THE 5TH MULTIDISCIPLINARY CONFERENCE ON REINFORCEMENT LEARNING AND DECISION MAKING

RLDM.ORG

RLDM2022

JUNE 8-11, 2022
BROWN UNIVERSITY

Invited speakers:

JOSH TENENBAUM (MIT)
MARCELO MATTAR (UCSD)
JILL O'REILLY (OXFORD)
NAO UCHIDA (HARVARD)
MELISSA SHARPE (UCLA)
NICOLA CLAYTON (CAMBRIDGE)
FREDERIKE PETZSCHNER (BROWN)
ORIEL FELDMANHALL (BROWN)
SCOTT NIEKUM (UT AUSTIN)
SATINDER SINGH BAVEJA (MICHIGAN AND DEEPMIND)
STEFANIE TELLEX (BROWN)
MARTHA WHITE (ALBERTA)
MATTHEW GOMBOLAY (GEORGIA TECH)
JEANNETTE BOHG (STANFORD)
JAKOB FOERSTER (FACEBOOK AI RESEARCH)

Program chairs:

ROSHAN COOLS & PETER STONE

Sponsors:

Sony AI
Microsoft
DeepMind
AI
ROBERT J. & NANCY D. CARNEY INSTITUTE FOR BRAIN SCIENCE BROWN UNIVERSITY
VECTOR INSTITUTE

Office of University Communications | MP-1646CARN22

CONTENTS

PREFACE

Welcome to Reinforcement Learning and Decision Making 2022!

Over the last few decades, reinforcement learning and decision making have been the focus of an incredible wealth of research in a wide variety of fields including psychology, animal and human neuroscience, artificial intelligence, machine learning, robotics, operations research, neuroeconomics and ethology. All these fields, despite their differences, share a common ambition—understanding the information processing that leads to the effective achievement of goals.

Key to many developments has been multidisciplinary sharing of ideas and findings. However, the commonalities are frequently obscured by differences in language and methodology. To remedy this issue, the RLDM meetings were started in 2013 with the explicit goal of fostering multidisciplinary discussion across the fields. RLDM 2022 is the fifth such meeting.

Our primary form of discourse is intended to be cross-disciplinary conversations, with teaching and learning being central objectives, along with the dissemination of novel theoretical and experimental results. To accommodate the variegated traditions of the contributing communities, we do not have an official proceedings. Nevertheless, some authors have agreed to make their extended abstracts available, which can be downloaded from the RLDM website.

We would like to conclude by thanking all past organizers, speakers, authors and members of the program committee. Your hard work is the bedrock of a successful conference.

We hope you enjoy RLDM2022.

Catherine Hartley and Michael L. Littman, *General chairs*
Roshan Cools and Peter Stone, *Program chairs*
Michael M. Frank and George Konidaris, *Local chairs*
Emma Brunskill, Peter Dayan, Yael Niv, Satinder Singh, Ross Otto and Rich Sutton, *Executive committee*
Quentin Huys and Marc Bellemare, *Area chairs*
Michael Browning and Katja Hoffman, *Workshop chairs*
Marcelo Mattar and Chelsea Finn, *Tutorial chairs*
Andrew Westbrook, *Awards and Spotlights chair*

Expressing Non-Markov Reward to a Markov Agent

David Abel
DeepMind
London, UK
dmabel@deepmind.com

André Barreto
DeepMind
London, UK
andrebarreto@deepmind.com

Michael Bowling
DeepMind
London, UK
bowlingm@deepmind.com

Will Dabney
DeepMind
London, UK
wdabney@deepmind.com

Steven Hansen
DeepMind
London, UK
stevenhansen@deepmind.com

Anna Harutyunyan
DeepMind
London, UK
harutyunyan@deepmind.com

Mark K. Ho
Princeton University
Princeton, NJ
mho@princeton.edu

Ramana Kumar
DeepMind
London, UK
ramanakumar@deepmind.com

Michael L. Littman
Brown University
Providence, RI
mlittman@cs.brown.edu

Doina Precup
DeepMind
London, UK
doinap@deepmind.com

Satinder Singh
DeepMind
London, UK
baveja@deepmind.com

Abstract

Markov Decision Processes are the standard model of sequential decision-making problems in reinforcement learning. However, as noted by Abel et al. [1], for some environments, there exist choices of task that cannot be expressed as a reward function that is Markovian on the environment's state space. We here address this limitation by studying a particular form of state-construction that is designed to systematically enrich the expressivity of reward. Concretely, we introduce the Split Markov Decision Process, a model of sequential decision-making problems with decoupled environment-state and reward-state, reminiscent of reward machines [2]. Using this model, we generalize one of the central questions of Abel et al. [1] regarding the expressivity of reward: given any task and Markovian environment, does there exist a reward function defined over some reward-state space that can express the task? Our main result answers this question in the affirmative for one type of case by offering a procedure that builds the realizing reward structures. We close by exploring basic aspects of reinforcement learning under these realizing reward structures in small-scale experiments, and call attention to open questions of interest.

Keywords: Reward, Reward Hypothesis, Expressivity, Markov Decision Process, MDP, Reinforcement Learning.

Acknowledgements

The authors would like to thank Hado van Hasselt, Alex Turner, Jacob Buckman, and Ben Van Roy for helpful conversations.

1 Introduction

Discrete time Markov Decision Processes (MDPs) are the standard model of environments in reinforcement learning (RL). Naturally, restricting attention to MDPs limits the space of representable RL problems—for instance, we might instead consider cases in which there is no available description of state, that time is not discrete, or that either the transition or reward function depend partially on history. Indeed, Abel et al. [1] explore the expressivity of Markov reward as a possible limitation to the space of tasks we can represent. Given an environment modeled as a Controlled Markov Process (CMP: Definition 1), Abel et al. [1] suppose that a designer (Alice) forms preferences over the CMP that she will translate into a reward function to incentivize a learning agent (Bob) to realize the chosen preferences. Abel et al. [1] then ask: for any choice of such preferences and CMP, will there always exist a Markov reward function that captures Alice's preferences?

One of the main results (Theorem 4.1) of Abel et al. [1] illustrates that there are two known types of failure cases in which the expressivity of Markov reward functions is lacking. The first they call the "steady state type" in which Alice holds preferences over events that occur with zero probability. For this reason, reward fails to elicit behavior that captures the given preferences, only because the behavior does not factor into the start-state value (or return) of the preferred behaviors. The second they call the "entailment type", in which the value of the desired preferences is entangled. For example, consider the "always move in the same direction" task in a grid world. Here, the CMP is a grid world with environment state defined according to a typical (x, y) pair. However, there is no Markov reward function that can properly incentivize an agent to prefer the four "go the same direction" policies above all alternatives—such reward functions depend on knowledge of either history or the future.

Results Overview. We here provide one kind of remedy to entailment issues. Our construction also yields a new and potentially interesting model of sequential decision-making problems we call the Split Markov Decision Process (Split-MDP) that bears heavy resemblance to *reward machines* [2]: a Split-MDP follows from the insight that rewards need not be based on the same notion of state as the environment's transitions. Concretely, a Split MDP is a CMP paired with an automaton-like structure we call a *reward bundle* (Definition 3) that produces reward. We use this model to provide a simple constructive proof that one aspect of the limited expressivity of Markov reward identified by Abel et al. [1] can be fixed by augmented state (or, said differently, that non-Markovian rewards do not suffer from entailment issues when encouraging Markov policies). Lastly, we conduct small-scale experiments that provide additional support to our findings, and highlight open questions of interest.

1.1 Preliminaries: CMPs, Tasks, and the Split-MDP

We first introduce relevant concepts needed to make our study precise, though we adopt many of the conventions from Abel et al. [1]. We begin with an environment described by a CMP, defined as follows.

Definition 1. *A **Controlled Markov Process** (CMP) is a model of an environment, $E = (\mathcal{S}_p, \mathcal{A}, p, \gamma, s_p^{(0)})$, where: $\mathcal{S}_p$ is a finite set of environment states, $\mathcal{A}$ is a finite set of actions, $p : \mathcal{S}_p \times \mathcal{A} \to \Delta(\mathcal{A})$ is a transition function, $\gamma \in [0, 1)$ is a discount factor, and $s_p^{(0)} \in \mathcal{S}_p$ is the environment start-state.*

Then, a designer (Alice) inspects the environment and forms some set of preferences over desired outcomes. Following Abel et al. [1], we will be initially focused on *sets of acceptable policies* (SOAPs), defined as follows, though our main result extends to policy orderings (POs) and trajectory orderings (TOs) defined by Abel et al. [1] as well.

Definition 2. *A **Set Of Acceptable Policies** (SOAP) is a non-empty subset of the deterministic policies, $\Pi_G \subseteq \Pi$, with Π the set of all deterministic mappings from $\mathcal{S}_p$ to $\mathcal{A}$ for a given E.*

As discussed, one of the central questions of Abel et al. [1] asks: for a given (E, Π_G) pair, will there exist a reward function that is Markov on $\mathcal{S}_p$ that can capture the given SOAP in E? A reward function *captures* the SOAP just when the start-state value induced by the given reward function adheres to the constraints of the SOAP. That is, the good policies all have strictly higher start-state value than the bad policies ("range" SOAP), or the good policies are all *optimal* and have strictly higher start-state value than the bad ("equal" SOAP). We here generalize this question to the case where Alice can choose not just a reward function, but also a state space for the reward function to operate over.

The Split Markov Decision Process. We next introduce two new structures: (1) A reward bundle (Definition 3): A collection of structures that produce reward, reminiscent of reward machines [2]; and (2) the Split-MDP (Definition 4): An MDP formed by a CMP paired with a reward bundle. This model is motivated by the observation that the basic laws of an environment might depend on *different* information than the reward function, and thus, we can decouple environment-state from reward-state.

Definition 3. *A **reward bundle** is a quadruple, $\mathscr{R} = (\mathcal{S}_r, r, f, s_r^{(0)})$, where: $\mathcal{S}_r$ is a finite set of reward states, $r : \mathcal{S}_r \times \mathcal{A} \times \mathcal{S}_r \to \mathbb{R}$ is a reward function, $f : \mathcal{S}_r \times \mathcal{S}_p \times \mathcal{A} \to \mathcal{S}_r$ is a deterministic reward-state dynamics function, and $s_r^{(0)} \in \mathcal{S}_r$ is the reward start-state.*

We may combine a CMP with a reward bundle to form a Split-MDP as follows.

Definition 4. *A **Split Markov Decision Process** (Split-MDP) is any MDP formed by pairing a CMP with a reward bundle, $M = (E, \mathscr{R})$, yielding $\mathcal{S} = \mathcal{S}_p \times \mathcal{S}_r$, $s^{(0)} = (s_p^{(0)}, s_r^{(0)})$, and $p_M : s_p \times s_r \times a \mapsto p(s_p, a), f(s_r, s_p, a)$.*

In cases where the reward-state space is unknown or unobservable to a learning algorithm, it might be natural to suppose an agent interacts with a Split-MDP but only observes the pair $(s_p^{(t)}, r(s_r^{(t)}))$. Such a setting induces a specific kind of partially observable MDP (POMDP) [3] in which the next observation, $s_p^{(t+1)}$, is predictable from $s_p^{(t)}$ and $a^{(t)}$ alone. We call such a model a *Split-POMDP*, and note that it defines a friendly class of POMDPs; we anticipate there are many interesting questions to pursue about Split-POMDPs beyond our scope.

2 Results.

First, we inspect whether there is *always* a reward bundle to realize a given SOAP. As mentioned in the introduction, we are focused on the "entailment cases", and set aside "steady state" issues by invoking the following assumption.

Assumption 1. *All tasks only contain preferences over outcomes that occur with non-zero probability.*

This assumption is just a concise way of limiting our attention to entailment cases, but we note that there are related arguments that go beyond this assumption that are out of scope for this paper.

Proposition 1. *Under Assumption 1, for any choice of CMP E and SOAP Π_G (with policies defined on $\mathcal{S}_p$), there exists a reward bundle $\mathscr{R}$ such that the optimal policies in $M = (E, \mathscr{R})$ are equivalent to those in Π_G.*

Proof of Proposition 1.

We are given a finite CMP, $E = (\mathcal{S}_p, \mathcal{A}, p, \gamma, s_p^{(0)})$, and a SOAP Π_G where each $\pi_g \in \Pi_G$ is a mapping from $\mathcal{S}_p$ to $\mathcal{A}$. We want to show that there is a reward bundle, $\mathscr{R} = (\mathcal{S}_r, r, f, s_r^{(0)})$, such that in the resulting Split-MDP $M = (E, \mathscr{R})$, the optimal policies, $\Pi_M^* = \arg\max_{\pi \in \Pi} V_M^\pi(s^{(0)})$, agree with the SOAP on each s_p,

$$(i) \quad \pi_M(s_p, s_r) = \pi_g(s_p), \ \forall_{s_p, s_r \in \mathcal{S}_p \times \mathcal{S}_r} \forall_{\pi_M \in \Pi_M^*} \exists_{\pi_g \in \Pi_G}, \tag{1}$$

$$(ii) \quad |\Pi_M^*| = |\Pi_G|. \tag{2}$$

We proceed by constructing such a reward bundle:

- *Reward state space.* Let $\mathcal{S}_r = \mathscr{P}(\Pi_G)$, where $\mathscr{P}(\mathcal{X})$ denotes the powerset of $\mathcal{X}$.
- *Reward start-state.* Let $s_r^{(0)}$ be Π_G.
- *Reward state dynamics.* We define f to remove any policy from Π_G that is inconsistent with the state-action taken, and to repopulate the full SOAP when all policies have been removed:

$$f(s_r, s_p, a) = \begin{cases} s_r \setminus \Pi_{\neq}(s_r, s_p, a) & |s_r| > 0 \\ \Pi_G & \text{otherwise.} \end{cases} \tag{3}$$

  where $\Pi_{\neq}(s_r, s_p, a)$ is the set of all policies in s_r that do *not* take a in s_p.
- *Reward function.* Lastly, let $r(s_r^{(t-1)}, a^{(t)}, s_r^{(t)}) = \mathbb{I}\left\{\exists_{\pi \in s_r} : \pi(s_p^{(t)}) = a^{(t)}\right\}$.

There are two key facts about this reward bundle. First, the reward function provides +1 reward to those state action pairs that *agree* with one of the acceptable policies. Second, the reward state dynamics function will *remove* any policy from the reward state that is inconsistent with an action taken. Thus, at any point in time, the *only* policies that remain in the reward state are those consistent with every action taken thus far. □

In some situations, we might imagine that the reward-state is either unknown or unobservable to the learning agent, thus inducing a Split-POMDP. We next highlight the fact that, by direct consequence of the previous construction, there will exist optimal policies in the Split-POMDP that only depend on environment-state.

Corollary 1. *Consider a Split-MDP constructed from an (E, Π_G) pair according to the procedure outlined in the proof of Proposition 1. When viewed as a POMDP with s_r the hidden state and s_p the observation, there will always exist an optimal deterministic policy that only depends on environment-state, s_p.*

Thus, when the agent only observes environment-state, there is still a *representable* optimal policy, unlike traditional POMDPs in which memoryless policies are known to be arbitrarily sub-optimal [6, 4]. We explore this consequence further in our experiments (subsection 2.1).

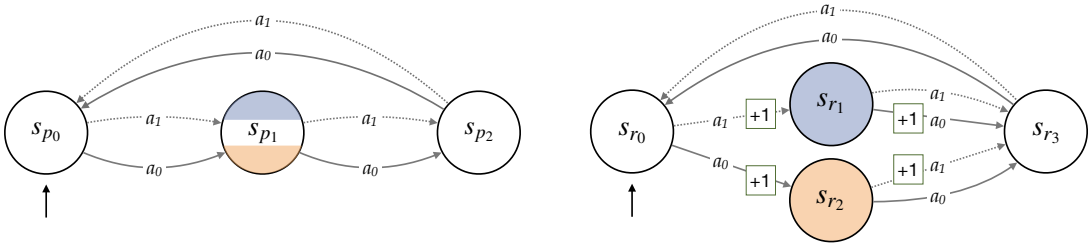
Furthermore, we find that the *trajectory ordering* and the *policy ordering* cases considered by Abel et al. [1] can be captured by a similar construction.

Corollary 2. *Under Assumption 1, for any choice of Split-CMP E and trajectory ordering $L_{\tau,N}$ or policy ordering L_Π (with trajectories and policies defined over $\mathcal{S}_p$), there exists a reward bundle $\mathscr{R}$ such that the start-state return of the trajectories of $L_{\tau,N}$ or start-state value of the ordering L_Π adheres to the constraints specified by the given task.*

We exclude the full proof due to space constraints, but note that the idea is a straightforward extension of the earlier proof. In the case of trajectory orderings, we define $\mathcal{S}_r$ to be the space of length $[1 : N]$ trajectories, and define $r : s_r \mapsto \mathbb{1}\{|s_r| = N\} \times (\text{RMAX} - \texttt{rank}(s_r))$ so that end-of-trajectory reward ranks the trajectories from best to worst. In the case of a policy ordering, let $\mathcal{S}_r = \mathscr{P}(L_\Pi)$, and provide reward each time-step proportional to the inverse-rank of the best policy still consistent with the behavior. Both constructions ensure that each ordering is respected.

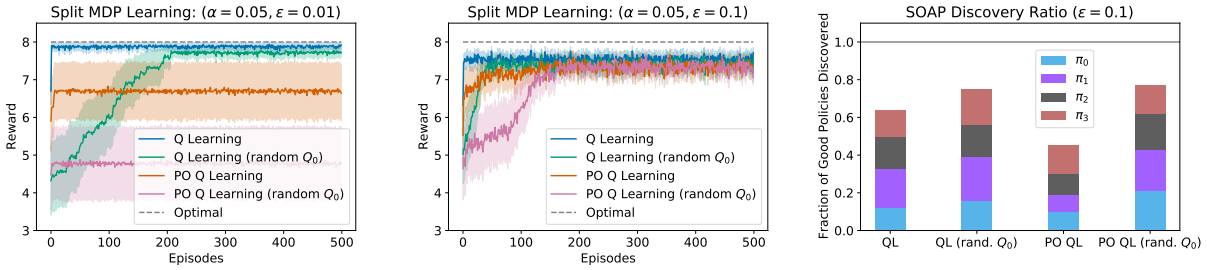
2.1 Experiments

We conduct experiments with a simple SOAP and CMP reminiscent of the XOR problem from Abel et al. [1]. The CMP has three environment-states and two actions, pictured in Figure 1a. The desired SOAP contains four policies, each requiring that the agent take *different* actions across two of the three environment-states. We construct a reward bundle with four-reward states (pictured in Figure 1b) according to the constructive procedure described by Proposition 1, and note that, as a consequence, there will exist four optimal policies over *just* s_p. Our experiments are intended to explore two questions. First, we examine learning curves of a variety of agents interacting with this environment to study the relationship between the representability of a good policy and its learnability—when viewed as a Split-POMDP, we know typical learning algorithms can *represent* each of the policies in the SOAP (by Corollary 1), but do not know whether learning these policies is also feasible (building on the results in Section 4.2 by McCallum [5]). Second, we inspect whether the proposed reward bundles can in fact incentivize learning algorithms to discover *any* of the acceptable policies, rather than just learn a single desired behavior. For simplicity, we experiment with tabular Q-learning of two variations: (1) When viewing the problem as an MDP, so $(s_p, s_r, r(s_r))$ are given as input each time step, and (2) When viewing the problem as a POMDP, so $(s_p, r(s_r))$ are given as input each time step. We further vary the initialization of Q between all zeros and uniform random from the interval $[0, 1]$, as well as different settings of the exploration parameter (ϵ, no annealing) used in ϵ-greedy action selection. We set the learning rate $\alpha = 0.05$ and discount factor $\gamma = 0.95$.



(a) Environment (b) Reward Bundle

Figure 1: The XOR-like environment used in the experiments (left), with the reward bundle (right) constructed by the procedure described in the proof of Proposition 1. The desired SOAP contains all policies that disagree on action choice across s_{p_0} and s_{p_1}. That is, $\Pi_G = \{\pi_{010}, \pi_{100}, \pi_{011}, \pi_{101}\}$, where π_{010} denotes $\{s_{p_0} \mapsto a_0 \mid s_{p_1} \mapsto a_1 \mid s_{p_2} \mapsto a_0\}$.



(a) $\epsilon = 0.01$ (b) $\epsilon = 0.1$ (c) SOAP Discovery Ratio

Figure 2: The first two figures present results from Q-learning interacting with an environment as a Split-MDP (blue) compared to a Split-POMDP (orange, "PO" prefix), when the rewards are generated from a well-constructed reward bundle as per Proposition 1. We further contrast performance relative to a randomly initialized Q function in both the MDP (green) and POMDP (pink, "PO" prefix) variations. The y-axis displays the mean, per-episode reward, averaged over 100 runs of the experiment with 95% confidence intervals, with optimal performance shown in the grey dashed line. The third plot illustrates the fraction of the four acceptable policies discovered by each learning algorithm in the final 50 episodes of learning across *all* runs of the experiment.

Results are presented in Figure 2. In Figure 2a and Figure 2b, we plot learning curves of all four agent varieties. We observe that in both zero-initialized and randomly-intialized learning for the standard MDP case (shown in blue and green, respectively), Q-learning can reliably discover optimal behavior, corroborating Proposition 1. In the POMDP case (orange and pink), the results suggest that, depending on ϵ, PO Q-learning will either achieve the *same level of performance* as its MDP counterpart (as in $\epsilon = 0.1$), or that there is a statistically significant gap separating the performance of the two (as in $\epsilon = 0.01$). This suggests that learning in a Split-POMDP is sometimes feasible; a natural direction for future work will further clarify the precise conditions under which effective learning is always possible. In Figure 2c, we visualize the fraction of time that each agent's greedy policy at the end of the episode is one of the SOAP policies (in just the final 50 episodes, averaged over 100 runs of the experiment). These results demonstrate that *all four* learning algorithms can reliably recover each one of the acceptable policies during learning, in roughly equal proportion. These results indicate that well-constructed reward bundles can in fact enhance what is learnable, even when the agent does not have access to the reward-state. A key direction for future work will identify simple learning procedures that can automatically construct agent-state to discover any kind of behavior expressible by reward.

2.2 Discussion

This work presents a simple state construction procedure that can enrich the expressivity of reward. Our results patch one of the holes identified by Abel et al. [1], and show that we can produce reward that is often conducive to learning. Lastly, we introduce the Split-MDP and Split-POMDP, which we believe may offer useful perspectives for studying state-construction, learning under partial-observability, and task complexity in RL.

References

[1] David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. On the expressivity of Markov reward. In *Advances in Neural Information Processing Systems*, 2021.

[2] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[4] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*, 1994.

[5] Andrew K. McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1996.

[6] Satinder Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the International Conference on Machine Learning*. 1994.

Human exploration balances approaching and avoiding uncertainty

Yaniv Abir
Department of Psychology
Columbia University
New York, NY 10027
yaniv.abir@columbia.edu

Michael N. Shadlen
Zuckerman Mind Brain and Behavior Institute,
Kavli Institute for Brain Science &
Howard Hughes Medical Institute
Columbia University
New York, NY 10027
shadlen@columbia.edu

Daphna Shohamy
Department of Psychology,
Zuckerman Mind Brain and Behavior Institute &
Kavli Institute for Brain Science
Columbia University
New York, NY 10027
ds2619@columbia.edu

Abstract

Humans are adept at exploring environments in which rewards are sparse, gathering information pertinent to their goals even if it cannot be immediately exploited to gain reward. We know very little about the computational basis of this capacity, since most studies of human exploration focus on tasks with immediate rewards for every choice. With immediate reward, value exploitation dominates behavior, rendering exploration too rare to examine. We developed a goal-directed exploration task in which information gathering is independent of value exploitation. We asked what computational principle guides participants in choosing between potential learning experiences, and how choice strategy is modulated by the computational difficulty of the task. To this end, we compared participants' choices to three hypothesized strategies, from sophisticated to simplified: (i) maximizing information gain, (ii) choosing the object associated with the highest current uncertainty, (iii) simply balancing the number of interactions with each object.

We found that current uncertainty was the best predictor of choice. Crucially, exploration was also strongly modulated by participants' overall knowledge of the goal, measured as their total uncertainty for both choice options. When participants' total uncertainty was low they chose the more uncertain option, as hypothesized. However, when total uncertainty was high, they avoided the more uncertain option, thereby slowing down the rate of incoming information. This strategy is accordant with managing mental effort of decision-making by reducing choice-switching costs. Indeed, participants preferred to repeat previous choices, and took longer to make choices counteracting this tendency. Altogether, our findings demonstrate that human exploration strategies are tailored to the limited computational capacities of our minds.

Keywords: exploration, decision making, capacity limitation, rational analysis, multi-armed bandit task

Acknowledgements

We are grateful for funding support from the NSF (award #1822619), NIMH/NIH (#MH121093) and the Templeton Foundation (#60844).

1 Introduction

Humans inhabit complex environments that are sparse in reward. To achieve most of our goals we have to learn the structure of the environment without the benefit of experiencing repeated extrinsic reward. Instead, we choose between potential learning experiences that carry no value per se, accumulating knowledge which may be used later to gain reward. Thus, before going to the theatre we read reviews and elicit recommendations from friends, rather than actually watching all running plays. We find out whether our choices were satisfactory only once, at the end of the night.

Most studies of human exploration, however, focus on tasks with immediate rewards for every choice [1–5]. Under such conditions, behaviour is dominated by the motivation to exploit existing knowledge, constricting the scientist's ability to observe and explain exploratory choices made to gain new knowledge. Thus, while much is known about how human choices are driven by sequential reward during learning, much less is known about how humans explore environments to gain new knowledge when rewards are not immediately available.

To examine human exploration, we developed a task allowing participants to learn about the environment through repeated interactions, gathering information but gaining no immediate reward. Participants had to maximize their learning in preparation for a future test, performance on which would be rewarded. We asked what computational strategy underlies participants' choices during exploration, and whether this strategy is modulated by the demands of the task.

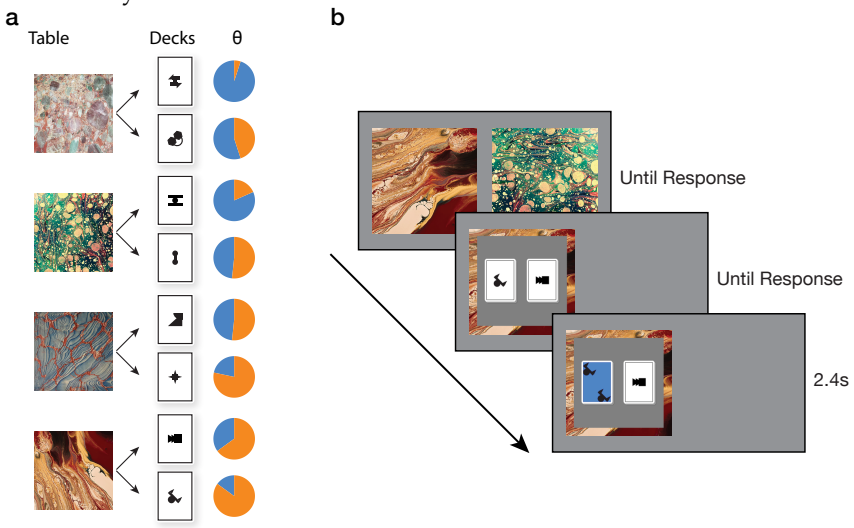


Figure 1: Task structure. (a) Participants explored four tables, each containing two decks with different proportions of blue/orange cards θ. The goal was to learn which deck had the higher (or lower) θ on each table, that is to learn $sgn(\Delta\theta_i)$, the sign of the difference between θs. (b) On a single exploration trial, participants chose between two tables, and then sampled a card from one of the decks on that table, observing its colour.

2 Task

The task simulated a room with four tables, with two decks of cards on each table. If a card was flipped, it was revealed to be either orange or blue. The proportion of orange vs. blue cards, θ, differed between the two decks on each table. Participants' goal was to learn $sgn(\Delta\theta_i)$, or which deck had had more orange (blue) cards on each table (Fig. 1a).

The task began with an exploration phase, followed by a test phase. On each trial of the exploration phase participants chose between two tables, and then chose to reveal one card from a deck on the table they had chosen (Fig. 1b). Participants were instructed that the exploration phase would be followed by a test phase after a random number of trials (drawn from a geometric distribution with rate $\frac{1}{45}$ to discourage pre-planning). During test, one of the colours was designated as rewarding. Participants were asked to indicate which deck had more of the rewarding colour on each table. For every correct choice they received \$0.25. Crucially, they received no reward during exploration, and were only rewarded for test performance.

A pre-registered sample of 194 participants played 22 rounds of our task over four online sessions. Each round employed new tables, decks and colours. We focus our analysis on table choices during exploration, since our candidate hypotheses do not make differing predictions for deck choices.

3 Hypotheses and models

3.1 Exploration strategy

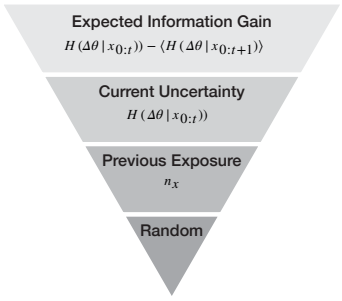


Figure 2: The hierarchy of considered exploration strategies.

To explain how participants choose between tables in the exploration phase, we employed rational analysis [6, 7] and derived the optimal exploration strategy, and two simpler approximate strategies (Fig. 2). These three strategies serve as benchmark hypotheses against which we compare participants' actual choices.

The optimal strategy, given at the top of Fig. 2, is choosing the table affording maximal **expected information gain** (EIG) [8–10]. EIG is the difference between the uncertainty in the value of the learning desideratum $sgn(\Delta\theta_i)$ given observed cards $x_{0:t}$, and the

expected uncertainty after observing the next card on trial $t + 1$. (We measure uncertainty using the entropy of the posterior).

Computing the second term in the EIG formula requires averaging over future unseen outcomes, which may be beyond the ability of participants. Neglecting this term amounts to simply choosing the table with the highest **current uncertainty** (Fig. 2, second tier) [5]. This has intuitive appeal: choose the table you know the least about. And yet, it might be implausible to expect participants to calculate uncertainties. A simpler heuristic is given on the third tier of Fig. 2: choosing the table with the least prior **exposure**, measured as the number of already observed cards n_x (sometimes known as the Upper Confidence Bound algorithm [11]). Lastly, participants may be random, rather than directed, explorers (e.g. employing ϵ-greedy, softmax, or Thompson sampling algorithms [2, 4, 5]).

To evaluate these three strategies, we derived each of the three quantities from an ideal Bayesian observer model that formed beliefs about θ based on the actual card sequence each participant had observed. We then tested whether the difference in the hypothesized quantity between the two choice options predicted participants' choices. We used regularized multilevel logistic regression models to this end.

3.2 Capacity limitations

Computing any of the strategies above may be taxing under limited resources. How do participants adapt to these limitations? One possibility is that when it is difficult to decide what to explore, participants' choices become more random [5]. In contrast, greater difficulty may prompt participants to adopt systematically biased heuristics that ameliorate the strain on cognitive resources [12].

Our main index of computational difficulty is the total uncertainty for both choice options. When total uncertainty is high, little is known about either option, and deciding between them has been empirically shown to be difficult [5]. We examine choice strategy along the continuum of total uncertainty, testing whether choices become noisier or more biased under high total uncertainty.

4 Results

4.1 Exploration strategy

Using approximate leave-one-out cross-validation for model comparison, we find that current uncertainty is the best predictor of participants' exploratory choices (Fig. 3d). Plotting the data confirms this conclusion: Current uncertainty for the table presented on the right relative to the table presented on the left predicts whether participants chose the table on the right as opposed to the table on the left (Fig. 3a). EIG provides a poorer fit to choices (Fig. 3b), and exposure was anti-correlated with choice, in exact opposite to the hypothesized relation (Fig. 3c).

Reaction times (RTs) for table choices further validated the role of current uncertainty as the decision variable [13] participants were using to make their choices: RTs were longer when the absolute difference in current uncertainty between the choice options was smaller b=-6.1×10^{-3}, 95% posterior interval (PI)=[-11.1×10^{-3},-1.2×10^{-3}].

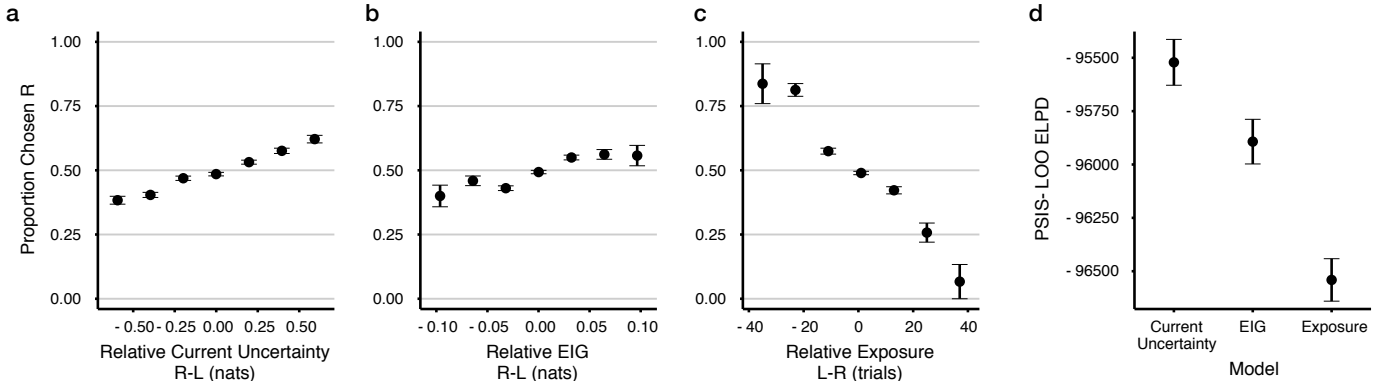


Figure 3: Current uncertainty is the best predictor of choice. (a) Current uncertainty predicts exploration-phase choices. The difference in uncertainty between the table on the right and the table on the left predicts choices of the table on the right. (b) Using the same method to plot choices as a function of EIG differences reveals that EIG fits the data less well. (c) The relationship between exposure and choice is negative, rather than the hypothesized positive correlation. (d) Model comparison shows that of the hypothesized strategies current uncertainty is the best predictor of exploratory choices: approximate leave-one-out cross-validation probability is highest for current uncertainty. Error bars ± 1 s.e.

4.2 Capacity limitations

From a normative perspective, exploration strategy should be exactly the same across all levels of total uncertainty: relative uncertainty should always be positively correlated with choice. Instead, we find a systematic bias in strategy: When total uncertainty was low or medium, participants chose the most uncertain table, as hypothesized. But when total uncertainty was high, they chose the least uncertain table, thereby slowing the rate of information-intake (Fig. 4). We validated this observation using a piecewise-regression model, allowing for the influence of relative uncertainty on choice to differ below and above a fitted threshold of total uncertainty. We observe a positive relationship between relative uncertainty and choice below the threshold b=0.24, 95% PI=[0.20,0.27]. But above the threshold we find that a significant negative interaction between relative and total uncertainty predicted choices b=-35.25, 95% PI=[-43.43,-28.31]. The value of the threshold is estimated to be 1.28 nats of total uncertainty (95% PI=[1.27,1.29]; the range of total uncertainty is 0 to 1.38).

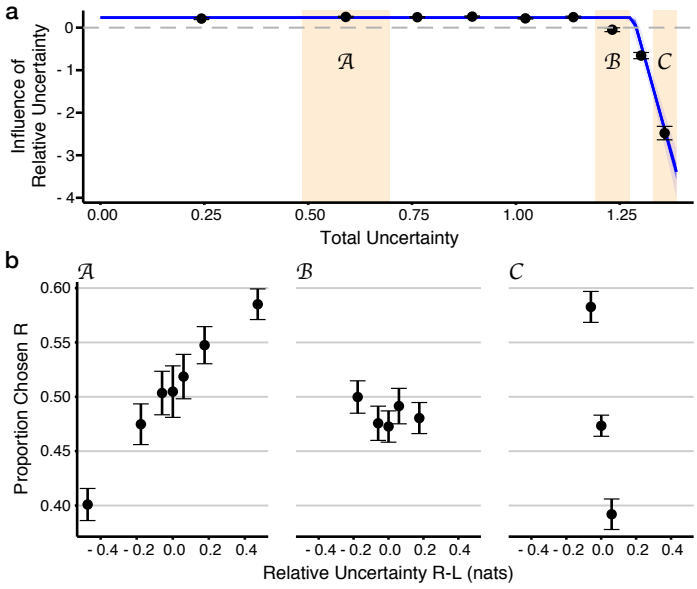


Figure 4: Total uncertainty regulates uncertainty approach vs. avoidance. (a) The influence of relative uncertainty on choice as a function of total uncertainty. The line marks the median prediction from a breakpoint regression model. Dots denote the median prediction for quantile bins. Ribbon and error bars denote 50% PI. Three regions of total uncertainty are marked in beige: low [A], medium [B] and high [C]. (b) Choice as a function of relative uncertainty is plotted for the three regions marked in panel a. Under low total uncertainty [A] participants tend to choose the table they are most uncertain about, as hypothesized. But that relationship is broken for medium levels of total uncertainty [B]. For high total uncertainty [C], participants strongly avoid the table they are most uncertain about. Error bars ± 1 s.e.

Thus, when total-uncertainty is high, participants avoid learning about uncertain tables. Does this strategy hinder their performance? We examined individual differences in exploration and test performance to answer this question. Pursuing relative uncertainty during the exploration phase is strongly associated with better test performance b=0.59, 95% PI=[0.54,0.65]. We do not see a test performance decrement for participants exhibiting greater uncertainty avoidance under high total-uncertainty. Participants who start avoiding relative uncertainty at a lower total uncertainty threshold actually have a weak tendency to do better at test b=-0.06, 95% PI=[-0.10,-0.01]. The magnitude of uncertainty avoidance under high total uncertainty is not associated with test performance b=0.02, 95% PI=[-0.02,0.07]. Thus, modulating strategy according to total uncertainty is not maladaptive.

This modulated strategy can be described as a tendency to revisit the best-learnt tables when total uncertainty is high. Indeed, we find a related pattern of choice that holds independently across all levels of relative or total uncertainty: Participants prefer to choose again the table they had last chosen. This tendency to repeat choices (b=0.27, 95% PI=[0.23,0.31]) is also reflected in RTs, which for repeat choices are shorter (b=-3.5×10^{-3}, 95% PI=[-4.0×10^{-3},-3.0×10^{-3}]) and less dependent on relative uncertainty (b=5.0×10^{-3}, 95% PI=[2.4×10^{-3},0.7×10^{-3}]). Hence, repeating a choice seems to be an additional heuristic participants employ to avoid the deliberation involved in choosing according to relative uncertainty.

5 Discussion

We examined the cognitive computations behind exploratory choices in a setting allowing for incremental memory-based learning. We find that overall, participants chose to learn more about the objects they were more uncertain about. However, when total uncertainty was high, participants chose to avoid objects with high uncertainty, learning instead about the objects they knew more about.

In this experiment, total uncertainty was correlated with the number of cards observed. While our results hold when trial number is added as a covariate to the regression models, only a version of the task in which total uncertainty and trial number are orthogonal can disentangle the contribution of each factor to uncertainty avoidance. We plan to employ such a task in our future work.

Avoiding uncertainty during learning amounts to down-regulating the rate of information intake, an ostensibly suboptimal strategy. The theory of resource rationality purposes that such deviations from optimality may be explained as adaptive to managing limited resources [12]. This pattern of behaviour is conceptually similar to confirmation biases in reasoning [7, 14] and the preference for massed over spaced learning documented in the meta-cognition literature [15]. This variety of phenomena suggests that balancing approaching and avoiding uncertainty is a pervasive principle of exploration.

Human exploration is often cast in terms of epistemic knowledge building. In contrast, studies of exploration in animals often focus on the balance between exploratory and fear-related responses [16, 17]. For example, it has been shown that rats exploring a novel arena [18, 19] or whisking to identify nearby objects [20] alternate between uncertainty approaching and safer, uncertainty avoiding strategies. Our findings suggest that the trade-off between exploration and avoidance is also a useful framework for understanding human behaviour.

References

1. Tversky, A. & Edwards, W. Information versus reward in binary choices. *Journal of Experimental Psychology* **71.** Place: US Publisher: American Psychological Association, 680–683. ISSN: 0022-1015 (1966).
2. Daw, N. D., O'doherty, J. P., Dayan, P., Seymour, B. & Dolan, R. J. Cortical substrates for exploratory decisions in humans. *Nature* **441.** Publisher: Nature Publishing Group, 876–879 (2006).
3. Cohen, J. D., McClure, S. M. & Yu, A. J. Should I stay or should I go? How the human brain manages the trade-off between exploitation and exploration. *Philosophical Transactions of the Royal Society B: Biological Sciences* **362,** 933–942. ISSN: 09628436 (2007).
4. Wilson, R. C., Geana, A., White, J. M., Ludvig, E. A. & Cohen, J. D. Humans use directed and random exploration to solve the explore–exploit dilemma. *Journal of Experimental Psychology: General* **143,** 2074 (2014).
5. Schulz, E. & Gershman, S. J. The algorithmic architecture of exploration in the human brain. *Current Opinion in Neurobiology* **55,** 7–14. ISSN: 18736882 (2019).
6. Anderson, J. R. *The adaptive character of thought* (Psychology Press, 1990).
7. Oaksford, M. & Chater, N. A Rational Analysis of the Selection Task as Optimal Data Selection. *Psychological Review* **101,** 608–631. ISSN: 0033295X (1994).
8. MacKay, D. J. C. Information-based objective functions for active data selection. *Neural computation* **4,** 590–604 (1992).
9. Gureckis, T. M. & Markant, D. B. Self-directed learning: A cognitive and computational perspective. *Perspectives on Psychological Science* **7,** 464–481 (2012).
10. Yang, S. C. H., Wolpert, D. M. & Lengyel, M. Theoretical perspectives on active sensing. *Current Opinion in Behavioral Sciences* **11,** 100–108. ISSN: 23521546 (2016).
11. Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* **3,** 397–422 (2002).
12. Lieder, F. & Griffiths, T. L. Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. en. *Behavioral and Brain Sciences* **43.** Publisher: Cambridge University Press. ISSN: 0140-525X, 1469-1825 (2020).
13. Shadlen, M. N. & Kiani, R. Decision making as a window on cognition. *Neuron* **80,** 791–806. ISSN: 08966273. `http://dx.doi.org/10.1016/j.neuron.2013.10.047` (2013).
14. Michel, M. & Peters, M. A. K. Confirmation bias without rhyme or reason. en. *Synthese* **199,** 2757–2772. ISSN: 1573-0964. `https://doi.org/10.1007/s11229-020-02910-x` (2021).
15. Simon, D. A. & Bjork, R. A. Metacognition in motor learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **27.** Publisher: American Psychological Association, 907 (2001).
16. Corey, D. T. The determinants of exploration and neophobia. *Neuroscience & Biobehavioral Reviews* **2.** Publisher: Elsevier, 235–253 (1978).
17. Ahmadlou, M. *et al.* A cell type–specific cortico-subcortical brain circuit for investigatory and novelty-seeking behavior. *Science* **372.** Publisher: American Association for the Advancement of Science, eabe9681 (2021).
18. Eilam, D. & Golani, I. Home base behavior of rats (Rattus norvegicus) exploring a novel environment. en. *Behavioural Brain Research* **34,** 199–211. ISSN: 0166-4328 (1989).
19. Botta, P. *et al.* An Amygdala Circuit Mediates Experience-Dependent Momentary Arrests during Exploration. en. *Cell* **183,** 605–619.e22. ISSN: 0092-8674 (2020).
20. Gordon, G., Fonio, E. & Ahissar, E. Emergent Exploration via Novelty Management. en. *Journal of Neuroscience* **34,** 12646–12661. ISSN: 0270-6474, 1529-2401 (2014).

Modular Policy Composition with Policy Centroids

Sandesh Adhikary
Paul G. Allen School of Computer Science
University of Washington
adhikary@cs.washington.edu

Byron Boots
Paul G. Allen School of Computer Science
University of Washington
bboots@cs.washington.edu

Abstract

We consider the task of aggregating policies in multi-objective decision making problems where multiple policies are trained to accomplish potentially conflicting tasks. While policy composition is a crucial component of multi-objective problems, commonly used techniques are restricted to simple averages that do not adhere to or exploit the structure of policy spaces. We present a new framework for policy composition viewing the problem as computing centroids in distance spaces where policies are embedded. These *policy centroids* not only subsume various existing composition techniques, but also provide a new means of inducing useful properties in composite policies through judicious choices of embedding spaces and distances. Additionally, we introduce policy centroids that extend existing compositions to new problem settings. For deterministic policies, we use distances between state-action value functions to define utility centroids that can be tuned to adjust the intensity of individual policy preferences. For stochastic policies, we introduce policy centroids based on statistical distances including the maximum mean discrepancy, which are particularly useful when policies are accessible only through samples. We evaluate our proposed policy centroids on various illustrative examples that highlight their benefits over existing approaches.

Keywords: policy composition, multi objective reinforcement learning, modular reinforcement learning

1 Introduction

As we continue to expand the scale and complexity of automated decision making, it becomes increasingly difficult to quantify all desired objectives into a single monolithic goal. We are routinely faced with multiple competing objectives with no concrete notion of a composite goal. Autonomous vehicles must quickly reach their destinations while avoiding crashes; economic policy should bolster growth but also minimize inequality; financial investments need to maximize returns while maintaining a diverse portfolio. Even when a composite objective does exist, it may still be difficult to learn a policy to accomplish it. Various works in multi-objective reinforcement learning have demonstrated the benefit of training separate agents on decompositions of complex reward signals. In robotics, a complex robot can be decomposed into multiple sub-agents, each reacting to changes in its local environment [1]. Even when tackling a single objective, we can benefit from training an ensemble of policies using different algorithms [2].

The problem of modular policy composition has been tackled in various fields, with numerous proposed solutions including sums and products of experts [2], compositions of utilities or potential fields [3], voting based ranking, and so on. The central idea behind these compositions is to form an average policy from a set of recommendations. However, existing approaches tend to use fairly limited notions of averages, generally restricted to arithmetic means, geometric means, and extremums. We consider the task of averaging policies in its full generality as computing centroids over arbitrary distance spaces that minimize distances to individual policies. This perspective can subsume many existing compositions, and also serve as a principled framework to design new compositions by picking appropriate embedding spaces and distances. We consider three existing approaches to policy composition, and show how policy centroids not only generalize them, but can also extend them to new problem settings.

2 Policy Centroids

We tackle policy composition as finding a single-policy solution to multi-objective Markov decision processes (MOMDPs). A MOMDP is a tuple $(\mathcal{S},\mathcal{A},T,\gamma,R)$ of state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T : S \times \mathcal{A} \times \mathcal{A} \to [0,1]$, discount factor $\gamma \in [0,1]$, and a *vector-valued* reward function $R : S \times A \times S \to \mathbb{R}^T$. The components of the rewards vector r_i are the rewards for the T separate objectives. A policy π determines the action executed at a given state; a deterministic policy $\pi(s) : \mathcal{S} \to \mathcal{A}$ returns a single action, and a stochastic policy $\pi(a|s) : \mathcal{A} \to [0,1]$ is a distribution over $\mathcal{A}$. For a state-action pair (s,a), $Q_i(a|s) = \mathbb{E}_{\pi_i}(\sum_{t=0}^{\infty}\gamma^t([r_i]_t)|s_0 = s, a_0 = a)$ denotes the state-action value function under the policy π_i for the i-th objective.

Policy Centroids Consider a set $\{\pi_i\}_{i=1}^T$ of T policies, such that each policy π_i has been formulated separately based only on the i-th component of the reward function. These policies must now be combined into a single composite policy $\bar{\pi}$, without assuming access to a composite reward function or a known decomposition of the composite objective into individual objectives. Let (M_i,d_i) be distance spaces[1] where π_i can be compared with alternatives via distances d_i. Policies are mapped onto these spaces using policy embeddings $\mu_i(\pi)$. Finally, given a set of optional non-negative weights $\{w_i\}_{i=1}^T$ with $\sum_i w_i = 1$, the composite policy $\bar{\pi}$ or the *policy centroid* is defined as follows

$$\bar{\pi} = \operatorname*{argmin}_{\pi} \sum_i w_i d_i(\mu_i(\pi_i), \mu_i(\pi)). \tag{1}$$

To design a policy centroid, we pick (1) embeddings μ_i that capture important features of policies, and (2) distances d_i that account for meaningful differences between them. The weights w_i are linear relative importances of the T objectives. Such weights are used in many existing policy compositions as the primary mode of controlling the behavior of the composite policy. These weights are either set using knowledge of the composite objective or learned with respect to a composite reward; otherwise, they are set to be uniformly distributed. Policy centroids provide additional means of affecting properties of the composite policy through the choice of the distance spaces (M_i,d_i). In the following sections, we show how various existing compositions can be interpreted as policy centroids. Moreover, we provide examples where different choices of embeddings μ_i and distances d_i can lead to composite policies with useful properties, and extend existing compositions to new problem settings.

2.1 Utility Centroids

In value-based reinforcement learning (RL), we often have access Q-values $Q(a|s)$ serving as utility maps over actions $a \in \mathcal{A}$ for a given state s, which are natural choices for policy embeddings. Unsurprisingly, averaging Q-values is already a popular composition strategy known as *utility fusion* [3], where the arithmetic average of Q-values is treated as the composite Q-value. We introduce *utility centroids* that subsume utility fusion. Given an exponent $\beta > 0$, the utility centroid[2] is defined as

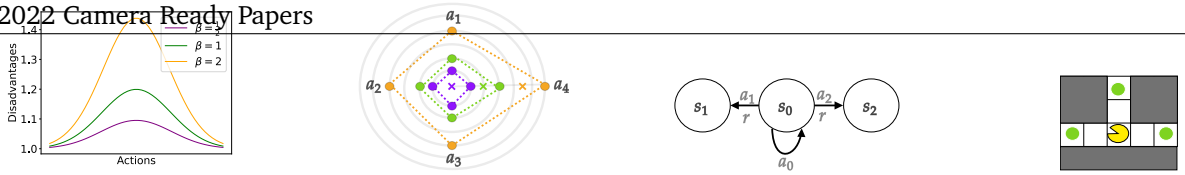
$$\bar{a} = \operatorname*{argmin}_{a} \sum_i w_i (Q_i(a_i|s) - Q_i(a|s) + 1)^{\beta}. \tag{2}$$

Here, task-specific embeddings are simply the Q-values $\mu_i(a|s) = Q_i(a|s)$ and distances are $d_i(Q_i(a_1|s), Q_i(a_2|s)) = (|Q_i(a_1|s) - Q_i(a_2|s)| + 1)^{\beta} - 1$. Equation 2 minimizes the average disadvantage or opportunity cost $(Q_i(a_i|s) - Q_i(a|s) + 1)^{\beta}$ of picking a over the preferred $a_i = \operatorname{argmax}_a Q_i(a|s)$. As illustrated in Figure 1a, the exponent β controls the intensity of individual preferences. Setting $\beta > 1$ (resp. $\beta < 1$) magnifies (resp. minimizes) differences in disadvantages leading to sharper (resp. flatter) disadvantage curves.

The exponent β plays a similar role to weights w_i; while w_i stretch or squeeze distances between policies linearly, β magnifies or minimizes distances between policies exponentially. However, unlike weights w_i, we can set the same exponent β across all policies and still affect relative distances. In Figure 1b, filled-circles of the same color correspond to the recommended

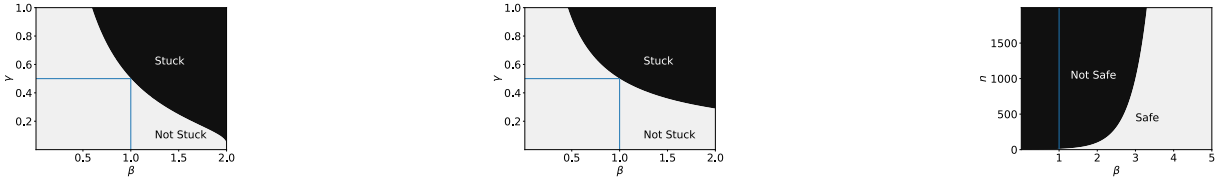
[1] A distance space (M,d) is a set M equipped with a non-negative, symmetric, and reflexive distance function $d(x,x')$ for $x,x' \in M$.

[2] The optional offset +1 simply sets the domain of the exponential to $\mathbb{R}_{\geq 1}$, avoiding the inverse behavior of the exponent for $(0,1)$.



(a) (b) (c) (d)

Figure 1: (a-b) Varying β in utility centroids in Equation 2 (c-d) Illustrations of MDPs with attractor states.



(a) Attractors: 3 state MDP (b) Attractors: PacBoy (c) Tyranny of the majority

Figure 2: Utility centroids to prevent the tyranny of the majority and escape attractors. The blue lines are reference markers for utility fusion, or equivalently utility centroids ($\beta=1$)

actions from four policies for a given choice of β. When $\beta=1$ (green), the policy recommending action a_4 has a slightly lower disadvantage, pushing the centroid closer to a_4. With $\beta>1$ (orange) differences in disadvantages are magnified, pushing the centroid further towards a_4. With $\beta<1$ (purple), the small difference in disadvantages is minimized, and the centroid is not biased towards any one policies. With $\beta=1$, Equation 2 reduces to standard utility fusion. Thus, β can serve as a useful hyperparameter, especially in the absence of non-uniform weights w_i. As we now discuss, adjusting β can alleviate some known problems with utility fusion in such settings.

Preventing Attractors When the Q-values of multiple policies are trained *egocentrically* (without taking into account the composite Q-value), utility fusion with uniform weights w_i overestimates the value of states where policies disagree on the optimal action [4]. For large discount factors γ, these states can become attractors where the agent gets stuck; restricting us to myopic suboptimal policies with low discount factors. In Equation 2, reducing β essentially minimizes disagreement between policies and can help us escape attractors. To demonstrate, we consider the two examples from [4] illustrating attractors in utility fusion. In the first example (Figure 1c), we consider a 3 state deterministic MOMDP where the agent in state s_0 can stay in place with action a_0, or move to one of two terminating goal states s_1 and s_2 with actions a_1 and a_2 respectively. We define separate objectives with reward $r>0$ for reaching s_1 and s_2 respectively, and no reward for staying in place. In the second example in Figure 1d, the PacBoy agent is equidistant from a reward in either of the three directions $\{\uparrow,\leftarrow,\rightarrow\}$ in a simple deterministic MOMDP; choosing $\downarrow$ keeps it in place. In both examples, egocentric utility fusion with uniform weights w_i gets stuck when $\gamma>0.5$ [4]. In Figures 2a and 2b, we plot the combinations of γ and β for which the policy *centroid* in Equation 2 is stuck (shaded black) and for which it reaches one of the two goal states (shaded white). As we decrease β, we can allow larger discount factors while still avoiding getting stuck. However, using too small a β may also lead to undesirable behavior where the relative preferences of policies are ignored.

Preventing the tyranny of the majority Utility fusion can suffer from a tyranny of the majority, where an objective dominated by a large number of competing objectives may never be fulfilled. If the minority objective is safety critical, it may be desirable to forego the other goals to prevent an accident. If the objectives involve human stakeholders, this majority bias can be unfair – the minority stakeholder may lose confidence in the system if their objectives are consistently ignored in service of the majority. As an illustration, consider a set of T objectives $\{O_i\}_{i=1}^{T}$ defined over a MOMDP with two actions $\{a_1,a_2\}$; O_1 corresponds to avoiding accidents, while the others correspond to reaching various goals. At some state, O_1 identifies action a_2 as unsafe with a disadvantage of d. In contrast, a_2 is slightly favorable over a_1 for the goal-seeking objectives, assigning disadvantage $\epsilon<d$ for a_1. With only two tasks $\{O_1,O_2\}$, utility fusion opts for the safe action a_1 as it has lower disadvantage. However, as we add other objectives to the mix, the unsafe action a_2 is picked when $n>d/\epsilon$. With the utility centroid in Equation 2, we can increase β to intensify individual preferences, leading to a larger threshold. In Figure 2c, we plot various combinations of β and n for which the agent picks the unsafe action a_2 (shaded black) and the safe action a_1 (shaded white) when $\epsilon=0.1d$, i.e., each goal-seeking agent only favors the unsafe action by a small margin of 10%. However, as β becomes very large, the minority may exert too much influence, preventing the agent from ever achieving its other goals. Thus β will likely have to be tuned to balance these effects.

2.2 Stochastic Policy Centroids

We now turn to the problem of composing policies represented as probability distributions $\pi(a|s)$ over actions, which we refer to as stochastic policies. Such policies can arise, for instance, due to exploration or as solutions to partially observable MDPs or adversarial games. Two common approaches of composing a set of stochastic policies $\{\pi_i\}$ are through sums and products of experts:

$$\bar{a}=h(\bar{\pi}(a|s)) \qquad \text{Sum of Experts (SoE):} \quad \bar{\pi}\propto\sum_i \pi_i(a|s) \qquad \text{Product of Experts (PoE):} \quad \bar{\pi}\propto\prod_i \pi_i(a|s). \tag{3}$$

The function $h(\cdot)$ could be an argmax operation over the composite density $\bar{\pi}$, or a function drawing samples from it. While not always described as such, these compositions are equivalent to element-wise arithmetic and geometric means, which are special cases of generalized f-means, which, in turn, are instances of Fréchet centroids. We can thus formulate policy centroids

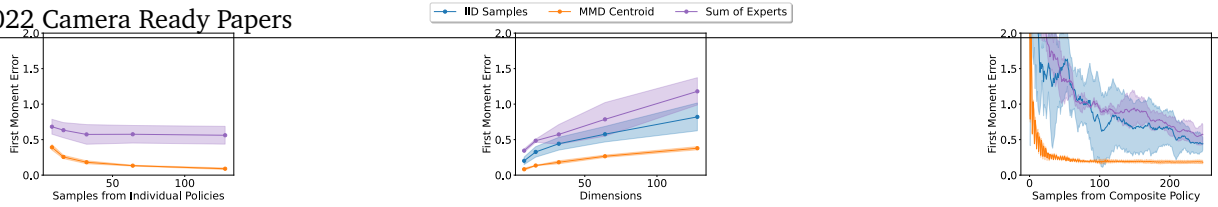


Figure 3: Errors for MMD centroids and SoE composition as a function of (left) samples from individual policies (middle) dimensionality of $\mathcal{A}$ and (right) samples from the composite policy. Error bars are standard deviations across 3 random seeds

corresponding to generalized-means-of-experts (GMoE) through a monotonic, continuous and invertible function f as follows:

$$\bar{\pi} \propto \operatorname*{argmin}_{\pi} \sum_i d^2(\pi_i, \pi) \quad \text{where} \quad d^2(\pi, \pi') = \sum_a |f(\pi(a|s)) - f(\pi'(a|s))|^2. \tag{4}$$

This centroid defines the well-known class of generalized f-means. With $f(x) = x^\beta$ (for $\beta \neq 0$), we get the family of *power means*, which includes SoE (arithmetic mean for $\beta = 1$) and PoE (geometric mean for $\beta \to 0$), as well as a spectrum of other means.

Composing Implicit Stochastic Policies While GMoEs are intuitive and ubiquitous, they may not always be ideal, for instance, when composing *implicit* stochastic policies. Implicit policies $\hat{\pi}$ are distributions without a known density function, but allow sampling. Such policies can be useful, for instance, to incorporate unknown noise distributions or as a flexible policy class for complex action distributions [5]. Given sets of actions $\{a_j^{(i)} \sim \hat{\pi}_i\}$ drawn from a set of implicit policies $\{\hat{\pi}_i\}$, we now wish to generate new actions $\{\bar{a}_m\}$ from the composite policy $\bar{\pi}$. As a concrete example, consider the following SoE composition problem where we wish to evaluate the expected value of some function $g(a)$ under the distribution of the composite policy. Assuming that both the function g and individual policies are computationally expensive to query, we seek a good estimate of $\mathbb{E}_{\bar{\pi}}\, g(a)$ with minimal samples from individual policies as well as the composite policy. SoEs are not directly applicable since they require the policies' probability densities. We would thus need some form of density estimation, which can be sample intensive for high-dimensional actions. With policy centroids, we can extend SoEs to this new problem setting. Specifically, we can embed implicit policies onto reproducing kernel Hilbert spaces, and use the maximum mean discrepency between distributions as our distance function.

MMD Policy Centroids The maximum mean discrepancy (MMD) is an integral probability (pseudo) metric (IPM) between probability measures that captures the maximal difference between expectations of functions. For a set of functions $\mathcal{G}$, the IPM between distributions π and π' is the supremum of the difference in expectations $\mathbb{E}_\pi(g) - \mathbb{E}_{\pi'}(g)$ over all $g \in \mathcal{G}$. We obtain the MMD when $\mathcal{G}$ is the unit-ball in a reproducing kernel Hilbert space (RKHS) [6]. For any symmetric positive-definite kernel $k(\cdot,\cdot)$ on a domain $\mathcal{A} \times \mathcal{A}$, there exists a corresponding RKHS $\mathcal{H}_k$ and k is the canonical feature mapping to $\mathcal{H}_k$. Additionally, we can use the kernel to also map probability distributions π defined over $\mathcal{A}$. These embedded distributions $\mu_\pi \in \mathcal{H}_k$, known as *kernel mean embeddings*, are defined as $\mu_\pi = \int k(a, \cdot) d\pi(a)$. The empirical estimates of these embeddings $\hat{\mu}_\pi = \frac{1}{n}\sum_{j=1}^n k(a_j \sim \pi, \cdot)$ converge in MMD to μ_π at a rate of $O(n^{-1/2})$ [6], which is independent of the dimensionality of $\mathcal{A}$ (in contrast to the curse of dimensionality for density estimation). The MMD between two kernel mean embeddings is the $\mathcal{H}_k$-norm of their difference: $\text{MMD}(\hat{\pi}, \hat{\pi}') = ||\, \hat{\mu}_\pi - \hat{\mu}_{\pi'} ||_{\mathcal{H}_k}$ Additionally, if the kernel is *characteristic* (e.g. Gaussian, Laplace), the MMD is a metric such that $\text{MMD}(\pi, \pi') = 0 \Leftrightarrow \pi = \pi'$ [6].

Given a kernel k, we define *MMD policy centroids* as $\bar{a} \sim \bar{\pi} = \operatorname{argmin}_{\hat{\pi}} \sum_i w_i \text{MMD}(\hat{\pi}_i, \hat{\pi}) = \operatorname{argmin}_{\hat{\mu}} \sum_i w_i ||\hat{\mu}_i - \hat{\mu}||^2_{\mathcal{H}_k}$. This objective is essentially what is optimized by the kernel herding algorithm [7] that draws samples from a kernel mean embedding; here, the target is instead the weighted sum $\sum_i w_i \hat{\mu}_i$. Using kernel herding, we can draw the t-th sample $\bar{a}_t$ from $\bar{\pi}$ as follows

$$\bar{a}_t = \operatorname*{argmin}_a \sum_{i,j} -2 w_i k(a_j^{(i)}, a) + I(t > 1) \sum_{m=1}^{t-1} \frac{1}{t} k(\bar{a}_m, a) \qquad \text{where } I(t>1) \text{ is the indicator function.} \tag{5}$$

While the first term in the objective encourages $\bar{a}_t$ to be similar to the set of samples $\{a_j^{(i)} \sim \hat{\pi}_i\}$, the second term encourages it to be *dissimilar* to samples $\{\bar{a}_m\}_{m=1}^{t-1}$ generated in prior iterations. This latter repulsive force improves sample diversity by inducing negative autocorrelations. For bounded kernels corresponding to finite dimensional $\mathcal{H}_k$, the herded *super* samples converge in MMD at a rate of $O(n^{-1})$ to the target distribution – faster than $O(n^{-1/2})$ for iid sampling from the true distribution [7]. Even though $\mathcal{H}_k$ is generally not finite, we still tend to get fast convergence in practice – even if Equation 5 is only solved approximately [7]. Thus, MMD centroids are well-suited for our purposes. Firstly, using the herded super samples, we should require fewer evaluations of g before $\mathbb{E}_{\bar{\pi}}(g)$ converges. Secondly, since we bypass an explicit density estimation, we should need fewer samples from individual policies as well. Moreover, since the convergence rate of the *implicit* density estimation $\hat{\mu} \approx \mu$ is independent of the dimensions of $\mathcal{A}$, MMD centroids should scale better to high-dimensional action spaces.

To demonstrate the advantages of MMD centroids, we consider a set of 3 Gaussian policies $\{\pi_i = \mathcal{N}(a_i, 0.1\mathbb{I})\}$ with mean actions $a_i \in \mathbb{R}^d$. Assuming uniform policy weights w_i, the composite distribution for the true weighted SoE policy is simply $\sum_i \frac{1}{3}\mathcal{N}(a_i, 0.1\mathbb{I})$. Our goal is to generate M samples $\{\bar{a}_m\}_{m=1}^M$ from the composite policy (only using samples from π_i) and use them to compute expected values of functions. Here, we fix the target function to be the first moment (mean action) of the composite policy, which is simply $\frac{1}{3}a_i$. In Figure 3, we compare estimation errors for traditional SoE, MMD centroids, as well as iid samples from the true composite policy. For MMD centroids, we use the Gaussian kernel with its bandwidth set using the median-trick heuristic, and generate samples via kernel herding. Since the true composite policy is generally unknown, we avoid tuning hyperparameters for the optimization in Equation 5 by opting for a brute force search within a random subset of M action samples drawn from individual policies. For the SoE composition (Equation 3), we draw samples from the kernel density estimate of the composite policy us-

ing a Gaussian kernel (with the median-trick bandwidth). In Figure 3, we fix $(d=32, M=250)$, $(M=250, N=32)$ and $(d=32, N=32)$ respectively, and plot errors with respect to varying N (samples from individual policies), d (dimensionality of actions), and M (samples from the composite policy). We see that MMD centroids offered better scaling with respect to all three parameters.

2.3 Riemannian Motion Policies

We have thus far focused on two policy compositions often encountered in multi-objective RL. We now turn our attention to a different setting within the framework of Riemannian motion policies (RMPs) from robotics, designed to combine multiple acceleration based policies for cohesive motion generation [1]. As described below, policy centroids share the intuition behind RMP-composition in using task-specific distance spaces, and can also extend RMP-composition to the setting of stochastic controllers.

RMP Composition While a robot's complete state is defined on a configuration space $\mathcal{Q}$, its various objectives can be defined on separate task spaces $\mathcal{X}_i$. Configuration space states $\mathbf{q}$ are mapped onto task-space states $\mathbf{x}_i$ via task-maps $\phi_i : \mathcal{Q} \to \mathcal{X}_i$. Using $\mathbf{J}_i$, the Jacobians of the task-maps ϕ_i, we can also map configuration space velocities and accelerations into their task-space counterparts as $\dot{\mathbf{x}}_i = \mathbf{J}_i \dot{\mathbf{q}}$ and $\ddot{\mathbf{x}}_i = \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}}$. A task-space RMP over $\mathcal{X}_i$ is a tuple $(\mathbf{f}_i, \mathbf{M}_i)$ indicating a recommended force $\mathbf{f}_i = \mathbf{M}_i \ddot{\mathbf{x}}_i$ given a PSD inertia matrix $\mathbf{M}_i$. The general idea behind the RMP framework is to combine task-space RMPs defining desired accelerations $\ddot{\mathbf{x}}_i$ into a composite RMP in the configuration space. Given a set of task-space RMPs $\{(\mathbf{f}_i, \mathbf{M}_i)\}$, the composite configuration-space RMP $(\mathbf{f}, \mathbf{M})$ is computed by *pulling-back* the task-space forces into the configuration space as $\texttt{pullback}(\mathbf{f}_i) = \mathbf{J}_i^T(\mathbf{f}_i - \mathbf{M}_i \dot{\mathbf{J}}_i \dot{\mathbf{q}}_i)$ and $\texttt{pullback}(\mathbf{M}_i) = \mathbf{J}_i^T \mathbf{M}_i \mathbf{J}_i$, and then adding up the pulled-back forces $\mathbf{f} = \sum_i^T \texttt{pullback}(\mathbf{f}_i)$ and inertia matrices $\mathbf{M} = \sum_i \texttt{pullback}(\mathbf{M}_i)$. The composite control or acceleration $\ddot{\mathbf{q}} = \mathbf{M}^+ \mathbf{f}$ is the solution to the following objective function

$$\operatorname*{argmin}_{\ddot{\mathbf{q}}} \frac{1}{2} \sum_{i=1}^{T} \| \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}} - \ddot{\mathbf{x}}_i \|^2_{\mathbf{M}_i} = \operatorname*{argmin}_{\ddot{\mathbf{q}}} \frac{1}{2} \sum_{i=1}^{T} \| \mu_i(\ddot{\mathbf{q}}) - \mu_i(\ddot{\mathbf{x}}_i) \|^2_{\mathbf{M}_i}. \tag{6}$$

In the second equality above, we have rewritten the RMP composition objective similar to a policy centroid with $\mu_i(\ddot{\mathbf{q}}) = \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}}$, where $\mu_i(\ddot{\mathbf{x}}_i) = \ddot{\mathbf{x}}_i$. Each term in this objective can be interpreted as a task-specific distance (defined via $\mathbf{M}_i$) between $\mu_i(\ddot{\mathbf{q}})$ and desired task-controls $\ddot{\mathbf{x}}_i$. Thus, policy centroids share RMP's underlying idea of employing task-specific mappings and computing centroids with respect to task-specific distances. However, policy centroids extend this idea beyond acceleration-based controls to generic settings including, for instance, discrete action spaces. Moreover, as we now discuss, the policy centroid perspective allows us to extend RMP composition to the case of stochastic controllers.

Wasserstein Policy Centroids The RMP framework assumes deterministic controls $\ddot{\mathbf{x}}_i$ or $\mathbf{f}_i$ and is not directly applicable when these controls are stochastic. Incorporating stochasticity into RMPs can be useful, for instance, when individual controllers are learned via RL with exploration, or to escape local stationary points where competing controls cancel each other out. To compose such stochastic RMP controllers, we can collect samples of desired controls from each individual RMP and pull them back onto the configuration space. As in Equation 6, the composite metric $\mathbf{M}$ defines a distance measure for pulled-back acceleration controls. In the deterministic setting with only one sample per RMP, we would essentially obtain the composite control formed via the metric-weighted averaging in Equation 6. But in the stochastic setting, we need to perform an average both in the space of controls (as in deterministic RMPs), but also in terms of probability masses spread across controls. Essentially, we require some form of stochastic averaging that can take into account the ground distance between controls in the pulled-back space, while also performing a probabilistic average. The family of Wasserstein distances between distributions is exactly such a statistical distance. The p-th Wasserstein distance $\mathcal{W}_p(\pi, \pi'; d)$ between two distributions π and π' defined with respect to a ground metric $d(\cdot, \cdot)$ represents the cost of transforming π into π' by moving probability mass in the ground space of samples; the cost of moving probability mass is measured using d. Using a Wasserstein distance as our statistical distance between policies, we can define Wasserstein policy centroids for RMPs with stochastic controllers as the Wasserstein barycenter of the individual pulled-back control distributions. Similar to how RMP composition computes sums or metric-weighted averages of controls, the Wasserstein policy centroids would allow us to average entire distributions over controls.

3 Conclusion

We presented a framework for modular policy composition, viewing the problem as computing centroids over distance spaces. We showed how this framework not only subsumes various existing policy compositions, but also provided examples of how it can be used to adapt them to new problem settings. In future work, we aim to implement the policy centroids presented here in large-scale policy composition problems, and also integrate policy centroids into policy learning pipelines.

References

[1] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "Rmpflow: A geometric framework for generation of multitask motion policies," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 968–987, 2021.

[2] M. A. Wiering and H. Van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 930–936, 2008.

[3] J. K. Rosenblatt, "Optimal selection of uncertain actions by maximizing expected utility," *Autonomous Robots*, 2000.

[4] R. Laroche, M. Fatemi, J. Romoff, and H. van Seijen, "Multi-advisor reinforcement learning," *arXiv:1704.00756*, 2017.

[5] Y. Tang and S. Agrawal, "Implicit policy for reinforcement learning," *arXiv preprint arXiv:1806.06798*, 2018.

[6] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *JMLR*, vol. 13, no. 1.

[7] Y. Chen, M. Welling, and A. Smola, "Super-samples from kernel herding," in *UAI*, 2010.

Prototyping three key properties of specific curiosity in computational reinforcement learning

Nadia M. Ady*
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8
Alberta Machine Intelligence Institute
nmady@ualberta.ca

Roshan Shariff
Department of Computing Science
University of Alberta
Alberta Machine Intelligence Institute
roshan.shariff@ualberta.ca

Johannes Günther
Alberta Machine Intelligence Institute
Department of Computing Science
University of Alberta
johannes@amii.ca

Patrick M. Pilarski
Department of Medicine
Department of Computing Science
University of Alberta
Alberta Machine Intelligence Institute
pilarski@ualberta.ca

Abstract

Curiosity for machine agents has been a focus of intense research. The study of human and animal curiosity, particularly *specific curiosity*, has unearthed several properties that would offer important benefits for machine learners, but that have not yet been well-explored in machine intelligence. In this work, we introduce three of the most immediate of these properties—directedness, cessation when satisfied, and voluntary exposure—and show how they may be implemented together in a proof-of-concept reinforcement learning agent; further, we demonstrate how the properties manifest in the behaviour of this agent in a simple non-episodic grid-world environment that includes curiosity-inducing locations and induced targets of curiosity. As we would hope, the agent exhibits short-term directed behaviour while updating long-term preferences to adaptively seek out curiosity-inducing situations. This work therefore presents a novel view into how specific curiosity operates and in the future might be integrated into the behaviour of goal-seeking, decision-making agents in complex environments.

Keywords: curiosity, specific curiosity, computational reinforcement learning

Acknowledgements

Thanks to all the amazing people who made this work better and clearer, especially Kate Pratt. The authors wish to thank their funding providers. NMA was supported by a scholarship from The Natural Sciences and Engineering Research Council of Canada (NSERC). Work by PMP was supported by grants or awards from the Canada CIFAR AI Chairs Program, the Alberta Machine Intelligence Institute (Amii), NSERC, and the Canada Research Chairs program.

*Corresponding author. Website: https://ualberta.ca/∼nmady/

1 Specific Curiosity: An Introduction

Curiosity is in vogue; it is considered socially desirable, relating to a set of behaviours that employers want in their employees, teachers want in their students, and life coaches encourage in our interpersonal relationships. It should come as no surprise that machine intelligence researchers have wanted to provide the benefits of curiosity to their computational creations for decades (Schmidhuber, 1991). Pursuing these benefits, researchers have developed numerous mechanisms.

While these mechanisms are different from each other (Ady & Pilarski, 2017; Linke *et al.*, 2019), many of them are centred on generating and using special reward-like signals called intrinsic rewards. Put simply, the learner, or agent, obtains intrinsic reward for encountering situations with good potential for learning. Researchers have associated different concepts with "potential for learning" and translated these concepts into metrics to shape the behaviour of their agents.[1] In general, these metrics are used to determine the amount of intrinsic reward the agent receives for a given experience. Intrinsic rewards have been shown to be very useful for increasing exploration in some situations (Pathak *et al.*, 2017).

Figure 1: Turn to pg. 2 to rotate the mug.

However, intrinsic rewards produce behaviour that is conceptually unlike curiosity. Critically, by rewarding a learner for getting into a particular situation, the designer is encouraging the learner to return to that same situation, or ones very like it. This behaviour is missing a conceptual aspect of curiosity.

When humans describe curiosity, they talk about their curiosity being piqued or triggered. They take action, not to observe the same situation that piqued their curiosity, but to observe a different situation that they believe will satisfy their curiosity (see Fig. 1). Humans take advantage of their learned knowledge of how the world works to learn something specific, answering a question that the curiosity-piquing situation led them to ask. This is sometimes referred to as *specific curiosity*.

As a main contribution, we distill from the related literature three key properties of specific curiosity, and argue that these properties are beneficial to machine learners; specifically, we introduce the appropriateness of a reinforcement learning approach for designing agents that exhibit these properties. These properties are as follows, and are a subset of five key properties described and argued for in more detail by Ady *et al.* (2022).

Directedness: By separating a curiosity-piquing situation from a curiosity-satisfying situation, we can think about the appropriate behaviour for a learner whose curiosity has been piqued: take a sequence of actions that seems most likely to take them to a situation that satisfies their curiosity. The tendency for learners experiencing specific curiosity to undertake directed behaviour has been well-documented in the literature (e.g. Hagtvedt *et al.* 2019, p. 2; Berlyne, 1960, p. 297). This view contrasts with many other machine reinforcement learning methods which rely on injecting randomness into their choices of actions to experience new situations, rather than heading directly for a situation they know they don't know.

Cessation when satisfied: Once a learner has found a situation that satisfies their curiosity, they don't need to experience the same situation again, so we have no need to incentivize returning to a curiosity-satisfying situation. Examples documenting the satisfiability of curiosity are prevalent in the literature, including the works of Wiggin *et al.* (2019, p. 1194), Buyalskaya & Camerer (2020, p. 141, who refer to it as 'fulfillment') and Dan *et al.* (2020, p. 150, who refer to curiosity being 'satiated').

Voluntary exposure: While the learner does not benefit from returning to a curiosity-satisfying situation (that would only answer a question for which they have a satisfactory answer!), they do benefit from experiencing curiosity-inducing situations. A curiosity-inducing situation offers an appropriate jumping-off point for learning, for metaphorically picking up a puzzle piece fitting into what the learner already knows. Choosing to partake in activities likely to induce curiosity—for example, picking up puzzles or mysteries or turning on Netflix—has been called voluntary exposure to curiosity (Loewenstein, 1994, p. 84). This property might be thought of as developing an increased preference for situations like those that have been curiosity-inducing in the past.

Finally, we provide a case study of an agent demonstrating these three properties together, actively separating curiosity-piquing situations from curiosity-satisfying situations—a separation which does not exist in the type of behaviour motivated by intrinsic rewards. This proof-of-concept shows that it is possible to create an agent with attributes of specific curiosity, but should not be thought of as *the* way to implement specific curiosity. Rather, we hope it inspires the community to build upon our ideas and think up improved machine agents that benefit from knowledge of specific curiosity.

[1] Examples include confidence (Schmidhuber, 1991), learning progress (Oudeyer *et al.*, 2007, p. 269), surprise (White *et al.*, 2014, p. 14), interest/interestingness (Gregor & Spalek, 2014, p. 435; Frank *et al.*, 2014, pp. 5-6), novelty (Gregor & Spalek, 2014, p. 435; Singh *et al.*, 2004, pp. 1, 5), uncertainty (Pathak *et al.*, 2017, pp. 1-2), compression progress (Graziano *et al.*, 2011, p. 44), competence (Oddi *et al.*, 2020, pp. 2417-2418), and information gain (Bellemare *et al.*, 2016, p. 4; Houthooft *et al.*, 2016, pp. 2-3).

2 The Appropriateness of a Reinforcement Learning Framework for Specific Curiosity

With the goal to offer the benefits of specific curiosity to a computational learner, we want to take advantage of an existing learning framework, ideally one that supports the properties we want to achieve. The properties of directedness and voluntary exposure require a learner to have some specific capabilities. For directedness, the learner must be able to decide how to act in the world so they can take the specific actions that they believe will allow them to satisfy their curiosity. For voluntary exposure, the learner must be able to decide how to act in the world so they can visit situations that they suspect will result in their curiosity being piqued, and also learn those enduring preferences for curiosity-inducing situations.

Figure 2: If you were curious about what was written on the mug, you probably didn't keep staring at Fig. 1. Instead, you took action to reach a situation that would satisfy your curiosity.

The framework of reinforcement learning supports these requirements, as reinforcement learning centres around learners who are able to choose actions that affect their experiences (Sutton and Barto, 2018, p. 3). Instantiations of computational reinforcement learning algorithms are often called agents because they shape their own experience in the world and learn from their actions. This quality makes the framework well-suited for the design of machine curiosity algorithms.

A reinforcement learning framework is particularly promising for the property of voluntary exposure, as learners can estimate the value of different situations (for more detail, see Sutton & Barto, 2018, p. 58). In this way, a learner may develop a preference (i.e., a higher value) for situations that are more likely to pique curiosity. This preference aligns with voluntary exposure as observed in humans.

While, to the best of our knowledge, our conceptual separation of curiosity-piquing situations from curiosity-satisfying situations and our argument for specific properties of specific curiosity are new contributions, computational reinforcement learning researchers have shown strong interest in aspects of these properties in other contexts. In particular, the property of directedness parallels work done on options (as early as Sutton *et al.*, 1999) and planning. Approaches using directedness also often exhibit cessation when satisfied. For example, the options framework includes termination conditions for each option (often naturally defined by goal states; Stolle & Precup, 2002, p. 212).

Prior work has aimed to address the lack of directedness that is a characteristic of intrinsic-reward methods. For example, the Model-Based Active eXploration algorithm presented by Shyam *et al.* (2019) focuses on planning behaviour to allow the agent "to observe novel events" (p. 1). Similarly, the Go-Explore family of algorithms centres on the idea of taking a direct sequence of actions to move to a specific state for the purpose of exploring from it (Ecoffet *et al.*, 2020).

3 Case Study & Experiments

In this section, we describe how we created a simple agent exhibiting some of the properties of specific curiosity. We specifically hope to show that, even in a simple and focused setting, using our properties as guidelines allows machine behaviour to emerge that approximates the specific curiosity of animal learners. This example is not intended as a final or definitive computational implementation of specific curiosity. The intended purpose of this section is for the reader to gain insight and motivation to further investigate how to integrate the properties of specific curiosity into different machine learning frameworks and problem settings.

A complete agent with specific curiosity will require some additional functionality that we have not implemented. A human can recognize that there is something they do not know, sometimes referred to as an *information gap* (Loewenstein, 1994) or an *inostensible concept* (Inan, 2012). Moreover, humans can imagine what they would need to observe to rectify such a gap, and even suggest actions that might lead to those observations. For example, you could reach out and turn the mug in Fig. 1, or move your own body to see the mug at a different angle. We do not tackle how agents can recognize information gaps or how they can predict what will satisfy their resulting curiosity. In our case study, we gave the agent a module that could recognize a curiosity-inducing situation and a corresponding curiosity-satisfying target. We hard-coded this module so only a single location in the world piqued curiosity and the targets were randomly selected locations from a pre-determined subset (see Fig. 3). We envision that the gaps in this architecture can be filled as more aspects of specific curiosity are understood and become computationally tractable.

Directedness: The first property we explored in the design of our agent was directedness. As we have already suggested, directedness seems to involve making and following a plan, which, in a reinforcement learning framework, suggested to us that we might use a model of the world. In computational reinforcement learning, a model-based approach using dynamic programming or an approximation of dynamic programming can provide a learner with a plan to follow from one state to another (Sutton & Barto, 2018, Ch. 4), suggesting existing algorithms we could exploit in our pursuit of the property of directedness.

Cessation when satisfied: Directedness in the pursuit of satisfying curiosity is temporary and should not persist when the agent is not in a state of curiosity.[2] We were inspired by an architecture introduced by Silver *et al.* in 2008: Dyna-2. We realized that Dyna-2 has a mechanism (originally used for playing games) that differentiates temporary behaviour from the long-term preferences of the agent: two value functions. In Silver's (2009, p. 87) work, a temporary value function captures unique aspects of the current match of a game, while a persistent value function captures principles of the game that hold across matches. In our design, we adapt this idea so the learner follows the temporary value function while curious but uses a separate persistent value function to capture long-term preferences. The temporary value function can lead the learner directly to a situation they believe will satisfy their curiosity. Since what will satisfy curiosity differs each time curiosity is piqued, the exact plan to satisfy curiosity can be discarded at satisfaction.

Voluntary exposure: A temporary value function doesn't have a lasting effect on the learner's preferences. However, we want an enduring effect associated with curiosity: voluntary exposure. Humans develop a preference for curiosity-inducing situations, so we wanted to experiment with algorithm modifications that would lead to agents with this property. We came up with a small modification to the standard TD learning algorithm (Sutton & Barto, 2018, p. 120) that would leave an enduring effect of having experienced curiosity on the persistent value function, V. Our modified temporal difference, δ, is:

$$\delta \leftarrow R + \gamma \cdot V(x') - \boldsymbol{[V(x) + V_{\text{curious}}(x)]} \tag{1}$$

where V_{curious} refers to the temporary value function. For this modification to result in the accrual of positive value in the persistent value function, V, it is necessary for V_{curious} to be non-positive everywhere. Given this modification, we ran experiments to see if the enduring effect looked like voluntary exposure.

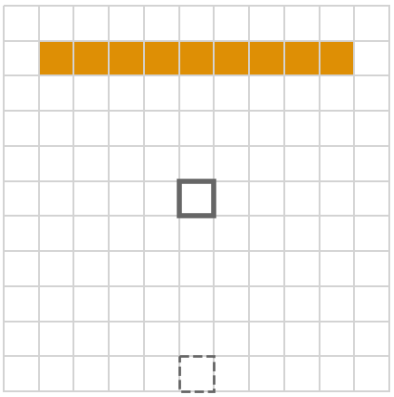


Figure 3: Domain used for the experiments. The orange boxes are the potential curiosity-satisfying locations. The heavily-outlined box is the curiosity-inducing location. If the agent leaves the top row, it teleports to the dashed box in the bottom row.

Experiment setup: We ran our experiments in an 11×11 grid world shown in Fig. 3. In this grid world, the learner is located in one of the squares and their actions let them move to adjacent squares, but they can only take sideward or upward actions (including diagonals). Any upward action from the top row of the grid teleports the agent to the middle of the bottom row. The agent never receives any reward from this domain (R in Eq. 1 is always zero).

As long as the agent is not in a state of curiosity, the temporary value function is zero and the agent acts ϵ-greedily with respect to its persistent value function. The centre of the grid is a curiosity-inducing location: when the agent visits it, a curiosity-satisfying target is generated and the agent is given a non-positive temporary value function (Fig. 4) that guides it directly to that target. The target is randomly selected with equal probability for each of the non-edge squares in the second row from the top (highlighted in Fig. 3). When the agent visits the target, the temporary value function is zeroed out again.

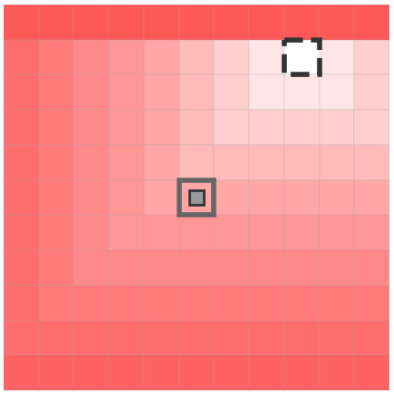


Figure 4: An example temporary value function, V_{curious}. The target has a heavy dashed outline and a value of zero. Magnitudes of negative values are shown using shades of red.

The results reported here reflect 30 trials of 5000 steps. The agent started each trial in the curiosity-inducing location. We ran further experiments varying the size and dynamics of the grid world, and the key results we present are representative of all these experiments.

Results & Discussion: Fig. 5 shows the persistent value function at the end of 5000 steps, averaged over 30 trials. Fig. 6 shows how the values of the curiosity-inducing location and the curiosity-satisfying locations change over time.

The curiosity-inducing location accrues the most value and the agent learns a trail of increasing value leading directly to that location. The curiosity-satisfying locations, on the other hand, do *not* accumulate much value over time. In effect, our modification to the learning update results in voluntary exposure while retaining the property of cessation when satisfied.

Looking at the behaviour of the agent over time (an example video can be viewed at `https://youtu.be/TDUpB7OefFc`) we can see the agent moves more and more directly towards the curiosity-inducing location.

[2] In the literature, there is a distinction between curiosity as a *state* experienced temporarily by a learner versus as a *trait*, or general propensity of the learner (Loewenstein, 1994, p. 78). Specific curiosity makes the most sense as a form of *state* curiosity because it only persists until the specific information of interest is found.

4 Conclusions

We argue that learners benefit from specific curiosity, something previously not demonstrated in the computational literature. In particular, specific curiosity has properties of directedness, cessation when satisfied, and voluntary exposure that yield important benefits that have been missing from other approaches to machine curiosity. In this work we developed a case study showing how these properties might be implemented in a computational reinforcement learning framework.

We appreciate reinforcement learning for its flexibility to express key properties of specific curiosity. We found, through our case study, that separating persistent and temporary value functions was a useful mechanism to encode some of the properties of biological curiosity. This separation gives our agent directed behaviour as well as learned preferences, which have traditionally been associated with model-based and model-free approaches to reinforcement learning, respectively. The way our agent balances these two approaches may offer a new perspective on how we might balance model-based and model-free learning.

Future directions: While directedness, cessation when satisfied, and voluntary exposure are the first properties we have explored empirically, our conceptual synthesis of the literature also demonstrates the importance of two more properties: transience and connection to long-term information search (Ady *et al.*, 2022). While our agent has begun to exhibit some aspects of specific curiosity, including these additional properties and continuing to incorporate new ideas from the study of human curiosity can only strengthen our computationally curious agents.

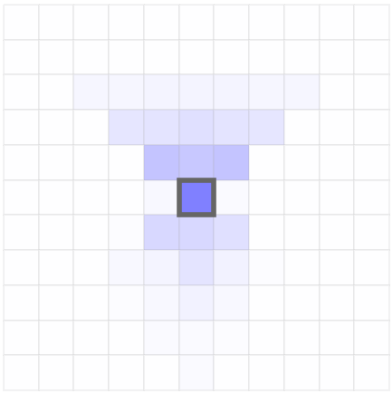


Figure 5: Persistent value function, V, learned by the end of 5000 steps, averaged over 30 trials. White squares have zero value. More positive values are shown in deeper shades of blue.

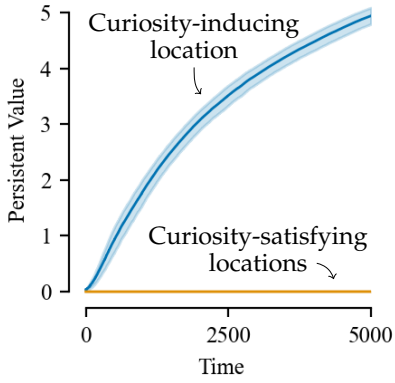


Figure 6: Persistent value of the curiosity-inducing location is in blue while the average persistent value of all of the curiosity-satisfying locations is in orange. The lines are the average over 30 trials while the shaded area shows the standard deviation.

References

Ady, Shariff, Günther, and Pilarski, in preparation for *J. Artificial Intelligence Research*, 2022.

Ady and Pilarski, *Multidiscip. Conf. Reinforcement Learning and Decision Making*, 2017.

Bellemare, Srinivasan, Ostrovski, *et al.*, *NeurIPS*, 2016, 1471–1479.

Berlyne, *Conflict, arousal, and curiosity.* McGraw-Hill Book Company, 1960.

Buyalskaya and Camerer, *Current Opinion in Behavioral Sciences*, 2020, 35:141–149.

Dan, Leshkowitz, and Hassin, *Current Opinion in Behavioral Sciences*, 2020, 35:150–156.

Ecoffet, Huizinga, Lehman, Stanley, and Clune, *Nature*, 2021, *590*, 580–586.

Frank, Leitner, Stollenga, *et al.*, *Frontiers in Neurorobotics*, 2014, 7:25.

Graziano, Glasmachers, Schaul, *et al.*, *Acta Futura*, 2011, *4*, 41–52.

Gregor and Spalek, *ELEKTRO 2014 Conference*, 2014, 435–440.

Hagtvedt, Dossinger, Harrison, *et al.*, *Org. Behav. and Human Decision Proc.*, 2019, 150:1-13.

Houthooft, Chen, Duan, *et al.*, *NeurIPS*, 2016, 1109–1117.

Inan, *The Philosophy of Curiosity*, 2012. Routledge.

Linke, Ady, Degris, *et al.*, *Multidiscip. Conf. Reinforcement Learning and Decision Making*, 2017.

Loewenstein, *Psychological Bulletin*, 1994, *116*:1, 75–98.

Markey and Loewenstein, *International handbook of emotions in education*, 2014, 238–255.

Pathak, Agrawal, Efros, *et al.*, *Int. Conf. on Machine Learning*, 2017, 2778–2787.

Oddi, Rasconi, Santucci, *et al.*, *European Conference on Artificial Intelligence*, 2020, 2417–2424.

Oudeyer, Kaplan, and Hafner, *IEEE Trans. on Evolutionary Computation*, 2007, *11*:2, 265–286.

Schmidhuber, *International Joint Conference on Neural Networks*, 1991, 1458–1463.

Shyam, Jaśkowski, and Gomez, *Int. Conf. on Machine Learning*, 2019, 5779–5788.

Silver, Sutton, and Müller, *Int. Conf. on Machine Learning*, 2008, 968–975.

Silver, "Reinforcement Learning and Simulation-Based Search in Computer Go," Doctoral Thesis, 2009.

Singh, Barto, and Chentanez, *NeurIPS*, 2004.

Spielberger and Starr, In: *Motivation: Theory and Research*, 1994, 221–243. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Stolle and Precup, *International Symposium on Abstraction, Reformulation, and Approximation*, 2002, 212–223.

Sutton and Barto, *Reinforcement Learning: An Introduction*, 2018. MIT Press, second edition.

White, Modayil, and Sutton, *Sequential Decision-Making with Big Data: AAAI-14 Workshop*, 2014, 19–23.

Wiggin, Reimann, and Jain, *Journal of Consumer Research*, 2019, 45(6):1194–1212.

Behavior Predictive Representations for Generalization in Reinforcement Learning

Siddhant Agarwal
Indian Institute of Technology Kharagpur

Aaron Courville
MILA, Université de Montréal

Rishabh Agarwal
Google Research, Brain Team
MILA, Université de Montréal

Abstract

Deep reinforcement learning (RL) agents trained on a few environments, often struggle to generalize on unseen environments, even when such environments are semantically equivalent to training environments. Such agents learn representations that overfit the characteristics of the training environments. We posit that generalization can be improved by assigning similar representations to scenarios with similar sequences of long-term optimal behavior. To do so, we propose behavior predictive representations (BPR) that capture long-term optimal behavior. BPR trains an agent to predict latent state representations multiple steps into the future such that these representations can predict the optimal behavior at the future steps. We demonstrate that BPR provides large gains on a jumping task from pixels, a problem designed to test generalization.

Keywords: Generalization in Reinforcement Learning, Representation Learning, Predictive Representations

1 Introduction

Deep reinforcement learning (RL) agents, even when trained on diverse environments with similar high level goals but different dynamics and visual appearances, often struggle to generalize on unseen environments, even when such environments are semantically equivalent to training environments [Farebrother et al., 2018, Cobbe et al., 2020, Agarwal et al., 2021a, Packer et al., 2018]. Such agents learn state representations from high-dimensional observations that typically overfit to the peculiarities of training environments [Song et al., 2019, Raileanu and Fergus, 2021] rather than capturing generalizable skills which can be transferred to unseen environments. Such overfitting hinders the real-world applicability of RL, making generalization in RL an important challenge.

To improve generalization using better representations, we revisit predictive representations [Littman et al., 2001, Rafols et al., 2005] that describe the environment in terms of predictions about future observations, such as representations that encode the underlying environments dynamics. While learning such temporally predictive representations has been shown to improve sample efficiency [Oord et al., 2018, Schwarzer et al., 2021] within a training environment, it is unclear whether such representations would improve performance in unseen environments. More recently, Agarwal et al. [2021a] enhance generalization by learning similar state representations for observations with similar long-term optimal behavior. Inspired by their findings, we posit that predictive representations that capture long-term optimal behavior might be better suited for generalization. We expect such *behavior predictive representations* to generalize as two observations, possibly across different environments, are assigned similar representations if they exhibit similar sequences of optimal behaviors, irrespective of their differences in obtained rewards, visual appearances, or even the underlying dynamics.

For learning behavior predictive representations (BPR), we train the agent to predict latent state representations multiple steps into the future such that these representations can predict the optimal behavior at the future steps (Figure 1). BPR can be viewed as a representation learning approach where the agent predicts the optimal behavior at future states resulting from following a sequence of actions from a given state. We show that BPR improves generalization upon existing methods including PSEs [Agarwal et al., 2021a] and SPR [Schwarzer et al., 2021] on the jumping task on pixels.

2 Preliminaries

We describe an environment as a Markov decision process (MDP) that corresponds to a tuple $\mathcal{M} = (S, A, P, R, \gamma)$ where S is the state space, A is the action space, $P : S \times A \times S \rightarrow [0, 1]$ is the state transition function, $R : S \times A \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is the discount factor. A policy $\pi(\cdot|s)$ maps a state $s \in S$ to a distribution over the action space A. A trajectory is defined as the sequence of states, actions and corresponding rewards i.e. $s_0, a_0, r_1, s_1, \cdots$. The goal of a reinforcement learning agent is to maximize the cumulative expected return $\mathbb{E}_{p(\tau)}[\sum_t \gamma^t r(s_t, a_t)]$ where $p(\tau) = p(s_0) \prod_t p(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$.

3 Behavior Predictive Representations

In this work, we aim to learn a policy that can generalize across related environments. Specifically, we train an agent using a finite number of environments (or tasks) sampled from a distribution of environments. The performance of this agent is evaluated using unseen environments sampled from the same distribution. For example, consider the generalization problem in a jumping task from pixels Tachet des Combes et al. [2018], where an agent needs to jump over an obstacle (Figure 2). Standard deep RL agents trained on a small number of training tasks with different obstacle positions struggle to generalize to unseen obstacle positions [Agarwal et al., 2021a].

Inspired from the recent success of representation learning to improve generalization [Agarwal et al., 2021a, Raileanu and Fergus, 2021, Zhang et al., 2020], we also focus on learning better representations to improve generalization. We posit that learning better representations requires understanding which states are similar in terms of their long-term optimal behavior. To do so, we aim to learn latent state representations that not only capture the behaviour at the current state but will also be able to predict the behaviour at future states, which we call *behavior predictive representations* (BPR). Since BPR simply uses an auxiliary objective L^{BPR}, it can be easily combined with any RL or IL setup, as shown in Figure 1.

To describe the auxiliary loss L^{BPR} for predicting the long-term optimal behavior, we define some notation first. Let s_t be the state at the time step t and z_t be the corresponding latent representation learned by the policy network f. The policy π predicts the action distribution given the latent representation z_t. We use an encoder network f to generate these latent representations from states as, $z_t = f(s_t)$. A transition function $h : S^* \times A \rightarrow S^*$ learns the state dynamics and predicts the representations at the next step, $\hat{z}_{t+1} = h(s_t, a_t)$.

We predict the representations for K future steps by iteratively applying the transition function. While such latent state dynamics are typically learned by minimizing the mean squared loss between $\hat{z}_{t+k}$ and z_{t+k}, we instead use these predicted representations to predict the optimal action distributions in the future steps $t + 1$ to $t + k$. To do so, the agent minimizes the cross entropy between the predicted action distribution and optimal action distributions at these steps.

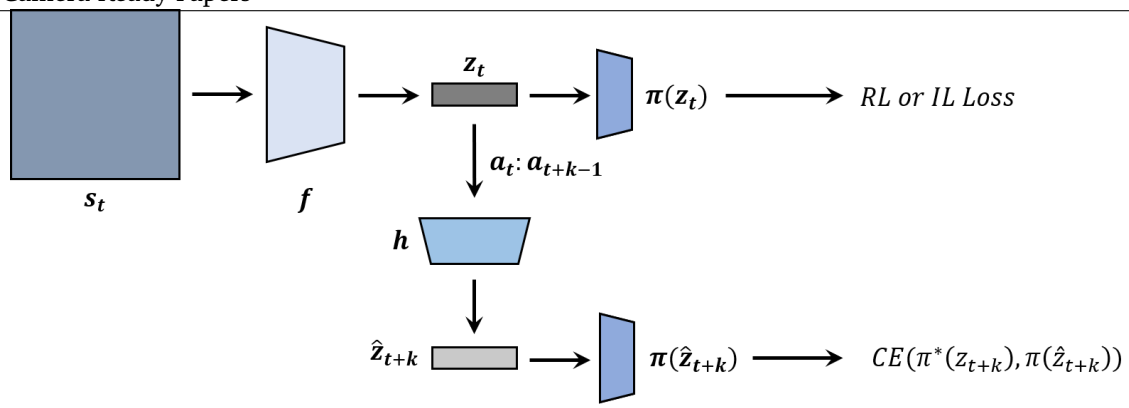


Figure 1: **Behavior Predictive Representations**. A schematic diagram showing how behavior predictive representations are learned using an auxiliary task on training environments. Representations z_t from the policy network are trained to predict the optimal behavior using either a reinforcement learning (RL) or imitation learning (IL) loss. These representations z_t, in conjunction with actions $a_t, a_{t+1}, \cdots, a_{t+k-1}$, are also trained to predicting latent representations $\hat{z}_{t+k}$ via the transition model h such that the $\hat{z}_{t+k}$ can predict the optimal behavior $\pi^*(z_{t+k})$ at time step $t+k$.

Specifically, given access to the optimal policy π^* on training environments, the auxiliary loss L^{BPR} is given by:

$$L^{BPR} = \sum_{k=1}^{K} L^{CE}(\pi^*(z_{t+k}), \pi(\hat{z}_{t+k})), \tag{1}$$

where $L^{CE}(\pi_1(\cdot|s), \pi_2(\cdot|s)) = -\sum_{a \in A} \pi_1(a|s) \log \pi_2(a|s)$.

In the general RL setting, where we do not have access to the optimal policy, we propose to use the learned policy for specifying the future behavior in the objective L^{BPR}. Specifically, the target distribution for the auxiliary cross entropy loss, L^{BPR}, comes from the same policy network that is being trained (π in Figure 1). To provide some stability from the continuous changes in the learned policy, we use a separate *target* policy network that is periodically updated with the learned policy network parameters, analogous to deep Q-learning [Mnih et al., 2013] and self-supervised learning [Grill et al., 2020]. So, the target representations $\hat{z}_{t+k}$ are derived from the learned transition function h while the action distribution $\pi_{\text{learned}}(z_{t+k})$ comes from the target policy network. For this setting, the auxiliary loss $\hat{L}^{BPR}$ is given by,

$$\hat{L}^{BPR} = \sum_{k=1}^{K} L^{CE}(\pi_{\text{learned}}(z_{t+k}), \pi(\hat{z}_{t+k})) \tag{2}$$

The final training objective is the combination of both of these loss functions. Let L^{RL} be the traditional model-free RL (or imitaion learning) objective. Then, the combined loss function for learning behavior predictive representations is $L = L^{RL} + \lambda_{BPR} L^{BPR}$, where λ_{BPR} is the weighting coefficient for the auxiliary loss.

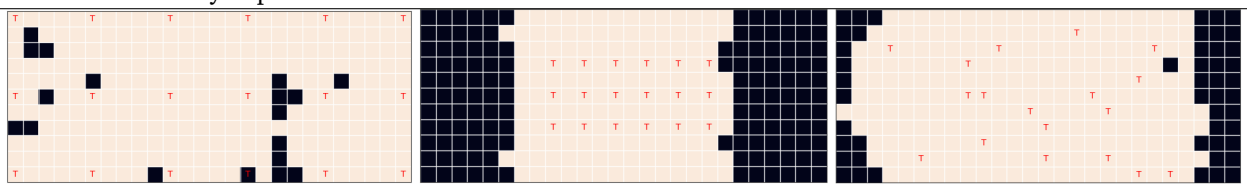
4 Experiments

We thoroughly investigate BPR on the jumping task [Tachet des Combes et al., 2018, Agarwal et al., 2021a] that captures whether agents can learn the correct invariances for generalization directly from image inputs.

4.1 Jumping Task From Pixels

Task Description. The task consists of an agent trying to jump over an obstacle using two actions: *right* and *jump*. Different tasks consist in shifting the floor height and/or the obstacle position (Figure 2). To generalize, the agent needs to be invariant to the floor height while jump based on the obstacle position.

Problem Setup. Following Agarwal et al. [2021a], we use three different configurations (Figure 3), each consisting of 18 seen (training) and 268 unseen (test) tasks, to test generalization in regimes without and with data augmentation using RandConv [Lee et al., 2020]. As discussed by Agarwal et al. [2021a], the different grids configurations capture different types of generalization: the "wide" grid tests generalization via "interpolation", the "narrow" grid tests out-of-distribution generalization via "extrapolation", and the random grid instances evaluate generalization similar to supervised learning where train and test samples are drawn i.i.d. from the same distribution.



(a) "Wide" grid (b) "Narrow" grid (c) Random grid

Figure 3: **Jumping Task**: Visualization of average performance of BPR with data augmentation across different configurations. We plot the median performance across 25 runs. Each tile in the grid represents a different task (obstacle position/floor height combination). For each grid configuration, the height varies along the y-axis (11 heights) while the obstacle position varies along the x-axis (26 locations). The red letter ⊤ indicates the training tasks. Random grid depicts only one instance, each run consisted of a different test/train split. Beige tiles are tasks BPR solved while **black** tiles are tasks BPR did not solve.

Baselines. We compare the efficacy of our method with a number of techniques that have been used to achieve generalization such as ℓ_2-regularization, dropout [Farebrother et al., 2018] and data augmentation [Lee et al., 2020].

Policy Similarity Embeddings (PSEs) [Agarwal et al., 2021a] are the state-of-the-art generalization method on the jumping task. PSEs form an important baseline for BPR as PSEs also use the future behaviour as a similarity metric between states. Specifically, PSEs learn contrastive metric embeddings using a policy similarity metric d (Equation 3) that uses policy to measure the long term behavior similarity between among states.

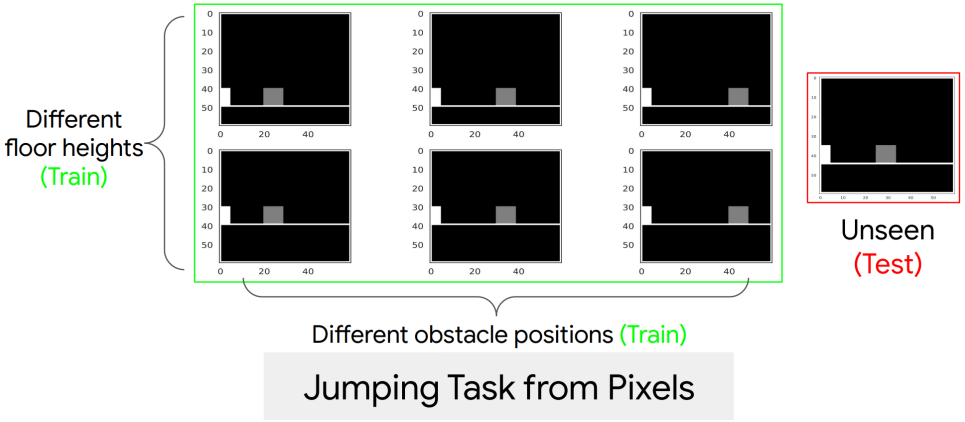


Figure 2: **Generalization on Jumping Task**. In this task, the agent needs to jump over an obstacle. The agent needs to time the jump precisely, at a specific distance from the obstacle, otherwise it will eventually hit the obstacle. Training environments consists of different obstacle positions as well as floor heights. At test time, the agent needs to generalize to environments with unseen positions and heights. The obstacle can be in 26 different locations while the floor has 11 different heights, totaling 286 environments.

$$d(x, y) = DIST(\pi^*(x), \pi^*(y)) + \gamma W_1(d)(p_{\pi^*}(\cdot|x), p_{\pi^*}(\cdot|y)) \tag{3}$$

Self-predictive representations (SPR) [Schwarzer et al., 2021] is another relevant baseline which has been shown to improve sample-efficiency on training environments on the Atari 100k benchmark [Kaiser et al., 2019, Agarwal et al., 2021b]. SPR's objective is that the agent learns to predict its own latent representations at future steps. Similar to BPR, it uses a transition function to iteratively generate these latent representations for the future steps. However, while BPR optimizes the latent representations to predict future behavior, SPR tries to maximize the similarity between the predicted latent representations $\hat{z}_{t+1} : \hat{z}_{t+K}$ with the true future state representations $z_{t+1} : z_{t+K}$. To do so, SPR uses a self-supervised learning objective [Grill et al., 2020] as the auxiliary loss,

$$L^{SPR}(s_t : s_{t+k}, a_t : a_{t+k}) = -\sum_{k=1}^{K} \left(\frac{q(g_o(\hat{z}_{t+k}))}{||q(g_o(\hat{z}_{t+k}))||_2} \right)^T \left(\frac{g_m(z_{t+k})}{||g_m(z_{t+k})||_2} \right) \tag{4}$$

where g_o, g_m and q are online projection network, target projection network and prediction networks respectively. SPR linearly combines the auxiliary objective, L^{SPR}, with the RL objective.

Results. Table 1 summarizes the performance of BPR and all the baselines with and without data augmentation. Without data augmentation, with only 18 training environments, BPR generalizes quite well in all the three grid configurations, significantly outperforming regularization and PSEs by a large margin. These results exhibit that BPR is effective even without data augmentation.

Data augmentation complements all the methods and boosts generalization performance. Comparing RandConv + BPR to RandConv, we see that BPR is much more effective on top of RandConv. Moreover, when used in conjunction with data augmentation, BPR performs comparably to the current state-of-the-art method PSEs. Compared to BPR, SPR degrades the generalization performance significantly and even performs poorly than simply using RandConv. We hypothesize that the self-supervised learning objective in SPR might be exacerbating the overfitting in learned representations by trying to predict the spurious features captured by the learned representations on training environments.

Table 1: Percentage (%) of test tasks solved by different methods w/o and w/ data augmentation. The "wide", "narrow", and random grids are described in Figure 2. For methods implemented in this work (BPR and SPR), we report average performance across 25 runs with different random initializations, with standard deviation between parentheses. Other results are taken from Agarwal et al. [2021a].

Data Augmentation	Method	Grid Configuration (%)		
		"Wide"	"Narrow"	Random
No	Dropout and ℓ_2 reg.	17.8 (2.2)	10.2 (4.6)	9.3 (5.4)
	PSEs	33.6 (10.0)	9.3 (5.3)	37.7 (10.4)
	BPR	**62.4** (18.6)	**15.3** (6.7)	**58.5** (20.0)
Yes	RandConv	50.7 (24.2)	33.7 (11.8)	71.3 (15.6)
	RandConv + SPR	23.3 (11.8)	30.6 (13.3)	64.1 (15.6)
	RandConv + PSEs	**87.0** (10.1)	**52.4** (5.8)	**83.4** (10.1)
	RandConv + BPR	**90.0** (18.6)	**52.0** (9.4)	**82.5** (15.1)

5 Acknowledgement

We acknowledge the support of Google cloud for providing GCP credits for running our experiments.

References

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021a.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *arXiv preprint arXiv:2108.13264*, 2021b.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056, 2020.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2020.

Michael L Littman, Richard S Sutton, and Satinder P Singh. Predictive representations of state. In *Neural Information Processing Systems*, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Eddie J Rafols, Mark B Ring, Richard S Sutton, and Brian Tanner. Using predictive representations to improve generalization in reinforcement learning. In *IJCAI*, 2005.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. *International conference on machine learning*, 2021.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. 2021.

Xingyou Song, Yiding Jiang, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2019.

Remi Tachet des Combes, Philip Bachman, and Harm van Seijen. Learning invariances for policy generalization. In *Workshop track at the International Conference on Learning Representations (ICLR)*, 2018.

Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block mdps. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Be Considerate: Avoiding Negative Side Effects in Reinforcement Learning (Extended Abstract)*

Parand Alizadeh Alamdari†
University of Toronto
Vector Institute
Toronto, Canada
parand@cs.toronto.edu

Toryn Q. Klassen†
University of Toronto
Vector Institute
Schwartz Reisman Institute for Technology and Society
Toronto, Canada
toryn@cs.toronto.edu

Rodrigo Toro Icarte
Pontificia Universidad Católica de Chile
Santiago, Chile
Vector Institute
Toronto, Canada
rodrigo.toro@ing.puc.cl

Sheila A. McIlraith
University of Toronto
Vector Institute
Schwartz Reisman Institute for Technology and Society
Toronto, Canada
sheila@cs.toronto.edu

Abstract

In sequential decision making – whether it's realized with or without the benefit of a model – objectives are often underspecified or incomplete. This gives discretion to the acting agent to realize the stated objective in ways that may result in undesirable outcomes, including inadvertently creating an unsafe environment or indirectly impacting the agency of humans or other agents that typically operate in the environment. In this paper, we explore how to build a reinforcement learning (RL) agent that contemplates the impact of its actions on the wellbeing and agency of others in the environment, most notably humans. We endow RL agents with the ability to contemplate such impact by augmenting their reward based on expectation of future return by others in the environment, providing different criteria for characterizing impact. We further endow these agents with the ability to differentially factor this impact into their decision making, manifesting behaviour that ranges from self-centred to self-less, as demonstrated by experiments in gridworld environments.

Keywords: AI Safety, Side Effects, Value Alignment

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. The third author acknowledges funding from ANID (Becas Chile). This work was done while he was a graduate student at the University of Toronto.

*The full paper appears in the *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022)* [1].

†equal contribution

1 Introduction

Recent work in AI safety has raised the concern that objectives are often underspecified, neglecting to take into account potential negative side effects. As Amodei et al. [2] explain, "an objective function that focuses on only one aspect of the environment may implicitly express indifference over other aspects of the environment." Stuart Russell gave the example of a robot, tasked to get coffee from a coffee shop, killing everyone in line to optimize its performance [9]. An example by Amodei et al. [2] is that of a robot breaking a vase on the optimal path between two points. Recent work, including in RL, has developed some techniques for avoiding or learning to avoid negative side effects [e.g., 7, 12, 8, 10].

Our concern in this paper is with how an RL agent can learn to act safely in the face of a potentially incomplete specification of the objective. Amodei et al. observe that "avoiding side effects can be seen as a proxy for the things we really care about: avoiding negative externalities. If everyone likes a side effect, there's no need to avoid it." In the spirit of this observation, we contend that *to act safely an agent should contemplate the impact of its actions on the wellbeing and agency of others in the environment.* Here we consider negative side effects to be those that impede the future wellbeing or agency of other agents.

The setup in this paper is *not* a multi-agent RL or cooperative AI setup, and this is done by design. We take the pragmatic stance that in many real world settings, an RL agent will not be able to compel humans to consistently and rationally cooperate, if they deign to cooperate at all. As such, the problem we address is in a single RL-agent setting in which the other agents — which may be the humans that operate in the environment — are just part of the environment, operating with fixed policies, and it is the acting RL agent that is constructing a policy that minimizes its impact on the future agency of these other (human) agents. An evocative example may be to consider university students who share a kitchen environment, and we wish our RL agent – the acting agent, with some conception of what others may typically do in the kitchen — to learn how to act in the kitchen in a manner that is considerate of others who may use the kitchen afterwards.

Here, we endow RL agents with the ability to consider in their learning the future welfare and agency of others in the environment. We do so by augmenting the RL agent's reward with an auxiliary reward that reflects different functions of expected future return of other agents. We contrast this with recent work on side effects that takes into account only how the agent's actions will affect its own future abilities [7, 12, 8]. Experiments in gridworld environments illustrate qualitative and quantitative properties of our proposed approach. The full version of this paper [1] explores some additional variants.

Notation: An MDP is a tuple $\langle S, A, T, r, \gamma \rangle$ where S is the state set, A is the action set, $T(s_{t+1}|s_t, a_t)$ gives transition probabilities, $r : S \times A \times S \to \mathbb{R}$ is the reward function, and γ is the discount factor. An MDP can also include a designated initial state $s_0 \in S$. A *terminal state* in an MDP is a state s which can never be exited and from which no further reward can be gained. A *value function* $V(s)$ gives the expected return of a state (when following some policy).

2 Problem and Approach

To incentivize the acting agent to consider the future wellbeing and agency of others, we augment its reward with an auxiliary reward that reflects the impact of its choice of actions on others. To reflect the acting agent's uncertainty about what is good for others, we make use of a distribution over value functions. In particular, suppose that we have a finite set $\mathcal{V}$ of possible value functions $V : S \to \mathbb{R}$, and a probability distribution $\mathsf{P}(V)$ over $\mathcal{V}$. Note that we don't have to commit to how many agents there are (or what exactly their actions are). It could be that each $V \in \mathcal{V}$ corresponds to a different agent, that the set reflects all possible value functions of a unique agent, or anything in between. Also, each $V \in \mathcal{V}$ could reflect some aggregation of the value functions of all or some of the agents.

We define the augmented reward function in Eq. (1), where r_1 is the acting agent's individual reward function, and F is some function. The hyperparameters α_1 and α_2, which we call "caring coefficents", are real numbers that determine to what extents the individual reward r_1 and the auxiliary reward $F(\mathcal{V}, \mathsf{P}, s')$ contribute to the overall reward. If $\alpha_1 = 1$ and $\alpha_2 = 0$, we just get the original reward function. Note that future activity does not have to start in exactly the same state at which the acting agent ended. V can be defined so that $V(s')$ gives the expected return of future activity considered over a known distribution of starting states, given that the acting agent ended in s'.

$$r_{\mathsf{value}}(s, a, s') = \begin{cases} \alpha_1 \cdot r_1(s, a, s') & \text{if } s' \text{ is not terminal} \\ \alpha_1 \cdot r_1(s, a, s') + \ \gamma \cdot \alpha_2 \cdot F(\mathcal{V}, \mathsf{P}, s') & \text{if } s' \text{ is terminal} \end{cases} \quad (1)$$

We consider three possible different definitions of $F(\mathcal{V}, \mathsf{P}, s')$, given in Eqs. (2), (3), and (4). In Eq. (2), $F(\mathcal{V}, \mathsf{P}, s')$ is the expected value of s', given the distribution on value functions. Under some conditions, this is a generalization of the auxiliary reward defined by Krakovna et al. [8], which assumed that the future value functions were ones of the acting agent (and so depended on the acting agent's own abilities). Meanwhile, Eq. (3) considers the value of s' if the "worst-case" value function from $\mathcal{V}$ is used. Note that those two reward augmentations may incentivize the acting agent to not only avoid negative side effects, but also to cause "positive side effects" – to help other agents (assuming $\alpha_2 > 0$). To focus on avoiding negative

$$\sum_{V \in \mathcal{V}} \mathsf{P}(V) \cdot V(s') \quad \text{expected future return} \quad (2)$$

$$\min_{V \in \mathcal{V}: \mathsf{P}(V) > 0} V(s') \quad \text{worst-case future return} \quad (3)$$

$$\sum_{V \in \mathcal{V}} \mathsf{P}(V) \cdot \min(V(s'), V(s_0)) \quad \text{penalize negative change} \quad (4)$$

side effects, Krakovna et al. [8] proposed comparing the state the agent ends up in against a *reference state*, and that is applicable to our approach as well. In Eq. (4), we use one of the simplest possible reference states, the initial state: the auxiliary reward is the lower of $V(s')$ and $V(s_0)$, where s_0 is the initial state. The idea is to decrease the acting agent's reward when it decreases the expected future return, but to *not* increase the acting agent's reward for increasing that.

A complication with our approach is that for some possible reward functions for the acting agent and future value functions, the acting agent may have an incentive to avoid terminating states, to avoid or delay the penalty for negative future return. This incentive would typically be undesirable. However, under some circumstances, the acting agent's optimal policy will be terminating. The proposition below and its proof are similar to a result of Illanes et al. [5, Thm. 1].

Proposition 1. *Let $M = \langle S, A, T, r_1, \gamma \rangle$ be an MDP where $\gamma = 1$, the reward function r_1 is negative everywhere, and there exists a terminating policy. Suppose r_{value} is the reward function constructed from r_1 according to Equation 2, using some distribution $\mathsf{P}(V)$. Then any optimal policy for the MDP $M' = \langle S, A, T, r_{\mathsf{value}}, \gamma \rangle$ with the modified reward will terminate with probability 1.*

Proof. Suppose for contradiction that there is an optimal policy π^* for M' that is non-terminating. Then there is some state $s \in S$ so that the probability of reaching a terminal state from s by following π^* is some value $c < 1$. Since rewards are negative everywhere, that means that $V^{\pi^*}(s) = -\infty$. On the other hand, any terminating policy gives a finite value to each state. Since there is a terminating policy for M there is one for M', and so π^* cannot be optimal. □

3 Experiments

We present experimental results using the previous section's formulations of reward functions that allow RL agents to contemplate the impact of their actions on others. In all the experiments, policies are learned using Q-learning. To aid exposition, we consider simple distributions over future value functions, in which the acting agent is certain of what the future value function is (or, in Figure 3, only considers a small number of possibilities). In our first set of experiments, we compare one of our formulations (Eq. (2)) of a considerate RL agent against two baselines. We illustrate that by considering others, the acting agent avoids causing negative side effects for them, and in some scenarios, yields positive side effects. Second, we illustrate the effect of the caring coefficient on the agent's behaviour and on other agents' reward. Finally, we share the results of a qualitative experiment that serves to illustrate how different reward function augmentations lead to different behaviours. Code is available at `https://github.com/praal/beconsiderate`.

3.1 The Impact of Considering Others

We explore how our choice of reward augmentation method affects the acting agent and the agent that goes next. We use a kitchen environment where agents aim to collect different ingredients from the fridge or shelves, and prepare a meal. Each agent, when it performs any action, gets -1 reward. We designed four different scenarios to illustrate properties of our approach. The results are shown in Table 1. We use a *step difference* metric, described in the caption.

Baselines: We compare our method, using Eq. (2) (with $\alpha_1 = \alpha_2 = 1$), with two reward augmentation baselines: not augmenting the reward, and a method based on Krakovna et al. [8]'s approach. The Krakovna-style baseline uses the same Eq. (2) to augment the rewards, except that the future value functions considered are always possible future value functions of the *acting agent itself* (as if it were trying to accomplish the tasks of other agents). So if other agents have differing abilities, those abilities are ignored in the Krakovna-style model. (Note that this does not incorporate Krakovna et al.'s notion of a "reference state" and may incentivize positive side effects in some cases, as our own method does.)

The first experiment (Salad) shows a scenario where the acting agent and next agent have the same abilities, and so our approach and the Krakovna-style baseline both avoid negative side effects and behave identically. The next experiments (Peanut and Salt) show that our approach, taking into account differing agent abilities, is sometimes more effective at

Table 1: Comparison of reward augmentation methods for acting and subsequent agents. Each row reflects a different method. Each column depicts results for a different experimental scenario. Each entry pair depicts "step differences" for the acting agent and the subsequently acting (next) agent. A "step difference" is the difference between the number of steps the agent required to execute their policy as compared to what they would have required if they had tried to complete their task from the initial state without considering other agents. ∞ indicates the task was unachievable.

Method	Salad acting agent, next	Peanut acting, next	Salt acting, next	Cookies acting, next
Non-augmented reward	0, ∞	0, ∞	0, ∞	0, 0
Based on Krakovna et al.	1, 0	1, ∞	1, 1	1, -2
Our approach [Eq. 2]	1, 0	2, 0	1, 0	1, -2

avoiding negative side effects than either baseline. The last experiment (Cookies) shows how our approach (and the Krakovna-style baseline) can cause positive side effects for the next agent. Each experiment is described more below.

In Salad, the acting agent needs to collect the ingredients from the fridge. If it doesn't consider side effects, it doesn't close the fridge and ruins all the remaining ingredients, preventing the next agent from completing its task. By considering future tasks (whether another agent's or its own), the acting agent learns to take an extra step to close the fridge. In Peanut, preparing food contaminates the environment, and for the next agent to cook requires that the environment first be cleaned (taking one step), or disinfected (taking two steps) if the next agent has allergies. Only our approach takes the two extra steps to disinfect the kitchen because it considers that the other agent (unlike itself) has allergies. In Salt, if the acting agent does not put the salt shaker back on the shelves, the next agent can't complete its task. By considering future agents (in the Krakovna-style baseline and our approach) this side effect is avoided. However, the acting agent is tall and may put the salt on the top shelf (making it take longer for the next, shorter, agent to get it) if it considers that the next agent will be itself, as in the Krakovna-style baseline. Finally, in Cookies, the next agent's task is to bake cookies in the oven. Two steps are required to preheat the oven (turning on the oven and waiting). By considering the future task of the next agent, the acting agent (who was not using the oven) can turn on the oven to start preheating it, and save the next agent two steps.

3.2 Varying the Caring Coefficient

We investigated the effect of choosing different caring coefficients (α_1 and α_2) in Eq. (2) by monitoring the average reward collected by each of the agents in the Minecraft^{TM}-inspired Craft-World Environment (Figure 1).

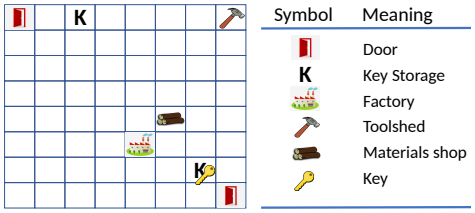


Figure 1: Craft-World Environment

Agents in this environment use tools and materials to construct artifacts such as boxes. Tools are stored in a toolshed in the upper right corner of the grid. Agents enter and exit the environment through doors in the upper left and lower right. They must collect materials and bring them to the factory for assembly. The factory requires a key for entry, and there is only one key, which can only be stored in one of two locations (marked with K). When considering other agents, the acting agent may elect to place the key in a position that is convenient for others, or may help other agents by anticipating their need for tools or resources and collecting them on their behalf.

In the experiment, agents enter at the top left door, tasked with making a box. The first agent learns a policy following Eq. (2). The second, subsequently acting, agent follows a fixed policy designed to optimize its own reward.

Figure 2 shows the reward that each agent gets (after training) as we vary the caring coefficient α_2. It also shows their average. When $\alpha_2 = 0$, the first agent is oblivious to others and exits the environment without returning the key, precluding the second agent from making a box. When $\alpha_2 > 0$, the agent becomes more considerate and returns the key on its way to the exit. As we increase the value of α_2, the first agent is incentivized to help the second agent, eventually (to its detriment) carrying extra materials to the factory for the second agent, garnering negative reward for this hard work and also, interestingly, lowering the average reward of the two agents.

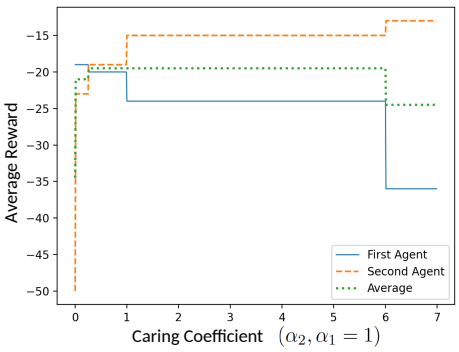


Figure 2: Effect of caring coefficients in the Craft-World environment.

3.3 Optimal Behaviours under Different Reward Augmentations

Figure 3 illustrates the difference between Eqs. (2), (3), (4), and the Krakovna-style baseline. In this experiment, the goal of the agents is to play with the doll and leave it somewhere in the environment for the next agent, and then exit the environment from their entry point; the agents get -1 reward for each step. There are six agents (circles 1-6 in Figure 3) in the environment with the same goal. They are shown at their individual entry points. Agents enter the environment separately; the acting agent is agent 1. In this scenario, $\alpha_1 = 1$, and $\alpha_2 = 10$. If we augment the acting agent's reward according to Eq. (2) (where the distribution of value functions is a uniform distribution over the optimal value function for each agent w.r.t. the goal of playing with the doll), the optimal policy is to place the doll as close as possible to the majority of the agents. If we use Eq. (3) the optimal policy is to place the doll so as to minimize the distance to the furthest agent. Finally, if we use Eq. (4) the optimal policy is to leave the doll where it is, because moving it causes negative side effects for agent 6. However if we use the approach based on Krakovna et al., the optimal policy is to leave the doll at agent 1's exit/entry point, so that the doll

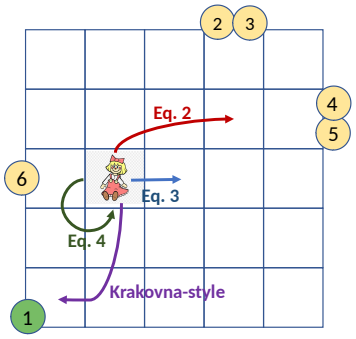


Figure 3: Each arrow points to where an optimal policy could leave the doll when Agent 1 receives auxiliary reward according to the approach labelling the arrow.

would be conveniently located for agent 1 if it were to re-enter. Furthermore, if we use non-augmented reward the agent does not have an incentive to place the doll in the environment and leaves with the doll (not shown in figure).

4 Related Work

We consider the relation of our work to Krakovna et al. [8] before briefly reviewing some other related work.

Krakovna et al. proposed modifying the agent's reward function to add an auxiliary reward based on its own ability to complete possible future tasks. A "task" corresponds to a reward function which gives reward of 1 for reaching a certain goal state, and 0 otherwise. In their simplest definition (not incorporating a baseline), the modified reward function is given by Eq (5), where r_1 is the original reward function, F is a distribution over tasks, V_i^* is the optimal value function for task i (when completed by the single agent itself), and β is a hyperparameter which determines the how much weight is given to future tasks. They interpret $1-\gamma$ (where γ is the discount factor) as the probability the agent will terminate its current task and switch to working on the future task, which leads to the $(1-\gamma)$ factor in the case where s' is not terminal. Their formulation is similar to (and inspired) our approach, though for r_K the value functions are restricted to be possible value functions for the agent itself (and so depend on what actions the agent itself can perform). In contrast, in our approach, we consider value functions that may belong to different agents with different abilities. Additionally, they assume the value functions are optimal. Below we show how under some conditions, our approach generalizes theirs.

$$r_K(s,a,s') = \begin{cases} r_1(s,a,s') + \beta(1-\gamma)\sum_i F(i)V_i^*(s') & \text{if } s' \text{ is not terminal} \\ r_1(s,a,s') + \beta\sum_i F(i)V_i^*(s') & \text{if } s' \text{ is terminal} \end{cases} \quad (5)$$

In the case where γ (the discount factor) is 1, r_K simplifies so that $r_K(s,a,s') = r_1(s,a,s')$ if s' is not terminal. Our Eq. (2) (substituted into Eq. (1)), in the case where $\gamma = 1$, can be rewritten as Eq. (6). Observe that if $\gamma = 1$, $\alpha_1 = 1$, $\alpha_2 = \beta$, and $\mathsf{P}(V) = \sum\{F(i) \mid V_i^* = V\}$ then $r_K = r_{\mathsf{value}}$. So in the undiscounted setting r_K is a special case of r_{value}.

$$r_{\mathsf{value}}(s,a,s') = \begin{cases} \alpha_1 \cdot r_1(s,a,s') & \text{if } s' \text{ is not terminal} \\ \alpha_1 \cdot r_1(s,a,s') + \alpha_2 \sum_{V\in\mathcal{V}} \mathsf{P}(V)\cdot V(s') & \text{if } s' \text{ is terminal} \end{cases} \quad (6)$$

Moving on, Turner et al.'s [12] Attainable Utility Preservation is an approach to avoiding side effects similar to Krakovna et al.'s. The agent's reward is modified, given a set $\mathcal{R}$ of other reward functions, to penalize actions that change the agent's *own* ability to optimize for the functions in $\mathcal{R}$. Considering other agents' abilities when avoiding side effects was informally discussed by Turner [11], and we have also investigated it in the context of symbolic planning [6]. Bussmann et al. [3] proposed "Empathetic Q-learning", an RL algorithm which learns not just the agent's Q-function, but an additional Q-function, Q_{emp}, which gives a weighted sum of the agent's value from taking an action, and the value that another agent will get. The value the other agent will get is approximated by considering what reward the first agent would get, if their positions were swapped. Finally, Du et al. [4] considered the problem of having an AI assist a human in achieving a goal. They proposed an auxiliary reward based on (an estimate of) the human's *empowerment* in a state. Empowerment is an information-theoretic quantity that measures ability to control the state (which could be influenced by the presence of irrelevant features that humans aren't interested in controlling).

References

[1] P. Alizadeh Alamdari, T. Q. Klassen, R. Toro Icarte, and S. A. McIlraith. Be considerate: Avoiding negative side effects in reinforcement learning. In *AAMAS*, pages 18–26, 2022.

[2] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[3] B. Bussmann, J. Heinerman, and J. Lehman. Towards empathic deep Q-learning. In *AISafety 2019*, volume 2419 of *CEUR Workshop Proceedings*, 2019.

[4] Y. Du, S. Tiomkin, E. Kiciman, D. Polani, P. Abbeel, and A. Dragan. AvE: Assistance via empowerment. In *NeurIPS*, 2020.

[5] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *ICAPS*, pages 540–550, 2020.

[6] T. Q. Klassen, S. A. McIlraith, C. Muise, and J. Xu. Planning to avoid side effects. In *AAAI*, 2022. To appear.

[7] V. Krakovna, L. Orseau, M. Martic, and S. Legg. Penalizing side effects using stepwise relative reachability. In *AISafety 2019*, volume 2419 of *CEUR Workshop Proceedings*, 2019.

[8] V. Krakovna, L. Orseau, R. Ngo, M. Martic, and S. Legg. Avoiding side effects by considering future tasks. In *NeurIPS*, 2020.

[9] J. Lebans. The threat from AI is not that it will revolt, it's that it'll do exactly as it's told. CBC Radio, April 2020.

[10] S. Saisubramanian, E. Kamar, and S. Zilberstein. A multi-objective approach to mitigate negative side effects. In *IJCAI*, pages 354–361, 2020. doi: 10.24963/ijcai.2020/50.

[11] A. Turner. Reframing impact. Blog post, `https://www.lesswrong.com/s/7CdoznhJaLEKHwvJW`, 2019.

[12] A. M. Turner, D. Hadfield-Menell, and P. Tadepalli. Conservative agency via attainable utility preservation. In *AIES*, pages 385–391, 2020. doi: 10.1145/3375627.3375851.

All You Need Is Supervised Learning: From Imitation Learning to Meta-RL With Upside Down RL

Kai Arulkumaran[*,1,2], **Dylan R. Ashley**[3,4,5], **Jürgen Schmidhuber**[3,4,5,6,7], **and Rupesh K. Srivastava**[7]

[1] ARAYA, Inc., Tokyo, Japan
[2] Imperial College London, London, UK
[3] The Swiss AI Lab IDSIA, Lugano, Switzerland
[4] Università della Svizzera Italiana (USI), Lugano, Switzerland
[5] Scuola Universitaria Professionale della Svizzera Italiana (SUPSI), Lugano, Switzerland
[6] King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[7] NNAISENSE, Lugano, Switzerland

Abstract

Upside down reinforcement learning (UDRL) flips the conventional use of the return in the objective function in RL upside down, by taking returns as input and predicting actions. UDRL is based purely on supervised learning, and bypasses some prominent issues in RL: bootstrapping, off-policy corrections, and discount factors. While previous work with UDRL demonstrated it in a traditional online RL setting, here we show that this single algorithm can also work in the imitation learning and offline RL settings, be extended to the goal-conditioned RL setting, and even the meta-RL setting. With a general agent architecture, a single UDRL agent can learn across all paradigms.

Keywords: supervised learning, imitation learning, offline reinforcement learning, goal-conditioned reinforcement learning, meta-reinforcement learning

Acknowledgements

This work was partially supported by the European Research Council (ERC, Advanced Grant Number 742870).

[*]Correspondence to kai_arulkumaran@araya.org. Work partially done while author was at NNAISENSE.

1 Introduction

Over time, the field of AI has exhibited significant convergence, with the increasing popularity of artificial neural networks (NNs) trained on large amounts of data resulting in improved performance across many benchmarks. While many earlier successes were based on supervised learning (SL), NNs have also revolutionised reinforcement learning (RL).

We begin by noting two further significant convergences in recent years. The first pertains to NN architectures. While convolutional NNs (CNNs) and recurrent NNs (RNNs) have for decades been applied to images and text, respectively, a new general-purpose architecture has emerged [16]. Originally applied to text, the Transformer architecture [34] has been successfully extended to images [6], as well as other structured input/outputs [16]. The advantage of such models is the ability to operate over *sets*, allowing a single, modular model to reuse knowledge over a dynamic set of inputs.

The second development is the rise of self-supervised learning (SSL) across different domains [36]: the use of "pretext tasks", constructing labels from unlabelled data in order to imitate SL training, has resulted in models that rival those trained with SL. SSL benefits from rich "supervisory signals", with models learning to *predict* transformations, f, of their inputs, x; loosely speaking, $p(f(x)|x)$, versus SL's $p(y|x)$, where y is a label. In a single domain, a variety of tasks can enable a model to gain complementary knowledge over different facets of the world.

We propose uniting these directions to create general learning agents, with the position that RL can itself be framed as an SL problem. This is not a novel proposition [25, 31, 32, 20, 12, 3, 17, 8, 11], but in contrast to prior works, we provide a general framework that includes online RL, goal-conditioned RL (GCRL) [28], imitation learning (IL) [26], offline RL [9], and meta-RL [29], as well as other paradigms contained within partially observed Markov decision processes (POMDPs) [23]. We build upon the proposal of upside down RL (UDRL) by Schmidhuber [31], the implementation of Srivastava et al. [32], and sequence modelling via Decision Transformers [3, 17]. We examine current implementations, discuss a generalisation of the framework, and then demonstrate a *single algorithm and architecture* on a standard control problem.

2 Upside Down RL

The core of UDRL is the policy, π, conditioned on "commands", c [31]. Given a dataset D of trajectories (states, s, actions, a, and rewards, r), the policy, parameterised by θ, is trained using an SL loss, $\mathcal{L}$, to map states and commands to actions:

$$\arg\min_{\theta} \mathbb{E}_{s,a,r\sim D}[\mathcal{L}(a, \pi(a|s, c; \theta))]. \tag{1}$$

In the initial implementation [32], $c = [d^H, d^R]$, where d^H is the desired (future) time horizon, $t_2 - t_1$, and d^R is the (time-bounded) return-to-go, $\sum_{t=t_1}^{t_2} r_t$. In a similar fashion to hindsight experience replay[1] [19, 1], the agent can be trained on data sampled from D, calculating d^H and d^R directly from the data. Trained thus, the agent can then achieve a desired return by sampling actions from its stochastic command-conditioned policy. Unlike typical RL agents, UDRL agents are capable of achieving low returns, medium returns, or high returns, based on the choice of d^R [32].

There are two key elements to the efficacy of UDRL—the first being the dataset (and its use) [27]. In the online RL setting, a UDRL agent can learn to perform reward maximisation in a manner akin to expectation maximisation: collect data using the policy, then train on the most rewarding data. This can be achieved by weighting the data [25], or controlling data storage/sampling [32]. Given an existing dataset of demonstrations, the agent can be trained in the offline RL setting [20, 17, 3]. D can be used for auxiliary tasks to improve learning/generalisation, such as via learning a world model [17].

The second key element is the use of commands. c can be any computable predicate that is consistent with the data [31]. In the initial implementation, the agent is trained to map observed actions a_t to the corresponding states s_t and $[d^H, d^R]$, where the latter is calculated from t to the terminal timestep T, allowing the agent to perform (undiscounted) credit assignment across the entire episode. During exploration in the environment (which generates more data for the agent), d^H is set to the mean of the most rewarding episodes in D, and d^R is sampled uniformly from values between the mean of the most rewarding episodes' returns, and the mean plus one standard deviation (encouraging optimism). After every environment interaction, d^H is decremented by 1 and d^R is decremented by r. If c is simplified to only contain the desired return, we recover reward-conditioned policies [20], and if $c = \emptyset$, we recover behavioural cloning (BC) [26], which is the simplest IL algorithm. Conversely, the trivial augmentation of c with a goal vector g extends UDRL to the GCRL setting [31, 12, 17], with zero-shot generalisation enabled via appropriate goal spaces (e.g., language [18]).

As the agent is trained using SL on observed data, UDRL bypasses several commmon issues in RL: bootstrapping (temporal-difference updates), off-policy corrections, and discount factors. This allows us to more easily focus on standard points in ML: the agent's ability to generalise to novel states and commands is based on the data available, the class of policies, and the optimisation process [8]. Srivastava et al. [32] trained fully-connected-/CNN-based UDRL agents using stochastic gradient descent on the cross-entropy loss, but replacing the architecture (e.g., with Transformers [17, 3])

[1] But without specific environment settings, e.g., symbolic state spaces or a distance-based goal-conditional reward functions.

or the optimisation (e.g., with evolutionary algorithms) are valid alternatives. While the move to sequence modelling allows the agent to benefit from greater contexts [17, 3], in general, a *stateful* agent is needed in order to deal with POMDPs, as well as more general computable predicates [31].

3 Generalised Upside Down RL in POMDPs

While many realistic problems cannot be captured by MDPs, they can be reasonably modelled by POMDPs, in which the agent only receives a partial observation o of the true state s. A principled approach to solving POMDPs is to keep a "belief" over the state, which can be achieved implicitly by training an RNN agent, which updates a hidden state vector h [37]. POMDPs encompass many other problems, such as hidden parameter MDPs [5], which consider related tasks, and even the general meta-RL setting (where h is typically augmented with previous action, reward and terminal indicator, $\mathbb{1}_{\text{terminal}}$ [7, 35]). POMDPs can also be related to generalisation in RL and the robust RL [22] setting, which considers worst-case performance. While various specialised algorithms exist for these different problem settings, recent results from Ni et al. [23] have shown that simple recurrent model-free RL agents are can perform well across the board. Motivated by this (and further related arguments by Schmidhuber [31]), we proceed by treating every environment as a POMDP, in which an agent attempts to learn using a general algorithmic framework.

A final ingredient enables a *single agent* to deal with all RL problem settings (plus IL) with *one model* — an architecture that can deal with arbitrary structured inputs and outputs (e.g., Perceiver IO [16]). Such a model allows c to be dynamic as needed: being null in the pure IL setting (where UDRL reduces to BC), being $[d^H, d^R, g]$ in the GCRL setting, and extending further beyond to incorporate SSL tasks. This alleviates the problem of using unlabelled demonstrations to bootstrap an RL agent, as the agent does not need to model the rewards achieved. Furthermore, the ability to adapt to different observation and action spaces allows such an agent to use third-person data for representation learning.[2]

Given this, we present a generalised algorithm for UDRL (Algorithm 1). Although nearly all RL problems consider the episodic setting (in which a terminal indicator is given before the environment resets), the following algorithm is applicable in both episodic and non-episodic MDPs, making it capable of continual/lifelong learning.

Algorithm 1 Generalised Upside Down RL

Require: E ▷ POMDP environment
Require: $\pi(a|o, c, h)$ ▷ Command-conditioned recurrent policy
Require: D ▷ Experience replay memory; existing data optional unless performing IL/offline RL

function RESET(E, π, D)
  Reset environment E and π's hidden state h ▷ h also contains the previous action, reward, and terminal indicator
  Get initial observation and goal (o, g) from E
  Sample c based on D and (o, g) ▷ Requires a procedure for sampling an initial command c based on observed data

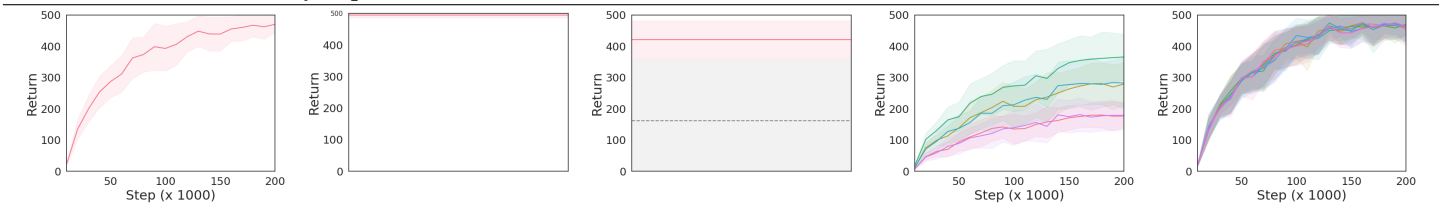
Train π on batches from D ▷ Data can be non-uniformly/adaptively sampled from D; auxiliary objectives can be used
if performing IL or offline RL without environment interaction **then return**

RESET(E, π, D)
while true **do**
  Act with $a, h \sim \pi(a|o, c, h)$
  Observe $(o', r, g', \mathbb{1}_{\text{terminal}})$ from environment transition ▷ g' given for goal-conditioned RL, $\mathbb{1}_{\text{terminal}}$ for an episodic MDP
  Update D with $(o, a, r, g, \mathbb{1}_{\text{terminal}})$ ▷ D may prioritise updates/remove old data
  Update h (to contain a and r) and c ▷ Requires a procedure for updating c, e.g., include g, decrement d^H
  Train π on batches from D
  if $\mathbb{1}_{\text{terminal}}$ **then** RESET(E, π, D)

4 Experiments

For our experiments, we use the classic CartPole control problem [2], where the agent receives a +1 reward for every timestep the pole is balanced, with a time limit of 500 timesteps. We adapt this environment to demonstrate how a single architecture can be used in the following settings: online RL, IL, offline RL, GCRL, and meta-RL. For the IL setting, we train the agent on 5 episodes with the maximum return of 500, collected from an online RL agent, and disable the reward and desired return inputs. For the offline setting, we train the agent on the worst 1000 trajectories from the online agent, with an average return of 162 ± 195. In the GCRL setting, at the beginning of each episode the agent is given an x goal position uniformly sampled from $[-1, 1]$, with the reward function set to $e^{-|x-g|}$. In the meta-RL setting, several environment parameters are randomly sampled at the beginning of each episode [21]. In the GCRL and meta-RL settings, during testing the agent is evaluated on a cross product of a uniform spread over goals/environment parameters.

[2] Correspondences between the agent and third party's observation/action spaces would be needed for true third-person IL [33]. To maintain such flexibility, the previous action and reward can be included in a dynamic c structure, instead of within h.



(a) Online RL (b) IL (c) Offline RL (d) GCRL (e) Meta-RL

Figure 1: Generalised UDRL agent trained under the (a) online RL (b) IL (c) offline RL (d) GCRL and (e) meta-RL settings in the CartPole environment. Results averaged over 10 test episodes per evaluation $\times$ 20 random seeds. The dashed line in (c) represents the distribution of returns in D. The different colours in (d) and (e) represent different goals/environment parameters, respectively.

As the observation and action spaces are constant, we use a simple base policy that consists of a linear layer, a sigmoid-gated linear layer, an LSTM [14], and a final linear layer to predict the logits of a categorical distribution. Each command (d^H and d^R), the goal, previous action, reward, and terminal indicator, are all embedded and concatenated with a learnable encoding vector, before being processed by a Transformer encoder layer [34]; the resulting vectors are aggregated using the $\max$ function and then used in the gated linear layer before the LSTM. For simplicity, all settings use the same architecture and hyperparameters; the code is available at `https://github.com/Kaixhin/GUDRL`.

As shown in Figure 1, the generalised UDRL agent is able to learn under all settings. While its performance may not match more specialised RL algorithms, it demonstrates that a single SL objective is sufficient for a variety of sequential decision making problems. Whilst it is possible to tune hyperparameters for individual tasks, a more compelling avenue for improving the performance of such agents is to incorporate further commands/tasks that relate to the environment's structure. This can include learning world models, SSL tasks [36], and more general computable predicates [31].

5 Discussion

Given the increased interest in the RL-as-SL paradigm, this work aims to construct a more general purpose agent/learning algorithm, but with more concrete implementation details and links to existing RL concepts than prior work [31]. A major question is whether such an idea can scale? As mentioned previously, optimisation and function approximation are key limiters. Other experiments with tabular representations have yielded UDRL agents that can learn more complex commands [31]; and in further experiments, the act of resetting weights has been useful for more complex agents. With less of the confounding problems of other RL algorithms, UDRL lays bare the problem of continual learning (and proactive interference, in particular) [10, 15].

Another discussion point UDRL introduces is the method by which (value) credit assignment can occur. While other agents typically consider the (discounted) episodic return, UDRL agents can incorporate a desired horizon. As such, UDRL might lend itself better to hierarchical RL [13]. In the current formulation, desired returns and goals are almost interchangeable in the command structure, but a more powerful formulation is to consider these as being variables that can be inferred themselves, i.e., a joint model (o, a, d^R, g). In the same way that inverse models can complement forward models, the joint distribution can interchange Q-functions, $Q(d^R|o, a, g)$ and return-conditioned policies, $\pi(a|o, d^R, g)$, utilising whichever is better for the situation (acting, or learning, in a data-dependent manner). Being able to infer the desired goal becomes particularly important when it comes to generalisation in the IL setting [4], and could even allow the use of "suboptimal" data [8]. Looking even further forward, with both hierarchy and a more intelligent command selection strategy, UDRL agents could be powerful vessels for implementing open-ended, goal-driven curiosity [24, 30].

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight Experience Replay. In *NeurIPS*, 2017.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Trans. SMC*, 13(5):834–846, 1983.

[3] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv:2106.01345*, 2021.

[4] P. De Haan, D. Jayaraman, and S. Levine. Causal Confusion in Imitation Learning. *NeurIPS*, 2019.

[5] F. Doshi-Velez and G. Konidaris. Hidden Parameter Markov Decision Processes: A Semiparametric Regression Approach for Discovering Latent Task Parametrizations. In *IJCAI*, 2016.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021.

[7] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv:1611.02779*, 2016.

[8] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. RvS: What is Essential for Offline RL via Supervised Learning? *arXiv:2112.10751*, 2021.

[9] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based Batch Mode Reinforcement Learning. *JMLR*, 6:503–556, 2005.

[10] W. Fedus, D. Ghosh, J. D. Martin, M. G. Bellemare, Y. Bengio, and H. Larochelle. On Catastrophic Interference in Atari 2600 Games. *arXiv:2002.12499*, 2020.

[11] H. Furuta, Y. Matsuo, and S. S. Gu. Generalized Decision Transformer for Offline Hindsight Information Matching. In *ICLR*, 2022.

[12] D. Ghosh, A. Gupta, A. Reddy, J. Fu, C. M. Devin, B. Eysenbach, and S. Levine. Learning to Reach Goals via Iterated Supervised Learning. In *ICLR*, 2021.

[13] N. Gürtler, D. Büchler, and G. Martius. Hierarchical Reinforcement Learning with Timed Subgoals. In *NeurIPS*, 2021.

[14] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[15] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. Transient Non-stationarity and Generalisation in Deep Reinforcement Learning. In *ICLR*, 2021.

[16] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, et al. Perceiver IO: A General Architecture for Structured Inputs & Outputs. *arXiv:2107.14795*, 2021.

[17] M. Janner, Q. Li, and S. Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In *NeurIPS*, 2021.

[18] Y. Jiang, S. S. Gu, K. P. Murphy, and C. Finn. Language as an Abstraction for Hierarchical Deep Reinforcement Learning. *NeurIPS*, 2019.

[19] L. P. Kaelbling. Learning to Achieve Goals. In *IJCAI*, 1993.

[20] A. Kumar, X. B. Peng, and S. Levine. Reward-conditioned Policies. *arXiv:1912.13465*, 2019.

[21] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Learning to Generalize: Meta-learning for Domain Generalization. In *AAAI*, 2018.

[22] J. Morimoto and K. Doya. Robust Reinforcement Learning. In *NeurIPS*, 2000.

[23] T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent Model-free RL is a Strong Baseline for Many POMDPs. *arXiv:2110.05038*, 2021.

[24] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Trans. Evol. Comput.*, 11(2):265–286, 2007.

[25] J. Peters and S. Schaal. Reinforcement Learning by Reward-weighted Regression for Operational Space Control. In *ICML*, pages 745–750, 2007.

[26] D. A. Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *NeurIPS*, 1988.

[27] M. Riedmiller, J. T. Springenberg, R. Hafner, and N. Heess. Collect & Infer-A Fresh Look at Data-efficient Reinforcement Learning. In *CoRL*, 2022.

[28] J. Schmidhuber. Learning Algorithms for Networks with Internal and External Feedback. In *Connectionist Models*, pages 52–61. Elsevier, 1990.

[29] J. Schmidhuber. On Learning How to Learn Learning Strategies. Technical Report FKI-198-94, Technische Universität München, 1994.

[30] J. Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE TAMD*, 2(3):230–247, 2010.

[31] J. Schmidhuber. Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions. *arXiv:1912.02875*, 2019.

[32] R. K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, and J. Schmidhuber. Training Agents Using Upside-down Reinforcement Learning. In *NeurIPS Deep RL Workshop*, 2019.

[33] B. C. Stadie, P. Abbeel, and I. Sutskever. Third-person Imitation Learning. In *ICLR*, 2017.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017.

[35] J. X. Wang, Z. Kurth-Nelson, H. Soyer, J. Z. Leibo, D. Tirumala, R. Munos, C. Blundell, D. Kumaran, and M. M. Botvinick. Learning to Reinforcement Learn. In *CogSci*, 2017.

[36] L. Weng. Self-Supervised Representation Learning. *lilianweng.github.io/lil-log*, 2019.

[37] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving Deep Memory POMDPs with Recurrent Policy Gradients. In *ICANN*, 2007.

Between Rate-Distortion Theory & Value Equivalence in Model-Based Reinforcement Learning

Dilip Arumugam
Department of Computer Science
Stanford University
dilip@cs.stanford.edu

Benjamin Van Roy
Department of Electrical Engineering
Department of Management Science & Engineering
Stanford University
bvr@stanford.edu

Abstract

The quintessential model-based reinforcement-learning agent iteratively refines its estimates or prior beliefs about the true underlying model of the environment. Recent empirical successes in model-based reinforcement learning with function approximation, however, eschew the true model in favor of a surrogate that, while ignoring various facets of the environment, still facilitates effective planning over behaviors. Recently formalized as the value equivalence principle, this algorithmic technique is perhaps unavoidable as real-world reinforcement learning demands consideration of a simple, computationally-bounded agent interacting with an overwhelmingly complex environment. In this work, we entertain an extreme scenario wherein some combination of immense environment complexity and limited agent capacity entirely precludes identifying an exactly value-equivalent model. In light of this, we embrace a notion of approximate value equivalence and introduce an algorithm for incrementally synthesizing *simple* and *useful* approximations of the environment from which an agent might still recover near-optimal behavior. Crucially, we recognize the information-theoretic nature of this lossy environment compression problem and use the appropriate tools of rate-distortion theory to make mathematically precise how value equivalence can lend tractability to otherwise intractable sequential decision-making problems.

Keywords: Bayesian reinforcement learning, Information theory, Model-based reinforcement learning, Efficient exploration

Acknowledgements

The authors gratefully acknowledge Christopher Grimm for initial discussions that provided an impetus for this work. Financial support from Army Research Office (ARO) grant W911NF2010055 is gratefully acknowledged.

1 Problem Formulation

We formulate a sequential decision-making problem as an episodic, finite-horizon Markov Decision Process (MDP) [4, 14] defined by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta, H \rangle$. $\mathcal{S}$ denotes a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a deterministic reward function providing evaluative feedback signals (in the unit interval) to the agent, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a transition function prescribing distributions over next states, $\beta \in \Delta(\mathcal{S})$ is an initial state distribution, and $H \in \mathbb{N}$ is the maximum episode length or horizon.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. As is standard in Bayesian reinforcement learning, both the transition function and reward function are not known to the agent and are consequently treated as random variables. With all other MDP components known a priori, the randomness in the model fully accounts for the randomness in the MDP, which is also a random variable. We denote by $\mathcal{M}^\star$ the true MDP with model $(\mathcal{R}^\star, \mathcal{T}^\star)$ that the agent interacts with and attempts to solve over the course of K episodes. Within each episode, the agent acts for exactly H steps beginning with an initial state $s_1 \sim \beta$. For each $h \in [H]$, the agent observes the current state $s_h \in \mathcal{S}$, selects action $a_h \sim \pi_h(\cdot \mid s_h) \in \mathcal{A}$, enjoys a reward $r_h = \mathcal{R}(s_h, a_h) \in [0, 1]$, and transitions to the next state $s_{h+1} \sim \mathcal{T}(\cdot \mid s_h, a_h) \in \mathcal{S}$.

A stationary, stochastic policy for timestep $h \in [H]$, $\pi_h : \mathcal{S} \to \Delta(\mathcal{A})$, encodes a pattern of behavior mapping individual states to distributions over possible actions. Letting $\{\mathcal{S} \to \Delta(\mathcal{A})\}$ denote the class of all stationary, stochastic policies, a non-stationary policy $\pi = (\pi_1, \dots, \pi_H) \in \{\mathcal{S} \to \Delta(\mathcal{A})\}^H$ is a collection of exactly H stationary, stochastic policies whose overall performance in any MDP $\mathcal{M}$ at timestep $h \in [H]$ when starting at state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$ is assessed by its associated action-value function $Q^\pi_{\mathcal{M},h}(s, a) = \mathbb{E}\left[\sum_{h'=h}^{H} \mathcal{R}(s_{h'}, a_{h'}) \;\middle|\; s_h = s, a_h = a\right]$, where the expectation integrates over randomness in the action selections and transition dynamics. Taking the value function as $V^\pi_{\mathcal{M},h}(s) = \mathbb{E}_{a \sim \pi_h(\cdot \mid s)}\left[Q^\pi_{\mathcal{M},h}(s, a)\right]$, we define the optimal policy $\pi^\star = (\pi^\star_1, \pi^\star_2, \dots, \pi^\star_H)$ as achieving supremal value $V^\star_{\mathcal{M},h}(s) = \sup_{\pi \in \{\mathcal{S} \to \Delta(\mathcal{A})\}^H} V^\pi_{\mathcal{M},h}(s)$ for all $s \in \mathcal{S}$, $h \in [H]$. We let $\tau_k = (s_1^{(k)}, a_1^{(k)}, r_1^{(k)}, \dots, s_H^{(k)}, a_H^{(k)}, r_H^{(k)}, s_{H+1}^{(k)})$ be a random variable denoting the trajectory experienced by the agent in the kth episode. Meanwhile, $H_k = \{\tau_1, \tau_2, \dots, \tau_{k-1}\} \in \mathcal{H}_k$ is a random variable representing the entire history of the agent's interaction within the environment at the start of the kth episode. Abstractly, a reinforcement-learning algorithm is a sequence of non-stationary policies $(\pi^{(1)}, \dots, \pi^{(K)})$ where, for each episode $k \in [K]$, $\pi^{(k)} : \mathcal{H}_k \to \{\mathcal{S} \to \Delta(\mathcal{A})\}$ is a function of the current history H_k. We note that no further restrictions on the state-action space $\mathcal{S} \times \mathcal{A}$, such as finiteness, have been made; notably, through our use of information theory, our algorithm may operate on any finite-horizon, episodic MDP although we leave the question of how to practically instantiate our algorithm for concrete settings of interest to future work.

2 Rate-Distortion Theory

We here provide a brief, high-level overview of rate-distortion theory [17] and encourage readers to consult [7] for more details. A lossy compression problem consumes as input a fixed information source $\mathbb{P}(X \in \cdot)$ and a distortion function $d : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}_{\geq 0}$ which quantifies the loss of fidelity by using a compression Z in place of the original X. Then, for any distortion threshold $D \in \mathbb{R}_{\geq 0}$, the rate-distortion function quantifies the fundamental limit of lossy compression as

$$\mathcal{R}(D) = \inf_{Z \in \Lambda} \mathbb{I}(X; Z) \triangleq \inf_{Z \in \Lambda} \mathbb{E}\left[D_{\mathrm{KL}}(\mathbb{P}\left(X \in \cdot \mid Z\right) \;||\; \mathbb{P}(X \in \cdot))\right] \qquad \Lambda \triangleq \{Z : \Omega \to \mathcal{Z} \mid \mathbb{E}\left[d(X, Z)\right] \leq D\},$$

where $\mathbb{I}(X; Z)$ denotes the mutual information and the infimum is taken over all random variables Z that incur bounded expected distortion, $\mathbb{E}\left[d(X, Z)\right] \leq D$. Naturally, $\mathcal{R}(D)$ represents the minimum number of bits of information that must be retained from X in order to achieve this bounded expected loss of fidelity. In keeping with the previous problem formulation, which does not assume discrete random variables, we note that the rate-distortion function is well-defined for information source and channel output random variables taking values on abstract alphabets [8]. Moreover, the problem of computing the rate-distortion function along with the channel that achieves its infimum is well-studied and solved by the classic Blahut-Arimoto algorithm [6, 1], which is computationally feasible for discrete channel outputs.

Just as in past work that studies satisficing in multi-armed bandit problems [15, 2, 3], we use rate-distortion theory to formalize and identify a simplified MDP $\widetilde{\mathcal{M}}_k$ that the agent will attempt to learn over the course of each episode $k \in [K]$. The episode dependence arises from utilizing the agent's current beliefs over the true MDP $\mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_k)$ as an information source to be lossily compressed.

3 The Value Equivalence Principle

As outlined in the previous section, the second input for a well-specified lossy-compression problem is a distortion function prescribing non-negative real values to realizations of the information source and channel output random variables $(\mathcal{M}^\star, \widetilde{\mathcal{M}})$ that quantify the loss of fidelity incurred by using $\widetilde{\mathcal{M}}$ in lieu of $\mathcal{M}^\star$. To define this function, we will leverage an approximate notion of value equivalence [10, 11]. For any arbitrary MDP $\mathcal{M}$ with model $(\mathcal{R}, \mathcal{T})$ and any stationary, stochastic policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, define the Bellman operator $\mathcal{B}^\pi_{\mathcal{M}} : \{\mathcal{S} \to \mathbb{R}\} \to \{\mathcal{S} \to \mathbb{R}\}$ as follows: $\mathcal{B}^\pi_{\mathcal{M}} V(s) \triangleq \mathbb{E}_{a \sim \pi(\cdot|s)} \left[\mathcal{R}(s,a) + \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)} \left[V(s')\right]\right]$. The Bellman operator is a foundational tool in dynamic-programming approaches to reinforcement learning [5] and gives rise to the classic Bellman equation: for any MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta, H \rangle$ and any non-stationary policy $\pi = (\pi_1, \ldots, \pi_H)$, the value functions induced by π satisfy $V^\pi_{\mathcal{M},h}(s) = \mathcal{B}^{\pi_h}_{\mathcal{M}} V^\pi_{\mathcal{M},h+1}(s)$, for all $h \in [H]$ and with $V^\pi_{\mathcal{M},H+1}(s) = 0, \forall s \in \mathcal{S}$.

For any two MDPs $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta, H \rangle$ and $\widehat{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, \widehat{\mathcal{R}}, \widehat{\mathcal{T}}, \beta, H \rangle$, Grimm et al. [10] define a notion of equivalence between them despite their differing models. For any policy class $\Pi \subseteq \{\mathcal{S} \to \Delta(\mathcal{A})\}$ and value function class $\mathcal{V} \subseteq \{\mathcal{S} \to \mathbb{R}\}$, $\mathcal{M}$ and $\widehat{\mathcal{M}}$ are value equivalent with respect to Π and $\mathcal{V}$ if and only if $\mathcal{B}^\pi_{\mathcal{M}} V = \mathcal{B}^\pi_{\widehat{\mathcal{M}}} V, \forall \pi \in \Pi, V \in \mathcal{V}$. In words, two different models are deemed value equivalent if they induce identical Bellman updates under any pair of policy and value function from $\Pi \times \mathcal{V}$. Grimm et al. [10] prove that when $\Pi = \{\mathcal{S} \to \Delta(\mathcal{A})\}$ and $\mathcal{V} = \{\mathcal{S} \to \mathbb{R}\}$, the set of all exactly value-equivalent models is a singleton set containing only the true model of the environment. The key insight behind value equivalence, however, is that practical model-based reinforcement-learning algorithms need not be concerned with modeling every granular detail of the underlying environment and may, in fact, stand to benefit by optimizing an alternative criterion besides the traditional maximum-likelihood objective [18, 12, 16]. Indeed, by restricting focus to decreasing subsets of policies $\Pi \subset \{\mathcal{S} \to \Delta(\mathcal{A})\}$ and value functions $\mathcal{V} \subset \{\mathcal{S} \to \mathbb{R}\}$, the space of exactly value-equivalent models is monotonically increasing.

For brevity, let $\mathfrak{R} \triangleq \{\mathcal{S} \times \mathcal{A} \to [0,1]\}$ and $\mathfrak{T} \triangleq \{\mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})\}$ denote the classes of all reward functions and transition functions, respectively. Recall that, with all uncertainty in $\mathcal{M}^\star$ entirely driven by its model, we may think of the support of $\mathcal{M}^\star$ as $\mathfrak{M} \triangleq \mathfrak{R} \times \mathfrak{T}$. We define a distortion function on pairs of MDPs $d : \mathfrak{M} \times \mathfrak{M} \to \mathbb{R}_{\geq 0}$ for any $\Pi \subseteq \{\mathcal{S} \to \Delta(\mathcal{A})\}$, $\mathcal{V} \subseteq \{\mathcal{S} \to \mathbb{R}\}$ as

$$d_{\Pi,\mathcal{V}}(\mathcal{M}, \widehat{\mathcal{M}}) = \sup_{\substack{\pi \in \Pi \\ V \in \mathcal{V}}} ||\mathcal{B}^\pi_{\mathcal{M}} V - \mathcal{B}^\pi_{\widehat{\mathcal{M}}} V||^2_\infty = \sup_{\substack{\pi \in \Pi \\ V \in \mathcal{V}}} \left(\max_{s \in \mathcal{S}} |\mathcal{B}^\pi_{\mathcal{M}} V(s) - \mathcal{B}^\pi_{\widehat{\mathcal{M}}} V(s)| \right)^2 .$$

In words, $d_{\Pi,\mathcal{V}}$ is the supremal squared Bellman error between MDPs $\mathcal{M}$ and $\widehat{\mathcal{M}}$ across all states $s \in \mathcal{S}$ with respect to the policy class Π and value function class $\mathcal{V}$.

4 Value-Equivalent Sampling for Reinforcement Learning

By virtue of the previous two sections, we are now in a position to define the lossy compression problem that characterizes a MDP $\widetilde{\mathcal{M}}_k$ that the agent will endeavor to learn in each episode $k \in [K]$ instead of the true MDP $\mathcal{M}^\star$. For any $\Pi \subseteq \{\mathcal{S} \to \Delta(\mathcal{A})\}$; $\mathcal{V} \subseteq \{\mathcal{S} \to \mathbb{R}\}$; $k \in [K]$; and $D \geq 0$, we define the rate-distortion function

$$\mathcal{R}^{\Pi,\mathcal{V}}_k(D) = \inf_{\widetilde{\mathcal{M}} \in \Lambda} \mathbb{I}_k(\mathcal{M}^\star; \widetilde{\mathcal{M}}) \triangleq \inf_{\widetilde{\mathcal{M}} \in \Lambda} \mathbb{E}\left[D_{\mathrm{KL}}(\mathbb{P}(\mathcal{M}^\star \in \cdot \mid \widetilde{\mathcal{M}}, H_k) \,||\, \mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_k)) \mid H_k \right], \tag{1}$$

where $\Lambda \triangleq \left\{ \widetilde{\mathcal{M}} : \Omega \to \mathfrak{M} \mid \mathbb{E}[d_{\Pi,\mathcal{V}}(\mathcal{M}^\star, \widetilde{\mathcal{M}}) \mid H_k] \leq D \right\}$. This rate-distortion function characterizes the fundamental limit of lossy MDP compression under our chosen distortion measure resulting in a channel

that retains the minimum amount of information from the true MDP $\mathcal{M}^\star$ while yielding an approximately value-equivalent MDP in expectation. Observe that this distortion constraint is a notion of approximate value equivalence which collapses to the exact value equivalence of Grimm et al. [10] as $D \to 0$. Meanwhile, as $D \to \infty$, we accommodate a more aggressive compression of the true MDP $\mathcal{M}^\star$ resulting in less faithful Bellman updates.

Algorithm 1 Posterior Sampling for Reinforcement Learning (PSRL) [19]

Input: Prior distribution $\mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_1)$
for $k \in [K]$ **do**
  Sample MDP $M_k \sim \mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_k)$
  Compute optimal policy $\pi^{(k)} = \pi^\star_{M_k}$
  Execute $\pi^{(k)}$ and observe trajectory τ_k
  Update history $H_{k+1} = H_k \cup \tau_k$
  Induce posterior $\mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_{k+1})$
end for

Algorithm 2 Value-equivalent Sampling for Reinforcement Learning (VSRL)

Input: Prior distribution $\mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_1)$, Distortion threshold $D \in \mathbb{R}_{\geq 0}$, Distortion function $d_{\Pi,\mathcal{V}} : \mathfrak{M} \times \mathfrak{M} \to \mathbb{R}_{\geq 0}$
for $k \in [K]$ **do**
  Compute channel $\mathbb{P}(\widetilde{\mathcal{M}}_k \in \cdot \mid \mathcal{M}^\star)$ achieving $\mathcal{R}_k^{\Pi,\mathcal{V}}(D)$ limit (Equation 1)
  Sample MDP $M^\star \sim \mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_k)$
  Sample compressed MDP $M_k \sim \mathbb{P}(\widetilde{\mathcal{M}}_k \in \cdot \mid \mathcal{M}^\star = M^\star)$
  Compute optimal policy $\pi^{(k)} = \pi^\star_{M_k}$
  Execute $\pi^{(k)}$ and observe trajectory τ_k
  Update history $H_{k+1} = H_k \cup \tau_k$
  Induce posterior $\mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_{k+1})$
end for

A standard algorithm for our problem setting is widely known as Posterior Sampling for Reinforcement Learning (PSRL) [19, 13], which we present as Algorithm 1, while our Value-equivalent Sampling for Reinforcement Learning (VSRL) is given as Algorithm 2. The key distinction between them is that, at each episode $k \in [K]$, the latter takes the posterior sample $M^\star \sim \mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_k)$ and passes it through the channel that achieves the rate-distortion limit (Equation 1) at this episode to get the M_k whose optimal policy is executed in the environment.

5 Discussion

Example 1 (A Multi-Resolution MDP). *For a large but finite $N \in \mathbb{N}$, consider a sequence of MDPs, $\{\mathcal{M}_n\}_{n\in[N]}$, which all share a common action space $\mathcal{A}$ but vary in state space ($\mathcal{S}_n$), reward function, and transition function. Moreover, for each $n \in [N]$, the rewards of the nth MDP are bounded in the interval $[0, \frac{1}{n}]$. An agent is confronted with the resulting product MDP, $\mathcal{M}$, defined on the state space $\mathcal{S}_1 \times \ldots \times \mathcal{S}_N$ with action space $\mathcal{A}$ and rewards summed across the N constituent reward functions. The transition function is defined such that each action $a \in \mathcal{A}$ is executed across all N MDPs simultaneously and the resulting individual transitions are composed to make a transition of $\mathcal{M}$. For any value of N, PSRL will persistently act to identify the transition and reward structure of all $\{\mathcal{M}_n\}_{n\in[N]}$.*

Example 1 presents a scenario where, as $N \uparrow \infty$, a complex environment retains a wealth of information, and yet, only a subset of that information may be within the agent's reach or even necessary for producing reasonably competent behavior. VSRL implicitly identifies a $M \ll N$ such that learning the subsequence of MDPs $\{\mathcal{M}_n\}_{n\in[M]}$ is sufficient for achieving a desired degree of sub-optimality.

The core impetus for this work is to recognize that, for complex environments, pursuit of the exact MDP $\mathcal{M}^\star$ may be an entirely infeasible goal. Consider a MDP that represents control of a real-world, physical system; learning a transition function of the associated environment, at some level, demands that the agent internalize laws of physics and motion to a reasonable degree of accuracy. More formally, take the random variable $M_1 \sim \mathbb{P}(\mathcal{M}^\star \in \cdot \mid H_1)$ reflecting the agent's prior beliefs over $\mathcal{M}^\star$. Denoting $\mathbb{H}(\cdot)$ as the entropy of a random variable, observe that identifying $\mathcal{M}^\star$ requires that a PSRL agent obtain exactly $\mathbb{H}(M_1)$ bits of information from the environment which, under an uninformative prior, may either be prohibitively large and exceed the agent's capacity constraints or simply be impractical under time and resource constraints.

6 Conclusion

In this work, we embrace the idea of *satisficing* [15, 2, 3]; as succinctly stated by Herbert A. Simon during his 1978 Nobel Memorial Lecture, "decision makers can satisfice either by finding optimum solutions for

a simplified world, or by finding satisfactory solutions for a more realistic world." Rather than spend an inordinate amount of time trying to recover an optimum solution to the true environment, VSRL pursues optimum solutions for a sequence of simplified environments. Future work will develop a complementary regret analysis that demonstrates how finding such optimum solutions for simplified worlds ultimately acts as a mechanism for achieving a satisfactory solution for the realistic, complex world. Naturally, the loss of fidelity between the simplified and true environments translates into a fixed amount of regret that an agent designer consciously and willingly accepts for two reasons: (1) they expect a reduction in the amount of time, data, and bits of information needed to identify the simplified environment and (2) in tasks where the environment encodes irrelevant information and exact knowledge isn't needed to achieve optimal behavior [9, 10, 11], a VSRL agent may still identify the optimal policy while maintaining greater sample efficiency than traditional PSRL.

References

[1] Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.

[2] Dilip Arumugam and Benjamin Van Roy. Deciding what to learn: A rate-distortion approach. In *International Conference on Machine Learning*, pages 373–382. PMLR, 2021.

[3] Dilip Arumugam and Benjamin Van Roy. The value of information when deciding what to learn. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[5] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[6] Richard Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972.

[7] Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

[8] Imre Csiszár. On an extremum problem of information theory. *Studia Scientiarum Mathematicarum Hungarica*, 9, 1974.

[9] Amir-massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494. PMLR, 2017.

[10] Christopher Grimm, Andre Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[11] Christopher Grimm, Andre Barreto, Gregory Farquhar, David Silver, and Satinder Singh. Proper value equivalence. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[12] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6120–6130, 2017.

[13] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *International Conference on Machine Learning*, pages 2701–2710. PMLR, 2017.

[14] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[15] Daniel Russo and Benjamin Van Roy. Satisficing in time-sensitive bandit learning. *Mathematics of Operations Research*, 2022.

[16] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, et al. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[17] Claude E. Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec., March 1959*, 4:142–163, 1959.

[18] David Silver, Hado Hasselt, Matteo Hessel, et al. The Predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pages 3191–3199. PMLR, 2017.

[19] Malcolm JA Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 943–950, 2000.

Learning Relative Return Policies With Upside-Down Reinforcement Learning

Dylan R. Ashley *[1,2,3] **Kai Arulkumaran** [4,5]

Jürgen Schmidhuber [1,2,3,6,7] **Rupesh Kumar Srivastava** [7]

[1] The Swiss AI Lab IDSIA, Lugano, Switzerland
[2] Università della Svizzera italiana (USI), Lugano, Switzerland
[3] Scuola universitaria professionale della Svizzera italiana (SUPSI), Lugano, Switzerland
[4] ARAYA Inc., Tokyo, Japan
[5] Imperial College London, London, UK
[6] AI Initiative, King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[7] NNAISENSE, Lugano, Switzerland

Abstract

Lately, there has been a resurgence of interest in using supervised learning to solve reinforcement learning problems. Recent work in this area has largely focused on learning command-conditioned policies. We investigate the potential of one such method—upside-down reinforcement learning—to work with commands that specify a desired relationship between some scalar value and the observed return. We show that upside-down reinforcement learning can learn to carry out such commands online in a tabular bandit setting and in CartPole with non-linear function approximation. By doing so, we demonstrate the power of this family of methods and open the way for their practical use under more complicated command structures.

Keywords: upside-down reinforcement learning, command-conditioned policies, reinforcement learning, supervised learning, artificial neural networks

Acknowledgements

This work was supported by the European Research Council (ERC, Advanced Grant Number 742870) and the Swiss National Supercomputing Centre (CSCS, Project s1090). We also thank both the NVIDIA Corporation for donating a DGX-1 as part of the Pioneers of AI Research Award and IBM for donating a Minsky machine.

*Correspondence to dylan.ashley@idsia.ch

1 Introduction

Artificial neural networks in their current incarnation are better suited to solving supervised learning problems than they are for solving reinforcement learning problems. Recently, a family of techniques based on upside-down reinforcement learning (Schmidhuber, 2019; Srivastava et al., 2019) has been proposed, which solve RL problems by framing them as supervised learning problems. These techniques all focus on directly learning a command-conditioned policy; they explicitly learn a mapping from states and commands to actions. Already, these methods have had remarkable success in solving offline RL problems (Chen et al., 2021; Janner et al., 2021; Kumar et al., 2019), but still struggle to achieve competitive results in online reinforcement learning problems.

This work investigates the learnability of *morethan* commands: commands that specify a goal in the form of a scalar value, a horizon, and the desired relation between the given scalar and the observed return under the given horizon. Here we show that these commands are learnable online with traditional UDRL in simple settings. By doing so, we demonstrate the practical potential of the flexibility of commands offered by the UDRL framework. We hope that this flexibility can be leveraged in the future to empower these methods to render them competitive in complicated online and continual learning problems.

2 Related Work

The idea of leveraging iterated supervised learning to solve reinforcement learning dates back to at least the work on reward-weighted regression by Peters and Schaal (2007), who brought the earlier work of Dayan and Hinton (1997) to the domain of operational space control and RL. However, Peters and Schaal (2007) only looked at the immediate-reward RL setting. This was extended to the episodic setting separately by Wierstra et al. (2008a) and then by Kober and Peters (2011). Wierstra et al. (2008a) went even further and also extended RWR to partially observable Markov decision processes, whereas Kober and Peters (2011) applied it to motor learning in robotics. Separately, Wierstra et al. (2008b) extended RWR to perform fitness maximization for evolutionary methods. Hachiya et al. (2009) and Hachiya et al. (2011) later found a way of reusing old samples to improve RWR's sample complexity. Much later, Peng et al. (2019) modified RWR to produce an algorithm for off-policy RL, using deep neural networks as function approximators.

Upside-down reinforcement learning as a command-conditioned method of using SL for RL emerged in Schmidhuber (2019) and Srivastava et al. (2019), with Ghosh et al. (2021) afterwards introducing a similar idea in a multi-goal context. Kumar et al. (2019) applied UDRL to offline RL and sometime later Chen et al. (2021) and then Janner et al. (2021) showed the potential of the UDRL framework to be competitive in this context when paired with the Transformer architecture of Vaswani et al. (2017). Furuta et al. (2021) generalized this Transformer variant to solve a broader class of problems.

3 Background

Reinforcement learning considers an agent receiving rewards through interacting with an environment. RL is usually modeled as a *Markov decision process* where, at each step t, the agent observes the current state of the environment $s_t \in S$, selects an action $a_t \in A$, and consequently receives a reward $r_{t+1} \in \mathbb{R}$. We say that the rule an agent follows to select actions is its *policy*, which we write as $\pi : S \times A \to \mathbb{R}$ where $\pi(s, a)$ is the probability of the agent taking action a when the environment is in state s. The objective is typically to find a policy that maximizes the sum of the temporally-discounted rewards—known as the return.

Upside-down reinforcement learning breaks the RL problem in two, using commands as an intermediary. Specifically, it divides the agent into two sub-agents: a sub-agent that interacts with the environment in an attempt to carry out a command, i.e., a *worker*; and a sub-agent that issues commands to the worker that maximize the expected value of the return, i.e., a *manager*. The problem of learning an optimal policy then becomes the problem of having both sub-agents learn to carry out their respective roles optimally. The key benefit of this formulation is that the worker solves a supervised learning problem which allows us to bring a portion of the RL problem's complexity into the domain of supervised learning, which—as we previously remarked—is the principal domain of ANNs.

Traditionally, commands in UDRL are given as desire-horizon pairs, i.e., (d, h) with $d \in \mathbb{R}$ and $h \in \mathbb{N}$. Semantically, these commands direct the agent to achieve a return equal to d in the next h steps. Training in UDRL is done with the hindsight method wherein the agent is trained to predict what action it took given the current state and command (g, h) where g is the actual return observed. For UDRL to solve online RL reward-maximization problems, the commands being issued by the manager should demand higher and higher returns as time goes on.

Morethan units—as first described in Schmidhuber (2019)—act as an additional element of the command. They serve to denote the relation between the true desired return and d. In their simplest form, morethan units capture a boolean relationship, i.e., "get a return larger than this value". However, the concept of morethan units is flexible enough to instead encode a ternary, additive, multiplicate, or more complicated relationship.

4 Experimental Setup

We experiment with two reinforcement learning settings here: a simple six-armed bandit and the well-known CartPole domain (Barto et al., 1983). Our toy bandit domain is intended to give us a deep look at how upside-down reinforcement learning operates with the morethan unit in a basic deterministic tabular setting. With our CartPole domain, we hope to evaluate the learnability of commands with morethan units when working with non-linear function approximation in more mainstream reinforcement learning domains.

In our six-armed bandit domain, pulling the i-th arm always results in a reward of exactly i. If the agent wanted to maximize the return, it should thus always pull the 6-th arm. Since we are working with UDRL, though, in both domains, we issue commands to the agent in the form (d, h, m) where d is the desired return, h is a horizon, and m is the *morethan unit*: a ternary digit which denotes the desired relation between d and h. If m is set to -1, then the agent is commanded to obtain a return of less than d in the next h steps; if m is set to 0, then the agent should obtain a return of exactly d in the next h steps; and, finally, if m is set to 1, then the agent should obtain a return greater than d in the next h steps.

Our implementation of UDRL follows the one used by Srivastava et al. (2019) with some domain-specific adaptations. For the bandit setting, we exploit the single-step episodes by learning a two-layered policy network that uses a one-hot encoding of the command as input. Here our network uses ReLU activation and orthogonal initialization and is trained using SGD under a step size of 0.01.

When acting in the bandit setting, we always issue a command with m set to 1. We record the results of these actions in a 100-episode experienced replay buffer. To train the policy network with this buffer, we sample a batch of 16 episodes and generate a permutation to pair each episode in the sample to another random episode in the sample. Recall that each of these samples will be in the form $b_i = ((d_i, h_i, m_i, g_i), a_i)$, where d_i is a desired return, h_i is a horizon (here always 1), m_i is the morethan unit (again here always 1), g_i is an observed return, and a_i is an action. For two samples b_i and b_j, we train the network to predict a_i given an input (g_i, h_i, m) where m set to 1 if $g_j > g_i$, 0 if $g_j = g_i$, and -1 if $g_j < g_i$. This novel sampling strategy is critical here as it provides us with a non-parametric way of providing in-distribution samples to the network. For the bandit setting, we train once using a randomly permuted batch and once with the batch permuted such that each b_i is matched with itself. We add 16 exploratory episodes to the buffer and repeat this training process 16 times for each iteration of the UDRL algorithm. We report the results of 10 runs of this with $25\,000$ steps each in Section 5.

Our network in the CartPole setting has a single hidden layer of 32 gated fast weights with Tanh activation and orthogonal initialization. We use Adam (Kingma and Ba, 2014) to train this network under a step-size of 0.0008 and with the other hyperparameters set as recommended in Kingma and Ba (2014).

For the CartPole setting, we use a similar training regimen as in the bandit setting but use an unbounded buffer to which we add 5 episodes in each iteration. Here, in each iteration, we train with 800 batches of size 256 for which we generate seven rather than two independent permutations (one of which matches samples to themselves as in the bandit setting). Note that the ability to generate exponentially more samples here is a key advantage of training with morethan units. We report the results of 30 runs of the above with $500\,000$ steps each in Section 5.

5 Results

To evaluate learning with morethan units in the bandit setting, we look at how the action probability density of the final learned command-conditioned policy changes as a function of d and m. The mean of these densities over the 10 runs is shown in Figure 1. Of crucial importance here, the learned action probabilities density is concentrated in the correct regions for each setting of the morethan unit. For example, when the morethan unit is set to 0, the action probability density is highly concentrated in the i-th action for each i desired return. Interestingly, when the morethan unit is set to -1, the action probability density clusters around the $(i-1)$-th action for each of the i desired returns, but when the morethan unit is set to 1, the action probability density is clustered around the most-rewarding action.

To evaluate learning with morethan units in the CartPole domain, we compare the observed return as a function of d and m when acting under the final learned command-conditioned policy. The results of this over the 30 runs is given in Figure 2. A similar but somewhat inverted relation appears here compared to the bandit results. When the morethan bit is set to 0, the observed returns roughly correspond to the desired returns and only start to taper off slightly as the desired return reaches the higher end of the spectrum. Likewise, when the morethan bit is set to 1, the observed return is consistently only marginally higher than the desired return for most of the values tested. However, when the morethan bit is set to -1, the observed returns are always drastically smaller than d.

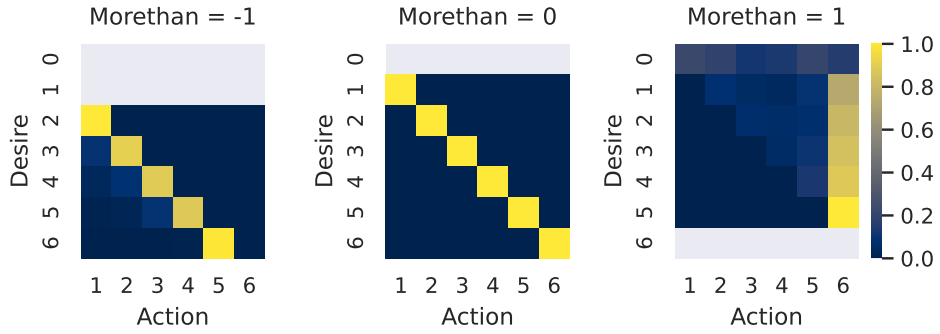


Figure 1: The mean action probabilities of the learned command-conditioned policy in the bandit setting. Note how the probability densities seem to cluster around the highest-valued valid action.

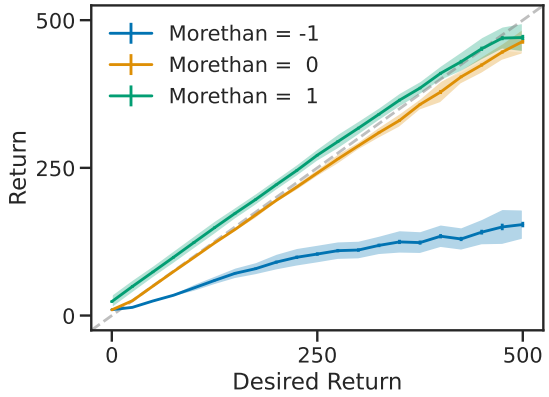


Figure 2: The desired return versus the observed return under the learned command-conditioned policy in the CartPole setting. Standard deviation is shown with shading. Note how the observed returns center around the lower-end of the valid returns.

6 Discussion

This work aimed to show that morethan units were a learnable concept in the context of upside-down reinforcement learning. The results presented in Section 5 clearly demonstrate that this is the case. Importantly, these results show that morethan units can be used with non-linear function approximation in mainstream online reinforcement learning domains and thus have the potential to be usable in some of the more advanced applications of reinforcement learning.

In our bandit experiments, the emphasis of the learned policy on maximal valid rewards throughout was unexpected. We hypothesize that this is due to the restrictive buffer size mixed with the training regimen whereby we perpetually demand increasingly large returns. This change in demand means that an experience replay buffer will be increasingly filled by large returns causing UDRL to increasingly oversamples large returns when training its policy network. The consequence of this is that UDRL will be prone to a specific form of forgetting. We hypothesize that this may be part of the reason that the UDRL paradigm has achieved its greatest successes primarily in the offline RL setting.

An initial inspection of the results of the CartPole experiment seems to produce something inconsistent with the above hypothesis. However, recall here that the buffer size is unbounded. This means that even though the buffer is increasingly filled with large values, the probability of a training sample having the morethan unit set to -1 grows as training progresses. Conversely, the probability of a training sample having the morethan unit set to 1 shrinks as training progresses. Together with our previous hypothesis, this suggests that the distribution of training samples drastically affects the exact nature of the learned policy here, even when not otherwise affecting its validity.

7 Conclusion and Future Work

In this work, we set out to demonstrate the flexibility afforded by the use of commands in upside-down reinforcement learning. We accomplished this by experimenting with commands in the form (d, h, m), where d is a return, h is a horizon, and m is a morethan unit which denotes whether the agent is being directed to receive a return greater than, less than, or equal to d in the next h steps. We showed that commands of this form are learnable in two reinforcement learning environments: a six-armed bandit and the well-known CartPole environment. Through doing this, we hope to pave the way for later work to leverage advanced command structures in the hopes of successfully applying UDRL and related methods to challenging online RL problems.

Future work will look at other advanced command structures. We also plan to experiment with the morethan unit under more complicated neural network architectures—such as the Transformer architecture. This would, in turn, allow us to understand better the utility of morethan units in significantly more challenging applications.

We note that the challenges faced by the worker portion of the upside-down reinforcement learning agent are similar to the challenges faced by continual learning agents. Future work will investigate how this relationship can be leveraged to specify good command structures for UDRL and related algorithms.

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(5), 834–846. https://doi.org/10.1109/TSMC.1983.6313077

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). *Decision transformer: Reinforcement learning via sequence modeling*. arXiv. https://arxiv.org/abs/2106.01345

Dayan, P., & Hinton, G. E. (1997). Using expectation-maximization for reinforcement learning. *Neural Computation*, *9*(2), 271–278. https://doi.org/10.1162/neco.1997.9.2.271

Furuta, H., Matsuo, Y., & Gu, S. S. (2021). *Generalized decision transformer for offline hindsight information matching*. arXiv. https://arxiv.org/abs/2111.10364

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C. M., Eysenbach, B., & Levine, S. (2021). Learning to reach goals via iterated supervised learning. *Proceedings of the International Conference on Learning Representations*. https://openreview.net/forum?id=rALA0Xo6yNJ

Hachiya, H., Peters, J., & Sugiyama, M. (2009). Efficient sample reuse in em-based policy search. *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, *5781*, 469–484. https://doi.org/10.1007/978-3-642-04180-8_48

Hachiya, H., Peters, J., & Sugiyama, M. (2011). Reward-weighted regression with sample reuse for direct policy search in reinforcement learning. *Neural Computation*, *23*(11), 2798–2832. https://doi.org/10.1162/NECO_a_00199

Janner, M., Li, Q., & Levine, S. (2021). *Reinforcement learning as one big sequence modeling problem*. arXiv. https://arxiv.org/abs/2106.02039

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv. https://arxiv.org/abs/1412.6980

Kober, J., & Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, *84*(1-2), 171–203. https://doi.org/10.1007/s10994-010-5223-6

Kumar, A., Peng, X. B., & Levine, S. (2019). *Reward-conditioned policies*. arXiv. https://arxiv.org/abs/1912.13465

Peng, X. B., Kumar, A., Zhang, G., & Levine, S. (2019). *Advantage-weighted regression: Simple and scalable off-policy reinforcement learning*. arXiv. https://arxiv.org/abs/1910.00177

Peters, J., & Schaal, S. (2007). Reinforcement learning by reward-weighted regression for operational space control. *Proceedings of the 24th International Conference on Machine Learning*, *227*, 745–750. https://doi.org/10.1145/1273496.1273590

Schmidhuber, J. (2019). *Reinforcement learning upside down: Don't predict rewards - just map them to actions*. arXiv. https://arxiv.org/abs/1912.02875

Srivastava, R. K., Shyam, P., Mutz, F., Jaskowski, W., & Schmidhuber, J. (2019). *Training agents using upside-down reinforcement learning*. arXiv. https://arxiv.org/pdf/1912.02877.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*, 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008a). Episodic reinforcement learning by logistic reward-weighted regression. *Proceedings of the 18th International Conference on Artificial Neural Networks*, *5163*, 407–416. https://doi.org/10.1007/978-3-540-87536-9_42

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008b). Fitness expectation maximization. *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, *5199*, 337–346. https://doi.org/10.1007/978-3-540-87700-4_34

General and Scalable Hierarchical Reinforcement Learning

Bernardo Avila Pires[*,*]
DeepMind
bavilapires@deepmind.com

Feryal Behbahani[*]
DeepMind
feryal@deepmind.com

Hubert Soyer[*]
DeepMind
soyer@deepmind.com

Kyriacos Nikiforou[*]
DeepMind
knikiforou@deepmind.com

Thomas Keck[*]
DeepMind
thomaskeck@deepmind.com

Zhengdong Wang
DeepMind
zhengdong@deepmind.com

Satinder Singh
DeepMind
baveja@deepmind.com

Abstract

Hierarchical Reinforcement Learning provides a framework for building agents with abstract, temporally extended behaviours that can be useful and generalise across tasks and timescales. In spite of these advantages, developing effective hierarchical agents has remained a challenge in visually complex, partially observable 3D environments. This work details a path towards a hierarchical agent with an explicit focus on scale and the long-term goal of zero-shot generalisation to tasks unseen during training. The guiding principles of scale and generalisation are reflected in the agent's design.

The agent features a goal conditioned low level controller which acts in the underlying environment based on grounded goals set by a high level controller. Given a trajectory of experience, the low level controller is trained to achieve future states chosen in hindsight. It has no notion of externally defined reward, and it has minimal assumptions about the origin and quality of its training data. The high level controller is trained to maximise extrinsic reward by setting goals for the low level controller to achieve. It can therefore act at a lower temporal resolution. This explicit separation of high and low level training objectives gives rise to scaling advantages in terms of data, task complexity and compute.

We highlight design choices, trade-offs and challenges for future research, and present results with a concrete implementation on the Hard Eight tasks, a set of challenging, visually complex and partially observable embodied 3D tasks. We demonstrate that our proof-of-concept implementation can learn temporal and behavioural abstractions from data, and use these abstractions as part of the solution to more complex tasks. Furthermore, our results show the flexibility of our approach to make use of pre-collected and uncurated data to boost performance, but also to be effective when the data for learning behavioural abstractions must be generated by the agent itself.

Keywords: hierarchical reinforcement learning, scalable reinforcement learning, generalisation

Acknowledgements

We would like to thank the following people for contribution ideas, as well as for their feedback and support: Doina Precup, Loic Matthey, Joe Marino and Steph Hughes-Fitt.

[*]corresponding author.

1 Introduction

While temporal and behavioural abstractions arising from Hierarchical Reinforcement Learning (HRL, Dayan and Hinton, 1992; Sutton et al., 1999) are appealing advantages, successfully training HRL agents to solve complex tasks in visually rich and partially observable 3D environments has remained challenging. This work details a path towards a hierarchical agent with a focus on scaling to such environments. We highlight design choices, trade-offs and challenges for future research and present an empirical evaluation of a proof-of-concept implementation in such complex procedurally generated, partially observable 3D tasks.

The agent design is based on two main principles

- Scalability
  - in terms of the environment, observation space and dynamics, as well as the agent
  - in terms of its architecture, model size and the data it can consume.
- Generalisation
  - with a focus on zero-shot generalisation to tasks unseen during training.

We believe that hierarchical reinforcement learning can be a prime ingredient in advancing both scalability and generalization to tasks unseen during training.

2 Agent Design Choices

The guiding principles of scalability and generalization are reflected in our agent's design. Similar to Dayan and Hinton (1992), we enforce a separation of the agent into a goal conditioned *Low Level Controller* (LLC) which directly acts in the environment, and a *High Level Controller* (HLC) which observes the environment at a lower temporal resolution and provides high level goals for the LLC to achieve. This separation of the two components is explicit, no gradients flow between them and they are trained using separate objectives. We refer to the high level actions that are communicated from high to low level controller as *goals* or *options*, and the actions available to the low level controller as *primitive actions* (Sutton et al., 1999).

The low level controller is a goal-conditioned policy trained in hindsight to reach a future state or observation (Andrychowicz et al., 2017). Given a trajectory of experience, a randomly selected future observation is encoded with a neural network and used as a goal that the policy is trained to reach. The space of goals is therefore continuous, rich, and grounded. For this reason, there is also no need to choose a fixed number of options a priori. The goal space can in principle express all possible behaviours and therefore lends itself naturally to continual learning.

The high level controller does not interact with the environment directly. Instead, it issues goals to the low level controller and waits for the low level controller to execute. Once the low level controller completes its execution, it sends a compressed version of what it has experienced to the high level controller, which then picks the next goal. Due to this temporal abstraction, episodes that are hundreds of low-level transitions long only require a handful of decisions from the high level controller. The low level controller with which the high level controller interacts can either be pre-trained or trained in tandem with the high-level controller, based on data generated by the agent itself. Due to the offline/online nature of training we call our agent **H**ierarchical **H**ybrid **O**ffline-**O**nline or **H2O2** for short.

Communication from the high to the low level is not limited to only goals. We grant the high level controller additional control by allowing it to modulate low level behaviour by e.g. manipulating the low level controller's policy entropy, setting a threshold for when the low level controller should stop, or bypassing the low level controller entirely and executing non-hierarchical, primitive actions in the environment directly. Similarly, we can also enrich the information that is communicated from the low to the high level. This ranges from simply the observation at the time the goal is reached, to compressed histories of what has been seen and additional information about e.g. option termination (time-out, goal reached, goal unachievable).

3 Consequences of Design Choices

The design choices introduced above have a number of implications on scalability and generalization to unseen tasks. The hierarchical structure opens up opportunities for better exploration using the temporarily extended options, efficient long term credit assignment, abstract reasoning about solutions in a semantically meaningful space, and may allow more efficient planning over much longer-horizons.

Explicitly separating the training objectives of the high and low level controller gives rise to scaling advantages:

- The low-level controller can be trained with various types of experience (e.g. pre-recorded in a dataset, human generated, collected from the high level controller's experience or exploratory policies)
- Training of the two components can run at a different rate and with different resources. Not being bound to experience produced at a certain rate allows scaling up the model.
- The high level can truly act and learn at a lower frequency than the low level. There is no gradient flowing between the two, data/goals are the only interface.
- Additional auxiliary predictors like affordance, transition and value models are learned alongside the LLC and could be leveraged by the HLC as components e.g. for exploration and planning.

The design choices also open up opportunities for better zero-shot generalisation to novel tasks:

- Low level policies are short horizon and as such, each behaviour will appear in a variety of situations, and be in-distribution even for novel and unseen tasks.
- How the HLC perceives the world and the space in which it takes decisions are determined by the output and goal space of the low level controller. Incorporating appropriate abstractions in these spaces can advance zero-shot generalisation or simplify learning dynamics models.
- The high level controller can compose options in novel ways to act, explore and plan in a temporally extended and structured manner. Jumpy planning is a particularly exciting direction to improve zero-shot generalization.

And finally, an explicit, grounded interface between low and high level controllers naturally provides a starting point for behaviour analysis.

4 Core Research Challenges

While this design brings many advantages, we have also identified core research challenges:

- **State and perceptual abstractions.** How the HLC perceives the environment may strongly impact the data efficiency and final performance of the agent.
- **Goal abstraction.** What is and is not represented in a goal has large implications on the agent's ability to re-use, model and re-combine its skills.
- **Goal coverage.** Grounding goals by defining them in hindsight during LLC training offloads the question of what behaviours/goals are covered onto the data distribution the LLC is trained on. Training the LLC on pre-collected data allows explicit control over this distribution but limits the data to a static set. On the other hand, training the LLC on data collected by the agent itself can lead to a lack of diversity.
- **HLC data efficiency.** Since the HLC takes only few steps per episode, it has to learn efficiently from a much smaller number of transitions than a non-hierarchical agent.
- **Learning and using models.** Learning models with temporal abstraction and doing jumpy planning is challenging, especially in partially observable environments.

5 Proof-of-Concept Implementation

We implemented a proof-of-concept H2O2 to demonstrate the merit of our approach, and as a first step in the direction of our vision. We built an agent that can learn options and then use them effectively to solve tasks it is trained on, in a complex, partially observable 3D environment.

Figure 1 shows a diagram of our implementation, and how its components interact with each other, with the dataset and the environment. The diagram also indicates what components are trained in offline and online fashion. The stream of online experience to feed back into the dataset is optional.

To train the LLC, we used offline V-Trace (Mathieu et al., 2021), a state-of-the-art offline RL method and hindsight experience replay (Lynch et al., 2020). We made as few assumptions as possible about quality and origin of the data, and made it a responsibility of the offline RL algorithm to handle the training data correctly. The high level controller is implemented using the state-of-the-art Muesli RL algorithm (Hessel et al., 2021).

We considered two regimes for training the agent:

- (from dataset) First the LLC is trained offline on a pre-collected dataset, and then the HLC is trained with a fixed LLC. The experience is collected by a trained agent acting in a large number of levels that share the same physics and general principles as the levels we train our agent on. We note that the data-collecting agent performs very poorly on tasks from the Hard Eight suite (Gulcehre et al., 2019).

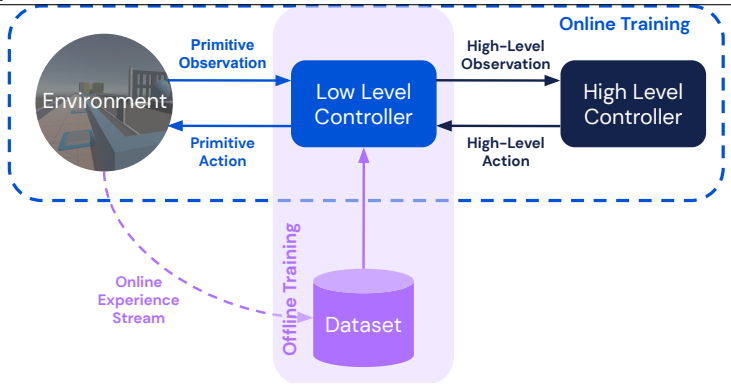
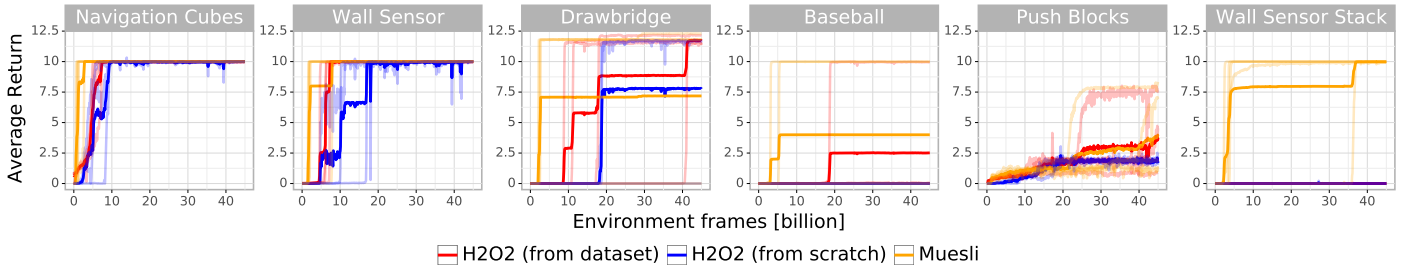


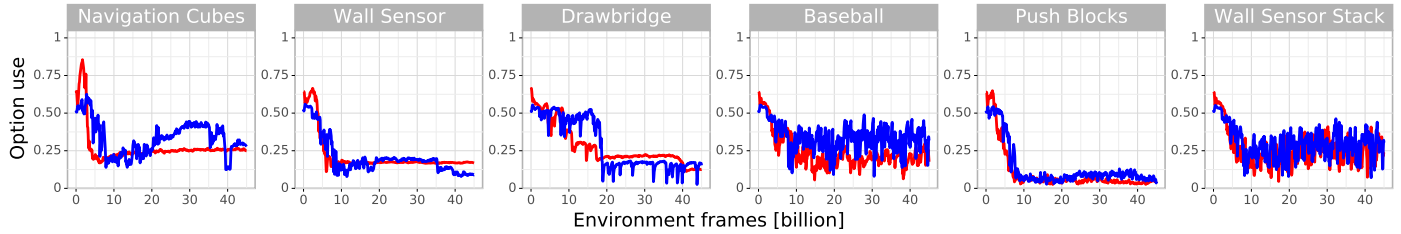
Figure 1: Proof of concept H2O2 implementation diagram.

- (from scratch) The LLC is trained in offline fashion, but in tandem with the HLC. The data used to train the LLC is generated by the H2O2 agent itself.

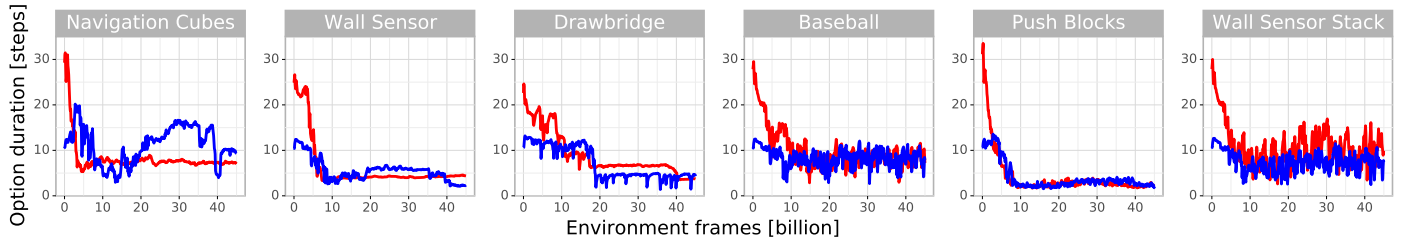
We evaluated H2O2 in the Hard Eight suite (Gulcehre et al., 2019). These are procedurally generated, partially observable 3D tasks that require exploration and complex behaviour to solve. Gulcehre et al. (2019) made some progress using human demonstrations. We found that a non-hierarchical Muesli agent (Hessel et al., 2021) can make progress in some of these tasks without requiring access to demonstrations, and we use it as a baseline for comparison.



(a) H2O2 and non-hierarchical Muesli average return trained multitask on Hard Eight.



(b) H2O2's frequency of using temporally extended options rather than primitive actions, throughout training.



(c) Average H2O2 option duration (excluding primitive actions) throughout training. Each step corresponds to four environment frames (where the same action is repeated).

Figure 2: Main results summary on six out of eight tasks in the Hard Eight suite (Gulcehre et al., 2019). No agent made progress in the remaining two tasks `Remember Sensor` and `Throw Across`.

Figure 2a shows the agents' average returns throughout training in six of the Hard Eight tasks. Neither agent could make meaningful progress on the other two, so they have been omitted. Figure 2b shows how often on average H2O2 used primitive actions in each of the tasks throughout training, and fig. 2c shows the average option duration per task.

These are the main takeaways about H2O2's performance in the Hard Eight suite:

Strengths

- **Effective Options.** The LLC learns complex temporally extended behaviour and the HLC learns to use it. Figure 2a shows competitive performance across tasks, and fig. 2c shows that options last for several steps on average. The average option duration also varies across tasks.
- **Options do not collapse.** The HLC uses a mix of primitive actions and temporally extended options, cf. fig. 2b. The frequency of primitive action use varies across tasks.
- **Robustness to uncurated data.** The agent that generates the pre-collected dataset has poor (near-zero or negative) average return in the Hard Eight tasks. Despite this, the LLC trained on this dataset allows H2O2 to reach competitive performance.
- **No strict need for a pre-collected dataset.** Even when running fully online, training the LLC based on data the agent itself generates, H2O2 achieves competitive performance while learning and using options.

Limitations

- **Data inefficiency.** The HLC in this implementation learns at a much lower temporal frequency and does not see all data generated during LLC execution. Every high level transition the HLC is trained on translates, on average, to more than forty transitions in the underlying environment. As a result, the HLC requires many more environment frames than the baseline to reach peak performance.
- **Matching baseline final performance, weak on three tasks.** Our H2O2 implementation has competitive average return after training, versus a non-hierarchical Muesli baseline, in the Hard Eight tasks. Of the six tasks where performance matches, in three tasks neither agent can make meaningful progress.

6 Conclusion

This work provides a new perspective on the contributions of goal-conditioned hierarchical RL to tackle challenging tasks in visually complex, partially observable procedural 3D environments. Throughout this research, we made the conscious choice of not including any domain specific knowledge or custom architectural choices. While we note that our analysis has been focused on the limited set of tasks reported in this work, we believe that many of our findings can be extended to other partially observable procedurally generated domains.

References

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

C. Gulcehre, T. Le Paine, B. Shahriari, M. Denil, M. Hoffman, H. Soyer, R. Tanburn, S. Kapturowski, N. Rabinowitz, D. Williams, et al. Making efficient use of demonstrations to solve hard exploration problems. In *International conference on learning representations*, 2019.

M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. van Hasselt. Muesli: Combining improvements in policy optimization. *arXiv preprint arXiv:2104.06159*, 2021.

C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.

M. Mathieu, S. Ozair, S. Srinivasan, C. Gulcehre, S. Zhang, R. Jiang, T. Le Paine, K. Zolna, R. Powell, J. Schrittwieser, et al. Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Predecessor Features

Duncan Bailey
Department of Cognitive Science
University of California
San Diego
dbailey@ucsd.edu

Marcelo G. Mattar
Department of Cognitive Science
University of California
San Diego
mmattar@ucsd.edu

Abstract

Any reinforcement learning system must be able to identify which past events contributed to observed outcomes, a problem known as credit assignment. A common solution to this problem is to use an eligibility trace to assign credit to recency-weighted set of experienced events. However, in many realistic tasks, the set of recently experienced events are only one of the many possible action events that could have preceded the current outcome. This suggests that reinforcement learning can be made more efficient by allowing credit assignment to any viable preceding state, rather than only those most recently experienced. Accordingly, we propose "Predecessor Features", an algorithm that achieves this richer form of credit assignment. By maintaining a representation that approximates the expected sum of past occupancies, our algorithm allows temporal difference (TD) errors to be propagated accurately to a larger number of predecessor states than conventional methods, greatly improving learning speed. Our algorithm can also be naturally extended from tabular state representation to feature representations allowing for increased performance on a wide range of environments. We demonstrate several use cases for Predecessor Features and contrast its performance with other similar approaches.

Keywords: Reinforcement Learning, TD-Learning, Eligibility Traces, Successor Representation

Acknowledgements

We would like to thank the Cognitive Science Department of the University of California San Diego.

1 Introduction

At the core of Reinforcement Learning (RL) is the problem of credit assignment: identifying which of the previously chosen events (actions and states) are causally related to an observed outcome. In the temporal difference (TD) learning algorithm, a cornerstone of RL, recently experienced events receive credit for the prediction errors encountered at each present moment. Thus, to distribute credit appropriately, a temporary record of recently experienced events – an eligibility trace – is maintained and updated continuously.

The assumption underlying algorithms based on eligibility traces is that only recent events can receive credit for an outcome. However, in many realistic tasks, the set of recently experienced events are only one of the many possible action events that could have preceded the current outcome. This suggests that reinforcement learning can be made more efficient by allowing credit to be assigned to any viable preceding state, rather than only those most recently experienced. But how to identify the correct set of preceding states in this richer form of credit assignment? To help answer this question we consider first the Successor Representation (SR), which quantifies, from each state, the expected (discounted) future occupancy of every other state [2]. This representation can be leveraged to compute values from rewards through a simple linear operation, and can be learned efficiently in the setting of linear function approximation [1]. We propose that this representation can also express, from each state, the set of all possible (discounted) preceding states, yielding a quantity analogous to an eligibility trace.

Here, we formalize precisely the relationship between the SR and the (expectation of) eligibility traces. We then propose a variant of temporal difference learning that uses this richer form of eligibility traces, an algorithm we call Predecessor Representation. To extend this method to the setting of function approximation, we describe an approach to learn the predecessors directly, giving rise to a second algorithm called Predecessor Features. In both cases, we demonstrate a few simulations showing that this learned quantity is helpful for credit assignment as it acts as an improved way of propagating error to relevant features than previous algorithms.

2 Formalism

The central task in Reinforcement Learning is to predict returns of future discounted rewards: $G_{t:T} = \sum_{i=1}^{T} \gamma_{t+i}^{(i-1)} R_{t+i}$ where T is the time the current episode terminates or $T = \infty$ for continuous tasks. The value $v_\pi(s) = \mathbb{E}_\pi \left[G_{t:T} \mid S_t = s \right]$ of state s is the expected return for a policy π when starting from state s. This value is approximated by the function $v_{\mathrm{w}}(s) \approx v_\pi(s)$, parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^d$. The value function can be specified as a table with each entry corresponding to a state, as a linear function of some defined input features, or as a non-linear function. The weight vector $\mathbf{w}$ is learned iteratively through the update rule $\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}_t$ such that $v_{\mathbf{w}}$ approaches the true v_π.

There are multiple methods for computing $\Delta \mathbf{w}_t$. The Monte Carlo algorithm defines $\Delta \mathbf{w}_t \equiv \alpha \left(R_{t+1} + \gamma_{t+1} G_{t+1:T} - v_{\mathbf{w}} \left(S_t \right) \right) \nabla_{\mathbf{w}} v_{\mathbf{w}} \left(S_t \right)$ and is able to succeed in the task of value function approximation, but does so with high variance. An alternative approach that estimates the value function with lower variance is TD learning. TD learning 'learns a guess from a guess' and replaces the return with the current expectation $v \left(S_{t+1} \right) \approx G_{t+1:T}$, where $\Delta \mathbf{w}_t \equiv \alpha \left(R_{t+1} + \gamma_{t+1} v_{\mathbf{w}} \left(S_{t+1} \right) - v_{\mathbf{w}} \left(S_t \right) \right) \nabla_{\mathbf{w}} v_{\mathbf{w}} \left(S_t \right)$.

Algorithms that employ a 'forward' looking approach, like the MC algorithm, use returns that depend on future trajectories and need to wait many time steps or until the end of an episode to create their updates. Alternatively, algorithms that employ a 'backward view' will look back on past experiences during an episode to update current estimates. For example, a classic backward-looking algorithm, $\mathrm{TD}(\lambda)$, defines $\Delta \mathbf{w}_t \equiv \alpha \delta_t \boldsymbol{e}_t$ where $\boldsymbol{e}_t = \gamma_t \lambda \boldsymbol{e}_{t-1} + \nabla_{\mathbf{w}} v_{\mathbf{w}} \left(S_t \right)$ is referred to as eligibility trace, and $\delta_t = R_{t+1} + \gamma_{t+1} v_{\mathbf{w}} \left(S_{t+1} \right) - v_{\mathbf{w}} \left(S_t \right)$ is referred to as the temporal difference (TD) error.

$\mathrm{TD}(\lambda)$ generalizes Monte Carlo and Temporal Difference methods, with $\mathrm{TD}(1)$ corresponding to an online implementation of the MC algorithm and $\mathrm{TD}(0)$ corresponding to a regular one-step TD update.

3 Predecessor Representation

We first offer a generalization of the concept of eligibility traces in the tabular case, leading to an algorithm called "Temporal differences – Predecessor Representation", or TD-PR. In the original $\mathrm{TD}(\lambda)$ algorithm, the main way of propagating value updates to relevant states is by keeping an eligibility trace. In the tabular $\mathrm{TD}(\lambda)$, the eligibility trace is defined by $e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$ where on each step the eligibility traces for all states decay by $\gamma\lambda$, and the eligibility trace for the one state visited on the step is incremented by 1. The eligibility trace defines the extent to which each state is 'eligble' for undergoing learning changes. The corresponding TD-error is $\delta_t = R_{t+1} + \gamma V_t \left(S_{t+1} \right) - V_t \left(S_t \right)$ and value function update $\Delta V_t(s) = \alpha \delta_t e_t(s),$ for all $s \in \mathcal{S}$.

Our starting point to generalize the concept of eligibility traces is the Successor Representation (SR) [2]. Given a stream of experience, the SR maitrx $\mathbf{M}$ represents a given state in terms of discounted occupancy to other states. The SR is defined as $\mathbf{M} = (\mathbf{I} - \gamma \mathbf{P})^{-1}$, where $\mathbf{P}$ is a transition matrix with entries corresponding to probabilities of transitioning from states (or state action pairs) to other states. Thus, if $\mathbf{M}_{ij}$ represents the expected (discounted) number of visitations from i to j, the ith row of $\mathbf{M}$ represents the expected (discounted) number of visitations from i to every state. Accordingly, the jth column of $\mathbf{M}$ represents the expected (discounted) number of visitations to state j, starting from every state. Indeed, while a row of $\mathbf{M}$ is "forward-looking", a column of $\mathbf{M}$ is "backward-looking". If we set the discount factor to $\gamma\lambda$:

$$\mathbf{M}_{ij} = \mathbb{E}\left[\sum_{p=0}^{\infty}(\gamma\lambda)^p \mathbb{1}_{s_{\{n+p+1=j\}}} \mid s_n = i\right] = \sum_{p=0}^{\infty}(\gamma\lambda)^p \mathbb{P}\left(s_{n+p+1} = j \mid s_n = i\right) = \frac{\mathbb{P}(s=j)}{\mathbb{P}(s=i)}\mathbb{E}\left[\sum_{p=0}^{\infty}(\gamma\lambda)^p \mathbb{1}_{\{s_{n-p-1}=i\}} \mid s_n = j\right]$$
$$= \frac{\mathbb{P}(s=j)}{\mathbb{P}(s=i)}\sum_{p=0}^{\infty}(\gamma\lambda)^p \mathbb{P}\left(s_{n-p-1} = i \mid s_n = j\right) = \frac{\mathbb{P}(s=j)}{\mathbb{P}(s=i)}\mathbb{E}\left[\sum_{p=0}^{\infty}(\gamma\lambda)^p \mathbb{1}_{\{s_{n-p-1}=i\}} \mid s_n = j\right] = \frac{\mathbb{P}(s=j)}{\mathbb{P}(s=i)}\mathbb{E}\left[e_{n-1}(i) \mid s_n = j\right]$$

In other words, the column of the SR is directly related to the expectation of the eligibility traces (see also van Hasselt et al [3], Pitis et al [5]). In contrast to a sample of the eligibility traces, the expected trace has the advantage that is contains all possible predecessor states, and thus can give rise to a more efficient TD algorithm.

Below we describe TD-PR, an example of such algorithm.

Algorithm 1 Tabular TD-PR

procedure TD-PR($episodes, \gamma, \lambda, \alpha, \beta$)
  $\mathbf{v} \leftarrow \mathbf{0}$
  $\mathbf{M} \leftarrow \mathbf{0}$ ($|S| \times |S|$ identity matrix)
  for episode in $1 \ldots n$ **do** (Note: $\mathbf{s}_k$ is one-hot vector with 1 at index k)
    $\mathbf{e} \leftarrow \mathbf{0}$ (eligibilty trace)
    for pair (s_i, s_j) and reward r in episode **do**
      $e(s_i) \leftarrow e(s_i) + 1$
      $\mathbf{M} \leftarrow \mathbf{M} + \beta \mathbf{e}\mathbf{s}_j^\intercal + \beta\gamma \mathbf{e}\mathbf{s}_j^\intercal \mathbf{M} - \beta \mathbf{e}\mathbf{s}_i^\intercal \mathbf{M}$
      $\mathbf{v} \leftarrow \mathbf{v} + \alpha \mathbf{M}_{\cdot i}(r + \gamma v_j - v_i)$
      $\mathbf{e} \leftarrow \gamma\lambda \mathbf{e}$
    end for
  end for
return $\mathbf{v}, \mathbf{M}$

Note that TD-PR learns the SR using temporal differences. At the end of the first episode, a column of the SR (predecessor representation) corresponds exactly to the usual eligibility trace. At the second episode and beyond, however, the two representations diverge. This is seen clearly in Figures 1b and 1c.

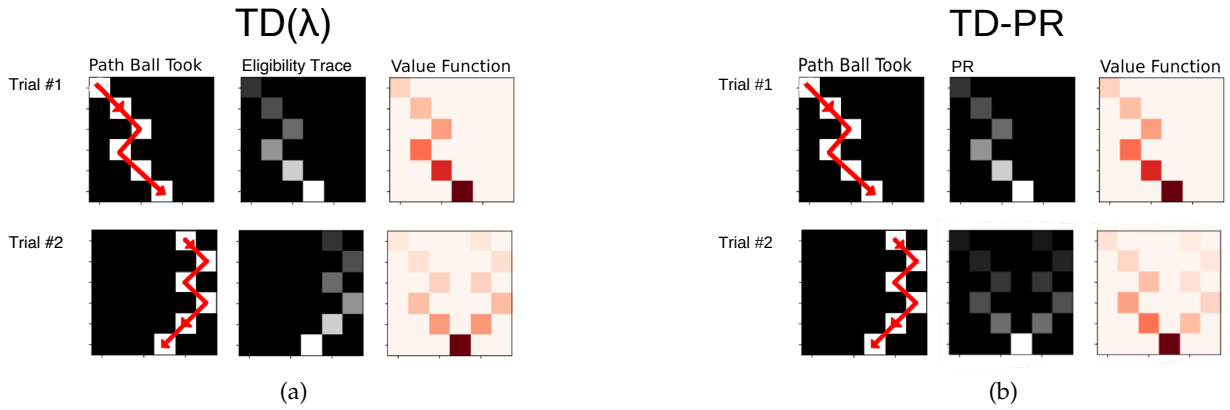


Figure 1: Illustrated is the first two trials where the ball hits the reward. Displayed from left to right is the path the ball took, the credit assignment vector (shown as a 6×6 matrix) used in each algorithm, and the value function at the end of the trial. It becomes apparent that the key difference between the two algorithms is TD-PR uses a credit assignment method that keeps a memory of past trial's state visits and updates all state values accordingly.

4 Predecessor Features

We now offer a generalization of the concept of eligibility traces in the general, function approximation case, leading to an algorithm called "Temporal differences – Predecessor Features", or TD-PF. We first define an expected eligibility trace as: $\boldsymbol{z}(s) \equiv \mathbb{E}\left[\boldsymbol{e}_t \mid S_t = s\right]$

This expected eligibility trace tries to approximate the sum of discounted predecessor features:

$$\boldsymbol{z}(s) = \mathbb{E}\left[\sum_{n=0}^{\infty} (\lambda\gamma)^n \mathbf{x}(S_{t-n}) \mid S_t = s\right]$$

$\lambda \neq 1$ should be included if a partial predecessor feature representation is desired. In TD-PF we try to learn approximations $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) \approx \boldsymbol{z}(S_t)$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. In doing so we will be using supervised learning techniques to minimize the empirical loss $\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}_{\boldsymbol{\theta}}(S_t))$. Where $\boldsymbol{y}$ is the target for each collected transition (s, a, r, s').

$$\boldsymbol{y} = \begin{cases} \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) & \text{if } S_t \text{ is an initial state} \\ \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) + \lambda\gamma \boldsymbol{z}_{\boldsymbol{\theta}}(S_{t-1}) & \text{otherwise} \end{cases}$$

Here $\mathbf{x}(S_t)$ is a feature vector of S_t. In this case, we will be using a TD like update of $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t)$. The learned representation will be map from a current feature vector to a vector of predecessor features. If we use a squared loss function, then

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)) = \mathbb{E}\left[\left\|\boldsymbol{y} - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)\right\|^2\right]$$

$$\Delta\boldsymbol{\theta} \leftarrow \nabla_{\boldsymbol{\theta}} \left\|\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) + \lambda\gamma \boldsymbol{z}_{\boldsymbol{\theta}}(S_{t-1}) - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)\right\|_2^2$$

For simplicity we will use a linear approximation of $\boldsymbol{z}_{\boldsymbol{\theta}}$ such that $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) = \boldsymbol{\Psi}\mathbf{x}(S_t)$ where $\boldsymbol{\Psi}$ is a square matrix. Note that any approximation method could be used here (i.e. a neural network). Computing the gradient of the squared loss function with respect to $\boldsymbol{\Psi}$ and performing gradient descent we get an update rule similar to linear TD-learning:

$$\boldsymbol{\Psi}_{t+1} = \boldsymbol{\Psi}_t - \beta\left(\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) - \boldsymbol{y}_{s,a,r,s'}\right)\mathbf{x}(S_t)^{\intercal}$$

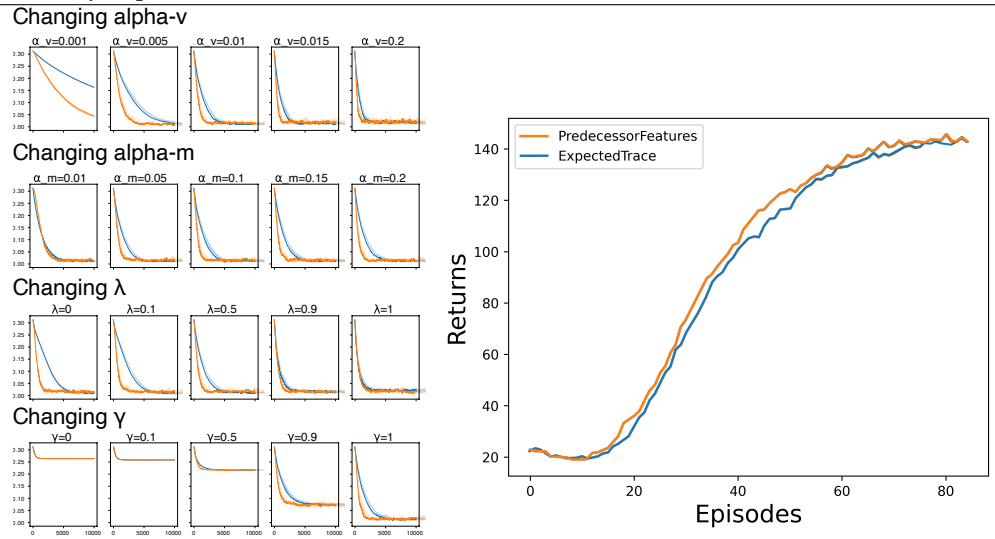
Note that Lehnert and Littman showed a similar learning rule for an approximation of Successor Features [4]. This TD-inspired update of the expected trace parameters differs from the Monte Carlo like update defined in van Hasselt et al[3]; where instead of updating towards a sampled eligibility trace we update towards this new representation added to the gradient of the value function of the current state. Simulations show that the $\boldsymbol{\Psi}$ matrix when using a tabular feature vector for each state approximately learns the SR matrix and gives similar performance to TD-PR.

Algorithm 2 Linear TD-PF

procedure LINEAR TD-PF($episodes, \mathbf{w}, \boldsymbol{\Psi}, \alpha, \beta$)
  initialize $\mathbf{w}, \boldsymbol{\Psi}$
  for episode in $1 \ldots n$ **do**
    $S_t \leftarrow$ initial state of episode
    $\mathbf{y} \leftarrow \mathbf{x}(S_t)$
    $\boldsymbol{\Psi}_{t+1} = \boldsymbol{\Psi}_t - \beta\left(\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) - \mathbf{y}\right)\mathbf{x}(S_t)^{\top}$ (Note: $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) = \boldsymbol{\Psi}\mathbf{x}(S_t)$)
    for pair (S_t, S_{t+1}) and reward r in episode **do**
      $\delta \leftarrow r + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$
      $\mathbf{y} \leftarrow \mathbf{x}(S_{t+1}) + \lambda\gamma \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)$
      $\boldsymbol{\Psi}_{t+1} = \boldsymbol{\Psi}_t - \beta\left(\boldsymbol{z}_{\boldsymbol{\theta}}(S_{t+1}) - \mathbf{y}\right)\mathbf{x}(S_{t+1})^{\top}$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)$
    end for
  end for
return $\mathbf{w}, \boldsymbol{\Psi}$

5 Experiments

The task we studied the methods of TD-PR on is the Japanese gambling game Pachinko (aka Plinko). Plinko is a 6×6 grid where a ball is placed with $1/6$ probability into one of the top slots. A ball will go down a row and to the left with probability $1/2$ and down a row and to the right with probability $1/2$, unless the ball is at a state at the edge of the grid. In this case, the ball will go down a row and away from the edge by one state with probability 1 on each time step. A reward of 0 was received on each time step for every state unless the ball landed in the fourth state from the left on the



(a) (b)

Figure 2: **(a)** Comparison of convergence to true value function for TD-PR (in orange) and TD(λ) (in blue). Each row refers to changing a different parameter of each function. Alpha-v and Alpha-m are the learning rates for the value function and SR matrix, respectively. Aside from the changing variable all other parameters were set to equivalent default values of (alpha-v = 0.01, alpha-m=0.1, $\lambda = 0.9$, $\gamma = 1$). **(b)** The "Predecessor Features" algorithm marginally outperforms the "ExpectedTrace" algorithm using a deep neural network approximation for the value function. There is only a slight difference between the two algorithms, however it leads to a difference in performance. The y axis is returns on each episode and the x-axis is the number of episodes the agent has experienced.

bottom row where a reward of 1 was received. Once the ball reached the final row, the episode was terminated. Every episode lasted 6 time steps and restarted from the top row. The state space had 36 elements ($|S| = 36$), meaning the size of the SR matrix was 36×36. A column of the SR matrix was a 36×1 sized vector and could be nicely reformatted to fit the size of the Plinko board of size 6×6.

In TD-PR applied to Plinko for value function approximation, after each step of each episode, the sample SR matrix is updated and a column of the sample SR is used to propagate the TD-error to qualifying states. In convergence to the true value function Figure 2 shows TD-PR outperforms TD(λ) for all parameter values. Note that the only parameter value where TD(λ) initially outperforms TD-PR is when alpha-m, representing the learning rate for the sample SR, is quite small (0.01), however it does converge to the optimal value function ahead of TD(λ). When applying the idea of Predecessor Features and Expected Traces [3] to Deep RL in the Cartpole task (Figure 2b), the minor difference in our algorithm shows slight improvements in learning.

6 Discussion

We propose a new method, based on the SR, for assigning credit to preceding states. We show that this method can also be implemented by learning the expected sum of discounted preceding states (or features) directly. This helps learning performance of the value function in both accuracy and speed, and relates to the predictive representations hypothesis[6].

References

[1] André Barreto et al. *Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement*. 2019. arXiv: 1901.10964 [cs.LG].

[2] Peter Dayan. "Improving Generalization for Temporal Difference Learning: The Successor Representation". In: *Neural Computation* 5.4 (1993), pp. 613–624. DOI: 10.1162/neco.1993.5.4.613.

[3] Hado van Hasselt et al. "Expected Eligibility Traces". In: *CoRR* abs/2007.01839 (2020). arXiv: 2007.01839.

[4] Lucas Lehnert and Michael L Littman. "Successor features support model-based and model-free reinforcement learning". In: *CoRR abs/1901.11437* (2019).

[5] Silviu Pitis. "Source Traces for Temporal Difference Learning". In: *CoRR* abs/1902.02907 (2019). arXiv: 1902.02907.

[6] Eddie J. Rafols et al. "Using Predictive Representations to Improve Generalization in Reinforcement Learning". In: *IJCAI*. 2005.

Asymmetric DQN for Partially Observable Reinforcement Learning

Andrea Baisero
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
baisero.a@northeastern.edu

Brett Daley
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
daley.br@northeastern.edu

Christopher Amato
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
c.amato@northeastern.edu

Abstract

Offline training and online execution is a reinforcement learning framework in which agents are first trained offline in a simulated environment, and then become operational online in the real environment. Offline training allows the learning agents to exploit privileged state information through a mechanism known as asymmetry, which is commonly associated with actor-critic methods, and has the potential to greatly improve the learning performance of partially observable agents if used appropriately. However, current research in asymmetric reinforcement learning is often heuristic in nature, and verified through empirical evaluations rather than theoretical analysis. In this work, we develop the theory of asymmetric policy improvement, and showcase a series of theoretically grounded value-based algorithms that are able to exploit privileged state information in a principled fashion, and often with formal convergence guarantees. These algorithms range from Asymmetric Policy Iteration and Asymmetric Action-Value Iteration, two exact model-based dynamic programming solution methods; to Asymmetric Q-Learning, a model-free reinforcement learning algorithm which introduces stochastic incremental updates; to Asymmetric DQN, a very practical state-of-the-art model-free deep reinforcement learning algorithm. We complement our theoretical analysis with an empirical evaluation performed on environments specifically selected to exhibit significant partial observability, which require both information-gathering strategies and memorization of the past, and which could not be solved by reactive agents, or without learning appropriate representations of the past. The results confirm the strength of our proposed Asymmetric DQN algorithm, which achieves better performances and/or converges faster than other related baselines, and further confirms issues associated with other common forms of asymmetry.

Keywords: Reinforcement Learning, Partial Observability, Offline Training, Privileged Information, Asymmetry, DQN.

Acknowledgements

This research was funded by NSF award 1816382.

1 Introduction

Offline training and online execution (OTOE) is a modern reinforcement learning (RL) paradigm in which a learning agent is trained *offline* (i.e., in simulation) before becoming operational *online* (i.e., in the "real" environment). Advantages of OTOE are broad and include operational safety guarantees, training speed, flexibility, and access to privileged information, which is the focus of this work. OTOE has even become the paradigm of preference in some research cliques, such as that of multi-agent RL, where it is often called centralized training and decentralized execution (CTDE).

Privileged information is data which is accessible during offline training, but not during standard online training and execution. This can take different forms depending on the type of control problem that is being addressed, e.g., other agents' actions and observations in multi-agent RL, or the system's state in partially observable RL (PORL). In OTOE, access to this information is a temporary privilege, available only during the offline phase due to access of the simulation's internal state. Despite being not available during online execution, such information has the potential (when used appropriately) to improve the agent's overall training performance and, therefore, its future online performance.

In PORL, OTOE and privileged information is most commonly associated with actor-critic methods through a mechanism called *asymmetry*. In actor-critic methods, two separate models are being trained: a *policy* model (representing the agent's behavior) and a *critic* model. Standard actor-critic can be said to be *symmetric* in the sense that both models receive the same information as input—in PORL, the agent's history. In *asymmetric* actor-critic, this symmetry is broken by providing the critic with privileged information, which is possible because the critic is exclusively a training construct.

On the other hand, asymmetry has yet to make the leap towards value-based methods, which typically involve training a single value model $\hat{Q}$ to estimate the optimal value function Q^*. Not only do value-based methods lack the duality of actor-critic (i.e., two separate models through which to apply asymmetry), but the value model itself is also restricted by the control problem to execute exclusively without access to privileged information. In this work, we first develop the theory of *Asymmetric Policy Iteration* (API), and then incorporate elements of stochastic training and non-linear function approximation which ultimately result in *Asymmetric DQN* (ADQN), the first theory-driven asymmetric value-based OTOE algorithm that is able to exploit privileged information.

2 Background

POMDPs A partially observable Markov decision process (POMDP) is a discrete-time control problem represented by tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, b_0, T, O, R, \gamma \rangle$, where (a) $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ are state, action, and observation spaces, (b) $b_0 \in \Delta\mathcal{S}$ is an initial state distribution, (c) $T\colon \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is a stochastic state transition function, (d) $O\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta\mathcal{O}$ is a stochastic observation emission function, (e) $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and (f) $\gamma \in [0, 1)$ is a discount factor.

PORL Partially observable RL (PORL) is based on observable *histories* $h \in \mathcal{H} \doteq (\mathcal{A} \times \mathcal{O})^*$. Policies are ranked based on how well they maximize the expected *return* $\mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$ and, without loss of generality, we focus on *deterministic* policies. The action-value function Q^π is the solution to the Bellman equation $Q^\pi(h, a) = R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[Q^\pi(hao, \pi(hao))\right]$, and the optimal action-value function Q^* is the solution to the Bellman *optimality* equation, $Q^*(h, a) = R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[\max_{a'} Q^*(hao, a')\right]$. We define the general space of Q functions as $\mathcal{Q} \doteq \{Q \mid Q\colon \mathcal{H} \times \mathcal{A} \to \mathbb{R}\}$, and use $g(Q)$ to denote the policy which acts greedily on Q, i.e., $\pi = g(Q) \implies \pi(h) = \operatorname{argmax}_a Q(h, a)$.

History-State Value Functions Recent work in asymmetric actor-critic has employed the notion of a history-state action-value function $U^\pi(h, s, a)$ [1, 2], which is the solution to the *history-state* Bellman equation $U^\pi(h, s, a) = R(s, a) + \gamma\, \mathbb{E}_{s',o|s,a}\left[U^\pi(hao, s', \pi(hao))\right]$ and is related to Q^π via $Q^\pi(h, a) = \mathbb{E}_{s|h}\left[U^\pi(h, s, a)\right]$; also, U^* denotes the value function associated with the optimal policy. Note that, while U^π uses state context to represent more informed values, the policy remains partially observable. We define the general space of U functions as $\mathcal{U} \doteq \{Q \mid Q\colon \mathcal{H} \times \mathcal{A} \to \mathbb{R}\}$.

Operators We use operator notation, and avoid introducing different symbols for similar operators on Q and U by overloading the respective symbols; the distinction remains clear from context, i.e., from the operator input/output.
$B_\pi\colon \mathcal{Q} \to \mathcal{Q}$, s.t. $B_\pi Q(h, a) \doteq R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[Q(hao, \pi(hao))\right]$ is the Bellman contraction with fixed point Q^π.
$B\colon \mathcal{Q} \to \mathcal{Q}$, s.t. $BQ(h, a) \doteq R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[\max_{a'} Q(hao, a')\right]$ is the Bellman optimality contraction with fixed point Q^*.
$B_\pi\colon \mathcal{U} \to \mathcal{U}$, s.t. $B_\pi U(h, s, a) \doteq R(s, a) + \gamma\, \mathbb{E}_{s',o|s,a}\left[U(hao, s', \pi(hao))\right]$ is the Bellman contraction with fixed point U^π.
$E\colon \mathcal{U} \to \mathcal{Q}$, s.t. $EU(h, a) \doteq \mathbb{E}_{s|h}\left[U(h, s, a)\right]$ is the *expectation* operator which converts a U function to a Q function.

Definition 2.1 (Mutual Consistency)**.** We say that functions Q and U are *mutually consistent* iff $Q = EU$ holds.

3 Asymmetric Value-Based PORL

Two fundamental issues make the use of states in value-based methods not directly possible: (a) action-value model $\hat{Q}$ may not access state information online; and (b) there is typically no other model for the purpose of offline training which may access privileged information (akin to the critic in actor-critic methods). As such, value-based methods seem

fundamentally incompatible with the notion of asymmetry and the use of privileged information. We resolve both issues by employing an auxiliary history-state model $\hat{U}$ used exclusively as a training construct through which to implement asymmetry. Our ultimate goal is to train $\hat{U}$ and $\hat{Q}$ jointly so as to converge to the optimal value functions U^* and Q^*.

3.1 Asymmetric Policy Iteration

Consider *Asymmetric Policy Iteration* (API), an iterative process which implements asymmetry by employing both history-state and history values. API starts from arbitrary initial values and policy U_0, Q_0, and π_0, and then uses the following update rules to generate sequences U_k, Q_k, and π_k,

$$U_{k+1} \leftarrow \lim_{n\to\infty} B^n_{\pi_k} U_k \,, \qquad \text{(U-evaluation)} \tag{1}$$

$$Q_{k+1} \leftarrow E U_{k+1} \,, \qquad \text{(Q-evaluation)} \tag{2}$$

$$\pi_{k+1} \leftarrow g(Q_{k+1}) \,. \qquad \text{(improvement)} \tag{3}$$

Theorem 3.1 (API Optimality)**.** *The sequences U_k, Q_k, and π_k generated by API converge to U^*, Q^*, and π^* (proof omitted.)*

Limitations While API is formally guaranteed to converge optimally, it also has significant practical limitations: (a) API is a solution method which requires a model of the environment, as well as efficient and accurate methods to compute the expectations in the U-evaluation and the Q-evaluation steps. (b) A practical approximation of the limit operator in the U-evaluation step might itself require multiple iterations to achieve an adequate precision. (c) API requires tabular models U and Q, which is not only impractical given that the space of histories grows exponentially with episode lengths, but also makes it not applicable to control problems which have continuous observations or states. (d) Perhaps most importantly, API does not offer any practical advantage compared to its non-asymmetric counterpart Policy Iteration (PI) [3]. Ultimately, both API and PI converge to the same optimal value function Q^*; if anything, API requires more memory and computation to achieve the same goal, resulting in a less practical solution method.

Why API? In light of the above limitations, what is then the purpose of API? We argue that API plays two crucial roles: (a) To show that privileged state information *can* be properly integrated into a value-based solution process while maintaining formal optimality guarantees. This theoretical aspect is often overlooked in modern asymmetric RL research and, to the best of our knowledge, API represents the first theoretical guarantee of this kind for value-based PORL. (b) To serve as a basis for other algorithms—such as the ones discussed next—that do provide practical advantages compared to their non-asymmetric counterparts. Next, we sequentially relax various aspects of API and develop asymmetric value-based algorithms that address each of API's limitations.

3.2 Asymmetric Action-Value Iteration

The first limitation of API which we address is the presence of the limiting operator in its U-step, which makes practical implementations inefficient and/or approximate. To this end, consider *Asymmetric Action-Value Iteration* (AAVI), an eager variant of API which uses the following update rules,

$$U_{k+1} \leftarrow B_{g(Q_k)} U_k \,, \qquad \text{(U-evaluation)} \tag{4}$$

$$Q_{k+1} \leftarrow E U_{k+1} \,. \qquad \text{(Q-evaluation)} \tag{5}$$

Compared to API, the improvement step has been folded in the U-evaluation step, also removing the need for an explicit policy representation. Further, the U-evaluation step has been simplified to apply operator $B_{g(Q_k)}$ a single time.

Theorem 3.2 (AAVI Optimality)**.** *The sequences U_k and Q_k generated by AAVI converge to U^* and Q^* (proof omitted.)*

3.3 Asymmetric Q-Learning

Like all dynamic programming methods, API and AAVI require an exact model of the POMDP environment. To bypass this often unrealistic requirement, we consider incremental stochastic updates based on sample transitions. We define *Asymmetric Q-Learning* (AQL) according to the following updates, where $\alpha_k \in [0, 1]$ is a sequence of stepsizes,

$$U_{k+1} \leftarrow (1-\alpha_k) U_k + \alpha_k (B_{g(Q_k)} U_k + w_k), \tag{6}$$

$$Q_{k+1} \leftarrow (1-\alpha_k) Q_k + \alpha_k (E B_{g(Q_k)} U_k + v_k). \tag{7}$$

Here, $w_k \in \mathcal{U}$ and $v_k \in \mathcal{Q}$ are zero-mean noise processes that represent the randomness in the environment and action selection at iteration k. These processes are not statistically independent since they are computed using shared transitions, and are actually related by the identity $v_k = E w_k$. We note that AQL (as presented above) utilizes *synchronous* updates; at each iteration k, the values associated with all tuples (h, s, a) are updated at the same time, each using an independent transition $(r, s', o) \sim \Pr(r, s', o \mid h, s, a)$. However, an *asynchronous* variant can also be formulated which updates a single element of U_k and Q_k, has the same optimality guarantees, and more closely resembles Q-Learning [4].

Theorem 3.3 (AQL Optimality)**.** *Assuming step sizes α_k which satisfy $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$, and mutually consistent $Q_0 = E U_0$, then the sequences U_k and Q_k generated by AQL converge to Q^* and U^* with probability 1 (proof omitted.)*

3.4 Asymmetric DQN

Our main algorithmic contribution is *Asymmetric DQN* (ADQN), an asymmetric variant of DQN [5] derived by introducing value function approximation to AQL. We first replace the tabular-lookup models U and Q of AQL with parametric differentiable models $\hat{U}$ and $\hat{Q}$. In practice, these are implemented as deep neural networks whose architectures are chosen on a case-by-case basis according to the structure of the states and observations emitted by the POMDP. To facilitate the substitution, we must reformulate the update rules of AQL as squared-error losses to be minimized. Given a single environment interaction (h, s, a, r, s', o), the corresponding losses can be defined as

$$\mathcal{L}_{\hat{U}} = \frac{1}{2}\left(r + \gamma \operatorname{stop}\left[\hat{U}(hao, s', \operatorname*{argmax}_{a'} \hat{Q}(hao, a'))\right] - \hat{U}(h, s, a)\right)^2, \tag{8}$$

$$\mathcal{L}_{\hat{Q}} = \frac{1}{2}\left(r + \gamma \operatorname{stop}\left[\hat{U}(hao, s', \operatorname*{argmax}_{a'} \hat{Q}(hao, a'))\right] - \hat{Q}(h, a)\right)^2, \tag{9}$$

where stop is the *stop-gradient* function, which indicates that gradient calculation should not consider the enclosed terms. The total loss $\mathcal{L}_{\hat{U}} + \mathcal{L}_{\hat{Q}}$ can be minimized with respect to the main parameters by a single backpropagation procedure.

When the function approximators are nonlinear (as is often the case for neural networks), training will fail if the gradient updates are conducted on sequentially collected POMDP transitions, since the data are not i.i.d. [5]. The second critical modification to AQL is therefore the adoption of experience replay [6] in order to decorrelate training experiences. Rather than training on a sample immediately when it is collected, each POMDP transition (h, s, a, r, s', o) is deferred to a first-in first-out replay memory. Periodically, when it is time to train the networks, a minibatch of several experiences is sampled at random from the replay memory; gradients for these samples are computed and averaged together to estimate the true gradient of the joint loss $\mathcal{L}_{\hat{U}} + \mathcal{L}_{\hat{Q}}$, which in turn is used to improve the parameters.

Why ADQN? Having finally addressed the practical disadvantages of API, it is worthwhile to reconsider the "why" question again, this time focusing on why one would prefer to use ADQN compared to DQN. Ultimately, the purpose of both algorithms is to compute Q^* through which optimal control can be executed, and both algorithms should *in theory* converge to the same Q^*. Then, what is the advantage of ADQN over DQN? Similarly to the asymmetric actor-critic case [1], the advantage is a practical one associated with the difficulties of learning an appropriate representation of history $\phi(h)$, which is one of the major learning bottlenecks in PORL. History representations are sequence models which require lots of data and computation. To make matters worse, the quality of the data used to train the history representation in PORL is directly related to the quality of $\hat{Q}$, which depends on the history representation itself, and it is quite hard to bootstrap the training of a history representation when starting from a poor history representation. On the other hand, learning an appropriate state representation $\phi(s)$ is much simpler due to the non-sequential nature of states. Therefore, we expect $\hat{U}$ to learn meaningful values faster in ADQN, without having to wait for a meaningful history representation. In turn, because $\hat{U}$ is a major component of the target for $\hat{Q}$, we expect $\hat{Q}$ to also learn faster.

3.5 State-Only Asymmetric DQN

Prior work in asymmetric PORL has sometimes adopted heuristic forms of asymmetry which uses state-only (i.e., history-less) forms of value functions and critics which have been recently shown to be linked to fundamental theoretical and practical issues [1]. These issues are inherently related to partial observability, include (but are not limited to) the introduction of bias into the learning process, and may severely compromise the learning performance. Nonetheless, we formulate a state-only variant of ADQN where $\hat{U}$ is redefined to ignore histories, and the losses are redefined as

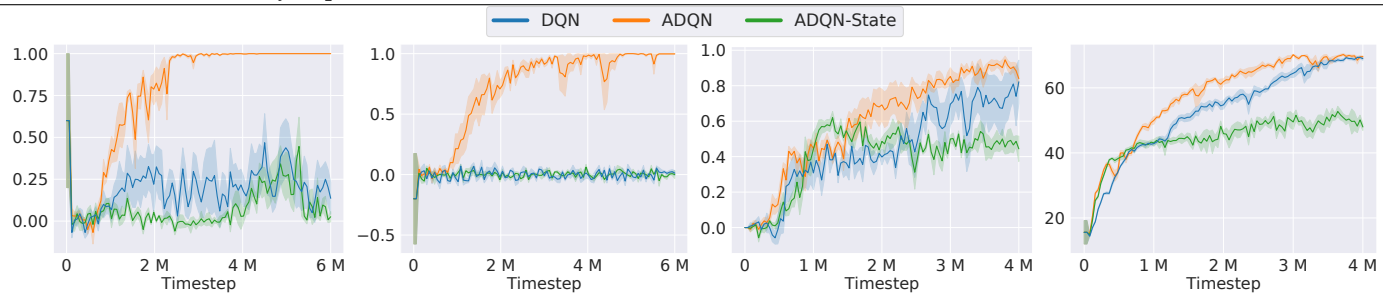
$$\mathcal{L}_{\hat{U}} = \frac{1}{2}\left(r + \gamma \operatorname{stop}\left[\hat{U}(s', \operatorname*{argmax}_{a'} \hat{Q}(hao, a'))\right] - \hat{U}(s, a)\right)^2, \tag{10}$$

$$\mathcal{L}_{\hat{Q}} = \frac{1}{2}\left(r + \gamma \operatorname{stop}\left[\hat{U}(s', \operatorname*{argmax}_{a'} \hat{Q}(hao, a'))\right] - \hat{Q}(h, a)\right)^2. \tag{11}$$

This state-only ADQN variant allows us to both confirm once again the issues generally associated with state-only value functions, and to demonstrate the theoretical validity and practical advantages of our proposed history-state ADQN.

4 Evaluation

We compare the performances of **DQN**, **ADQN**, and **ADQN-state** in four partially observable navigation tasks which exhibit significant levels partial observability and require both information-gathering strategies and memorization of the past: **Heaven-Hell-3** and **Heaven-Hell-4** [7], **Car-Flag** [8], and **Cleaner** [9]. For each environment and algorithm, we perform an independent grid-search over some hyper-parameters of interest, and select the combination of hyper-parameters which results in the best final performance and learning stability (prioritizing final performance if necessary).



(a) **Heaven-Hell-3** (b) **Heaven-Hell-4** (c) **Car-Flag** (d) **Cleaner**

Figure 1: Performance curves showing episodic returns averaged over the last 100 completed episodes, with statistics computed over 5 independent runs. The shaded areas represent one standard error around the mean.

The results shown in Figure 1 confirm the theoretical analysis for both **ADQN** and **ADQN-state**. In all control problems, **ADQN** is superior to **DQN** and **ADQN-state** in final performance and/or convergence speed. In Figures 1a and 1b the contrast is particularly noteworthy, as only **ADQN** is able to solve the control problems at all, and is able to do so optimally. In Figures 1c and 1d, **DQN** is also able to eventually solve the control problem, albeit converging slower than **ADQN**. Finally, **ADQN-state** is suboptimal and performs worse than **ADQN** and/or **DQN** in all the control problems.

5 Conclusions

OTOE is a RL framework where agents are trained offline in a simulated environment, which allows for the possibility of accessing privileged information which would otherwise be unavailable, like the partially observable environment's state. Asymmetry is a mechanism through which such privileged information is used during training, but is otherwise unnecessary during online execution, and has the potential to greatly boost learning performance and efficiency, when executed correctly. However, modern work in asymmetric RL tends to focus on unproven heuristics which lack a theoretical justification. In this work, we bridge this gap and develop the theory of asymmetric value-based RL. We first derive the guiding principle of asymmetric policy improvement through API, a theoretical solution method with strict optimal convergence guarantees. Then, we apply a series of relaxations to API which ultimately lead to ADQN, a practical and competitive deep RL algorithm. We perform an empirical evaluation of ADQN in a series of environments which specifically selected to exhibit high levels of partial observability, which require memorization of the past, and in which specific information-gathering strategies need to be executed. In these environments, ADQN achieves better performances and/or faster than other baselines. Future work may focus on extending ADQN to the multi-agent control case, which poses further learning challenges, and on finding applications where state-only ADQN may thrive.

References

[1] A. Baisero and C. Amato, "Unbiased asymmetric reinforcement learning under partial observability," in *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, 2022.

[2] X. Lu, Y. Xiao, A. Baisero, and C. Amato, "A deeper understanding of state-based critics in multi-agent reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] C. J. C. H. Watkins, "Learning from delayed rewards," PhD Thesis, King's College, Cambridge, 1989.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, and A. K. Fidjeland, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[6] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3, pp. 293–321, 1992.

[7] B. Bonet, "Solving large POMDPs using real time dynamic programming," in *Proc. AAAI Fall Symp. on POMDPs*, 1998.

[8] H. Nguyen, "POMDP Robot Domains," 2021. [Online]. Available: https://github.com/hai-h-nguyen/pomdp-domains

[9] S. Jiang and C. Amato, "Multi-agent reinforcement learning with directed exploration and selective memory reuse," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 777–784.

Decision difficulty modulates the re-use of computations across trials in non-sequential decision tasks.

Nidhi V. Banavar
Department of Cognitive Sciences
UC - Irvine
nbanavar@uci.edu

Aaron M. Bornstein
Department of Cognitive Sciences
UC - Irvine
aaron.bornstein@uci.edu

Abstract

Decision making under uncertainty necessitates complex computations which are traded-off with the need for efficiency. This is particularly relevant in the context of experiments where individuals make a sequence of choices and previous computations may be leveraged to support efficiency. However, it is an open question as to whether humans do indeed reference the recent past, especially in complex environments where it is task-incongruent to do so (e.g. non-sequential experiments). In behavioral economic experiments with randomized or unstructured choice sets, trial-level sequential dependencies are generally assumed to be present only in motor or perceptual operations. Here, we explicitly model trial-property-driven sequential effects in response time data in two data sets: intertemporal choice and risky and ambiguous choice. We find evidence for widespread sequential effects. These effects are modulated by decision difficulty and trial-level uncertainty in both tasks. Our results add to the growing literature demonstrating trial-level sequential dependencies in higher-order cognition.

Keywords: sequential decision making; economics; Bayesian cognitive models; response time analysis

Acknowledgements

We thank Catherine A. Hartley for providing the intertemporal choice data. We also thank Michael D. Lee and Joachim Vandekerckhove for helpful discussions.

1 Introduction and Background

When making decisions under uncertainty and with limited time, humans and animals must balance efficiency with completeness. We have previously shown that humans exhibit 'spillover' between adjacent trials in non-sequential decision tasks[1]. Here, we examine whether this effect is modulated by the relative properties of the decision computations required on successive trials. Such effects could be the result of rational computational approaches – for instance if people are showing sequential effects in nonsequential environments, it could be because they are trying to infer the underlying Markov Decision Process (MDP) [4, 14]. While this idea has been studied extensively in lower-order decision making, it is a major open question whether individuals reference recent history in higher-order non-sequential decision tasks [2]. This is conceptually distinct from notions of motor or perceptual perseveration, which may themselves also be present during complex choice [1].

We therefore consider economic decision making in two non-sequential tasks: intertemporal choice (ITC) – decision making under temporal uncertainty – and risky and ambiguous choice (RAC) – choice under immediate known (risk) or unknown (ambiguous) uncertainty. A common method of measuring these constructs in behavioral economics is through experiments: typically, individuals make a sequence of choices in a randomized choice set and are explicitly instructed to treat each decision independently [8, 11]. The data is then analyzed as if each choice was indeed made independently and not in sequence. Our previous work suggests that this may be a flawed assumption [1].

First we consider the cognitive processes involved in making decisions about immediate or temporal uncertainty. Popular theories of intertemporal choice suggest that decisions are in part informed by the deliberative simulation of potential future outcomes [12]. These sorts of computations and simulations are expensive and time-consuming, so when successive decisions are made under time pressure and involve similar future dates and values it may be inefficient to construct preferences anew on each trial [3]. We may further expect such re-using of computations or, more broadly, reference to the past, as recent work suggests that neurons that code for value do so relatively (i.e. what has *changed* now compared to what was seen before) [16]. Thus, taken together, individuals may *act as though* there are structured temporal dependencies during decisions made in a sequence – even if those decisions are not explicitly related to each other.

The mechanisms that generate, or propagate these sequential effects are an open question even in perceptual and visual working memory, where such task in-congruent sequential behavior has been studied extensively [5, 9]. Candidate mechanisms include serial biases in attention [5] and efficient coding influences on working memory representations [9].

However, it remains unknown whether such 'spillover' effects apply to choice behavior more broadly and, if so, whether they are mediated by a general inclination to optimize speed/accuracy tradeoff in behavior, or by local uncertainty about task demands. To address this question, we leverage our previously introduced methodology to test and account for trial properties-driven sequential effects in such environments [1]. In our previous analysis, we observed that some sequential effects were driven by decision difficulty: they were present when the current trial was more difficult than the preceding. In particular, we found that a subset of subjects were sensitive to relative changes in Expected Value (i.e. when the previous trial had a large difference in Expected Value - "easy" - and the current trial had a small difference in Expected Value - "hard"). However, decision difficulty in ITC tasks, for example, is potentially conflated with the inherent uncertainty in receipt of future reward outcomes [7]. Here, we further extend our previous framework by re-parameterizing the stimulus space (all potential trial properties) in a simpler fashion and present a re-analysis of the intertemporal choice data in [1]. Then, we apply this approach to a newly collected risky and ambiguous choice task to test directly whether relative changes in decision difficulty or uncertainty more generally drive sequential effects. Across these datasets, we find evidence in favor of sequential effects, and in the RAC task, specifically that these effects are being driven by a tradeoff against choice difficulty, rather than by a need to resolve contextual uncertainty.

2 Methods

2.1 Experiments and Data

Intertemporal Choice (ITC). We model $n = 482$ adult subjects who made a sequence of 102 binary choices between a same-day monetary reward (SS: "smaller sooner", range: $\$1 - \$85$) and a larger delayed reward (LL: "larger later", range: $\$10 - \$95$; delay range: $4 - 180$ days). All data were collected previously (for details, see [8]). The experiment was fully randomized, with no experimentally-designed trial-level dependencies. All stimuli were displayed numerically and counterbalanced so that the SS and LL options occurred equally often on either side of the computer screen. Subjects had 6s after stimulus onset to make a choice.

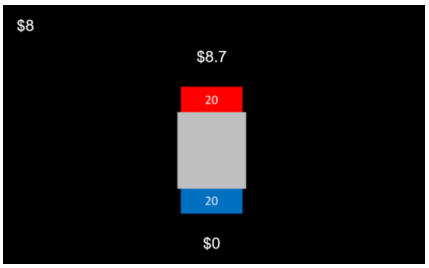


Figure 1: An example ambiguous trial from the RAC task, where the subject has 3 seconds from stimulus onset to make a choice between a certain reward of $8 and a chance to win $8.7 by playing the lottery.

Risky and Ambiguous Choice (RAC). We model $n = 98$ adult subjects who made a sequence of 196 binary choices between a certain reward (range: $\$3 - \$9.5$) and a lottery (range: $\$5 - \$24$), in 4 blocks (Figure 1). The amount a subject could win by choosing the lottery was almost

always larger than the certain reward, except during 16 catch trials. All data were collected on Amazon Mechanical Turk. Lotteries were either *risky* (1/7) where the full probability distribution was presented graphically (win probabilities: 25%, 50%, 75%) or *ambiguous* (6/7) where partial information was presented (ambiguity levels: 15%, 40%, 60%, 85%). While we did not design any trial-level dependencies, we ensured that 50% of successive trials increased in ambiguity, and 50% decreased. A risk trial followed by an ambiguous trial is considered as an increase in ambiguity, as risky trials are unambiguous with respect to the probability of reward. Likewise, an ambiguous trial followed by a risk trial would be considered a decrease in ambiguity. Further, we controlled for median risk/ambiguity levels, lottery reward, and fixed reward across blocks. As with ITC, the stimulus options occurred equally often on either side of the computer screen. Subjects had 3s after stimulus onset to make a choice.

No outcomes were realized over the course of either experiment (i.e no feedback after each choice other than a confirmation of the subject's selection). However, the experiments were incentive compatible and a single trial was selected at random, realized, and paid out at the end of the experiment as a bonus. For the RAC experiment, participants on Amazon MTurk were paid 10% of their winnings to be consistent with pay rates on the platform.

We exclude any responses that were made in less than 300ms. We also exclude any missed trials and the trial immediately after them from the following analyses.

2.2 Model

2.2.1 Baseline Models

We implement a hierarchical Bayesian drift diffusion model (DDM) to model response times using the Wiener module [15] in JAGS [13]. That is, for subject i and trial j, we model observed response time as Wiener first passage time *(wfpt)* distributed:

$$RT_{ij} \sim wfpt(\alpha_i, \tau_i, \beta_i, \delta_{ij})$$

Here, α_i represents the subject-level threshold or boundary separation, τ_i is the subject-level non-decision time (processes unrelated to the value-based decision process), β_i is the subject-level bias ($\beta_i < 0.5$: bias towards immediate option in ITC and towards the fixed option in RAC), and δ_{ij} is the subject-and-trial-level drift rate (the rate of evidence accumulation). We model all these parameters as hierarchical Normals in order to better capture individual differences [10]. For α_i, τ_i, and β_i, we use the same prior and hyperprior specifications for both tasks, referencing [15] for mean hyperpriors and using 'noninformative' priors for the standard deviation:

$$\mu_\alpha \sim \mathit{Uniform}(0.001, 3) \quad \mu_\tau \sim \mathit{Uniform}(0, 0.6) \quad \mu_\beta \sim \mathit{Uniform}(0.01, 0.99) \quad \sigma_\alpha, \sigma_\tau, \sigma_\beta \sim \mathit{Uniform}(0.001, 4)$$

We take a cognitive psychometrics approach to modeling the drift rate by allowing it to be driven by (combinations of) trial properties. We keep the broad functional relationship between trial properties as dictated by behavioral economic models of choice behavior (e.g. allowing an inverse relationship between the drift rate and delay for ITC). We also normalize all stimulus properties such that they fall between 0 and 1. Then, for subject i and trial j:

$$\delta_{ITC,ij} = \beta_{0,i} + \beta_{1,i} \cdot (\mathit{value}_{LL,ij} - \mathit{value}_{SS,ij}) + \beta_{2,i} \cdot \mathit{delay}_{ij}^{-1} \tag{1}$$

$$\delta_{RAC,ij} = \beta_{0,i} + \beta_{1,i} \cdot (\mathit{Expected\ Value}_{\mathit{Lottery},ij} - \mathit{Expected\ Value}_{\mathit{Fixed},ij}) + \beta_{2,i} \cdot A_{ij} \tag{2}$$

In Equation 2, the Expected Value of a choice option is given by $EV = p \cdot v$, where p is the probability of reward and v is the monetary value. For a risky trial, p is the percentage chance of winning reward. For an ambiguous trial, $p = 0.5$ as in [11]. A represents the degree of ambiguity of a given lottery. On ambiguous trials, this is the fraction of the lottery that is occluded by the grey bar as seen in Figure 1. On risky trials, $A = 0$. Finally, we allow all drift rate decomposition parameters ($\beta_0, \beta_1, \beta_2$) to be hierarchical Standard Normals.

2.2.2 Sequential Effects

We build stimulus-driven sequential effects into each of the β terms on the drift rate decomposition and on the bias. Specifically, as we only consider influence of recent history that is transient (1-trial-back), we incorporate relative changes in stimulus values between the previous and current trial. For the ITC task, we consider successive trials that *increase* or *decrease* in (objective) value difference, delay difference, and value x delay difference. For the RAC task, we similarly consider value differences, ambiguity difference, and value x ambiguity difference. We use Indicator Variables to subset sequences of trials that follow any of the above specified patterns, resulting in a 8-fold tiling of stimulus space for both experiments. We then augment our baseline models by allowing these stimulus properties to exert linear additive influences on the parameters of interest. For example, if we consider trials that increase in value from trial $j-1$ to trial j:

$$\beta'_{0,ij} = \beta_{0,i} + \gamma_i \cdot 1([V_{a,j} - V_{b,j}] > [V_{a,j-1} - V_{b,j-1}]) \tag{3}$$

We allow all sequential effect parameters to be hierarchical Standard Normals. We simultaneously assess the influence of relative stimulus properties on all drift rate decomposition and bias parameters. Thus, in Equation 3, $\beta_{0,i}$ becomes the sequential-effect-adjusted drift rate regression intercept for individual i and the indicator variable is 1 if there is an increase in value difference from trial $j-1$ to trial j. To answer our core question of interest, we test whether the sequential effect parameters (e.g. γ_i) is non-zero using the Savage-Dickey ratio to approximate the Bayes Factor (BF). We interpret any $BF > 3$ as evidence in favor of sequential effects.

3 Results

3.1 Drift Rate Decompositions

We fit a variety of different drift rate decompositions and present DIC differences in two model comparisons between the decompositions presented in the *Methods* section (Equations 1 and 2) that may be of particular interest to the reader. For ITC, we report that our drift rate decomposition that allows an inverse relationship between delay and value, as generally given in economic models of temporal discounting, outperforms the decomposition that where both value and delay have directly proportional relationships with drift rate ($DIC_{ITC,\text{Inverse}} = 116886$, $DIC_{ITC,\text{Direct}} = 11875$). In RAC task, we report that the drift rate that only includes the Expected Value difference weight (β_1) performs more poorly than the drift rate that incorporates all trial properties $DIC_{RAC,\text{All}} = 21824$, $DIC_{RAC,\text{EV Only}} = 22134$).

ITC	Mean(CI)	RAC	Mean(CI)
β_0	$-0.66(-1.26, -0.06)$	β_0	$-0.72(-2.99, 0.37)$
β_1	$3.45(1.19, 5.70)$	β_1	$0.84(-0.87, 6.14)$
β_2	$2.34(-2.15, 6.84)$	β_2	$-0.02(-0.72, 0.67)$
α	$2.47(1.70, 3.24)$	α	$1.87(0.97, 2.47)$
τ	$0.76(0.36, 1.16)$	τ	$0.40(0.19, 1.14)$
$bias$	$0.51(0.39, 0.63)$	$bias$	$0.50(0.38, 0.59)$

Figure 2: Aggregate posterior means (95% Credible Intervals) for Drift Rate decompositions and other DDM parameters. Note that these values represent the spread across individuals.

We find that for the ITC task, subjects tend to accumulate evidence more quickly when the value difference increases (β_1), all else held constant. Similarly, subjects tend to accumulate evidence more quickly when the delay decreases (β_2), all held constant. Both make sense, as larger value differences might push individuals towards selecting the LL option, and delayed rewards offered in the far future may not be worth the wait.

As in the ITC task, subjects appear to be sensitive to all trial properties in the RAC task. All else held constant, the average subject's drift rate increases as the Expected Value difference between choice options increase (β_1). Subjects are also sensitive to the degree of ambiguity, with drift rate decreasing with increasing ambiguity (β_2) (all else held constant). However, we highlight the credible interval ranges and note that there are considerable individual differences.

3.2 Sequential Effects

We observed reliable trial-property-driven sequential effects on almost all sequences, regardless of task. Indeed, for almost every possible combination of stimulus sequences, 100% of subjects showed evidence of sensitivity to previous stimuli in the ITC task. Consistent with our hypothesis, effects in the RAC task are more specific to the combination of trial properties considered – in particular, most individuals show sequential effects when there are relative increases and decreases in value.

ITC	Proportion	RAC	Proportion
$value \uparrow$	1	$value \uparrow$	0.80
$value \downarrow$	0.998	$value \downarrow$	0.93
$delay \uparrow$	1	$amb \uparrow$	0.05
$delay \downarrow$	1	$amb \downarrow$	0.01
$v. \uparrow d. \uparrow$	1	$v. \uparrow a. \uparrow$	0
$v. \uparrow d. \downarrow$	1	$v \uparrow a. \downarrow$	0.18
$v. \downarrow d. \uparrow$	1	$v. \downarrow a. \uparrow$	0
$v. \downarrow d. \downarrow$	0.84	$v. \downarrow a. \downarrow$	0.57

Figure 3: Proportion of subjects that demonstrated sequential effects ($BF > 3$) on any one of the drift rate decomposition parameters or bias. Each row represents specific successive trial properties (e.g. $value \uparrow$ subsets successive trials that increased in value difference as noted in Equation 3). The top four rows can be thought of as "main effects" of specific trial properties and the bottom four "interactions."

In Figure 5, we list the posterior group-level means and standard deviations of the sequential effect parameters themselves for the RAC task only, omitting ITC results due to space constraints. We find broad individual variability in terms of which parameters seem to be most sensitive to sequential trial properties: while one subject may manifest this sensitivity on the drift rate intercept parameter, another may see the same on multiple parameters.

We show this more concretely in the ribbon plots Figure 4. Indeed, we find variability in the magnitude and directionality of sequential effects even when considering related sequences (value difference $\downarrow$ vs. value difference $\downarrow$ *and* amb. $\downarrow$). For subjects that showed non zero sequential effects on successive trials that increased or decreased in value difference regardless of ambiguity, we recovered these effects only on β_1, the Expected Value difference parameter on the drift rate. This serves as a sanity check of sorts, as reward value is explicitly weighted by β_1 through Expected Value. Similarly, for the handful of subjects that were only sensitive to relative differences in cross-trial ambiguity, we only recovered

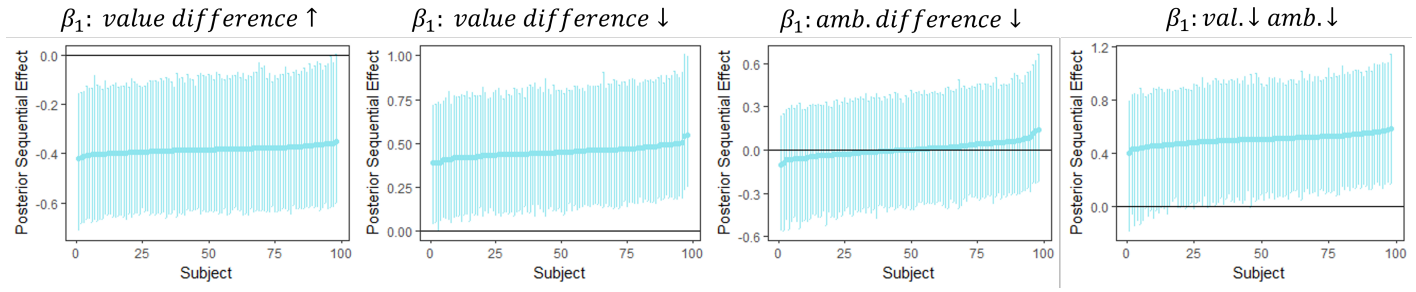


Figure 4: Sorted posterior 95% Credible Intervals of sequential effects on the drift rate Expected Value term (β_1) when successive trials (a) increase in value difference, (b) decrease in value difference, (c) decrease in ambiguity, and (d) decrease in both value difference and ambiguity difference. (Equation 2, RAC task).

effects on β_2, the weight of ambiguity on the drift rate. For decrease in value difference and decreases in both value and ambiguity difference, inferred Bayes Factors were near threshold ($BF_v = 2.97, BF_{va} = 2.9$).

4 Conclusion

Using a generative model and a simple parameterization of trial properties, we have demonstrated that individuals are influenced by recent history during economic decision making, even when it is task-incongruent to do so. We do not argue that this influence is a conscious one, but more specifically that these sorts of sequential influences are fundamental to human cognition and information processing, and thus ought to be explicitly accounted for. We find near-ubiquitous evidence of sequential effects in the ITC task. Our results from the RAC task showed more specific sequential effects and suggest that these effects are driven by relative differences in difficulty, and not necessarily uncertainty (ambiguity). We further show that in this task, explicitly incorporating trial properties into response time modeling via drift rates is fundamental to recovering these sequential effects (Figure 5).

Properties	β_0	β_1	β_2	*bias*
$value\uparrow$	H_0	$-0.38(0.12)$	H_0	H_0
$value\downarrow$	NEE	$0.45(0.11)$	NEE	H_0
$amb\uparrow$	H_0	NEE	$-0.29(0.07)$	H_0
$amb\downarrow$	H_0	H_0	$0.32(0.09)$	H_0
$v.\uparrow a.\uparrow$	H_0	NEE	H_0	H_0
$v\uparrow a.\downarrow$	H_0	NEE	$0.59(0.13)$	H_0
$v.\downarrow a.\uparrow$	H_0	NEE	NEE	H_0
$v.\downarrow a.\downarrow$	NEE	$0.5(0.16)$	H_0	H_0

Figure 5: Posterior sequential effect group means (γ_i) for Risk and Ambiguity Choice Task. All numerical means and standard deviations presented in a cell (*mean (standard deviation)*) have a BF > 3 that they are non-zero. If instead there is a H_0 then we find evidence (BF > 3) in favor of the null. Finally, if a cell contains "NEE" then the data does not contain enough evidence to favor either the null or alternative hypothesis (NEE: Not enough evidence).

We also note the value of simple parametrizations of trial properties: by explicitly modeling only relative increases or decreases, we were able to recover substantially more individuals who were sensitive to sequential effects. Thus, we have found strong evidence of effects that are likely widespread and therefore should be explicitly accounted for in all measures (Figure 3). Indeed, this could be one reason why task measures do not seem to correlate with each other or have as good test-retest reliability as other measures like surveys [6].

References

[1] N.V. Banavar, M.D. Lee, and A.M. Bornstein. Sequential effects in non-sequential tasks. *Proceedings of the 19th International Conference on Cognitive Modeling*.

[2] H.R. Brooks and P. Sokol-Hessner. Quantifying the immediate computational effects of preceding outcomes on subsequent risky choices. *Scientific reports*, 10(1):1–10, 2020.

[3] I. Dasgupta, E. Schulz, N.D. Goodman, and S.J. Gershman. Remembrance of inferences past: Amortization in human hypothesis generation. *Cognition*, 178:67–81, 2018.

[4] N.D. Daw, A.C. Courville, and P. Dayan. Semi-rational models of conditioning: The case of trial order. *The probabilistic mind*, pages 431–452, 2008.

[5] J. Fischer and D. Whitney. Serial dependence in visual perception. *Nature neuroscience*, 17(5):738–743, 2014.

[6] R. Frey, An. Pedroni, R. Mata, J. Rieskamp, and R. Hertwig. Risk preference shares the psychometric structure of major psychological traits. *Science advances*, 3(10):e1701381, 2017.

[7] X. Gabaix and D. Laibson. Myopia and discounting. Technical report, National bureau of economic research, 2017.

[8] L.E. Hunter, A.M. Bornstein, and C.A. Hartley. A common deliberative process underlies model-based planning and patient intertemporal choice. *bioRxiv*, page 499707, 2018.

[9] A. Kiyonaga, J.M. Scimeca, D.P. Bliss, and D. Whitney. Serial dependence across perception, attention, and memory. *Trends in Cognitive Sciences*, 21(7):493–497, 2017.

[10] M.D. Lee. Bayesian methods in cognitive modeling. *The Stevens' handbook of experimental psychology and cognitive neuroscience*, 5:37–84, 2018.

[11] I. Levy, J. Snell, A.J. Nelson, A. Rustichini, and P.W. Glimcher. Neural representation of subjective value under risk and ambiguity. *Journal of neurophysiology*, 103(2):1036–1047, 2010.

[12] J. Peters and C. Büchel. The neural mechanisms of inter-temporal decision-making: understanding variability. *Trends in cognitive sciences*, 15(5):227–239, 2011.

[13] M. Plummer et al. Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, volume 124, pages 1–10. Vienna, Austria., 2003.

[14] F. Schuur, B.P. Tam, and L.T. Maloney. Learning patterns in noise: Environmental statistics explain the sequential effect. In *CogSci*, 2013.

[15] D. Wabersich and J. Vandekerckhove. Extending jags: A tutorial on adding custom distributions to jags (with a diffusion model example). *Behavior research methods*, 46(1):15–28, 2014.

[16] J. Zimmermann, P.W. Glimcher, and K. Louie. Multiple timescales of normalized value coding underlie adaptive choice behavior. *Nature communications*, 9(1):1–11, 2018.

Accelerating Reinforcement Learning using Frequently Used Transition Pathways

Edward W. Barker
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
ebarker@student.unimelb.edu.au

Charl J. Ras
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
cjras@unimelb.edu.au

Abstract

An important challenge in current reinforcement learning research is finding ways to enable agents to learn more-quickly, to help mitigate the need for large amounts of training. We introduce a novel technique to accelerate the rate at which a reinforcement learning algorithm can learn a value function (VF) approximation. The technique involves keeping a dynamic list of a relatively small collection of pathways through the state-action space. This list is periodically updated. Pathways are selected for inclusion based on the frequency with which they are traversed by the agent as it learns. Whenever the agent follows one of these pathways, as soon as it exits the pathway, a full backup of the VF estimates along the pathway is performed. This means that state-action pairs which sit on these pathways have VF estimates which rapidly lose any initial bias. The compute cost of doing these full backups is constrained by maintaining, at each state-action pair on a stored pathway, an estimate of the value of leaving the pathway at that point.

We provide a theoretical analysis which demonstrates that our technique can, under appropriate conditions, significantly accelerate learning compared to classical temporal difference methods, whilst creating only modest additional computational overhead. We also provide preliminary experimental results which corroborate our theoretical analysis. We posit that the technique is effective because learning the approximate frequency with which transitions through the state-action space are occurring is relatively quick and easy, compared to learning the VF itself. Once learned, however, information regarding transition frequencies can be exploited in such a way that it is possible to more efficiently construct an accurate VF estimate. The technique is most effective when an agent tends to favour certain regions of the state-action space, although these favoured regions can change over time and do not need to be known in advance.

Keywords: Reinforcement learning, SARSA, Learning rate, Model-based learning, Trajectory sampling

Acknowledgements

This research was supported by an Australian Government Research Training Program Scholarship.

1 Introduction

Whilst reinforcement learning techniques have recently shown considerable success on a range of complex problems [5, 7], many of the most note-worthy successes have come about, in part, as a result of providing agents with enormous numbers of samples on which to train. As a result, researchers are currently interested in identifying ways to enable algorithms to learn more efficiently, and to make better use of training samples [3].

In this paper we propose a new and simple technique designed to increase learning efficiency. We introduce the technique as an extension to tabular SARSA, however (as discussed below) the technique can be extended to cases where value function (VF) approximation is used. We will: (a) introduce an algorithm, SAR-σ, which formally implements our technique; (b) provide theoretical results which demonstrate that SAR-σ has only modest computational complexity compared to SARSA, and that it will, given a fixed policy, converge to an accurate VF estimate considerably faster than SARSA under appropriate conditions; and (c) provide preliminary experimental results to corroborate our theoretical results. We also provide a motivational discussion where we explain the intuition behind our proposed technique, and compare it to related techniques in the literature.

The technique assumes that an agent will store a list of frequently-traversed pathways through the state-action space. This list will be periodically updated. Whenever an agent begins to traverse one of these pathways, it delays updating the VF estimate until it leaves the pathway, at which point it performs a full backup of the VF along the pathway it has just traversed. These full backups have the effect of circumventing the often inefficient bootstrapping which occurs when using temporal differences, allowing a more-rapid decay of the bias in the VF estimate along these pathways. The technique has connections to a number of existing techniques in the literature, however it is arguably most closely related to trajectory sampling [6] for model-based learning of a VF. Our method differs in that it tightly constrains the manner in which full backups are performed, allowing us to provide guarantees around computational cost, and allowing easier integration with function approximation methods. A key feature of our method is its reliance on measuring the frequency of transitions to guide the manner in which full backups are performed. This heavy reliance on "visit frequency" makes our method conceptually quite closely aligned with the class of methods which use visit frequency to guide how to construct a VF approximation architecture in the case of large state or action spaces [2].

Let us use $p(X)$ to denote the set of probability distributions over some set X. Formally, we define a *Markov Decision Process* as a tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of *states* and *actions* respectively (generalisation to the continuous case is possible but beyond our current scope), $R : \mathcal{S} \times \mathcal{A} \to p(\mathbb{R})$ is a *reward function*, $P : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$ is a *transition function*, and $\gamma \in [0, 1]$ is a *discount factor*. A *policy* is a mapping $\pi : \mathcal{S} \to p(\mathcal{A})$. We have a sequence of iterations $t \in \{1, 2, \ldots\}$. At each t an agent is in a particular state $s_t \in \mathcal{S}$. The agent samples an action a_t from the distribution $\pi(s_t)$ for some policy π. The agent then transitions to a new state s_{t+1} sampled from $P(s_t, a_t)$, and a reward r_t sampled from $R(s_t, a_t)$ is generated. Our overarching formal objective is to design an algorithm which will find a policy which, assuming s_0 is sampled from some predefined element of $p(\mathcal{S})$, maximizes expected discounted reward.

2 The SAR-σ algorithm

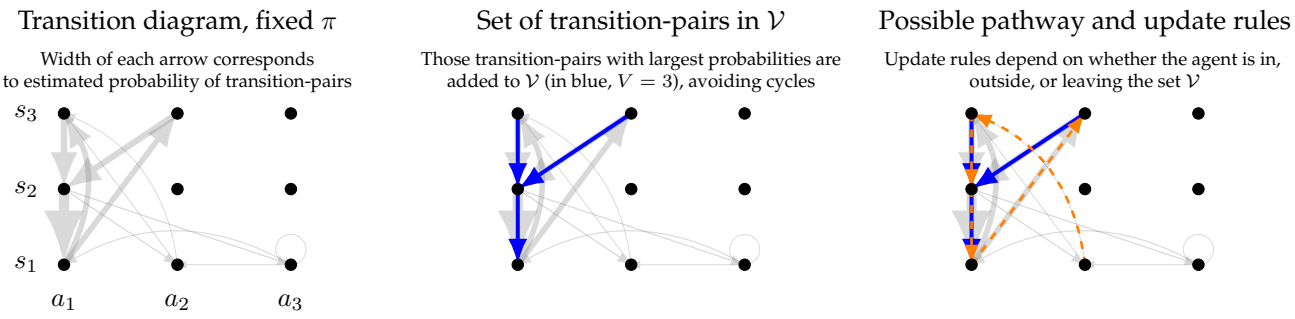


Figure 1: Example of construction by SAR-σ of frequently visited pathways, $|\mathcal{S}| = 3$ and $|\mathcal{A}| = 3$.

In this section we outline a new algorithm, SAR-σ, to formally implement our proposed technique. SAR-σ is equivalent to tabular SARSA, however with some additions. In implementing our technique, we do not explicitly store a list of frequently-traversed pathways. We instead store a list $\mathcal{V}$, of fixed size V, of the most frequently-occurring *transition-pairs*, defined as state-action-state-action 4-tuples. Exploiting the Markov property, our list of "frequently-traversed pathways" will be implicit, and will be taken to be any pathway which can be constructed from elements of the list $\mathcal{V}$ of transition-pairs (we do not need to explicitly construct the implicit list of frequently-traversed pathways at any point). We apply a constraint so that the list $\mathcal{V}$ cannot include any set of transition-pairs which would result in a cyclical pathway. This ensures that any individual implied pathway will not exceed length V. After every ν iterations, where ν is a parameter

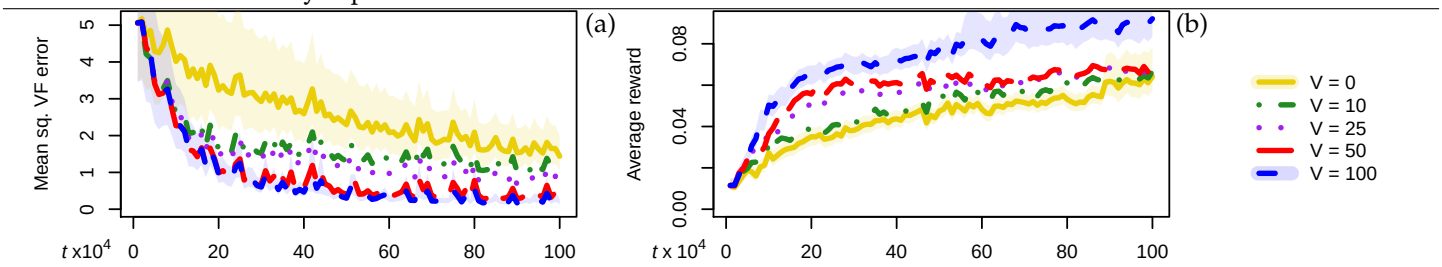


Figure 2: (a) Mean squared error over 50 independent runs with a fixed policy, (b) Average reward over 50 independent runs with an ϵ-greedy policy. Shaded regions, for $V = 0$ and $V = 100$, indicate one standard deviation.

which is typically large, we replace $\mathcal{V}$ with the V transition-pairs with the highest estimated probability under the current policy. An estimate $\hat{P}$ of P can be constructed in a number of ways, however in this paper we have used an $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ array to estimate P (keeping a count of each transition, suitably normalised, and with greater weighting given to recent transitions). This array, in combination with π, provides an estimate for every transition-pair probability.

Algorithm 1 The SAR-σ algorithm. Called at each iteration t. Initialise $\mathcal{L}$ as empty and $\mathcal{V}$ arbitrarily. We use Π to represent an estimate of the probability of some state-action-state-action sequence. It can be calculated using π and $\hat{P}$.

Select a_t according to some policy π (e.g. ϵ-greedy)
if $t \bmod n = 0$ **then**
  Clear $\mathcal{V}$
  while $|\mathcal{V}| < V$ **do**
    Find $v = (s, a, s', a') \notin \mathcal{V}$ satisfying:
      v has maximum probability $\forall\, (s, a, s', a') \notin \mathcal{V}$
      v, if added to $\mathcal{V}$, will not create a cycle
    Add v to $\mathcal{V}$
    $\vec{Q}(s', a') \leftarrow \check{Q}(s', a') -$
      $\sum_{(s'',a'')\in\mathcal{S}\times\mathcal{A}} \Pi(s', a', s'', a'')\hat{Q}(s'', a'')$
  end while
end if
$\check{R}(s_{t-1}, a_{t-1}) \leftarrow \check{R}(s_{t-1}, a_{t-1}) +$
  $\eta(R(s_{t-1}, a_{t-1}) - \check{R}(s_{t-1}, a_{t-1}))$
$v_t \leftarrow (s_{t-1}, a_{t-1}, s_t, a_t)$
if $v_{t-1} \notin \mathcal{V}$ and $v_t \notin \mathcal{V}$ **then**
  $\check{Q}(s_{t-1}, a_{t-1}) \leftarrow \check{Q}(s_{t-1}, a_{t-1}) +$
    $\eta(\gamma\check{Q}(s_t, a_t) - \check{Q}(s_{t-1}, a_{t-1}))$
else if $v_t \in \mathcal{V}$ **then**
  Add (s_t, a_t) to front of $\mathcal{L}$
  $\vec{Q}(s_{t-1}, a_{t-1}) \leftarrow \vec{Q}(s_{t-1}, a_{t-1}) - \eta\check{Q}(s_{t-1}, a_{t-1})$
else ▷ Implies $v_{t-1} \in \mathcal{V}$ and $v_t \notin \mathcal{V}$
  while $|\mathcal{L}| > 0$ **do**
    $l \leftarrow$ pair at front of $\mathcal{L}$
    $\check{Q}(l) \leftarrow \vec{Q}(l) +$
      $\sum_{(s',a'):(l_1,l_2,s',a')\in\mathcal{V}} \Pi(l_1, l_2, s', a')\hat{Q}(s', a')$
    Remove l from front of $\mathcal{L}$
  end while
  $\vec{Q}(s_{t-1}, a_{t-1}) \leftarrow \vec{Q}(s_{t-1}, a_{t-1}) +$
    $\eta(\hat{Q}(s_t, a_t) - \check{Q}(s_{t-1}, a_{t-1}))$
end if
$\hat{Q}(s_t, a_t) \leftarrow \check{Q}(s_t, a_t) + \check{R}(s_t, a_t)$
Update $\hat{P}$ (e.g. using a count of transitions)

Like SARSA, SAR-σ uses a fixed step size η. The SAR-σ algorithm also maintains, for each state-action pair (s, a), a value estimate $\hat{Q}$, much like SARSA. However SAR-σ stores $\hat{Q}$ separated into two terms, $\check{R}$ and $\check{Q}$, where $\check{R}(s, a)$ estimates expected reward and $\check{Q}(s, a)$ estimates the expected discounted value of the next state-action pair, so $\hat{Q} = \check{R} + \check{Q}$. For each pair (s', a') such that $(s, a, s', a') \in \mathcal{V}$, we also maintain an estimate $\vec{Q}$. The value $\vec{Q}$ represents the state-action value conditional upon the agent's next transition being outside the set $\mathcal{V}$. Storing this value is instrumental in curtailing the computational cost of doing full backups, as it forms a temporary estimate of the value of "leaving" a particular pathway. Details of how each of these values are updated is in Algorithm 1 (see also Figure 1).

3 Theoretical results

We make some comments on algorithm complexity. The time complexity of SAR-σ in each iteration will be $O(V)$ due to the backup process described in (C) above, compared to $O(1)$ for SARSA. However this very much represents a "worst-case". Provided that the pathways implied by $\mathcal{V}$ do not branch heavily, which is the situation we would expect to observe more commonly in practice, the computational demands of SAR-σ will be close to SARSA. The more-complex operations regarding the formation of $\mathcal{V}$ occur infrequently and so will not significantly impact processing time. If, to estimate P, we store a separate value for every state-action-state sequence, then space complexity for SAR-σ will be $O(S^2A)$, compared to $O(SA)$ for SARSA. However this relatively large space complexity can be avoided by using function approximation as well as more efficient techniques to identify high probability transitions (see below). The next result explores the increase in learning speed realisable by using SAR-σ as opposed to SARSA. For simplicity we assume that P is available and note that, in environments of sufficient complexity, an accurate estimate, $\hat{P}$, for P can be generated quickly compared to $\hat{Q}$. At iteration t we say that the agent is *in* $\mathcal{V}$ if $(s_{t-1}, a_{t-1}, s_t, a_t) \in \mathcal{V}$, otherwise the agent is *not in* $\mathcal{V}$.

Theorem 1. Suppose that π is a fixed policy and that ζ is the proportion of the time the agent is in $\mathcal{V}$ when following π. For both SARSA and SAR-σ define $J \coloneqq \sum_{s\in\mathcal{S},a\in\mathcal{A}} \Pr(s,a)(\hat{Q}(s,a) - Q^\pi(s,a))^2$, where $\Pr(s,a)$ is the stationary distribution induced by π. Suppose that P is available. Define τ_0 and τ_σ as the number of iterations required such that J is less than ε_1 with probability at least $1-\varepsilon_2$ for SARSA and SAR-σ respectively. Then, provided $\min\{|\hat{Q}(s,a) - Q^\pi(s,a)| : (s,a) \in \mathcal{S}\times\mathcal{A}\} > 0$ there exists $\varepsilon_2 > 0$ such that $\tau_\sigma/\tau_0 \in O(1-\zeta)$ as $\varepsilon_1 \to 0$.

Sketch of proof. The following captures the main ideas although omits several details. The proof relies on the fact that (on average, and provided the agent tends to stay within $\mathcal{V}$) bootstrapping will occur over longer sequences of states and actions when using SAR-σ compared to SARSA. For SARSA, we can use continuous time ODEs to approximate the behaviour of $\hat{Q}$. Denote as $\hat{Q}_t$ a continuous time analogue of $\hat{Q}$ for $t \in \mathbb{R}^+$:

$$\hat{Q}_t(s_\tau, a_\tau) = \sum_{i=1}^{\infty} \left\{ \gamma^i \mathrm{E}\left(R_{\tau+i}\right) + \underbrace{\frac{\mathrm{E}\left(\hat{Q}_0(s_{\tau+i}, a_{\tau+i}) - \sum_{j=0}^{\infty}\gamma^j R_{\tau+i+j}\right)(\gamma\eta t)^i}{(i-1)!e^{\eta t}}}_{\text{Error term}} \right\} \quad \text{where } R_\tau \coloneqq R(s_\tau, a_\tau). \tag{1}$$

This formula can be derived, for example, by starting with the ODE for an "episode" of length one:

$$\frac{d\hat{Q}_t(s,a)}{dt} = \eta(\mathrm{E}(R(s,a)) - \hat{Q}_t(s,a)) \quad \text{to which the solution is} \quad \hat{Q}_t(s,a) = \left(\hat{Q}_0(s,a) - \mathrm{E}(R(s,a))\right)e^{-\eta t} + \mathrm{E}(R(s,a)) \tag{2}$$

and extending this formula recursively to episodes of increasing length. We now examine the total error, J, which is given by the sum over the second term in the right hand side of the above equation. We have:

$$J = \sum_{s\in\mathcal{S},a\in\mathcal{A}} \Pr(s,a) \left(\sum_{i=1}^{\infty} \frac{\mathrm{E}\left(\hat{Q}_0(s_{\tau+i}, a_{\tau+i}) - \sum_{j=0}^{\infty}\gamma^j R_{\tau+i+j}\right)(\gamma\eta t)^i}{(i-1)!e^{\eta t}} \right)^2 \geq C \left(\sum_{i=1}^{\infty} \frac{(\gamma\eta t)^i}{(i-1)!e^{\eta t}} \right)^2 =: J', \tag{3}$$

where C denotes some constant. The series inside the brackets is the same as the expectation over a Poisson random variable with parameter ηt, with a weighting (governed by γ) which has a value of one at the origin and decays geometrically. Consider SARSA. In order for J to be small, ηt must be large (so the the mean of the corresponding Poisson distribution is shifted a long way from the origin, and hence from larger weightings). Since ηt will large, and the majority of the probability mass of the Poisson distribution will be for large values of i we can approximate the error using a normal distribution with mean ηt and variance ηt. Accordingly:

$$J' \approx C \left(\int_0^\infty e^{-\gamma x} \frac{1}{\sqrt{2\pi\eta t}} e^{-\frac{(x-\eta t)^2}{2\eta t}}\, dx \right)^2. \tag{4}$$

Now consider SAR-σ. Define $\rho \coloneqq 1/(1-\zeta)$, the average number of iterations which pass before a single backup by SAR-σ. If we assume that the number of states frequently visited is much smaller than $1/(1-\gamma)$ and that ηt is large, we can obtain an inequality for SAR-σ the same as that in equation (3) but with each coefficient γ^i replaced by a new coefficient $\gamma^{\rho i}$, the inequality reversed, and a new constant c replacing C. In effect, we have scaled the weighting function by a factor of ρ. It is then apparent that we have a bound on comparative error for SAR-σ and SARSA if, for SAR-σ, we transform ηt so that $\eta t \to \eta t\rho = \eta t/(1-\zeta)$. This is the result for the underlying ODE. By assuming that η is small, we can rely on Theorem 1.1 in Chapter 10.1 from [4] to ensure that the value of $\hat{Q}$ will remain in a small region of the value of the ODE for all t, a region defined by a normal distribution with variance a function of η. From this the result follows. □

The result means that, as ζ approaches one (that is, the agent spends most of its time in $\mathcal{V}$), then the samples required by SAR-σ to converge becomes small compared to SARSA. Since $\mathcal{V}$ cannot contain transition-pairs forming a cycle we avoid the degenerate case where $\zeta = 1$. In fact the learning rate increase has an upper bound of $O(V)$. Note, of course, that in many problem settings, reducing training iterations is a more critical concern than reducing compute time per iteration.

4 Experimental results

We have run preliminary experiments to provide some initial validation of our ideas.[1] We define U as a uniform random variable over the range $[-1,1]$. The problem we examine here is a randomly generated environment with deterministic transitions and with $|\mathcal{S}| = 1000$ and $|\mathcal{A}| = 2$. We assign $R(s,a) = 1 + U$ for 20 randomly chosen state-action pairs and $R(s,a) = U$ for the remaining pairs. We set $\gamma = 0.99$, $\nu = 10^4$, and initialised $\hat{Q}$ uniformly randomly over the range $[-1,1]$. We have run two sets of experiments. In the first experiment we held π fixed, with π such that action 1 is selected with 0.99 probability and otherwise action 2 is selected, and set $\eta = 0.001$. As Figure 2(a) illustrates, SAR-σ makes a marked difference to rate at which VF error drops (SARSA is equivalent to $V = 0$). As expected, as V is increased, the

[1] All associated code and experimental data is available here: https://github.com/edwbarker/RLDM2022

impact increases. In the second experiment we set $\eta = 0.01$ and applied an ϵ-greedy policy ($\epsilon = 0.01$). As Figure 2(a) illustrates, SAR-σ also improves performance. This is presumably because, if $V > 0$, the agent will have, as its policy evolves, more rapid access to accurate VF estimates. This will aid the agent in selecting appropriate actions. The average compute time per iteration, excluding the periodic updates to $\mathcal{V}$, was approximately equivalent for all values of V.

5 Discussion and related methods

SAR-σ exploits the fact that state-action pairs which are visited with high frequency will contribute more to total error (as measured, for example, by J) than state-action pairs which are visited infrequently. It learns which transition-pairs occur most frequently, which is much easier to learn than the VF itself since it does not depend on multiple complex pathways through the state-action space. By performing full backups on pathways composed from these transition-pairs, SAR-σ can generate accurate estimates on these pathways with far fewer samples. Conversely, by performing full backups on *only* these pathways, and using $\vec{Q}$ to summarise the effect of leaving a pathway, SAR-σ avoids the computational cost associated with doing full backups. The selection of V is important. If V is too large, then the backups will become computationally expensive, and if V is too small, then the full backups may confer little advantage. As Theorem 1 makes clear, the technique is better suited to situations where an agent tends to spend the majority of its time in a relatively limited region of the state-action space, though this region can change, and does not need to be known in advance.

SAR-σ bears similarities with some other techniques from the literature. Insofar as SAR-σ performs updates over pathways, as opposed to relying purely on single-step bootstrapping, it bears a superficial similarity to SARSA(λ) and n-step SARSA. These latter methods, however, use individual samples of the *aggregate* reward over pathways, and as such, whilst they side-step the inefficiency involved in bootstrapping, the samples they collect may be subject to large variance, which may need to be compensated for by smaller step sizes. SAR-σ is arguably most closely related to the set of model-based methods which seek to learn an estimate for P and then perform backups using this estimate [6]. What differentiates SAR-σ from existing model-based methods (for example prioritised sweeping or trajectory sampling), however, is that SAR-σ uses transition probabilities in a strictly targeted way. In particular, transition probabilities are only used to construct common transition pathways for the purpose of backups over small sets of state-action pairs. As such one of the central challenges associated with model-based learning, which is the cost of doing backups, is addressed. Since SAR-σ constructs, albeit implicitly, sequences of state-action pairs, and concentrates its learning on the behaviour of the VF on these sequences, some connections can also be drawn to the concept of options [1]. However options must either be pre-defined, or else are learned by using VF error as direct feedback, which is conceptually quite distinct from the dependence of SAR-σ on transition probabilities.

SAR-σ can be generalised beyond the tabular case. For example, SAR-σ can be adapted to operate with aggregations of states and actions, instead of individual states and actions, including where these aggregations are dynamically updated [2]. When this is done, we can use SAR-σ to accelerate learning, even when the state and action spaces are very large. In fact, starting from coarse state representations, estimates of visit frequency can be used to guide the creation of more precise state representations, more precise representations of state-action pairs, and finally *sequences* of state-action pairs (as with SAR-σ), all within the same linear, and therefore stable, approximation framework. It is not immediately clear, however, how easily the technique could be applied to complex non-linear approximation architectures, such as deep neural networks, although this would be an interesting direction for further research.

References

[1] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[2] E. Barker and C. Ras. Unsupervised basis function adaptation for reinforcement learning. *Journal of Machine Learning Research*, 20(128):1–73, 2019.

[3] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[4] H. Kushner and G. G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*, volume 35. Springer Science & Business Media, 2003.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[6] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, second edition, 2018.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

PARSR: Priority-Adjusted Replay for Successor Representations

Samuel A. Barnett*
Department of Computer Science
Princeton University
Princeton, NJ 08540
samuelab@princeton.edu

Ida Momennejad
Microsoft Research
300 Lafayette Street
New York, NY 10012
idamo@microsoft.com

Abstract

Intelligent agents are capable of transfer and generalization. This flexibility in adapting to new tasks and environments often relies on representation learning and replay. Among these algorithms, successor representation learning and memory replay offer biologically plausible solutions. However, replay prioritization algorithms remain largely limited to prediction errors. Here we propose PARSR (pronounced *PARS*-er), Priority-Adjusted Replay for Successor Representations, to address this caveat. Decoupling learning of the environment dynamics and rewards, PARSR can use prediction errors from either representation learning or rewards to prioritize memory replay. We compare PARSR to prioritized sweeping, Dyna, and a number of state of the art algorithms using replay and successor representations in cognitive neuroscience.

Keywords: reinforcement learning, prediction error, replay, planning, decision-making, prioritization, human-like behavior, Dyna

Acknowledgements

We are grateful for the support and feedback provided by Arthur Juliani, Brandon Davis, Katja Hofmann, Sam Devlin, Harm van Seijen, Evan Russek, Marcelo Mattar, and Nathaniel Daw.

*Research performed as an intern at Microsoft Research.

1 Introduction

Intelligent agents are capable of transfer and generalization. Imagine driving to a coffee shop to meet a friend. If we encounter a blocked road, we are able to quickly adapt in choosing a new route that will get us there. Or if we decide that a different coffee shop might be preferable, we can just as easily change course.

To accommodate this flexibility to changes in the environment, contemporary reinforcement learning algorithms often rely on representation learning and replay [1, 2, 3, 4]. **Learning flexible yet compact representations of the environment allows us to adapt to changes in its rewards, and replay enables us to adapt to changes in transition structures without the need for significant amounts of further real experience.**

One such algorithm proposes a biologically plausible solution [5], combining successor representation (SR) [6, 7] for representation learning supplemented by memory replay (inspired by Dyna architectures [8]). This algorithm, Dyna-SR for short, captures human behavioral accuracy and reaction times across a number of tabular tasks. A key advantage of the Dyna-SR algorithm is that it is almost as flexible a model-based RL algorithm, while at decision time it is almost as inexpensive as model-free RL. By caching multi-step trajectories of states offline, Dyna-SR remains more flexible than MFRL and SR alone, while avoiding MBRL's high cost of rolling out entire state-action-state-reward trajectories at decision time.

As a caveat, the replay prioritization in current implementations of Dyna-SR focus on more recent memories for efficiency. While this heuristic may capture certain aspects of human memory recall [9], previous work in cognitive neuroscience suggest human-like replay may be better modelled by an error-based replay prioritization [10]. **Thus, here we extend the scope of algorithms combining representation and replay, using both reward-based and representation-based errors for replay prioritization.** While current approaches remain largely limited to tagging memories with reward prediction errors (PE) for priority [11, 12, 13, 14, 15], our proposed algorithm is inspired by Dyna-SR but can prioritize replay using either reward PE and successor PE.

We propose PARSR (pronounced *PARS*-er), Priority-Adjusted Replay for Successor Representations, which improves on Dyna-SR offering more human-like replay prioritization with no effective increase in hyperparameters. As an SR-based algorithm, PARSR learns a representation of the transition structure (i.e., the environment dynamics) and the reward structure separately, allowing either to be quickly relearned for greater generalization. Critically, by decoupling reward and transition representations, PARSR can use the prediction errors from either to prioritize memory replay.

We propose two variants of PARSR based on the choice of prediction error: M-PARSR (prioritizes memories using successor PE) and Q-PARSR (prioritizes memories using value PE). This prioritization for memory selection distinguishes PARSR from Dyna-SR [5], which performs replay-enhanced SR learning with random memory selection with a recency bias.

We test how well PARSR captures human behavior in small tabular experiments (with 6 states) [1] as well as a scaled version of the experiments (with 121 states). We compare PARSR to a number of state of the art algorithms using replay on simple benchmark transfer learning (or *revaluation*) tasks in cognitive neuroscience (Figs. 1a, 1b, 1c) and a scaled up version of these tasks (Figs.1d, 1e, 1f). We find that PARSR matches human-like behavior as well as other algorithms' efficiency in learning speed of the prioritization-based algorithms. To clarify differences among the solutions different replay heuristics provide, we visualize which experiences are prioritized as more important to recall by different prioritization algorithms (Fig. 1e). The code for implementing the algorithm and benchmark experiments is available at `https://github.com/s-a-barnett/PrioritizedSR`.

1.1 Related work

Experience replay has been incorporated as a core part of many reinforcement learning algorithms, both in the tabular setting [8, 16], and in deep reinforcement learning [17]. The use of prioritization based on prediction error as a heuristic for faster learning has also been explored in both settings [11, 12, 13, 14, 15].

The successor representation was first introduced in [6], forming the basis for a number of algorithms in both tabular and function approximation settings [1, 5, 18, 4]. It has also been studied for its neuroscientific plausibility in [7]. For comprehensive overviews of the successor representation and its properties, refer to [19] and [2].

2 Algorithms

We focus on RL algorithms that integrate the advantages of both model-based planning (flexibility) and model-free learning methods (speed). For a full treatment on the RL setting, we refer to the classic text by Sutton and Barto [20], in particular Chapter 8. We use as baselines the Dyna-Q algorithm [8, 20] and prioritized sweeping (PS) algorithm [12, 11, 20], which interleave learning from real experience with learning from simulated experience, or *replay*, sampled from a model learned from previous interactions with the environment. In the deterministic setting considered in this paper, this model is a dictionary whose entries are state-action pairs, and whose values are the next state and received reward.

While Dyna-Q and prioritized sweeping both exploit replay to integrate new experiences into their Q function more efficiently, they differ in their selection mechanisms: Dyna-Q samples past experiences uniformly from its model, whereas PS selects experiences based on a queue ordered according to the magnitude of the temporal difference (TD) error on the Q function that was recorded when the state was most recently encountered. This is then propagated through to states that precede the replayed state, allowing the largest changes to flow backwards through the model. This approach is consistent with human studies suggesting that larger prediction errors are followed by more offline replay, and offline replay of predecessors of states tagged with prediction error is correlated with future revaluation behavior [10].

Algorithm 1 PARSR

1: **Hyperparameters:** Number of replay cycles n, exploration parameter ε, SR learning rate α_M, reward weights learning rate $\alpha_{\boldsymbol{w}}$, prioritization type *PriType*.
2: Initialize $M(a, s, s')$, $\boldsymbol{w}(s)$, *Model*(s, a) for all s, a and *PQueue* to empty.
3: **while** True **do**
4: $s \leftarrow$ current (nonterminal) state.
5: $Q \leftarrow \boldsymbol{w}(:)^\top M(a, s, :)$.
6: $a \leftarrow \varepsilon$-*greedy*(s, Q).
7: Take action a; observe reward, r, and state, s'.
8: *Model*$(s, a) \leftarrow r, s'$.
9: $Q \leftarrow \boldsymbol{w}(:)^\top M(a, s, :)$.
10: $a' \leftarrow \arg\max_{a''}(Q(s', a''))$.
11: $\boldsymbol{\delta}_M \leftarrow \left[\mathbf{1}_s + \gamma M(a', s', :) - M(a, s, :)\right]$. ▷ SR ($M$) prediction error.
12: $M(a, s, :) \leftarrow M(a, s, :) + \alpha_M \boldsymbol{\delta}_M$.
13: $\boldsymbol{w}(s) \leftarrow \boldsymbol{w}(s) + \alpha_{\boldsymbol{w}}\left[r - \boldsymbol{w}(s)\right]$.
14: **if** *PriType* is M-PARSR **then**
15: $p \leftarrow \|\boldsymbol{\delta}_M\|$
16: **else if** *PriType* is Q-PARSR **then**
17: $p \leftarrow \delta_Q \equiv \boldsymbol{\delta}_M^\top \boldsymbol{w} - \boldsymbol{w}(s) + r$ ▷ Q prediction error.
18: Insert s, a into *PQueue* with priority p.
19: **loop** n times
20: **if** *PQueue* is not empty **then**
21: $\bar{s}, \bar{a} \leftarrow$ *first*(*PQueue*).
22: **else**
23: $\bar{s} \leftarrow$ random previously observed state.
24: $\bar{a} \leftarrow$ random action previously taken in $\bar{s}$.
25: $\bar{r}, \bar{s}' \leftarrow$ *Model*$(\bar{s}, \bar{a})$.
26: $Q \leftarrow \boldsymbol{w}(:)^\top M(\bar{a}, \bar{s}, :)$.
27: $\bar{a}' \leftarrow \arg\max_{\bar{a}''}(Q(\bar{s}', \bar{a}''))$.
28: $\boldsymbol{\delta}_M \leftarrow \left[\mathbf{1}_s + \gamma M(\bar{a}', \bar{s}', :) - M(\bar{a}, \bar{s}, :)\right]$.
29: $M(\bar{a}, \bar{s}, :) \leftarrow M(\bar{a}, \bar{s}, :) + \alpha_M \boldsymbol{\delta}_M$.

Though replay alone improves an agent's ability to relearn local aspects of the task at hand, the representation of this task (namely, the Q function) is inflexible inasmuch as it can obscure the different kinds of changes that might take place. This can lead to slower learning, and does not correspond to the representations of such tasks used by biological agents [21, 19, 2]. To provide further flexibility, we define the successor representation (SR) as the discounted sum over expected future state visitations:

$$M(a, s, s') = \mathbb{E}\left[\sum_{t=T}^{\infty} \gamma^{t-T} \mathbf{1}(s_t = s') \mid s_T = s, a_T = a\right]. \quad (1)$$

The Q function can now be expressed as a linear combination of the SR and the rewards of each state, $\boldsymbol{w}(s')$:[1]

$$Q(s, a) = \sum_{s'} M(a, s, s')\boldsymbol{w}(s'). \quad (2)$$

The SR is learned with its own TD update (algorithm 1, line 11), and the weights are learned using a direct update (algorithm 1, line 13).

In SR-based RL algorithms each experience simultaneously updates both the SR and reward weights, enabling flexibility to changes in rewards (reward transfer). However, transition transfer requires updating SR for states with reevaluated transitions, via real or replayed experience. Thus, combining the representational flexibility of the SR with efficient forms of planning through replay achieves both reward and transition transfer.

The Dyna-SR algorithm [5] is such a hybrid, sampling experiences randomly with a recency-weighted bias in order to update SR. However, we hypothesized that sampling past experiences according to a prioritization schema, similar to prioritized sweeping, could further improve performance while still capturing human-like behavior.

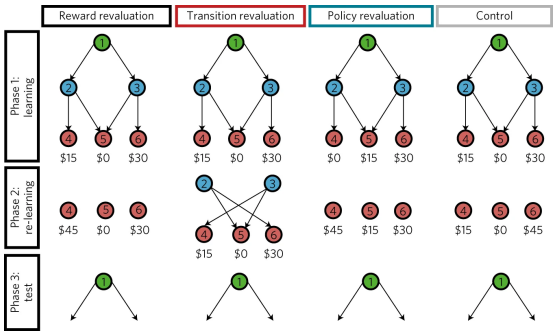
We call this novel algorithm *PARSR*: Priority-Adjusted Replay for Successor Representations (algorithm 1 and Fig. 2, changes from Dyna-SR in blue).

Unlike prioritized sweeping, which only relies on reward prediction errors (PE), PARSR can prioritize replay using PE for either the SR or reward weights. When using the latter priority measure for PARSR we call this variant *Q-PARSR*, and when using successor prediction errors ($\|\boldsymbol{\delta}_M\|$), we refer to the variant as *M-PARSR*. Each algorithm represents different choices about what kinds of experience to prioritize, potentially leading to different behavior and training times.

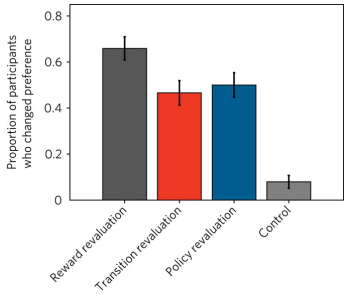
3 Experiments

We evaluate the performance of both variants of the PARSR algorithm in comparison to other SR-based and replay-based RL algorithms on revaluation tasks on different scales. Each experiment is evaluated over 10 seeds with $\varepsilon \in \{0.1, 0.3, 0.5, 1.0\}$. The Six States (resp. Six Rooms) experiment is performed for 100 (resp. 10) runs per seed, with 10 (resp. 1000) replay cycles per timestep for each algorithm. Note that PARSR has the same number of hyperparameters as other algorithms, so any improvement in human-like performance is not due to increased model complexity.

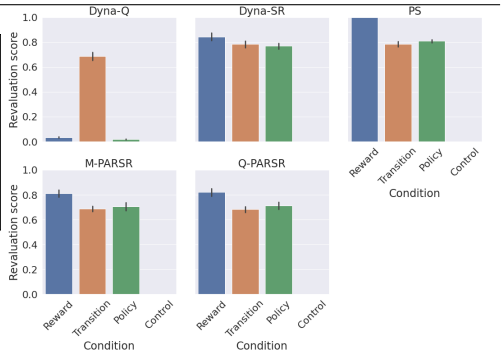
[1] We can extend this analysis to rewards defined on state-action pairs, $\boldsymbol{w}(s', a')$, and define our SR on 4-tuples as $M(a, s, a', s')$.



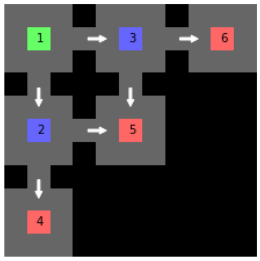
(a) Six States design.



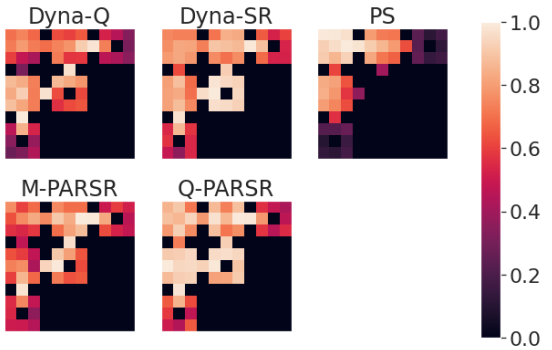
(b) Six States human results.



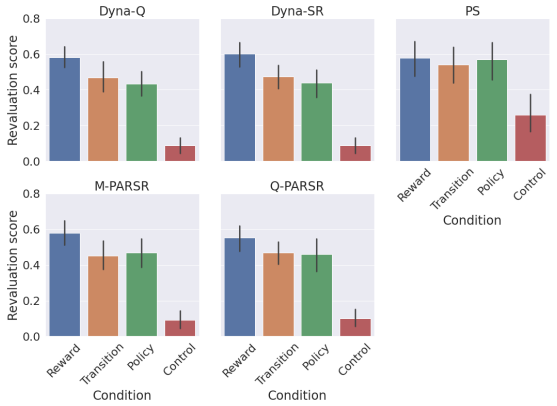
(c) Six States model results (10 replay cycles).



(d) Six Rooms (121 states) design.



(e) Six Rooms priority visualization, phase 2 of transition revaluation.



(f) Six Rooms model results (1000 replay cycles).

Figure 1: **Top row**: structure and results for the Six States experiment, reproduced from Experiment 2 in [1]. In (a), numbered circles denote different states, and arrows denote the unidirectional actions available at each state. For a given phase of a given condition, trials begin only in the earliest-stage states that are displayed in the figure for that condition and phase. (b) shows the proportion of participants in the human experiment ($n = 88$) who changed preference following the re-learning phase for each condition. Participants show greater ability to transfer in the case of reward revaluation than they do for the transition or policy conditions, though they are also capable of performing those tasks in some proportion. (c) reproduces these results for different algorithms. **Bottom row**: results for the Six Rooms experiment. (d) shows the map of the Six Rooms environment, with white arrows denoting "trapdoors" between rooms. (f) shows the revaluation scores for each model: all algorithms with the exception of PS attain results that are analogous to those achieved by human participants in Experiment 2 of [1] (Six States). (e) shows the relative frequency of the states prioritized during replay for the transition revaluation condition during phase 2. *M-PARSR has a much narrower focus on the bottlenecks between the rooms at which the revaluation is taking place, whereas Q-PARSR has a more uniform distribution over the prioritized states despite nonetheless employing a priority queue.* Both achieve similar performance in these tasks in spite of these differences.

3.1 Six States

We first reproduce and extend the results of Experiment 2 of [1], which we refer to as the "Six States" experiment. In this decision-making task (in which, unlike Experiment 1, the participant takes actions at every step), participants complete four games, each corresponding to a different experimental condition (Fig. 1a). The six states of the environment are structured as a unidirectional, three-stage decision tree, where two actions are available at the first two stages and a scalar reward is received at the terminal states in the final stage.

Each task is divided into three phases: one must pass each phase three times consecutively in order to progress to the next phase. In phase 1, participants are trained on a specific reward and transition structure. In phase 2, a change in either the reward or transition structure is changed, and participants learn about the changed structure *without revisiting the starting state*. Hence, participants do not get to experience these new contingencies following an action taken from the first stage. In phase 3, participants perform a single test trial beginning from the starting state, with the revaluation score corresponding to the probability (over multiple experimental runs) that the participant changes their action in state 1 between the end of phase 1 and the single trial in phase 3. Results from a human study in [1] show a greater revaluation score for the reward condition than transition or policy conditions, and no significant difference in revaluation between the latter two. Revaluation in the control condition is significantly lower than all of these.

For both Experiments 1 and 2, we find that both variants of PARSR are able to capture the human revaluation behavior in the task equally as well as Dyna-SR.

3.2 Six Rooms

We wanted to investigate whether the findings from the Six States experiment scaled to tasks with larger state spaces. To test this, we designed Six Rooms, a gridworld analog with 121 states. In this environment, the states in the Six States experiment correspond to the centers of the six rooms, laid out in a similar structure to the smaller environment. Unidirectionality is enforced through one-way corridors between each room, in order to retain the solution structure of the smaller environment. Each of the conditions and phases of the Six States experiment can be defined analogously for the Six Rooms domain.

Despite their similar latent structure, the Six Rooms environment represents a greater challenge for the agents since moving between the rooms requires a sequence of *several* actions, thus requiring more updates to the agents' representations. To successfully pass each phase, therefore, requires not only that the agent navigates to the correct state given its starting state, but moreover that it do so using the shortest path. During phase 2, agents are initialized in the center states of the second-stage rooms and are required to navigate to center states of the correct final rooms using the quickest path. This matches the difficulty of the exclusion criteria across the four conditions.

Fig. 1f shows the results of the experiment for each algorithm. We observe that the revaluation scores for these tasks match that of the human performance on the Six States experiment for all but Priortized Sweeping and most faithfully by PARSR. This suggests that PARSR's performance scales to more complex tasks with a similar latent structure. This further offers the testable prediction that human behavior in the Six Rooms experiment should scale accordingly.

In Fig. 1e, we visualize the relative frequencies at which each state was prioritized by each algorithm during transition revaluation (phase 2). **We observe a difference between the two PARSR variants in their prioritization strategies: while M-PARSR prioritizes replaying bottleneck states at which the revaluation is occurring, Q-PARSR's prioritization focus is more diffuse.** Future work is required to test which conditions and tasks are best served by each prioritization scheme. For instance, adding meta-learning to PARSR could control which type of prioritization is appropriate depending on the task at hand.

4 Discussion and Future Work

We have introduced PARSR: an algorithm that combines successor representation based learning with novel and neurally plausible replay prioritization heuristics. PARSR's two variants prioritize experience replay using either representation-based or reward-based prediction errors. Both PARSR variants show human-like behavior on benchmark tasks with 6 states (Figs. 1a, 1b 1c) as well as as scaled tasks (the Six Rooms environment) with 121 states (Figs.1d, 1e, 1f). The latter offers novel predictions for human behavior in scaled experiments.

In future work we will extend PARSR beyond tabular environments, as a deep RL algorithm with function approximation, similar to that of Prioritized Experience Replay [13]. In addition to extending PARSR to deep learning and more complex environments, future work would investigate further the nature of the two prioritization signals in PARSR variants, e.g. to investigate sequence memory activations at given moments in the task [18]. Moreover, we have visualized how error signals determine the relative frequency of prioritized experiences (Fig. 1e). Future work is required to investigate which problems are better served by which prioritization schemes. One solution is a novel algorithm combining PARSR with meta-learning of a control parameter learned across tasks and environments that, given the problem at hand, determines which PE signal is appropriate for efficient replay prioritization.

References

[1] Ida Momennejad et al. "The successor representation in human reinforcement learning". In: *Nature Human Behaviour* 1.9 (2017), pp. 680–692.

[2] Ida Momennejad. "Learning structures: Predictive representations, replay, and generalization". In: *Current Opinion in Behavioral Sciences* 32 (2020), pp. 155–166.

[3] Lennart Wittkuhn et al. "Replay in minds and machines". In: *Neuroscience & Biobehavioral Review* 129 (2021), pp. 367–388.

[4] Andre Barreto et al. "Transfer in deep reinforcement learning using successor features and generalised policy improvement". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 501–510.

[5] Evan M Russek et al. "Predictive representations can link model-based reinforcement learning to model-free mechanisms". In: *PLoS Computational Biology* 13.9 (2017), e1005768.

[6] Peter Dayan. "Improving generalization for temporal difference learning: The successor representation". In: *Neural Computation* 5.4 (1993), pp. 613–624.

[7] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. "The hippocampus as a predictive map". In: *Nature Neuroscience* 20.11 (2017), pp. 1643–1653.

[8] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[9] M.J Kahana, M.W Howard, and S.M Polyn. *2.26 - Associative Retrieval Processes in Episodic Memory*. Elsevier, 2008.

[10] Ida Momennejad et al. "Offline replay supports planning in human reinforcement learning". In: *Elife* 7 (2018), e32548.

[11] Andrew W Moore and Christopher G Atkeson. "Prioritized sweeping: Reinforcement learning with less data and less time". In: *Machine Learning* 13.1 (1993), pp. 103–130.

[12] Jing Peng and Ronald G Williams. "Efficient learning and planning within the Dyna framework". In: *Adaptive Behavior* 1.4 (1993), pp. 437–454.

[13] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[14] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[15] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. *Prioritized Level Replay*. 2021. arXiv: 2010.03934 [cs.LG].

[16] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8.3 (1992), pp. 293–321.

[17] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

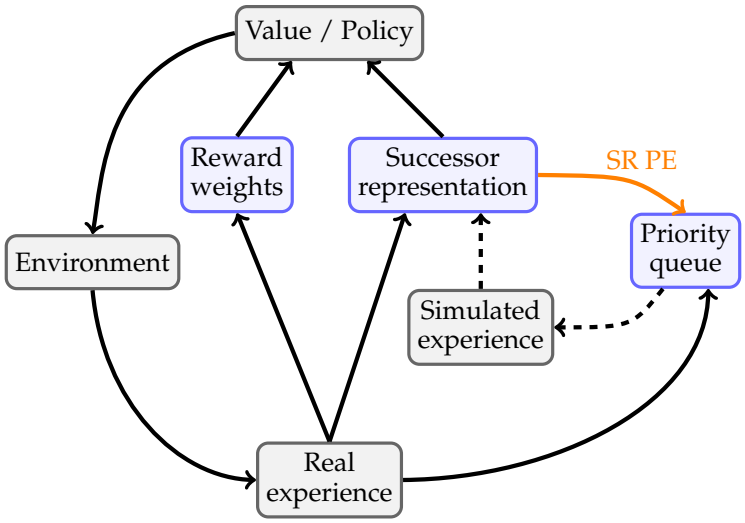
[18] Marcelo G Mattar and Nathaniel D Daw. "Prioritized memory access explains planning and hippocampal replay". In: *Nature Neuroscience* 21.11 (2018), pp. 1609–1617.

[19] Samuel J Gershman. "The successor representation: its computational logic and neural substrates". In: *Journal of Neuroscience* 38.33 (2018), pp. 7193–7200.

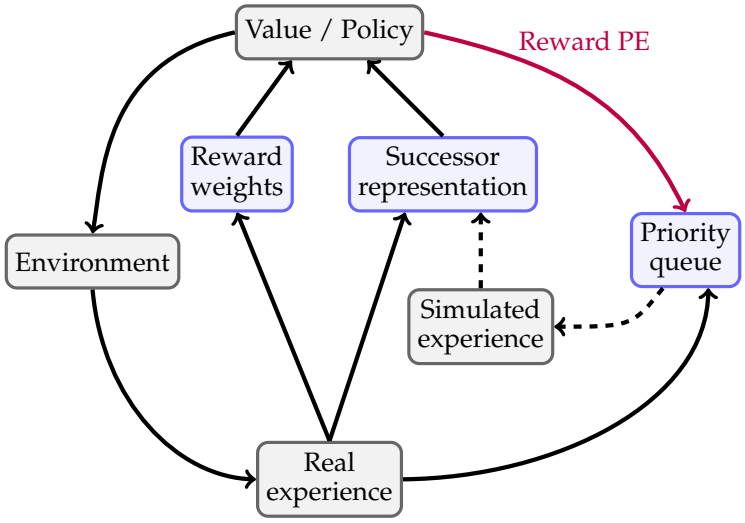
[20] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[21] Edward C Tolman. "Cognitive maps in rats and men". In: *Psychological Review* 55.4 (1948), pp. 189–208.

5 Appendix



(a) M-PARSR



(b) Q-PARSR

Figure 2: Flowcharts for both variants of the PARSR algorithm. Components of the agent are in blue. **Real experience** (s, a, r, s') is fed into the priority queue, and used to update the **reward weights** $\boldsymbol{w}(s')$ and **successor representation** $M(a, s, s')$ via temporal difference learning. The **priority queue** returns **simulated experiences** $(\overline{s}, \overline{a}, \overline{r}, \overline{s}')$, which are used to provide further updates to the successor representation. M-PARSR (a) prioritizes according to the successor representation prediction error (SR PE), whereas Q-PARSR (b) prioritizes according to the reward prediction error (Reward PE). The reward weights and successor representation determine the state-action **value** function $Q(s, a) = \sum_{s'} M(a, s, s')\boldsymbol{w}(s')$, which in turn determines the **policy** for acting in the **environment**.

PAE-POMDP: POMDP with Prolonged Action Effects

Sumana Basu
School of Computer Science
McGill University (Mila)
sumana.basu@mail.mcgill.ca

Marc-André Legault
School of Computer Science
McGill University (Mila)
marc-andre.legault@mila.quebec

Adriana Romero
School of Computer Science
McGill University
adriana.romsor@gmail.com

Doina Precup
School of Computer Science
McGill University & Mila
dprecup@cs.mcgill.ca

Abstract

In this paper, we identify *PAE-POMDP*, a subclass of Partially Observable Markov Decision Processes (POMDPs) in which the Markov assumption is broken due to the fact that actions have prolonged effects, often proportional to action amplitude. This occurs in many practical scenarios involving homeostatic control, such as regulating blood glucose levels by administering insulin. In this case, for example, the effect of administering a dose of medication is felt over several hours.

We propose a simple approach to converting PAE-POMDPs into MDPs, enabling the use of existing RL algorithms to solve such problems, without the need for explicit recurrence in the function approximation. We designed a simple toy environment which allows us to define prolonged action effects for discrete and continuous action spaces. We demonstrate the performance of our approach on this toy environment.

Keywords: Reinforcement Learning, POMDP, Prolonged action effect

Acknowledgements

Sumana Basu has been suported by an IVADO PhD fellowship. We are grateful to Mila IDT for their extraordinary IT support, and Emmanuel Bengio for the productive discussions during his Mila Office Hours.

1 Introduction

The formal paradigm for sequential decision making using Reinforcement Learning (RL) is the Markov Decision Process (MDP), which assumes that the agent's observations contain sufficient information to model both the immediate rewards and the transition distribution to the next state. Real life problems frequently violate this assumption, due to some sources of information being unavailable, or past information being relevant, resulting in partial observability. In this setting, observations indirectly provide information about the state. Such problems are formalized as Partially Observable Markov Decision Processes (POMDPs). Partial observability can be due to incomplete state information, temporarily available information, perceptual aliasing and noise [1, 6].

A general model-free approach to solving POMDPs, regardless of the nature of partial observability, is to learn a policy represented by a recurrent network [6]. While this approach handles the non-Markovian nature of the environment, training the hidden state representation has its own challenges, especially in environments in which trajectories are very long. An alternative approach is to use domain information in order to capture relevant information from the history in hand-crafted features, eliminating the need for computationally expensive training of a recurrent policy. For example, stacking frames from videos can capture information such as object velocity, which is not readily available in single frames [2].

In this paper, we define a special type of POMDP, in which the process of defining good features and training can be systematic and also less onerous computationally than in the general case. In particular, we study the case in which the Markovian property is lost specifically because actions have a prolonged effect on the environment, beyond the time step at which they were taken. We call this POMDP subclass *Prolonged Action Effect POMDP (PAE-POMDP)*.

An example of PAE-POMDP is blood glucose regulation with injections of insulin. Once insulin is released in the bloodstream, it will have a lasting effect on blood glucose until it is degraded. The magnitude and duration of the effect of a fixed amount of insulin depends on idiosyncratic differences in individual metabolism (for example, insulin sensitivity or degradation rate). In this setting, the insulin to be administered at the next step should be not only a function of the current blood glucose, but also of the unobserved concentration of active insulin in the bloodstream. Note that the entire insulin history is not relevant, only the residual active insulin.

While the problem of controlling a PAE-POMDP could be solved by usual POMDP means, more efficient solution methods can be used if we are willing to make some assumptions on the duration of action effects and their interaction. We show that for the framework that we define, a PAE-POMDP can be converted into an MDP with a modified state space, so all standard RL solution methods are applicable (instead of requiring recurrent networks).

In order to study this class of problems and associated solution methods in a clean setting, we define a small environment, which is a PAE-POMDP and can be instantiated with both discrete and continuous actions. We show that our approach of converting the problem into an MDP and then solving it is successful in this toy environment. The environment itself should be a useful benchmark for future work in this area.

2 Related Work

Non-Markovian processes are commonplace in real-world applications of RL. A promising research approach for dealing with this problem in the model-free RL has been to learn a flexible representation of the full observed history, which can both account for deviations from the Markovian property and be beneficial under partial observability. The vast majority of methods use recurrent networks to learn feature representation from a history of the observed state and/or actions. The policy and/or value functions are then defined on top of these features. For example, the Deep Recurrent Q-Network (DRQN) uses a Long Short-Term Memory (LSTM) architecture to integrate arbitrarily long histories of observations, in order to estimate a policy robust to partial observability in the setting where Deep Q-learning would be used otherwise [4]. One improvement over this model, described by Zhu et al. [10], is to include previous actions, in addition to the observations, in the LSTM that serves as input to the Q-Network. Extensions to actor-critic algorithms have also been proposed. For instance, the Deterministic Policy Gradient (DPG) algorithm has been augmented with LSTMs of previous states and actions to yield Recurrent DPG [1]. Further work building on the DPG framework addresses the overestimation problem in actor-critic methods by using Twin Delayed Deep DPG [5]. In summary, most methods to address partial observability in deep model-free RL have included recurrent networks with memory to represent the observation and action history. Here, we will use a complementary, memoryless and computationally simpler approach focusing on the specific case where actions have a prolonged effect on the environment. This is similar to the memoryless approach taken by Jonker et al. [9] to handle control delay in real systems. They showed that a memoryless algorithm designed using knowledge of the source of partial observability (the delay) and its length outperforms SARSA(λ) without adding any further computational complexity.

3 Background

The typical formal framework for reinforcement learning (RL) is the Markov Decision Process (MDP). A MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, r, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is a (finite) set of actions, $\mathcal{P}$ is the state transition probability $\mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a)$, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is the discount factor. As the name suggests, an MDP obeys the Markov assumption that future is in independent of the past given the present, which means that transitions and rewards depend only on the current state and action and not on the past history.

A Partially Observable Markov Decision Process (POMDP) is a 7-tuple $(\mathcal{S}, \mathcal{A}, r, \mathcal{P}, \Omega, \mathcal{O}, \gamma)$, where Ω is the set of observations, $\mathcal{O}$ is the set of conditional observation probabilities, $O(\omega|s)$, also known as the emission function, and the rest of the elements are the same as in an MDP. In a POMDP, the agent does not have direct access to the identity of the states, instead needing to infer them through the observations.

While in a MDP, an agent which aims to act optimally with respect to the expected long-term return only needs to consider Markovian policies, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, in a POMDP, policies need to either rely on the entire history of action and observations, or to infer the hidden state from this history. In recent work, recurrent networks have become the standard for implementing such policies. We call recurrent policy a mapping $\pi : \mathcal{T} \times \mathcal{A} \to [0, 1]$ where $\mathcal{T}$ is the space of trajectories $\tau = \{(o_t, a_t, r_t)\}_{t=0}^{T}$ of up to T time steps.

4 PAE-POMDP

In this section we introduce PAE-POMDP, a subclass of POMDPs in which an action's effect lasts more than one time step. This statement is a kind of forward view. From a backward view perspective, the state at a time step t is constructed by super-imposing the effects of the actions from several preceding time steps. From the perspective of the agent, this means that it needs to keep track of the history of actions over a preceding period of time. Note that action effects that are simply delayed by a certain amount of time [9, 8] are a special case of this setup .

In general, this problem is no simpler than a regular POMDP, as an agent that keeps track of its history may need to remember all the actions taken since the beginning of time. However, we are going to consider a more circumscribed problem formulation which is relevant for situations such as administering medications. Specifically, we are going to use a decay assumption inspired from the rate laws in chemical kinetics [7]. The rate law is a formal expression defining the relationship between the forward rate of a chemical reaction and the initial concentration of the compounds. The integrated rate law for a first order reaction depending on the concentration of a single reactant is defined as:

$$ln[A] = -kt + ln[A]_0$$

where $[A]_0$ is the initial concentration, $[A]$ is the concentration at time t and k is the rate constant. This type of law governs for example insulin concentration which can be used to control diabetes [3]. This kind of natural phenomenon is common in natural dynamical systems (e.g. heat dissipation or radioactive decay). If we consider the initial concentration as an action, this equation can be viewed as describing the evolution of this action's effect over time. By exponentiating both sides, we can see that the future effect of an action is proportional to its initial magnitude, and it decays exponentially over time. Hence, we will define a PAE-POMDP using these modelling assumptions.

One further specification relates to the way in which action effects compose with each other. For simplicity, we will assume that action effects are additive and independent of each other, which means that we do not need to model interactions.

More precisely, suppose the action a_t taken at time step t continues to change the future states of the environment for $\kappa \in \mathbb{Z}^+$ time steps. The value of κ is environment specific and also depends on the amplitude of the action a_t. We will also assume that the initial action effect will decay at a constant rate λ, which is specific to the environment. Formally, we assume that $a_{>t} = \lambda a_t$. Note κ could e.g. be defined as the time interval necessary for a_t's effect to fall below a threshold. Here we implicitly assume that in such a case, the action's effects can safely be ignored thereafter.

The observation of the agent at a particular time step can therefore be defined conditional on the current state and all the past actions: $o_{t+1} \sim \mathcal{O}(.|s_t, a_{<t})$. Note that although this is a generalized notation, not all the actions $a_{<t}$ that occurs before t have an effect on o_{t+1} (only those within a κ window). Moreover, note that if action effects are additive, this conditional probability distribution could be modelled as a sum of effects due to the actions. This assumption is less important but lead to further computational gains (which we would like to investigate in future work).

5 Transforming PAE-POMDP into an MDP

If the amplitude of an action decays by a factor of λ for each of the κ time steps when it stays active, then at any timestep t, the environment state is a function of a subsequence of the previous actions $\{a_{\leq t}\}$. We call this the *effective action* $a_{\mathcal{E}}$,

which is given by::

$$a_{t_\mathcal{E}} = \begin{cases} 0 & \text{if } t = 0 \\ \lambda a_{(t-1)_\mathcal{E}} + a_{t-1} & \text{otherwise} \end{cases} \tag{1}$$

If we augment the state space of the original problem $\mathcal{M}$ with the effective actions, such that $s_{t_\mathcal{E}} = (s_t, a_{t_\mathcal{E}})$, the Markov assumption is once again restored and the revised MDP $\mathcal{M}_\mathcal{E} = (\mathcal{S}_\mathcal{E}, \mathcal{A}, r, \mathcal{P}, \gamma)$ can be solved as a proxy of the original MDP $\mathcal{M}$. Note that $\mathcal{M}_\mathcal{E}$ and $\mathcal{M}$ only differ in the state space. Since we have access to $\mathcal{M}_\mathcal{E}$ we can use traditional RL algorithms to solve it, including Q-learning, actor-critic etc.

In the experiments below, we will use Q-learning on the constructed MDP as a simple, memory-less baseline for solving PAE-POMDP. We call this the *effective baseline*, a simple yet valuable way to compute a policy that avoids computationally costly alternatives.

6 Experiments

6.1 Environment

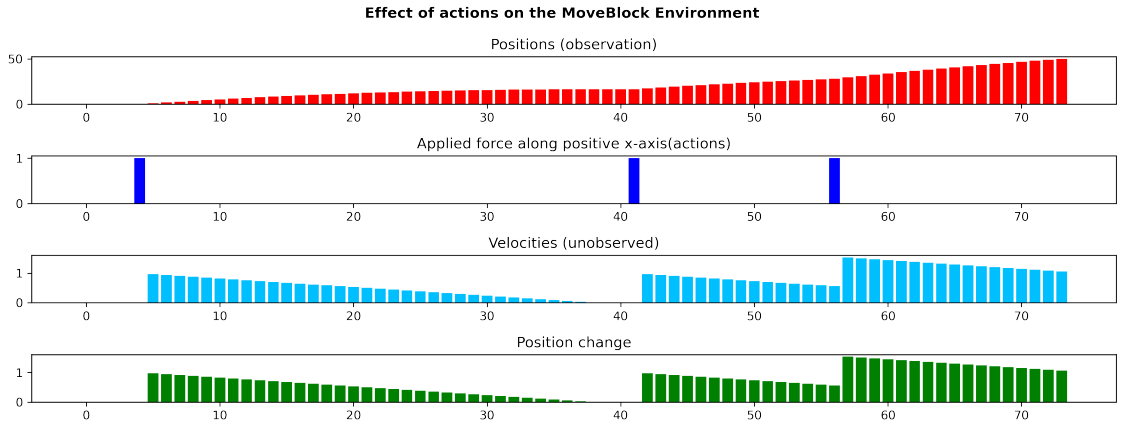
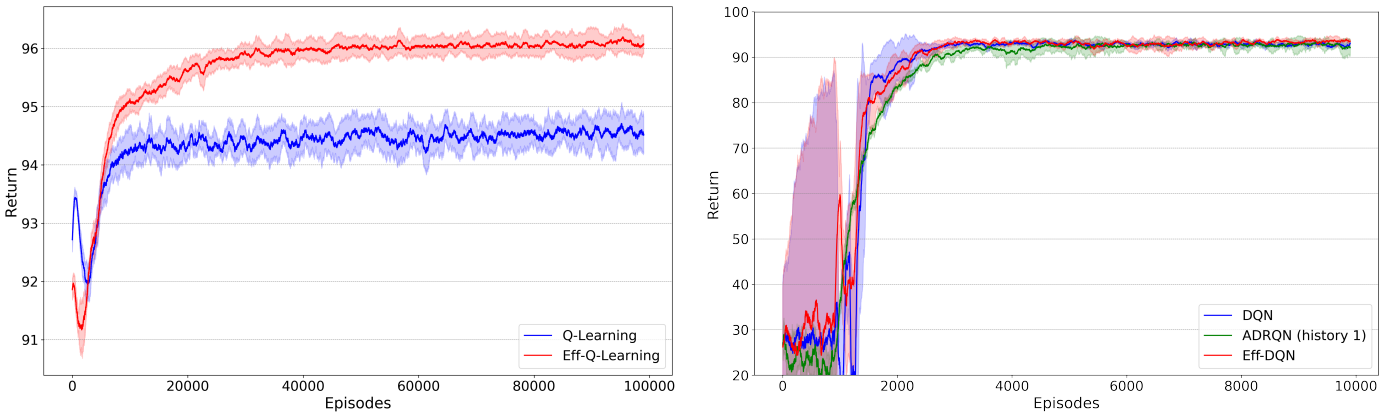


Figure 1: **Prolonged and Additive Action Effect.** *Prolonged Effect:* Position (observed variable) does not change initially in the absence of force (action). Unit force is exerted at t=3 followed by no force till t=40. But since velocity (unobserved variable) keeps on changing under the previously exerted force until friction brings it to zero, position keeps on changing too in the absence of any further force. *Additive Effect:* Forces exerted at t=41 and t=57 exhibit additive nature.

Although we were motivated in defining PAE-POMDP by the control system of glucose regulation with insulin injections, that problem has other complexities and sources of partial observability hampering the assessment of our approach. Hence, we created a toy environment, *MoveBlock*, where the only violation of Markovian property comes from the prolonged effect of actions (Figure 1). The task is to move a block from its initial position to a final position on a slippery surface with minimum possible effort. The only control available is force along the horizontal axis in the direction of the goal. Due to the slippery nature of the floor, an exerted force will cause the object to move until it is stopped by friction, prolonging the effect of actions over multiple timesteps. The reward system is similar to the Mountain Car environment, where there is a penalty for every action proportional to its amplitude and a high positive reward for reaching the destination. The discrete action space version has 11 action choices between [0, 10]. The observation is the position (continuous) while the velocity remains unobserved to the agent. While a random policy will solve the task, the optimal policy depends on the action history. An example of the action effects is depicted in Fig. 1.

6.2 Results and Discussion

We discretized the continuous position values and clipped the effective action value to the max action value. A tabular Q-learning agent trained with ϵ-greedy exploration on this discrete domain serves as a baseline. As shown in figure 2a, an effective baseline outperforming the Q-learning agent suggests it might be comparable to a recurrent baseline, even if not at par. In figure 2b, we compared a DQN agent with an ADRQN [4] agent and an Effective-DQN agent. In addition to the current state information, ADRQN takes into account the action history of a specific length. All the agents follow the same state, action representation and Q-network architecture of a single hidden layer. In ADRQN, the fully connected layer is replaced by an LSTM unit. Our results show that all the agents solve the task equally well. This performance is justified given the history based policy only needed a history of length 1 to solve it. So, this experiment only concludes that despite the presence of prolonged action effect, deep RL methods are capable of solving easy PAE-POMDP tasks



(a) Tabular results

(b) Function Approximation results

Figure 2: **Performance on MoveBlock.** (a) Move block for upto 50 units of distance. Tabular Q learning and our algorithm combined with Q learning. Mean and std deviation of performance over 10 seeds. Performance is smoothed with a window of size 1000. (b)Move block for upto 500 units of distance. Performance comparison of DQN, Effective-DQN and recurrent baseline ADRQN. Solid line is mean and shaded region is standard deviation over 5 seeds. All curves smoothed with a window of 100.

leveraging the representational capacity of neural networks. We need to compare the methods on a more difficult task, such as the glucose control environment to be able to comment on the competency of the method.

7 Conclusion and Future Work

We have identified the PAE-POMDP subclass, and shown that an effective baseline can be devised to solve it. Tabular results show that our approach is better than a Q-learning agent. However, we were unable to make any conclusion in the function approximation setting when compared against a recurrent baseline. Adding a diverse set of environments of varying difficulty level, such as a glucose controller, thermostat etc. where the actions are prolonged but the decay models are different would also be useful, especially as it would allow us to evaluate how well the stated decay assumption approximates real cases.

References

[1] Nicolas Heess et al. Memory-based control with recurrent neural networks. In *NIPS Deep Reinforcement Learning Workshop 2015*, 2015.

[2] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] E Ferrannini etc al. Pattern of insulin delivery after intravenous glucose injection in man and its relation to plasma glucose disappearance. *The Journal of Clinical Investigation*, 64(1):243–54, July 1979.

[4] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017.

[5] Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps, 2021.

[6] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps, 2022.

[7] Atkins Peter and de Paula Julio. The rates of chemical reactions. In *Atkins' Physical chemistry*, chapter 22, page 791–823. W.H. Freeman, New York, 8 edition, 2006.

[8] Simon Ramstedt, Yann Bouteiller, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays, 2021.

[9] Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3226–3231, 2010.

[10] Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. On improving deep reinforcement learning for pomdps, 2018.

Likelihood Approximation Networks enable fast estimation of generalized sequential sampling models as the choice rule in RL

Krishn Bera
Dept. of Cognitive, Linguistic & Psychological Sciences
Carney Institute for Brain Science
Brown University
krishn_bera@brown.edu

Alexander Fengler
Dept. of Cognitive, Linguistic & Psychological Sciences
Carney Institute for Brain Science
Brown University
alexander_fengler@brown.edu

Michael J. Frank
Dept. of Cognitive, Linguistic & Psychological Sciences
Carney Institute for Brain Science
Brown University
michael_frank@brown.edu

Abstract

Sequential sampling models (SSM) are a powerful class of models used to summarize cognitive process dynamics underlying decision-making in various task settings. In reinforcement learning (RL), while researchers typically assume a simple softmax choice rule, more recently, studies have used the drift diffusion model (DDM), a popular SSM to jointly model choice and response time distributions during learning (Pedersen, Frank, & Biele, 2017; Pedersen & Frank, 2020; Fontanesi, Gluth, Spektor, & Rieskamp, 2019). Such an approach allows researchers to study not only the across-trial dynamics of learning but the within-trial dynamics of choice processes, using a single model.

However, a practical problem in parameter estimation is the lack of closed-form likelihoods for a large class of models. Such intractable likelihoods render typical Bayesian inference methods infeasible. Alternative likelihood-free inference methods need to be invoked, which often tend to incur enormous computational costs, thereby limiting their application. To enable Bayesian estimation for a broad class of RL-SSM models, we leverage the recently developed Likelihood Approximation Networks (LAN) (Fengler, Govindarajan, Chen, & Frank, 2021). The LAN approach involves training neural networks that learn approximate likelihoods for arbitrary generative models, allowing fast posterior sampling with only a one-off cost for model simulations that are amortized for future inference. Once amortized, the approximate likelihoods can be used for tractable inference via MCMC across arbitrary experiment designs, while allowing a much larger class of SSMs to serve as the behavior generating mechanisms of reinforcement learning agents.

Using synthetic datasets, we show here, a proof of concept that this method can be utilized to estimate the true posterior parameter distributions for the RL-DDM. Furthermore, we show accurate parameter recovery in hierarchical settings. We conclude by proposing a LAN-based reinforcement learning extension to the widely used HDDM Python toolbox (Wiecki, Sofer, & Frank, 2013), which would allow us to leverage LANs for arbitrary RL-SSM models.

Keywords: Sequential Sampling Models, Reinforcement Learning, Parameter Estimation, Approximate Bayesian Computation

Acknowledgements

This work was funded by NIMH grant P50 MH 119467-01.

1 Introduction

Hybrid computational models, combining the conceptual ideas of sequential sampling of evidence with reinforcement learning models have greatly expanded the cognitive modeller's toolbox in recent years (Pedersen et al., 2017; Fontanesi et al., 2019). The main idea behind these models is to allow a reinforcement learning (RL) process to drive the trial-by-trial parameters of a sequential sampling model (SSM) such as the basic drift diffusion model (DDM) (Ratcliff & McKoon, 2008), to jointly capture reaction time and choice behavior in complex tasks which involve learning from feedback (see Figure 1). As a result, RL-SSM is a much more powerful and expressive class of models. It naturally lends itself for use in computational modeling of numerous cognitive tasks where the 'learning process' informs the 'decision-making process'.

The combination of these two classes of models however inherits a shortcoming which plagues the world of sequential sampling models to begin with. Great interest in model variants, such as models which move beyond 2-choice scenarios, models which exchange the noise distribution (Wieschen, Voss, & Radev, 2020) in the diffusion, models which include attractor dynamics (Usher & McClelland, 2001) as well as models which deal with non-constant evidence criteria (Cisek, Puskas, & El-Murr, 2009), is contrasted with empirical data analysis being mostly limited to basic versions of the drift diffusion model, which uses evidence criteria that are constant over time as well as a linear accumulation process perturbed by Gaussian noise. This state of affairs falls out of the simple analytic convenience provided by the basic DDM, for which fast-to-compute likelihood functions (Navarro & Fuss, 2009) exist to make inference tractable, while no such closed-form likelihood functions exist for most variations of interest.

Hence, while recent work has enabled the development of easy to use software to combine the DDM and RL (Pedersen & Frank, 2020; Fontanesi, 2021), the natural extension to a combination of RL with a larger class of SSMs has been hampered by the lack of easy to compute likelihoods. The main bottleneck is formed by the need for expensive model simulations during inference, making statistical inference for such models intractable for anyone but highly computationally sophisticated experts.

Recent advances in the field of likelihood-free inference (Papamakarios & Murray, 2016; Gutmann, Dutta, Kaski, & Corander, 2018; Papamakarios, Nalisnick, Rezende, Mohamed, & Lakshminarayanan, 2019; Papamakarios, Sterratt, & Murray, 2019; Lueckmann, Bassetto, Karaletsos, & Macke, 2019; Radev, Mertens, Voss, & Köthe, 2020; Fengler et al., 2021), provide a suite of new tools that utilize the power of deep learning to improve the computational efficiency of statistical inference in models which are accessible only via simulations (but lack a analytical likelihood function).

We leverage here one such tool, likelihood approximation networks (LANs) (Fengler et al., 2021), to bridge the gap towards the application of a wider class of SSMs conjointly with RL. In this preliminary work, we provide a proof of concept that LANs can be used fruitfully for this type of modelling.

In the 'Methods' section, we provide a brief description of the specific LAN that we employed and how it was trained. The next subsection outlines the test bed which we use to generate the synthetic dataset and run parameter recovery. The 'Results' section shows proof-of-concept parameter recovery and the results of our approach when applied to hierarchical settings. We conclude by highlighting the significance/benefits of the proposed method and outlining the future directions.

2 Method

2.1 Likelihood Approximation Network

We trained a multilayer perceptron with three layers to approximate the likelihoods, as per the procedures outlined in the LAN reference paper (Fengler et al., 2021). The resulting network takes the data and model parameters (features during training) as input and outputs trial-wise (approximate) log-likelihood values (labels, based on kernel density estimates of empirical likelihood functions). The specific network used in this article was trained on 3×10^5 parameter combinations (with 2×10^5 simulations of each run) of the drift-diffusion model using Pytorch (Paszke et al., 2019).

2.2 Test Bed

We test our method on synthetic datasets of the multi-armed bandit task. The experiment is a two-armed bandit task with binary outcomes. The model employed a simple delta learning rule (Rescorla, 1972) to update the action values

$$q_{action}(t+1) = q_{action}(t) + \alpha * [r(t) - q_{action}(t)],$$

where $q_{action}(t)$ denotes expected reward (Q-value) for the chosen action at time t, $r(t)$ denotes reward obtained at time t and α denotes the learning rate. As an initial benchmark to compare with analytical solution, we began with using a LAN for the vanilla two-choice drift-diffusion model with three parameters - boundary seperation (a), non-decision time

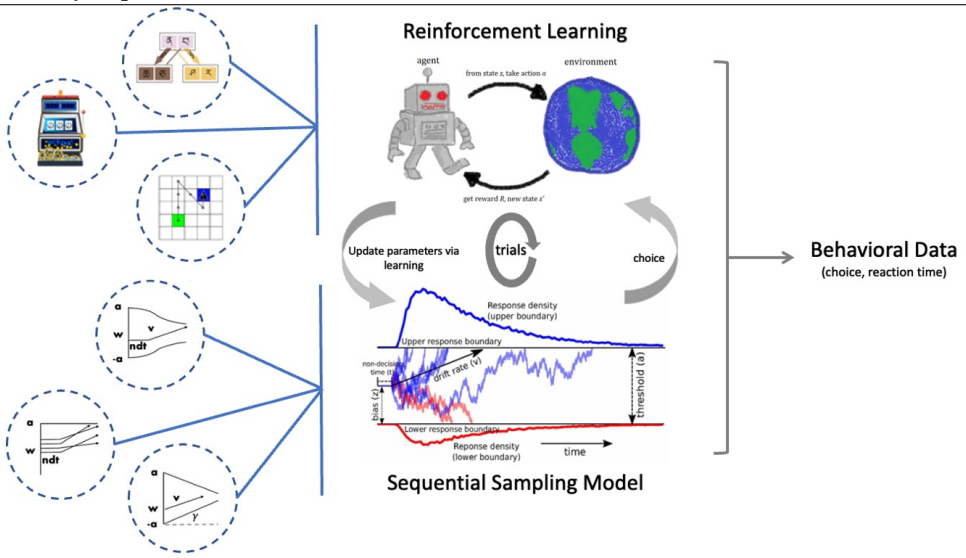


Figure 1: RLSSM - combining reinforcement learning and sequential sampling models.

(t) and drift rate (v). The trial-by-trial drift rate depends on the expected reward value learned by the RL rule. The drift rate is therefore a function of Q-value updates, and is computed by the following linking function

$$v = (q_{action1} - q_{action2}) * s,$$

where s is a scaling factor of the difference in Q-values. In other words, the scaler s is the drift rate when the difference between the Q-values of both the actions is exactly one.

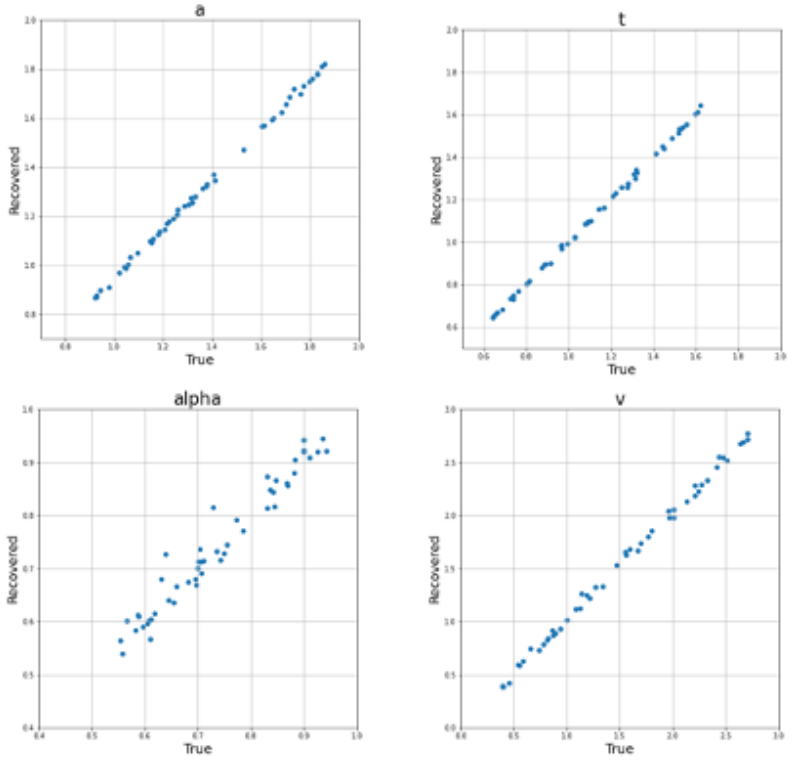


Figure 2: Parameter recovery on synthetic datasets. The true group mean parameter values (used to generate the datasets) are plotted on the x-axis. The recovered group mean parameter values are plotted on the y-axis. a is boundary separation, t is non-decision time, $alpha$ is the learning rate and v is the scaling factor for drift rate.

3 Results

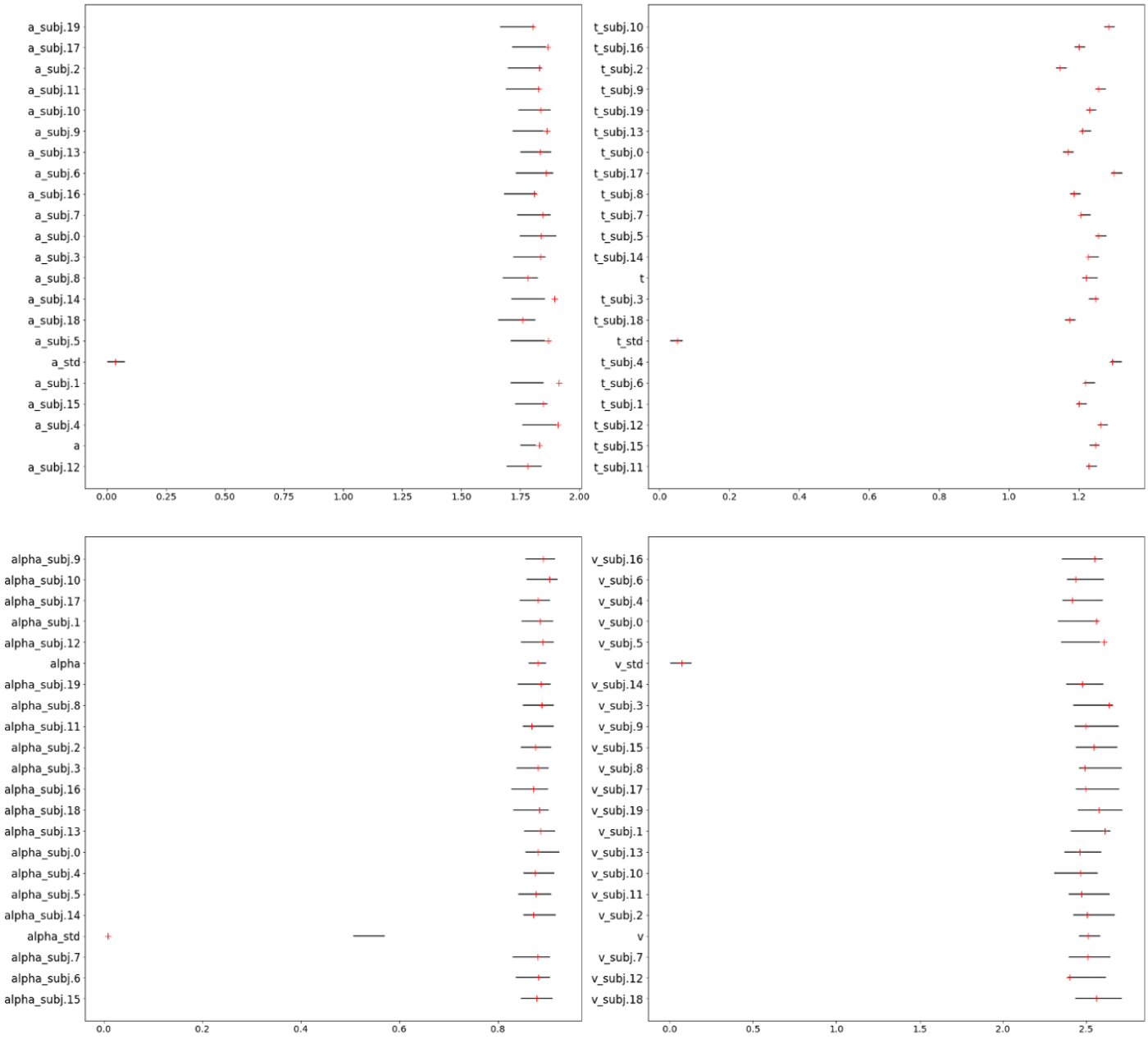


Figure 3: Hierarchical parameter recovery. Posterior distributions (denoted by caterpillar plots) of recovered parameters of a dataset with 20 subjects. The black line corresponds to 90% highest density interval. The ground truth parameter values are denoted by red crosses. Group mean and standard deviation are also plotted (for ex. see *a* and *a_std*).

3.1 Parameter Recovery

For parameter recovery experiments, 50 synthetic datasets were generated using the RL-DDM model. Each dataset contained 20 subjects with 500 simulated trials per subject. The mean and standard deviation of group parameters were sampled from a uniform distribution over a reasonable range of parameters. The subject means were sampled from

a truncated normal distribution, which was parameterized by group mean and standard deviation. The probabilities of reward for choosing the action corresponding to the upper and lower boundary were 0.8 and 0.2, respectively. To generate samples from the respective posterior distributions, we employed MCMC, specifically coordinate-wise slice sampling (Neal, 2003) as implemented in the HDDM toolbox (Wiecki et al., 2013). Figure 2 shows parameter recovery on the generated datasets.

3.2 Application to Hierarchical Settings

Figure 3 shows hierarchical parameter recovery on a sample dataset. We show that the inference method can be generalized to simultaneously estimate reinforcement learning parameters and decision parameters within a fully hierarchical Bayesian estimation framework. The LAN-based inference was able to accurately recover individual and group level parameters.

3.3 Posterior Predictive Checks

An important step in computational modeling is validating the model at hand. We check for model validity using posterior predictive checks, which involves simulating data using estimated parameters and comparing observed and simulated results. The simulated dataset was obtained by repeating the simulation process 500 times for each subject in a sample dataset. To evaluate the choice proportion for best option across learning for observed and simulated data, we bin the trials and plot 90% highest density intervals of the mean responses. Figure 4 shows mean response rate and reaction time densities in simulated and observed datasets.

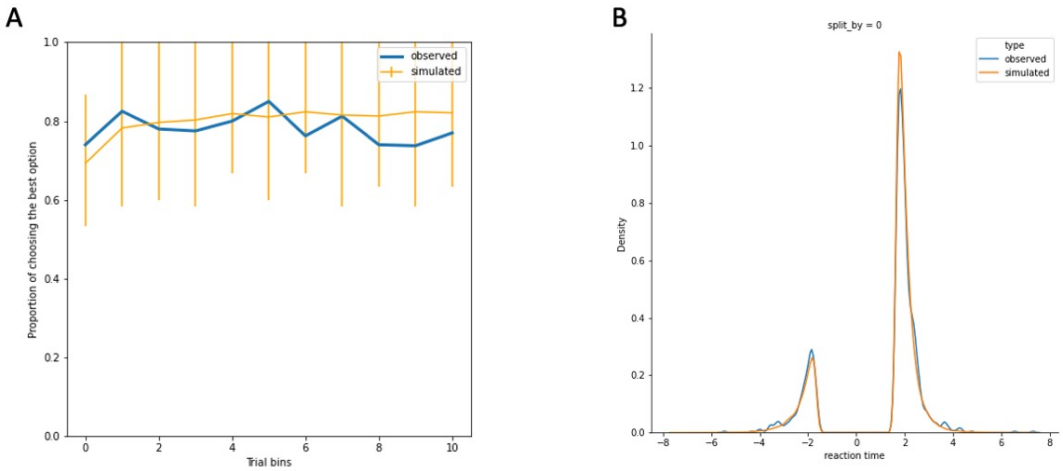


Figure 4: Posterior predictive checks. (A) Rate of choosing the best option across learning. Uncertainty in the generated data is captured by the 90% highest density interval of the means across simulated datasets. (B) Density plots of observed and predicted reaction time across conditions. RTs for lower boundary choices (i.e. worst option choices) are set to be negative (0-RT) to be able to separate upper and lower bound responses

4 Conclusion and Future Directions

The present work serves as a basic proof of concept for the joint application of LANs for the estimation of RL-SSM models. Given the encouraging results we plan on a number of extensions. Firstly, we incorporated LAN-based RL-SSM into the widely used HDDM python toolbox (Wiecki et al., 2013), a step towards easy community access. This addition to HDDM is also user-augmentable, so that researchers can test and ultimately share their own versions of RL-SSM models with the community through HDDM. And as a next step, it also naturally lends itself to testing hypotheses regarding how neural

dynamics correlate with learning or decision parameters, via HDDMnnRegression (Fengler et al., 2021). Second, we plan to test our approach on a much larger bank of models ourselves. In the context of RL-SSM, we can e.g. test n-choice multi-armed bandits, SSMs with dynamic decision bounds, SSMs with non-gaussian noise, opening the possibility for richer tests of theoretical models that can be informed by neural dynamics. Lastly, on the RL side, we will also expand the learning rules that can be immediately combined with SSMs, one goal being to determine how these improve or hinder parameter identifiability for various combinations of RL and SSM agents.

References

Cisek, P., Puskas, G. A., & El-Murr, S. (2009). Decisions in changing conditions: the urgency-gating model. *Journal of Neuroscience*, *29*(37), 11560–11571.

Fengler, A., Govindarajan, L. N., Chen, T., & Frank, M. J. (2021). Likelihood approximation networks (lans) for fast inference of simulation models in cognitive neuroscience. *Elife*, *10*, e65074.

Fontanesi, L. (2021, February). *laurafontanesi/rlssm: First dev release.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.4562217` doi: 10.5281/zenodo.4562217

Fontanesi, L., Gluth, S., Spektor, M. S., & Rieskamp, J. (2019). A reinforcement learning diffusion decision model for value-based decisions. *Psychonomic bulletin & review*, *26*(4), 1099–1121.

Gutmann, M. U., Dutta, R., Kaski, S., & Corander, J. (2018). Likelihood-free inference via classification. *Statistics and Computing*, *28*(2), 411–425.

Lueckmann, J.-M., Bassetto, G., Karaletsos, T., & Macke, J. H. (2019). Likelihood-free inference with emulator networks. In *Symposium on advances in approximate bayesian inference* (pp. 32–53).

Navarro, D. J., & Fuss, I. G. (2009). Fast and accurate calculations for first-passage times in wiener diffusion models. *Journal of mathematical psychology*, *53*(4), 222–230.

Neal, R. M. (2003). Slice sampling. *The Annals of Statistics*, *31*(3), 705 – 767. Retrieved from `https://doi.org/10.1214/aos/1056562461` doi: 10.1214/aos/1056562461

Papamakarios, G., & Murray, I. (2016). Fast ε-free inference of simulation models with bayesian conditional density estimation. In *Advances in neural information processing systems* (pp. 1028–1036).

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference. *arXiv preprint*, *arXiv:1912.02762*.

Papamakarios, G., Sterratt, D., & Murray, I. (2019). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd international conference on artificial intelligence and statistics* (pp. 837–848).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Pedersen, M. L., & Frank, M. J. (2020). Simultaneous hierarchical bayesian parameter estimation for reinforcement learning and drift diffusion models: a tutorial and links to neural data. *Computational Brain & Behavior*, *3*, 458–471.

Pedersen, M. L., Frank, M. J., & Biele, G. (2017). The drift diffusion model as the choice rule in reinforcement learning. *Psychonomic bulletin & review*, *24*(4), 1234–1251.

Radev, S. T., Mertens, U. K., Voss, A., & Köthe, U. (2020). Towards end-to-end likelihood-free inference with convolutional neural networks. *British Journal of Mathematical and Statistical Psychology*, *73*(1), 23–43.

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation*, *20*(4), 873–922.

Rescorla, R. A. (1972). A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Current research and theory*, 64–99.

Usher, M., & McClelland, J. L. (2001). The time course of perceptual choice: the leaky, competing accumulator model. *Psychological review*, *108*(3), 550.

Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). Hddm: Hierarchical bayesian estimation of the drift-diffusion model in python. *Frontiers in neuroinformatics*, *7*, 14.

Wieschen, E. M., Voss, A., & Radev, S. (2020). Jumping to conclusion? a lévy flight model of decision making. *The Quantitative Methods for Psychology*, *16*(2), 120–132.

Adaptive Online Value Function Approximation with Wavelets

Michael Beukman
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
michael.beukman1@students.wits.ac.za

Michael Mitchley
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
barcoded@gmail.com

Dean Wookey
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
wookey.dean@gmail.com

Steven James
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
steven.james@wits.ac.za

George Konidaris
Department of Computer Science
Brown University
Providence RI, 02912
gdk@cs.brown.edu

Abstract

Using function approximation to represent a value function is necessary for continuous and high-dimensional state spaces. Linear function approximation has desirable theoretical guarantees and often requires less compute and samples than neural networks, but most approaches suffer from an exponential growth in the number of functions as the dimensionality of the state space increases. In this work, we introduce the wavelet basis for reinforcement learning. Wavelets can effectively be used as a fixed basis and additionally provide the ability to adaptively refine the basis set as learning progresses, making it feasible to start with a minimal basis set. This adaptive method can either increase the granularity of the approximation at a point in state space, or add in interactions between different dimensions as necessary. We prove that wavelets are both necessary and sufficient if we wish to construct a function approximator that can be adaptively refined without loss of precision. We further demonstrate that a fixed wavelet basis set performs comparably against the high-performing Fourier basis on Mountain Car and Acrobot, and that the adaptive methods provide a convenient approach to addressing an oversized initial basis set, while demonstrating performance comparable to, or greater than, the fixed wavelet basis. To aid in reproducibility, we publicly release our source code.[1]

Keywords: reinforcement learning, value function, wavelets, linear function approximation

Acknowledgements

Computations were performed using High Performance Computing infrastructure provided by the Mathematical Sciences Support unit at the University of the Witwatersrand.

[1] https://github.com/Michael-Beukman/WaveletRL

1 Introduction

Representing a value function in reinforcement learning (RL) in continuous state spaces requires function approximation. One approach is non-linear function approximation, where a neural network is trained to learn useful features from raw observations. However, this class of methods requires many samples, large amounts of computation [1], and does not possess theoretical or convergence guarantees [2]. Another common scheme is *linear function approximation*, where the value function is approximated by a weighted sum of non-learnable basis functions. This results in simple algorithms and an error surface convex in the weights, but still allows for the representation of complex value functions because the basis functions themselves can be arbitrarily complex. The obvious question is then: which basis functions should one use?

Several fixed basis schemes have been introduced (e.g., [3–5]), all with the inherent disadvantage of a combinatorial explosion that makes them unsuited to high-dimensional domains. Consequently, research has focused on either constructing basis functions from data in both discrete and continuous state spaces, or selecting an appropriate set from a fixed dictionary of candidate basis functions with the aim of producing basis function sets that are subexponential in the number of dimensions. A third approach is *adaptive methods* (e.g., [6–9]), which are a hybrid of the above. Adaptive methods create new basis functions that complement or replace an existing set of basis functions on which learning is performed. They have the advantage in that they are online, which reduces the sample and computational complexity, and do not require a large dictionary of candidate functions, since they add representation complexity incrementally.

We extend existing adaptive methods to basis functions based on *wavelets*, which are able to approximate functions at various scales and locations, and which can be refined to build a more accurate representation where such detail is necessary. We further prove that wavelets are both necessary and sufficient for function splitting approaches. We present an algorithm for value function approximation that begins with a minimal set of basis functions, and only adds interactions between different dimensions as required. We test our approach on Mountain Car and Acrobot and our results show that a fixed wavelet basis is competitive with the high-performing Fourier basis [4]. Furthermore, the adaptive techniques perform comparably to the fixed basis, while not starting out with a complete basis set.

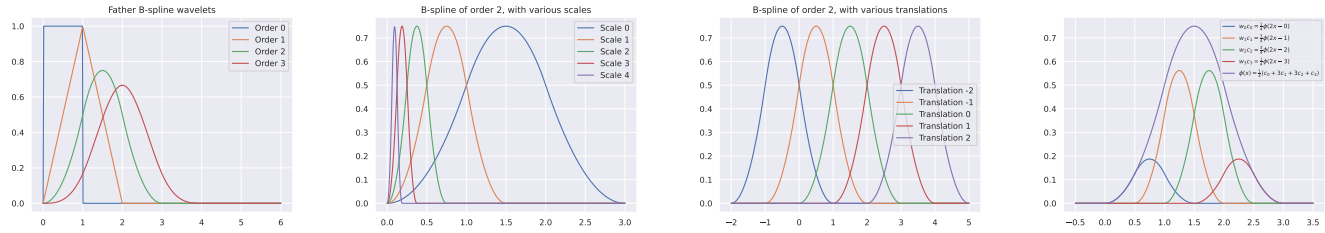
2 Value Function Approximation

The reinforcement learning problem in continuous domains is typically modelled as a Markov Decision Process and described by a tuple (S, A, P, R, γ), where $S \subseteq \mathbb{R}^d$ is a d-dimensional state space, A is a set of actions, $P(s'|s, a)$ describes the probability of transitioning to state s' after having performed action a in state s, $R(s, a)$ describes the reward received for executing such a transition, and γ is the discount factor. The agent is required to learn a policy π mapping states to actions that maximises the return (discounted future sum of rewards) at time t: $G_t = \sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i})$ [2]. Given a policy π, RL algorithms often estimate a *value function*: $V^\pi(s) = \mathbb{E}_\pi\big[\sum_{i=0}^{\infty} \gamma^i R(s_i, a_i)|s_0 = s\big]$, where action a is selected according to policy π. Linear value function approximation represents V^π as weighted sum of n basis function $\mathbf{\Phi}$: $V^\pi(s) \approx \mathbf{w} \cdot \mathbf{\Phi} = \sum_{i=1}^{n} w_i \phi_i(s)$. This approximation is linear in the components of the parameter (or weight) vector, $\mathbf{w}$, which results in simple update rules and a quadratic error surface.

Basis Functions In function approximation, a *basis* is a set of orthonormal functions spanning a function space, such that anything within that function space can be exactly reconstructed using a unique weighted sum of the basis functions. Typically, we are interested in orthonormal bases for $\mathcal{L}_2(\mathbb{R})$, the space of all finite square-integrable functions. In Euclidean spaces, $f(x) \in \mathcal{L}_2(\mathbb{R})$ implies $\int_{-\infty}^{\infty} f^2(x)dx$ is finite, a property satisfied by all value functions with finite return. Formally, if a basis is overcomplete (i.e. not all functions are orthogonal), then it is known as a *frame*.

A simple choice of basis function is *tile coding* [5], where the state space is discretised into tile basis functions that evaluate to 1 within a fixed region, and 0 elsewhere. Although tile coding forms a basis, it restricts the value function to be piecewise constant. Another approach is the *polynomial basis* [10], which sets $\phi_i(s) = \Pi_{j=1}^{d} s_j^{c_{i,j}}$, where each $c_{i,j} \in [0, \ldots, n]$ and n denotes the order of the basis. While orthonormal polynomials exist, simple ones are not, and may lead to redundancies in their representations. A more common scheme is *radial basis functions* (RBFs) [3], where each function is a Gaussian with a given mean and variance. RBFs have local support and are therefore well-suited to representing value functions with local discontinuities. The *Fourier basis* [4] uses Fourier series terms as basis functions, setting $\phi_i(s) = \cos(\pi \mathbf{c}_i \cdot s)$ where for order n, $\mathbf{c}_i = [c_1, \ldots, c_d]$ is a vector of coefficients between 0 and n. The Fourier basis requires choosing just a single parameter (the order) and in practice outperforms RBFs and polynomials on some common low-dimensional benchmarks [4].

The main shortcoming of these fixed-basis approaches is that the number of basis functions grows exponentially with the dimension of the state space. This has led to feature selection approaches that use a fixed basis as a feature dictionary, but only select a small subset of terms to compactly represent the value function. Ideally, such methods would allow the function approximator to represent extra detail where necessary and handle local discontinuities. However, the above basis function schemes are poorly suited to this: tile coding requires careful discretisation and can only represent piecewise-constant functions, RBFs lack an obvious way of setting their centres and variances, and the Fourier and



(a) Different orders (b) Scaling (c) Translating (d) Refineability

Figure 1: B-Spline wavelets of different (a) orders, (b) scales and (c) translations. In (d), the original function (purple) can be represented as a weighted sum of its "children" functions. For clarity, these functions are not normalised.

polynomial bases produce functions with global support. We therefore require a new basis scheme suitable for general function approximation that allows us to add spatially local basis functions to incrementally add detail to the value function representation.

3 Wavelets

Through the use of a *Fourier transform*, we can represent any periodic function as an integral of sines and cosines with varying frequencies. However, since sines and cosines have global support, the Fourier transform becomes cumbersome when representing *transient* or local phenomena [11]. To deal with this issue, we can instead consider *wavelets*. Wavelets are simply functions $\phi : \mathbb{R} \to \mathbb{R}$, some families of which are compactly supported [12] (nonzero in a finite interval) allowing us to model local phenomena. There are many types of wavelets but we focus on B-Spline father wavelets here, the first three orders of which are given by:

$$\phi^0(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases} \qquad \phi^1(x) = \begin{cases} x & 0 \le x \le 1 \\ 2 - x & 1 \le x \le 2 \\ 0 & \text{otherwise} \end{cases} \qquad \phi^2(x) = \begin{cases} 0.5x^2 & 0 \le x \le 1 \\ 0.75 - (x - 1.5)^2 & 1 \le x \le 2 \\ 0.5(x-3)^3 & 2 \le x \le 3 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In practice, these functions are normalised to satisfy $\int_{-\infty}^{\infty}(\phi(x))^2 dx = 1$. We note that the zero$^{\text{th}}$ order B-Spline function is also referred to as a Haar wavelet [13], and can produce functions equivalent to disjoint tile coding. We can scale (dilate) an arbitrary wavelet of order n by multiplying the input $x \in \mathbb{R}$ by 2^j to obtain $\phi^n_j(x) = \phi^n(2^j x)$. We can further translate it by $k \in \mathbb{Z}$ to obtain $\phi^n_{j,k}(x) = \phi^n(2^j x - k)$. Some examples are shown in Figure 1. One appealing property of wavelets is that they are *refinable*, meaning that they can be replaced with a weighed sum of smaller copies of themselves at regular intervals. Concretely, as shown in Figure 1d, they satisfy the *refinability equation*:

$$\phi(x) = \sum_k w_k \phi(mx - k) \text{ with } m > 1, k \in \mathbb{Z} \tag{2}$$

We now prove that wavelets are both necessary and sufficient as a basis that obeys this equation.

Theorem 1. *If one wishes to split a basis function $\phi(x) \in \mathcal{L}_2(\mathbb{R})$ (that is, the basis function is finite square-integrable) into a finite number of smaller copies of itself such that the value function remains unchanged, using Equation 2 where $m > 1$ and $k \in \mathbb{Z}$, with a finite number of w_k nonzero, it is necessary and sufficient to use a compactly supported father wavelet as ϕ.*

Proof. The above conditions define a father wavelet function [14]. Further, Equation 2 is obeyed by all father wavelet functions with refinement mask m [12]. If a father wavelet has compact support, it will have a finite dilation series. □

3.1 Wavelets as a basis for linear function approximation in reinforcement learning

To construct the wavelet basis for RL, given a state space $S \subseteq [0, 1]^d$, we choose a specific order n and scale j. For each dimension of the state space, we create wavelets with scale j, order n and translations $-n \le k < 2^j$. The full basis then consists of all combinations (products) of d of these functions, such that each term has a unique dimension. This is a fixed basis, providing a drop-in replacement for the methods described above. As an example, with $d = 2, n = 1, j = 0$, we have state $[s_1, s_2]$ and the atomic functions are $\{\phi(s_1 - 0), \phi(s_1 + 1), \phi(s_2 - 0), \phi(s_2 + 1)\}$. All pairwise products with distinct dimensions yield $\mathbf{\Phi} = [\phi(s_1 - 0)\phi(s_2 - 0), \phi(s_1 - 0)\phi(s_2 + 1), \phi(s_1 + 1)\phi(s_2 - 0), \phi(s_1 + 1)\phi(s_2 + 1)]$.

Adaptive Methods: This still has an exponential number of terms, however. To remedy this, we can start with a minimal basis set and add extra resolution *as and when necessary*. This can be done using two atomic operations: *splitting* and

combining. The former takes a feature ϕ and replaces it with its children, redistributing its weight among the children, such that the value function remains unchanged. This allows each child's weight to be updated individually, adding more representational capacity. In the above example, if we were to refine $\mathbf{\Phi}_1$ in dimension 1, we obtain the children of $\phi(s_1)$: $\phi(2s_1), \phi(2s_1 - 1), \phi(2s_1 - 2)$ and multiply each child with $\phi(s_2)$ to obtain 3 new functions, which would replace $\mathbf{\Phi}_1$. The second operation, *combining*, adds in products between different dimensions dynamically, instead of starting with a full combination. Intuitively, this allows us to start with a decoupled basis set (with $(n + 2^j)d$ terms as opposed to $(n + 2^j)^d$ for a fully coupled basis), where interactions between different state variables are ignored, and only added in when doing so would improve our approximation of the value function. For example, the initial basis set would simply be $\mathbf{\Phi} = [\phi(s_1), \phi(s_1 + 1), \phi(s_2), \phi(s_2 + 1)]$, and conjunctions could be subsequently added.

However, this raises a new question: *how do we choose which functions to split and which to combine?* One solution is to estimate the usefulness or *relevance* of a candidate function as we learn the value function. We first define $H(\phi, E) = \frac{T-1}{T}||\Omega||(1-\epsilon)\sum_{t=0}^{T}\epsilon^{(T-t)}E(s_t)\phi(s_t)$ where T is the sample count of ϕ (the number of times that ϕ was nonzero for a given state), $||\Omega||$ is the size of the domain in which ϕ is nonzero and ϵ is a hyperparameter which controls how much weight in the moving average is put on recent samples. We then extend the relevance measure of Geramifard et al. [8] to obtain the *relevance* $\rho(\phi) = H(\phi, E(s_t) = \delta(s_t))$ and the *observed error* $O(\phi) = H(\phi, E(s_t) = |\delta(s_t)|)$, where δ_t is the *TD error* $\delta_t = R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$. These quantities are estimates of the inner products $\langle\phi, \delta\rangle$ and $\langle\phi, |\delta|\rangle$, respectively. Intuitively, functions with high relevances will be frequently nonzero in the presence of error and will thus reduce this error when added to the basis set. Finally, we define $C(\phi) \doteq O(\phi) - \rho(\phi)$ for convenience.

Using the above ideas of *splitting*, *combining* and *relevance*, we first present AWR (Algorithm 1), in which we start with a full basis set at some scale and adaptively increase the scale for functions that have the highest C above some tolerance. This is similar to the method proposed by Li and Zhu [9] (who used Haar wavelets, although the theoretical analysis is more generally applicable), but we split based on the function relevance instead of using the coefficient magnitudes. The next algorithm is IBFDD (Algorithm 2), which starts with a decoupled basis set and adds in interactions (i.e. products) of different basis functions as the method progresses. Finally, we combine these two to obtain the Multiscale Adaptive Wavelet Basis (MAWB), which simply starts with a decoupled basis set, and alternates between IBFDD and AWR as learning progresses—either increasing the detail in a local region or adding in useful interactions.

Algorithm 1 AWR: Adaptive Wavelet Refinement

Require: Basis set $\mathbf{F}$, s, δ_t, $\rho(\phi)$, $T(\phi)$, splitting tolerance τ_s.
  for all Functions ϕ activated by state (s, a) **do**
    Update $\rho(\phi)$, $O(\phi)$, $T(\phi)$ & $\rho(\phi_c), T(\phi_c)$ $\forall$ children ϕ_c of ϕ.
  end for
  if $C(\phi) \geq \tau_s$ and $C(\phi) = \max_k(C(\phi_k))$ **then**
    Replace ϕ with its children in dimension d, $\phi^d_{j,k}$, where d maximises the average relevance of the children of ϕ.
  end if

Algorithm 2 IBFDD: Incremental Basis Function Dependency Discovery

Require: Basis set $\mathbf{F}$, s, δ_t, $\rho(\phi)$, $T(\phi)$, combination tolerance τ_c.
  for all $\phi_f = \phi_g\phi_h$ s.t. $\phi_g, \phi_h \in \mathbf{F}, \phi_f \notin \mathbf{F}$ and $\phi_f(s) \neq 0$ **do**
    Update $\rho(\phi_f), T(\phi_f)$
  end for
  if $|\rho(\phi_f)| \geq \tau_c$ and $|\rho(\phi_f)| = \max_k |\rho(\phi_k)|$ **then**
    $\mathbf{F} \leftarrow \mathbf{F} \cup \{\phi_f\}$
  end if

3.2 Preliminary Experiments

In Figure 2 we demonstrate the performance of the wavelet basis on two tasks using Sarsa(λ). We see that, for Mountain Car, the fixed B-spline basis with 36 terms in total is competitive with the Fourier basis with 36 terms. For Acrobot, most methods do well, and again, the B-Spline basis is competitive with the Fourier basis. The adaptive methods generally perform comparably to the fixed ones, while not needing to start with an exponentially sized basis set. We do note, however, that using a decoupled basis (which is equivalent to MAWB with $\tau_c, \tau_s = \infty$) often outperformed a coupled one, motivating the need to investigate more complex environments. Finally, we show example value functions in Figure 3.

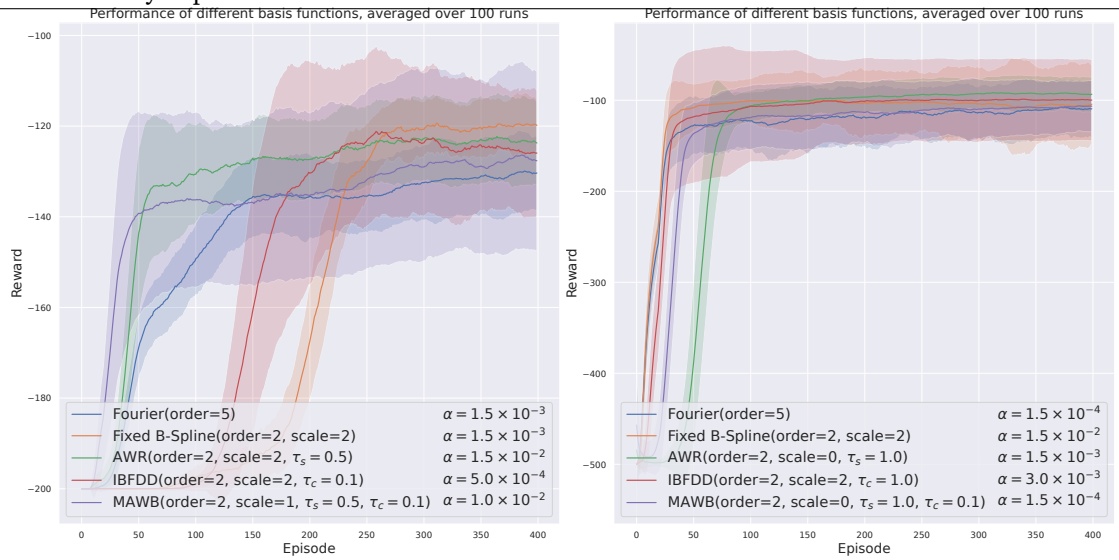
4 Conclusion

We introduced a linear function approximation scheme whose features are wavelet functions. We proved that the use of wavelets is both necessary and sufficient to obtain a refinable basis of $\mathcal{L}_2(\mathbb{R})$ that can perform multi-scale function approximation. Preliminary experimental results demonstrate that wavelets are competitive with other fixed-basis function approximation schemes.

References

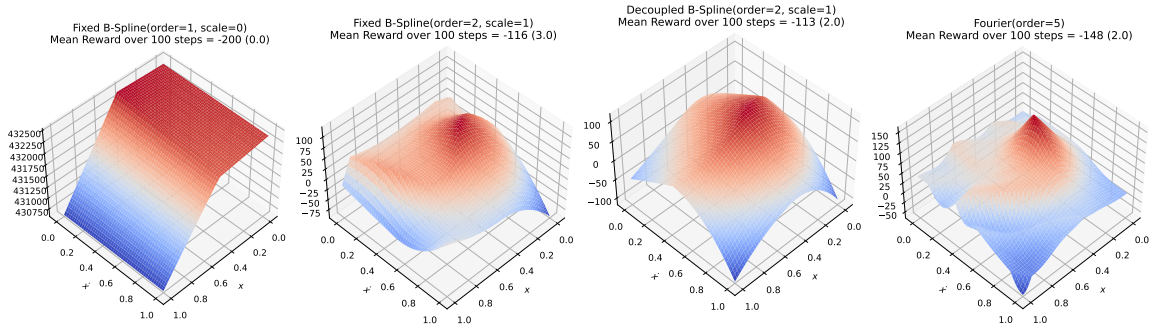
[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, 2017.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.



(a) Mountain Car (b) Acrobot

Figure 2: The performance of different basis functions on (a) Mountain Car and (b) Acrobot. α was chosen for each basis function based on a small grid search to maximise the mean reward over the last 100 episodes. We use Sarsa(λ) ($\gamma = 1, \lambda = 0.9, \epsilon = 0$) and we plot the average reward over the previous 20 episodes, with standard deviation shaded.



(a) Fixed B-Spline (b) Fixed B-Spline (c) Fixed B-Spline, Decoupled (d) Fourier

Figure 3: Negative value functions for Mountain Car showing mean reward (standard deviation) over 10 runs of 100 episodes with the shown value function being fixed. These were achieved after performing $10,000$ episodes with a small α. An insufficient scale as in (a) performs poorly, and the resulting value function in (b) is smoother than that of (d), while achieving improved results. The decoupled basis set also results in high performance, with a simple value surface.

[3] M. J. Powell, "Radial basis functions for multivariable interpolation: a review," *Algorithms for approximation*, 1987.

[4] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[5] J. S. Albus, "A theory of cerebellar function," *Mathematical biosciences*, vol. 10, no. 1-2, pp. 25–61, 1971.

[6] S. Whiteson, "Adaptive tile coding for value function approximation," Tech. Rep., 2007.

[7] S. Lin and R. Wright, "Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[8] A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. P. How, "Online discovery of feature dependencies," in *ICML*, 2011.

[9] T. Li and Q. Zhu, "On convergence rate of adaptive multiscale value function approximation for reinforcement learning," in *29th IEEE International Workshop on Machine Learning for Signal Processing, 2019*. IEEE, 2019, pp. 1–6.

[10] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[11] S. Mallat, *A wavelet tour of signal processing*. Elsevier, 1999.

[12] I. Daubechies, *Ten Lectures on Wavelets*. SIAM, 1992. [Online]. Available: https://doi.org/10.1137/1.9781611970104

[13] A. Haar, *Zur theorie der orthogonalen funktionensysteme*. Georg-August-Universitat, Gottingen., 1909.

[14] G. Strang, "Wavelets and dilation equations: A brief introduction," *SIAM review*, vol. 31, no. 4, pp. 614–627, 1989.

Hierarchical structure learning for perceptual decision making in visual motion perception

Johannes Bill
Department of Neurobiology
Harvard Medical School
Boston, MA 02115
johannes_bill@hms.harvard.edu

Samuel J. Gershman
Department of Psychology
Harvard University
Cambridge, MA 02138
gershman@fas.harvard.edu

Jan Drugowitsch
Department of Neurobiology
Harvard Medical School
Boston, MA 02115
jan_drugowitsch@hms.harvard.edu

Abstract

Successful behavior in the real world critically depends on discovering the latent structure behind the volatile inputs reaching our sensory system. Our brains face the online task of discovering structure at multiple timescales ranging from short-lived correlations, to the structure underlying a scene, to life-time learning of causal relations. Little is known about the mental and neural computations driving the brain's ability of online, multi-timescale structure inference. We studied these computations by the example of visual motion perception owing to the importance of structured motion for behavior. We propose online hierarchical Bayesian inference as a principled solution for how the brain might solve multi-timescale structure inference. We derive an online Expectation-Maximization algorithm that continually updates an estimate of a visual scene's underlying structure while using this inferred structure to organize incoming noisy velocity observations into meaningful, stable percepts. We show that the algorithm explains human percepts qualitatively and quantitatively for a diverse set of stimuli, covering classical psychophysics experiments, ambiguous motion scenes, and illusory motion displays. It explains experimental results of human motion structure classification with higher fidelity than a previous ideal observer-based model, and provides normative explanations for the origin of biased perception in motion direction repulsion experiments. To identify a scene's structure the algorithm recruits motion components from a set of frequently occurring features, such as global translation or grouping of stimuli. We demonstrate in computer simulations how these features can be learned online from experience. Finally, the algorithm affords a neural network implementation which shares properties with motion-sensitive cortical areas MT and MSTd and motivates a novel class of neuroscientific experiments to reveal the neural representations of latent structure.

Keywords: hierarchical structure, online learning, Bayesian inference, visual perception, biological neural network

Acknowledgements

The authors thank Anna Kutschireiter for valuable discussions and feedback on the theory. This research was supported by grants from the Harvard Brain Science Initiative (Collaborative Seed Grant, J.B., S.J.G. & J.D.), the Center for Brains, Minds, and Machines (CBMM; funded by NSF STC award CCF-1231216), and a James S. McDonnell Foundation Scholar Award for Understanding Human Cognition (Grant 220020462, J.B. & J.D.).

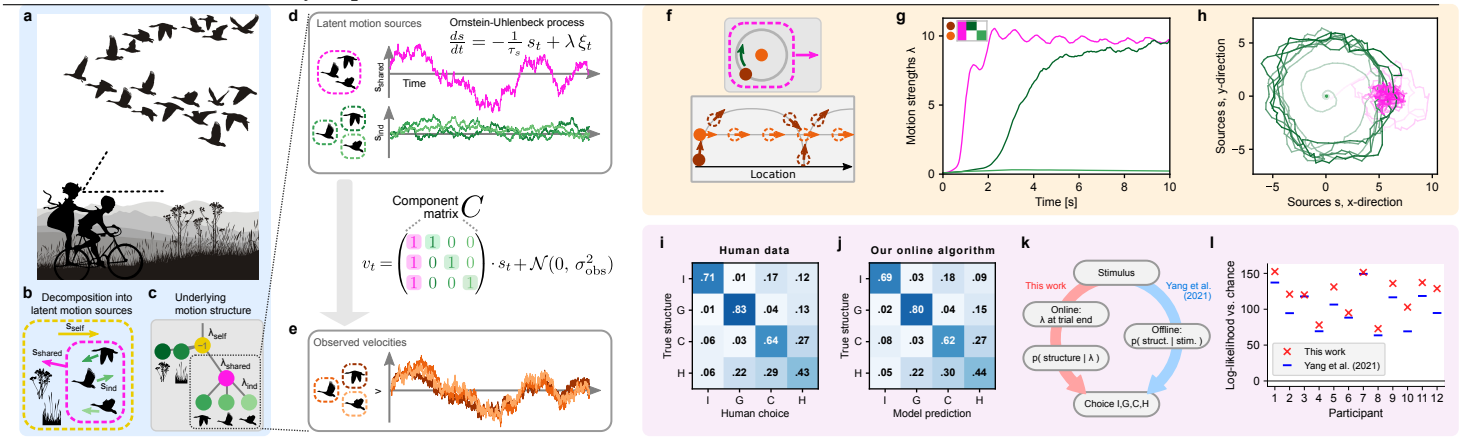


Figure 1: Explaining perception of structured motion as online hierarchical inference. ***(a – e)*** *Generative model of structured motion. Our online algorithm inverts the generative model to simultaneously identify the underlying structure and to decompose observed velocities into (latent) motion sources. See main text for details.* ***(f – h)*** *Demonstration of the algorithm by the example of the Duncker wheel.* ***(f)*** *The Duncker wheel is a rolling wheel with only two visible point lights at the hub and rim.* ***(g)*** *Like humans, the algorithm discovers a shared component for both lights (pink) and an individual component for the rim light (dark green).* ***(h)*** *The lights' velocities are decomposed into joint rightward motion and rim light-rotation. Brightness = time.* ***(i – l)*** *Classification task of ambiguous motion scenes, analyzing behavioral data from [9].* ***(i)*** *Human confusion matrix when classifying ambiguous motion scenes as Independent, Global, Clustered, and Hierarchically nested motion.* ***(j)*** *Confusion matrix of the algorithm on the same trials as [9]. The algorithm quantitatively explains human percepts and even captures the fine-structure in the confusion matrix.* ***(k)*** *For this, we fed the value of $\boldsymbol{\lambda}_t$ at trial end into a logistic regression classifier (trained on the ground truth, not human responses), and fitted the same choice model as [9], who had employed the ideal posterior on the full input trajectory.* ***(l)*** *Log-likelihood of human responses under both models. Our algorithm explains human responses better for every participant. The results in panels (j) and (l) are cross-validated.*

Introduction

Real-world scenes feature rich spatial and temporal structure. Understanding this structure allows humans and animals to make sense of their environment by organizing complex and often ambiguous sensory input streams into stable, meaningful percepts. The emerging compressed representations benefit goal-directed actions and decision making. While machine learning algorithms have been developed to infer the structure of large datasets offline [1], an understanding of how biological agents discover structure online is only beginning to emerge [2].

We studied online structure inference across multiple timescales by the example of visual motion perception. Motion structure, that is, statistical relations in velocities, carries essential information about the spatial and temporal evolution of the environment. For instance, the features composing an object typically move coherently, or self-motion adds optic flow to all observable velocities in a scene. To benefit behaviors such as navigation, tracking, prediction, and pursuit, the visual system must decompose observable velocities, $\boldsymbol{v}_t$, (see **Fig. 1a**) into their putative latent origins, e.g., the observer's self-, shared flock-, and each bird's individual motion (**Fig. 1b**).

Bayesian inference has provided a successful normative framework for understanding human visual motion perception in spatially constrained (local) patches [3, 4] and for simple structures [5, 6]. For structured motion spanning multiple objects, larger areas of the visual field, and longer timescales, however, a comprehensive theoretical description is only beginning to emerge. Recent work [7] has introduced tree structures for the mental organization of observed velocities into nested hierarchies, yet the inference process over structures had to be performed by a biologically unrealistic offline sampling algorithm. Theory-driven experiments have revealed that the human visual system makes use of hierarchical structure when solving visual tasks [8], and that aspects of motion structure perception can be explained by Bayesian structural inference [9], yet the algorithms underlying the structure discovery remained elusive.

We address the questions of how the visual system solves the chicken-and-egg problem of parsing motion in a scene in real time while simultaneously inferring the scene's underlying structure, and how the involved probabilistic computations can be performed by neural circuits. We propose an online Expectation-Maximization (EM) algorithm which leverages the fact that instantaneous motion (e.g., the speed and direction of flocking birds) and a scene's structure (e.g., the presence of a flock) evolve on different timescales. We demonstrate that the derived algorithm replicates a range of psychophysics experiments both qualitatively and quantitatively. Further, we present an implementation of the algorithm by recurrent neural networks, which feature connection and response properties of cortical areas implicated in visual motion processing, and propose a targeted experiment to test model predictions in neural recordings. Finally, we explore how the set of

typical motion components, such as shared motion or grouping, which the algorithm draws upon when explaining the structure of a scene, could be learned from experience on long timescales, rather than being given to it as a parameter.

Part of this work has been posted as a pre-print, `https://www.biorxiv.org/content/10.1101/2021.10.21.465346v1`, which presents the derivations and results in detail. The simulation results on online learning of the motion components are a new contribution.

Online hierarchical inference in a generative model of structured motion.

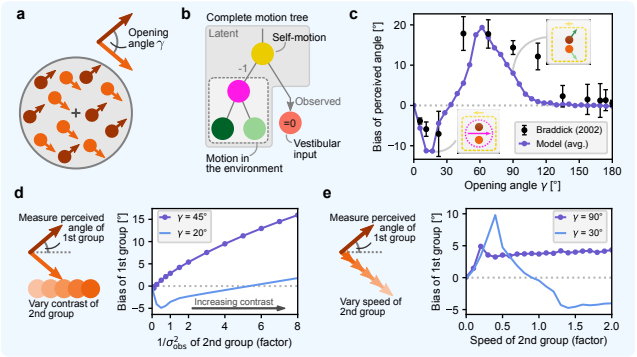
We build on the generative model of structured motion from [8] in which observable velocities, $\boldsymbol{v}_t$, are generated from latent causes, $\boldsymbol{s}_t$. The so-called *motion sources*, $\boldsymbol{s}_t$, can have volatile speed and direction, and usually respect a temporally more robust, tree-shaped *motion structure*: in **Fig. 1c**, graph connectivity defines how sources (nodes in the graph) affect observable objects (black object outlines), and vertical edge length, called *motion strength*, λ_m, indicates the long-term average speed of the associated source, s_m. Sources are a-priori assumed to evolve as Ornstein-Uhlenbeck processes in each spatial dimension (**Fig. 1d**, only 1 dim. shown) leading to stationary distributions, $s_m \sim \mathcal{N}\left(0,\, \frac{\tau_s}{2}\lambda_m^2\right)$. Observed velocities, $\boldsymbol{v}_t$, are noisy versions of the sum of all ancestral sources in the graph with graph connectivity represented by the *component matrix*, $\boldsymbol{C}$ (see **Fig. 1d & e** for illustration of 3 flocking birds). For most of the following, we assume that the component matrix, $\boldsymbol{C}$, is given and fixed, e.g., because it has been learned from experience.

We derived an online EM algorithm for simultaneously inferring estimates of the sources, $\boldsymbol{s}_t$, and of the underlying structure, $\boldsymbol{\lambda}_t$, from a stream of observations, $\boldsymbol{v}_t$ (assuming that time-constants and observation noise are fixed parameters). Note that, with the component matrix, $\boldsymbol{C}$, given, the motion structure is fully defined by $\boldsymbol{\lambda}$. The derivation exploits the different timescales of typical changes in $\boldsymbol{s}_t$ (volatile, E-step) and $\boldsymbol{\lambda}$ (stable, M-step). With mild approximations, we obtained:

$$\partial_t \boldsymbol{\lambda}_t^2 = -\tfrac{1}{\tau_\lambda}\,\boldsymbol{\lambda}_t^2 + \boldsymbol{\alpha} \odot \left(\boldsymbol{\mu}_t^2 + \boldsymbol{f}_\Sigma(\boldsymbol{\lambda}_t^2)\right) + \boldsymbol{\beta} \quad \text{with} \quad f_\Sigma(\lambda_m^2) = \frac{\sigma_{\text{obs}}^2}{\tau_s \,\|\boldsymbol{c}_m\|^2}\left(-1 + \sqrt{1 + \frac{{\tau_s}^2\,\|\boldsymbol{c}_m\|^2}{\sigma_{\text{obs}}^2}\,\lambda_m^2}\right), \tag{1}$$

$$\partial_t \boldsymbol{\mu}_t = -\tfrac{1}{\tau_s}\,\boldsymbol{\mu}_t + \boldsymbol{f}_\Sigma(\boldsymbol{\lambda}_t^2) \odot \boldsymbol{C}^\mathsf{T}\,\boldsymbol{\epsilon}_t \quad \text{with prediction error} \quad \boldsymbol{\epsilon}_t = \frac{\boldsymbol{v}_t}{\sigma_{\text{obs}}^2} - \frac{\boldsymbol{C}\,\boldsymbol{\mu}_t}{\sigma_{\text{obs}}^2}\,. \tag{2}$$

Here, $\boldsymbol{\mu}$ and $\boldsymbol{f}_\Sigma$ are the posterior parameters of $p(\,\boldsymbol{s}_t \,|\, \boldsymbol{v}_{0:t}\,) = \mathcal{N}\left(\boldsymbol{\mu}_t,\, \mathrm{diag}\left[\boldsymbol{f}_\Sigma(\boldsymbol{\lambda}_t^2)\right]\right)$, $\odot$ denotes elementwise multiplication, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ stem from a sparsity-inducing prior on $\boldsymbol{\lambda}^2$, $\|\boldsymbol{c}_m\|^2$ is the norm of $\boldsymbol{C}$'s mth column, and we require that $\tau_\lambda > \tau_s$. As common for online EM, eqn. (1) + (2) define a coupled dynamical system. Intuitively, the algorithm measures the range in which the motion sources vary, $\langle \boldsymbol{s}_t^2 \rangle$, to estimate the structure parameters, $\boldsymbol{\lambda}_t^2$, during the M-step in eqn. (1). At the same time, during the E-step in eqn. (2), the system's expected input, $\boldsymbol{C}\boldsymbol{\mu}_t$, leads to prediction errors, $\boldsymbol{\epsilon}_t$, which are projected into the domain of sources, $\boldsymbol{C}^\mathsf{T}\,\boldsymbol{\epsilon}_t$, and gated by $\boldsymbol{f}_\Sigma(\boldsymbol{\lambda}_t^2)$ for credit assignment. The E-step performs the velocity decomposition *conditioned on* the scene's structure which is inferred during the M-step. This is, to our knowledge, the first online inference model of Bayesian motion structure perception.



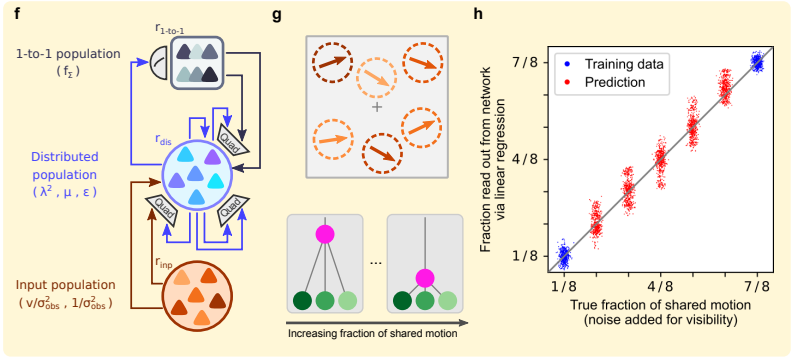


Figure 2: The model captures human motion direction repulsion and supports a neural implementation. *(**a–e**) Simulations of motion direction repulsion. **(a)** Motion direction repulsion is a systematic bias in the perceived angle between the motion of two groups of dots moving linearly in an aperture. **(b)** We endowed the model with self-, shared and group motion (yellow, pink, and green), as well as a noisy vestibular input signaling the observer's stationarity. **(c)** The algorithm replicates the biphasic opening angle-dependence of the bias measured by [10]. **(d & e)** We further make testable predictions for varying contrast (d) and speed (e) of the 2nd dot group. **(f)** Recurrent network model implementing the online algorithm. **(g)** Proposed experiment to measure neural representations of latent structure. Moving dots in several apertures follow our generative model (top). Different trials use different fractions of shared and individual motion (bottom). **(h)** The model predicts that the fraction of shared motion in the stimulus can be read out by a linear regression model (red points) which was only trained on a subset of trials (blue points).*

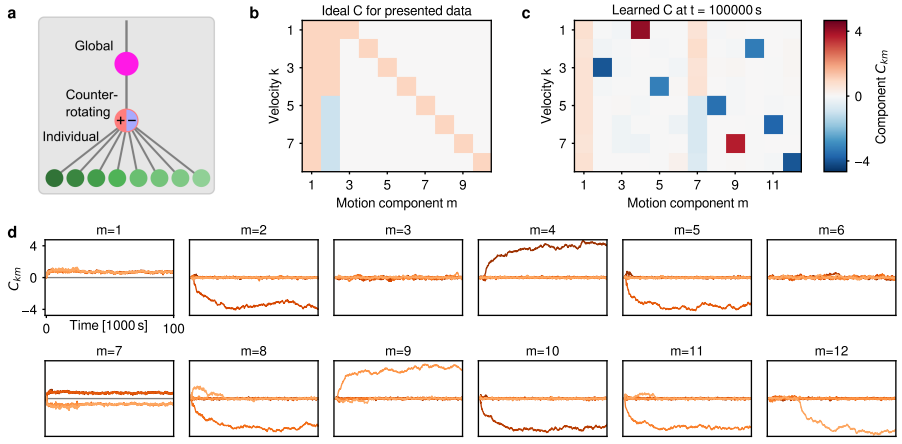


Figure 3: Learning of motion components from experience. ***(a)*** *Full structure of presented velocities. From the available features (global motion, counter-rotation, 8 individual motions) only a random subset is presented at any time.* ***(b)*** *Theoretically optimal $\boldsymbol{C}$-matrix for this stimulus.* ***(c)*** *Learned $\boldsymbol{C}$-matrix. All components have qualitatively been identified (note that the sign and m-order play no role). We gave the algorithm two more components in $\boldsymbol{C}$ than required. These extra components have, as expected, been left unused by the learning algorithm.* ***(d)*** *Time evolution of all 12 motion components during learning (one panel per component m; one line per velocity k).*

The algorithm explains experiments of human motion perception

Due to limited space, we here only present results on the Duncker wheel (**Fig. 1f – h**), motion structure classification of ambiguous scenes (**Fig. 1i – l**; data from [9]), and biased perception known as motion direction repulsion (**Fig. 2a – e**; data from [10]). How the algorithm replicates and explains these experiments is detailed in the figure captions.

Neural network model and proposed neuroscience experiment

Eqs. (1) and (2) rely on only linear and quadratic operations (by adding $\boldsymbol{\epsilon}_t$ as a represented auxiliary variable). Following a derivation similar to [11], who showed how up-to-quadratic dynamics of computational variables can be implemented by recurrent rate-based networks, we devised a network model with biologically realistic neural interactions to implement the inference algorithm. The network architecture is shown in **Fig. 2f**. The network represents (inferred) motion strengths, $\boldsymbol{\lambda}_t^2$, and motion source means, $\boldsymbol{\mu}_t$, in a linear population code. The only variable not fitting into this framework is the square-root function in f_Σ which is thus represented by a dedicated neural population. The full derivation of the model and a demonstration of its ability to implement the inference algorithm are provided in the bioRxiv preprint.

Motivated from the network model, we propose a new class of theory-driven neuroscience experiments to probe the neural representations of latent structure. In each trial, moving dots, which follow the generative model from **Fig. 1d&e**, are presented in several apertures, see **Fig. 2g (top)**. Different trials use different fractions of shared and individual motion, such that the expected dot speed, $\langle v^2 \rangle \propto \lambda^2_{\text{shared}} + \lambda^2_{\text{ind}}$, is held constant across trials, see **Fig. 2g (bottom)**. The network model encodes $\boldsymbol{\lambda}_t^2$ linearly in its neural activity, and thus predicts that the fraction of shared motion, defined as $\lambda^2_{\text{shared}} \,/\, (\lambda^2_{\text{shared}} + \lambda^2_{\text{ind}})$, can be read out by a linear regression model. As shown in **Fig. 3h**, a linear regression model trained on two fractions (blue points) correctly reads out also other fractions (red points).

Learning of common motion components on long timescales

Could the motion components, $\boldsymbol{C}$, be learned from observations in an unsupervised manner? We derived a learning rule for the matrix elements, C_{km}, through gradient-based online EM. Since learning of the motion components, $\boldsymbol{C}$, aims to maximize the data likelihood beyond what could be explained by the sources, $\boldsymbol{s}_t$, and the scenes' structures, $\boldsymbol{\lambda}_t$, learning of $\boldsymbol{C}$ must evolve on long timescales. We think of it as lifetime learning of a set of features which commonly occur in natural scenes. To keep the emerging components sparse and interpretable, we further have the freedom to impose a regularizing prior, $p(\boldsymbol{C})$, during learning. The full learning rule reads:

$$\partial_t \boldsymbol{C} = \eta_C \left[\frac{1}{\sigma^2_{\text{obs}}} \left(\boldsymbol{v}_t \, \boldsymbol{\mu}_t^\mathsf{T} - \boldsymbol{C} \, \left(\boldsymbol{\mu}_t \, \boldsymbol{\mu}_t^\mathsf{T} + \boldsymbol{\Sigma}_t \right) \right) + \frac{1}{N_C} \nabla_{\boldsymbol{C}} \log p(\boldsymbol{C}) \right] \quad . \tag{3}$$

This rule establishes the intuition to compare the observed covariance between inputs and inferred motion sources against their expected covariance. Here, η_C is a small learning rate, N_C weights the prior vs. the likelihood, and $\boldsymbol{\Sigma}_t = \text{diag}\left[\boldsymbol{f}_\Sigma(\boldsymbol{\lambda}_t^2) \right]$ is the variance of the $\boldsymbol{s}$-posterior.

In **Fig. 3**, we demonstrate, for the first time, online learning of hierarchically nested motion relations in a computer simulation. Observed velocities are generated according to the model in **Fig. 1d & e**, with the nested graph structure shown in **Fig. 3a**. The corresponding ground truth-component matrix is shown in **Fig. 3b**. At any time, a random subset of the available components from **Fig. 3b** is present (average: 3 active components), and a new random structure is drawn every 20 s. For the learning system, the component matrix is initially empty, i.e., $\boldsymbol{C}(t=0) = \boldsymbol{0}$, and we provide two

additional (empty) components to this matrix to test whether the algorithm finds a sparse, minimal solution. The system evolves according to eqn. (1)–(3) for $100{,}000\,\mathrm{s}$ using the following regularizing prior:

$$\log p(\boldsymbol{C}) \propto -\frac{1}{2l_2}\sum_{m=1}^{M}\left(\sum_{k=1}^{K}|C_{km}|\right)^2 - \frac{1}{l_1}\sum_{m=1}^{M}\sum_{k=1}^{K}|C_{km}| \quad . \tag{4}$$

This prior facilitates motion components to remain small (L2 regularization of full components), and individual matrix elements to be sparse (L1 regularization of C_{km}). Furthermore, we add small, zero-mean exploration noise in every time step during learning.

At the end of the simulation, all components were correctly identified, see **Fig. 3c**. The time evolution during learning is shown in **Fig. 3d** for all $M=12$ components. The orange lines denote the $K=8$ velocities ($k=1$: darkest). A horizontal gray line marks $C_{km}=0$ (if visible). Global motion ($m=1$) and counter-rotation ($m=7$) are discovered quickly. The individual components take more time.

Interaction of priors across timescales

All dynamic variables, that is, $\boldsymbol{s}$, $\boldsymbol{\lambda}$, and $\boldsymbol{C}$, are softly constrained by prior distributions. We briefly discuss the modeling assumptions arising from those priors as well as how the priors interact with another. On the fasted timescale of the generative model, the motion sources, $\boldsymbol{s}_t$, obey a Gaussian prior, $p(s_m) = \mathcal{N}\left(0,\, \frac{\tau_s}{2}\lambda_m^2\right)$, which favors slow velocities and is supported for modeling visual motion perception by experimental work [3]. On an intermediate timescale, motion strengths, $\boldsymbol{\lambda}$, are governed by priors from the family of scaled inverse chi-squared distributions leading to the constants $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in eqn. (1). For this work, we employed two members of this family: for all allocentric motion of objects, that is, every λ_m except self-motion, we chose the scale-free Jeffreys prior, $p(\lambda_m^2) \propto \lambda_m^{-2}$. This prior induces sparsity of inferred motion structures by preferring vanishing motion components ($\lambda_m^2 = 0$) and interacts with $p(s_m)$ by setting its variance. For self-motion, we chose a uniform distribution, $p(\lambda_{\text{self}}^2) = const.$, capturing that frequent saccades and other eye movements give rise to fast retinal velocities for all observables, $\boldsymbol{v}_t$. On the longest timescale, motion components, $\boldsymbol{C}$, follow the regularizing prior of eqn. (4) which has been discussed earlier. Notably, in interaction with $p(\boldsymbol{\lambda}^2)$, this prior resolves an invariance in the generative model: multiplying $\boldsymbol{C}$ with any constant can be fully compensated by dividing $\boldsymbol{\lambda}$ by the same constant, thereby leaving $p(\,\boldsymbol{v}\,|\,\boldsymbol{\lambda},\,\boldsymbol{C}\,)$ unchanged. It is the exponential penalty on large components in $p(\boldsymbol{C})$ that counteracts the preference of $\boldsymbol{\lambda}$ for small values, thereby leading to the (now unique) equilibria observed in **Fig. 3d**.

Conclusion

We have proposed a comprehensive theory of online hierarchical inference for structured visual motion perception. The derived algorithm decomposes an incoming stream of retinal velocities into latent motion components which in turn are organized in a nested, tree-like structure. A scene's inferred structure provides the visual system with a temporally robust scaffold to organize its percepts and to resolve momentary ambiguities in the input stream. Applying the theory to human visual motion perception, we replicated diverse phenomena from psychophysics and made concrete predictions for new experiments. The algorithm afforded a recurrent neural network model motivating targeted neuroscience experiments to reveal the neural representations of latent structure. Finally, we demonstrated in a computer simulation that also the set of motion components could be learned online from experience.

References

[1] Charles Kemp and Joshua B Tenenbaum. "The discovery of structural form". In: *Proceedings of the National Academy of Sciences* (2008).

[2] Andrew M Saxe, James L McClelland, and Surya Ganguli. "A mathematical theory of semantic development in deep neural networks". In: *Proceedings of the National Academy of Sciences* (2019).

[3] Yair Weiss, Eero P Simoncelli, and Edward H Adelson. "Motion illusions as optimal percepts". In: *Nature neuroscience* (2002).

[4] Alan A. Stocker and Eero P. Simoncelli. "Noise characteristics and prior expectations in human visual speed perception". In: *Nature Neuroscience* (Apr. 2006).

[5] Andrew E. Welchman, Judith M. Lam, and Heinrich H. Bülthoff. "Bayesian motion estimation accounts for a surprising bias in 3D vision". In: *Proceedings of the National Academy of Sciences* (2008).

[6] James H Hedges, Alan A Stocker, and Eero P Simoncelli. "Optimal inference explains the perceptual coherence of visual motion stimuli". In: *Journal of vision* (2011).

[7] Samuel J Gershman, Joshua B Tenenbaum, and Frank Jäkel. "Discovering hierarchical motion structure". In: *Vision research* (2016).

[8] Johannes Bill et al. "Hierarchical structure is employed by humans during visual motion perception". In: *Proceedings of the National Academy of Sciences* (2020).

[9] Sichao Yang et al. "Human visual motion perception shows hallmarks of Bayesian structural inference". In: *Scientific reports* (2021).

[10] Oliver J Braddick, Keith A Wishart, and William Curran. "Directional performance in motion transparency". In: *Vision research* (2002).

[11] Jeffrey M Beck, Peter E Latham, and Alexandre Pouget. "Marginalization in neural circuits with divisive normalization". In: *Journal of Neuroscience* (2011).

Graduate Student Descent Considered Harmful? A Proposal for Studying Overfitting in Reward Functions

Serena Booth
Bosch, MIT CSAIL
sbooth@mit.edu

W. Bradley Knox
Bosch, The University of Texas at Austin
bradknox@cs.utexas.edu

Julie Shah
MIT CSAIL
julie_a_shah@csail.mit.edu

Scott Niekum
The University of Texas at Austin
sniekum@cs.utexas.edu

Peter Stone
The University of Texas at Austin
and Sony AI
pstone@cs.utexas.edu

Alessandro Allievi
Bosch
alessandro.allievi@us.bosch.com

Abstract

For better or for worse, reward functions are typically designed through an ad-hoc process involving extensive trial and error, in which an engineer tries different reward functions and observes how RL agents perform in their task. Trial-and-error reward design is known to facilitate unsafe reward shaping [1, 2], wherein a shaped reward function changes the optimal solution for the task with respect to the "true" reward function. Still, unsafely-shaped reward functions are sometimes useful or even necessary since formulating an application as an RL problem remains non-trivial.

We put forth a proposal for studying a related question: can reward functions that have been designed by trial-and-error be *overfit* to RL algorithms and hyperparameters? We define a reward function to be overfit when the function is optimized (perhaps imperfectly) with respect to some distribution of learning algorithms, hyperparameters for those algorithms, or environments, but is likely to be used or tested with a different distribution, resulting in a relative performance drop compared to learning with other possible reward functions on this different distribution.

We propose two studies for assessing overfitting in trial-and-error reward design. First, we propose a computational study to assess the risk and frequency of overfitting; second, we propose a user study which tests overfitting in ad-hoc reward design with imperfect optimizers (humans!). Through these studies, we can explore whether reward functions risk overfitting to the choice of RL algorithm (e.g., Q-learning, PPO, DDQN, or A2C), hyperparameters (e.g., learning rate(s), number of episodes), or environment (e.g., discount factor, state distributions). We report some early results wherein we provide evidence that the best-performing reward functions can vary with the discount factor (γ) and with the choice of RL algorithm (between PPO, DDQN, and A2C).

It is notoriously difficult to assess RL contributions due to stochasticity in environments and intrinsic variance in the algorithms and code [3, 4]. Overfitting in reward functions is equally a concern for RL reproducibility, as it can lead to false comparisons between RL algorithms or to suboptimal performance. Reward functions are often defined once through a trial-and-error process and subsequently reused by the community at large; if these reward functions are overfit to the distribution at design time, future comparisons may be structurally disadvantaged. Ultimately, the work proposed in this extended abstract aims to contribute to the RL reproduciblity literature by studying how the choice of reward function complicates the assessment and comparison of RL methods.

Keywords: Reward Function Design; Evaluation; Overfitting

1 Introduction

Reward functions are notoriously hard to design. In their quintessential introductory text on RL, Sutton and Barto assert: "The reward signal is your way of communicating to the agent what you want achieved, not how you want it achieved" [5]. Nonetheless, the practice of reward design almost never adheres to this adage.[1] Sutton and Barto's statement implies that reward functions should often be sparse; in practice, this is rarely the case, since it can be hard to learn from sparse signals. Instead, reward functions are typically designed through an ad-hoc process of trial and error.

One of the known, common problems with ad-hoc reward design is reward shaping. In reward shaping, the reward function is overloaded to both communicate the underlying performance metric and guide an agent's learning toward a desired policy. However, ad-hoc reward shaping is known to be typically *unsafe*—meaning that a shaped reward function is likely to change the optimal solution to a given reinforcement learning task [1, 2].

But, unsafe reward shaping is not the only concern when employing trial-and-error reward design. Using a series of computational experiments and user studies, we plan to assess whether trial-and-error reward design also induces *overfitting* of reward functions. We define a reward function to be overfit when the function is optimized (perhaps imperfectly) with respect to some distribution $\mathcal{D}_1$ of learning algorithms, hyperparameters for those algorithms, or environments, but is likely to be used or tested with a different distribution $\mathcal{D}_2$, resulting in a relative performance drop compared to learning with other possible reward functions.

We propose a computational study to assess overfitting of reward functions by testing whether different reward functions perform best for different instantiations of RL algorithms, hyperparameters, and environments. We have conducted a preliminary study and have found evidence that reward function overfitting can occur when varying the discount factor or algorithm. While this proposed study can assess the potential, risk, and frequency of overfitting in reward function design, it uses a computationally-expensive grid search over reward functions and, as such, is rarely applicable to RL practice. Instead, most reward functions are manually crafted through an ad-hoc trial-and-error design process.

As such, we also propose conducting a user study which assesses whether overfitting of reward functions is also a concern in the presence of imperfect optimizers: humans. For this user study, participants should be given an RL task where their goal is to train the best agent they can. The user must choose between different deep RL algorithms (A2C, PPO, DDQN) and hyperparameters (e.g., the discount factor, learning rate(s), number of episodes, etc.), and must design their own reward function. Analyzing the reward functions participants design will allow us to assess whether a different choice of algorithm and hyperparameters can lead to relative underperformance. We have conducted a pilot study and have found preliminary evidence of reward function overfitting in this setting.

From these studies, we will gain new insights into how reward functions affect the assessment and comparison of RL methods. This abstract presents a blueprint of our planned studies and early results; we invite feedback and discussion.

2 Domain

We consider the Hungry Thirsty domain [6]: a 6×6 gridworld in which an agent exists for a fixed time horizon of n steps. Food is available in one randomly-chosen grid corner; water in another. Some movement transitions are blocked by "walls" in the grid (red lines in Fig. 1). At each timestep, the agent can choose one of six actions: move (left, right, up, down), eat, or drink.

The agent's goal is to have sated hunger for as many timesteps as possible. If the agent has not eaten in the last timestep, it becomes hungry. However, the agent can only eat if it is not thirsty; and, if the agent has not drunk in the last timestep, it becomes thirsty with 10% probability. In this domain, the 'true reward function' or score consists of non-zero reward for each timestep when the agent is not hungry, e.g., $\sum_{t=0}^{n} \mathbb{1}(\neg \mathtt{H} \in s_t)$, where s_t is the agent's state at time t, n is the length of the episode, and $\mathtt{H}$ is a boolean predicate corresponding to hunger.

When the agent is thirsty, under the optimal policy the agent navigates to the water or, if the agent is co-located with the water tile, the agent drinks. When the agent is not thirsty, the agent should either navigate to the cell containing the food or eat if the agent is co-located with the food tile. A key property of this

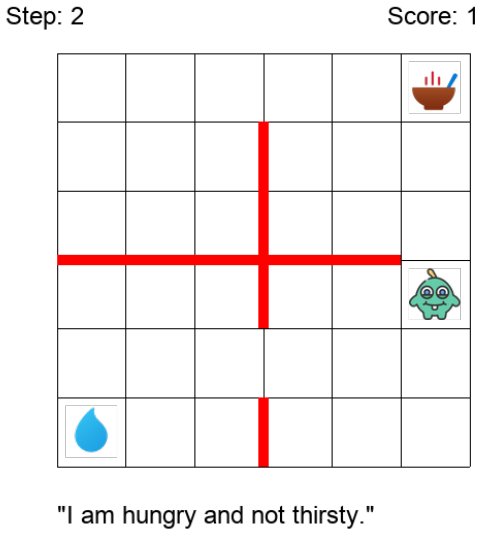


Figure 1: The Hungry Thirsty gridworld domain.

[1] Even within this text, Sutton and Barto disregard this advice when designing a Dyna-Q+ agent. To encourage exploration, they replace the reward function r with $r + \kappa\sqrt{\tau}$, where κ is a hyperparameter and τ is the number of steps the agent has taken [5, p. 168].

108

domain is that with just the true reward function, it is computationally challenging for many RL algorithms to find this optimal policy due to the reward's sparseness and the domain's high stochasticity. However, shaping functions (i.e., by rewarding the agent for learning to drink) let many RL algorithms solve this domain relatively easily.

3 Proposed Computational Study

Singh et al. [6] introduced the notion of an 'optimal reward function' as the reward function that maximizes the true reward over a tested distribution of environments. Given an input distribution, they conducted a grid search over possible reward functions and assessed the performance of each possible function against the true reward function. In Hungry Thirsty, they searched over reward functions that consist of four values corresponding to possible states: not hungry and not thirsty ($\neg\mathtt{H} \wedge \neg\mathtt{T}$), hungry and thirsty ($\mathtt{H} \wedge \mathtt{T}$), etc. They then compare the performance of agents trained with these reward functions by assessing their undiscounted cumulative true reward.

$$r_{\text{true}}(s) = \begin{cases} 1.0 & s = \neg\mathtt{H} \wedge \neg\mathtt{T} \\ 1.0 & s = \neg\mathtt{H} \wedge \mathtt{T} \\ 0.0 & s = \mathtt{H} \wedge \neg\mathtt{T} \\ 0.0 & s = \mathtt{H} \wedge \mathtt{T} \end{cases}$$

$$r_{\text{optimal}}(s) = \begin{cases} 1.0 & s = \neg\mathtt{H} \wedge \neg\mathtt{T} \\ 0.5 & s = \neg\mathtt{H} \wedge \mathtt{T} \\ -0.01 & s = \mathtt{H} \wedge \neg\mathtt{T} \\ -0.05 & s = \mathtt{H} \wedge \mathtt{T} \end{cases}$$

To determine the optimal reward function, they average the agent's performance over N randomly-sampled environments. In their study, the agent uses Q-learning with a fixed learning rate α, an ϵ-greedy action selection with fixed random action probability, ϵ, and a fixed discount factor γ.

Since the true reward function for Hungry Thirsty is sparse, this signal can be hard to learn from. In contrast, the optimal reward function incentivizes the agent to learn to drink as a subgoal. With this reward, drinking is preferred to not doing so since the reward is greater; this subgoal is described as a form of intrinsic motivation.

We plan to reapply this framework to instead assess overfitting in reward functions. We hypothesize that the optimal reward function defined above will be sub-optimal for a different learning algorithm or hyperparameter instantiation. Specifically, we propose running this optimal reward function search with varied Q-learning hyperparameters: γ (the environment discount factor) and α (the learning rate). We additionally propose studying whether the reward functions are overfit to the learning algorithm itself by comparing the optimal reward function for a Q-learning agent to those for several deep RL alternatives: A2C [7], DDQN [8], and PPO [9]. For this study, we propose the following hypotheses:

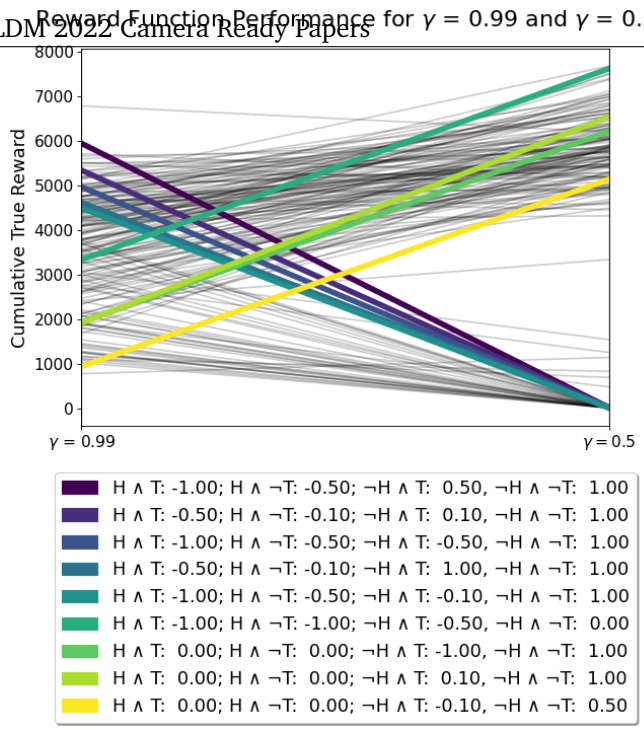
- **Hypothesis 1:** Given an 'optimal' reward function selected with respect to a distribution $\mathcal{D}_1$, a different reward function will be 'optimal' if selected with respect to a different distribution, $\mathcal{D}_2$.
  This is the weakest hypothesis: it looks only for the existence of an alternative optimal reward function for a different RL algorithm or hyperparameter instantiation. While the distributions can be chosen adversarially (e.g., by choosing an inappropriate learning algorithm), we assume both distributions include support for learning appropriately if given a viable reward function. This assumption can be formulated as a sanity test.
- **Hypothesis 2:** If the reward functions are ranked according to the true reward function for each distribution (e.g., $r_i > r_j > r_k > ...$ for $\mathcal{D}_1$), the ranked reward functions from $\mathcal{D}_1$ will not be correlated with the ranked reward functions from $\mathcal{D}_2$. We propose computing Kendall's tau rank correlation to assess this; Kendall's tau measures the strength and direction of the monotonic association between rankings. One limitation of this statistical test is that it does not take the difference in magnitude of the reward function performance into consideration.

3.1 Preliminary Results

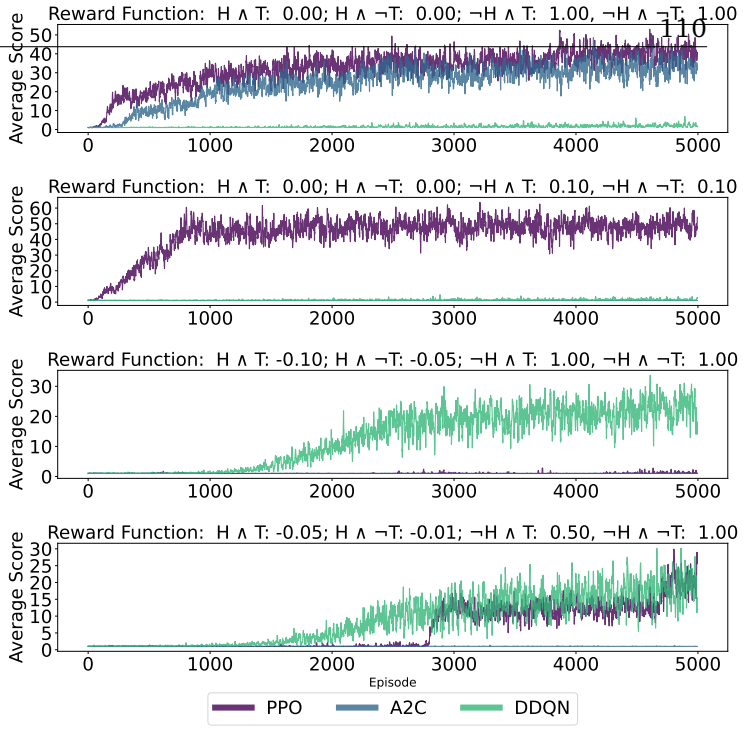
We first study the role of the discount factor, γ, in reward function design for Hungry Thirsty. We use a Q-learning agent which learns over $75,000$ timesteps with learning rate $\alpha = 0.05$. We take $\gamma = 0.99$ for $\mathcal{D}_1$ and $\gamma = 0.5$ for $\mathcal{D}_2$, and we average over 100 environments and Q-table instantiations to account for variance. We conduct a grid search over 2401 reward functions, where $r(s) \in [-1, -0.5, -0.1, 0, 0.1, 0.5, 1.0]$. Of these reward functions, the vast majority do not lead to substantive learning; only 184 reward functions achieve scores of > 1000 points over $75,000$ steps for either value of γ.

- **Hypothesis 1:** In support of hypothesis 1, we find the 'optimal' reward functions do indeed differ for $\gamma = 0.99$ and $\gamma = 0.5$. While both of these reward functions perform quite well for both experiment conditions (i.e., in the top $\approx 25\%$ of reward functions for both conditions), this is not true of the second best reward function for $\gamma = 0.99$; this reward function fails to support learning for $\gamma = 0.5$. See Fig. 2a.
- **Hypothesis 2:** In support of hypothesis 2, we find the ranked lists of reward functions for each distribution ($\gamma = 0.99$ and $\gamma = 0.5$) are uncorrelated, as assessed by Kendall's tau ($\rho_\tau(184) = 0.047, p = 0.341$).

We next study the role of the algorithm choice in determining the best reward function. For this experiment, we modify the Hungry Thirsty environment to be episodic, with 200 steps per episode. A grid search over deep RL algorithms is exceedingly expensive; as such, as a first study, we select four reward functions with different properties for comparison. We choose two sparse reward functions (r_{true} and $0.1 \times r_{\text{true}}$) and two shaped reward functions (including r_{optimal}). From



(a) A parallel coordinate plot comparing reward function performance for $\gamma = 0.99$ and $\gamma = 0.5$, where each line is a different reward function. Several of the best reward functions for $\gamma = 0.99$ yield learning failures for $\gamma = 0.5$. Sparse reward functions, where increased reward is only received in response to the agent being not hungry, perform better when $\gamma = 0.5$. The second best performing reward function for $\gamma = 0.99$ ($\texttt{H} \wedge \texttt{T}: -1, \texttt{H} \wedge \neg\texttt{T}: -0.5, \neg\texttt{H} \wedge \texttt{T}: 0.5, \neg\texttt{H} \wedge \neg\texttt{T}: 1.0$), fails when $\gamma = 0.5$. The nine colored lines correspond to reward functions which had the largest change in absolute performance between the two conditions.



(b) We select 4 reward functions: the true sparse reward function (r_{true}), a smaller magnitude sparse reward function ($0.1 \times r_{\text{true}}$), and two shaped reward functions (one of which is r_{optimal}). For each reward function, we train an A2C agent, a PPO agent, and a DDQN agent using standard hyperparameters, and we average performance over 5 training runs. For the first reward function (r_{true}), only PPO and A2C learn effectively. For the second reward function ($0.1 \times r_{\text{true}}$), only PPO learns. For the third reward function, only DDQN learns. For the last reward function (r_{optimal}), both DDQN and PPO learn, while A2C does not.

Figure 2: Computational Study of Reward Function Performance

this preliminary study, we find evidence in support of our hypotheses; namely, that the ranking of reward function performance differs when compared across different learning algorithm. Of these four reward functions, we find A2C is only successful in learning on the larger magnitude sparse reward function. In contrast, DDQN fails to learn for both sparse reward functions, but is the best performing algorithm for the shaped reward functions. See Fig. 2b.

4 Proposed User Study

In practice, reward functions are typically designed manually, guided by a combination of engineer intuition and trial-and-error experimentation. In a survey of 24 RL practitioners from the University of Texas at Austin, SonyAI, and MIT, $22/24$ reported using a trial-and-error process to design their most recent reward function; of these, 17 reported using domain knowledge or intuition to guide their initial reward design. We thus propose a complementary user study to assess whether overfitting of reward functions also occurs in naturalistic settings.

For this user study, we will ask participants to design the best-performing Hungry Thirsty agents they can. Every participant will be compensated at a base rate of $\$40$ USD/hour, and the 5 participants who design the top-performing agents will be awarded an additional $\$10$ USD bonus. Participants will be required to have completed a class in reinforcement learning, but have not previously used the Hungry Thirsty domain. This protocol has received IRB approval.

Participants will be presented with an interface which asks them to instantiate a reward function for Hungry Thirsty. Their reward functions are constrained to weights in the range $[-1, 1]$ for each of four logical state instantiations: not hungry and not thirsty, hungry and thirsty, etc. In addition, the participant is asked to select a deep RL algorithm from a choice of A2C, PPO, and DDQN, as well as learning hyperparameters for their selected learning algorithm and environment. A demonstration of the study is shown in Appendix A.

Since human participants are imperfect optimizers and are likely to only perform trial-and-error assessments with a small number of reward functions, the question of overfitting in this user study context is complicated. If the participant's selected reward function is sub-optimal relative to some other plausible reward function, can their selected function be overfit? Since we define overfitting in terms of relative and not absolute performance, this indeed consti-

tutes overfitting—i.e., if the participant's reward function is biased by their algorithm and hyperparameter selection such that it underperforms on a different algorithm or hyperparameter selection, their selected reward function is overfit regardless of whether it is the singular best-performing reward function for their tested distribution.

Similarly, it is unfair to compare a participant's selected reward function to the space of all possible reward functions, since the participant will likely select a reward function which is sub-optimal for *both* their test distribution and for a target distribution. We thus restrict the set of reward functions to consist of only the reward functions used during the participant's search. We introduce a hypothesis accounting for this inherent sub-optimality:

- **Hypothesis 3:** If a participant tries the set of reward functions $r_1, r_2, ..., r_n$ using a distribution $\mathcal{D}_1$, and selects the reward function r_i from this set, this reward function is overfit if, for some alternate distribution $\mathcal{D}_2$, a reward function r_j in the set outperforms r_i according to the true reward (where $r_j \neq r_i$).

4.1 Preliminary Results

We conducted two pilot user studies. User #1 trained 7 RL agents with 4 reward function (3 of which are shaped), different algorithms (DDQN and A2C), and different hyperparameters for every trial. Their selected 'best' agent used a sparse reward function ($r(\texttt{H} \in s) = -1; r(\neg\texttt{H} \in s) = 1$) with DDQN. However, we found that one of their earlier overlooked reward functions ($r(\texttt{H} \wedge \texttt{T}) = -1; r(\texttt{H} \wedge \neg\texttt{T}) = -0.5; r(\neg\texttt{H} \wedge \texttt{T}) = 0.5; r(\neg\texttt{H} \wedge \neg\texttt{T}) = 0.6$) performs equally well with their final DDQN hyperparameter selection (i.e., over 5 trials, the performance of both functions is within the standard deviation of the other). While this pilot does not definitively confirm overfitting in trial and error reward design according to our hypothesis, it shows how viable reward functions can be passed over.

User #2 trained 3 RL agents, using a consistent shaped reward function and algorithm (PPO), instead varying hyperparameters (specifically, the update step frequency, number of episodes, and actor network learning rate).

5 Discussion

In this extended abstract, we have introduced two proposed studies for assessing overfitting in reward function design. Through a preliminary investigation, we find evidence of reward function overfitting. One limitation of this proposal is its focus on a single domain; to better understand the risk of reward function overfitting more generally, we must analyze multiple domains. In addition, the computational study we propose is costly; as such, this method of reward function design is rare (though has been done [10]). But, user studies are inherently constrained to lab settings, which may not mimic real-world practices. An additional study could assess overfitting in the reward functions used in RL applications.

This problem of reward function overfitting has, to our knowledge, not previously been discussed. Since overfitting impacts the assessment and comparison of RL methods, this is an important avenue for improving RL reproducibility.

References

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[2] W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis) design for autonomous driving. *arXiv preprint arXiv:2104.13906*, 2021.

[3] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[4] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *ICLR*, 2019.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

[7] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE RA-Letters*, 4(2):2007–2014, 2019.

A User Study Protocol

Hungry-Thirsty Domain

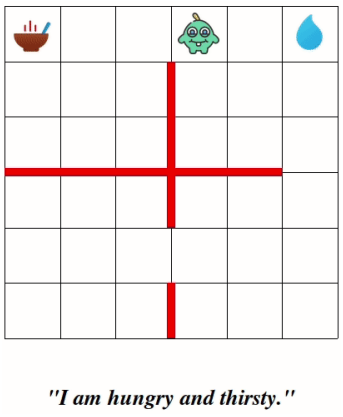
The goal of the hungry-thirsty domain is to teach an agent to eat as much as possible.
There's a catch, though: the agent can only eat when it's not thirsty.
Thus, the agent cannot just “hang out” at the food location and keep eating because at some point it will become thirsty and eating will fail.

- The agent always exists for 200 timesteps.
- The grid is 6x6. Food is located in one randomly-selected corner, while water is located in a different (random) corner.
- At each timestep, the agent may take one of the following actions: move (up, down, left, right), eat, or drink. But actions can fail:
  - The drink action fails if the agent is not at the water location.
  - The eat action fails if the agent is thirsty, or if the agent is not at the food location.
  - The move action fails if the agent tries to move through one of the red barriers (depicted below).
- If the agent eats, it becomes not-hungry for one timestep.
- If the agent drinks, it becomes not-thirsty.
- When the agent is not-thirsty, it becomes thirsty again with 10% probability on each timestep.

See an example of the game below.

In [1]:
```
from IPython.display import Image
Image("../Assets/h-t.gif", width=300)
```

Out[1]:



"I am hungry and thirsty."

Your Study ID

Replace the YOUR_NAME string with your full name to generate a unique, privacy-preserving study ID.

In [2]:
```
YOUR_NAME = "YOUR_NAME"
import hashlib
study_id = hashlib.md5(YOUR_NAME.encode()).hexdigest()
print ("To preserve privacy, your study ID is: ", study_id)
```

```
To preserve privacy, your study ID is:  85f5b220c4c9521a21ee9ecd967795ea
```

Your Task

We haven't specified the reward function, learning algorithm, or algorithm hyperparameters; you'll need to fill in these details using the code cells below.

In [3]:
```
# import notebooks; do not edit
%run setup_reward_and_learning_alg.ipynb
%run training_and_model_eval.ipynb
```

Design your Reward Function and Select Your RL Algorithm

For your reward function, you need to assign a value r(s) to each state. The state is composed of the thirst and hunger status of the agent. You must assign a value for each state:

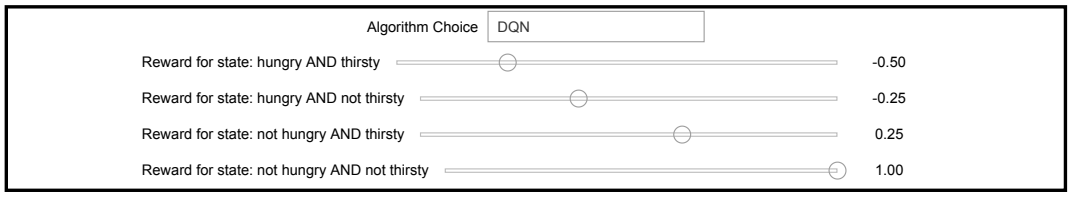
- r(Hungry AND Thirsty) = ???; r(Hungry AND Not Thirsty) = ???; r(Not Hungry AND Thirsty) = ???; r(Not Hungry AND Not Thirsty) = ???

For your RL algorithm, you will need to choose your algorithm. Your options are:

- A2C, DQN, PPO

After choosing your RL algorithm, you will need to select the hyperparameters. If you change your learning algorithm, you may need to re-run the hyperparamater selection (below).

In [4]:
```
# select the learning algorithm and reward function parameters
selectors = reward_and_alg_selector()
selectors
```

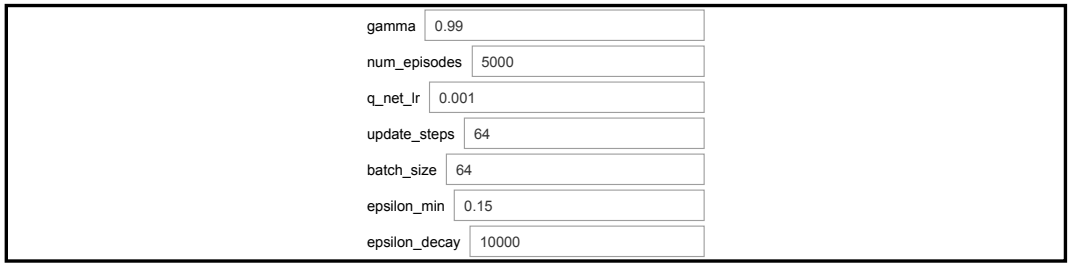


Hyperparameters

!!! If you change your learning algorithm selection, you may need to rerun the following hyperparameter selection cell as well. !!!

- Shared Hyperparameters:
  - **Gamma**: the discount factor for the environment. A small Gamma means the agent prioritizes only immediate rewards, while a larger Gamma means the agent tends to also consider future rewards.
  - **Num_Episodes**: the number of episodes to train for. A smaller number means the experiments are faster, but contain less experience to learn from.
  - **Learning Rates**: for DQN, there is only one learning rate, the Q-network learning rate (**q_net_lr**). For both A2C and PPO, there are two learning rates, one for the actor network (**actor_lr**) and one for the critic network (**critic_lr**). Smaller learning rates make smaller updates to the network weights (and hence optimization is slower), while larger learning rates make larger updates.
  - **Update_Steps**: Only applicable to PPO and DQN, this is the frequency with which to perform updates. A smaller number means more frequent updates, which is slower but more information dense.
- DQN:
  - **Epsilon_Min**: DQN uses an epsilon-greedy strategy. Epsilon decreases over time to encourage initial exploration (starting at epsilon=1). epsilon_min corresponds to the floor for the epsilon value. A larger epsilon_min means more exploration, less exploitation. A smaller epsilon min means less exploration, more exploitation.
  - **Epsilon_Decay**: Over time, epsilon decreases from 1 to epsilon_min. Every time step, epsilon decreases by 1/epsilon_decay.
  - **Batch_Size**: The number of samples to take from the experience replay buffer from which to calculate the loss and update the deep Q Network. A smaller number is faster to run but contains less experience.
- PPO:
  - **Eps_Clip**: In PPO, the estimated advantage function is clipped to handle variance. If the probability ratio between the new policy and the old policy falls outside the range (1 — ε) and (1 + ε), the advantage function is clipped. A smaller eps_clip value is more permissive; a larger eps_clip value is more restrictive and allows for less substantial policy changes.

In [14]:
```
# select the learning algorithm hyperparameters
# !!! If you change the algorithm choice, you will need to re-run this !!!
alg = get_params(widget_ref=selectors)["Algorithm Choice"]
select_learning_alg_params = construct_hyperparam_selector(alg_name = alg)
select_learning_alg_params
```



Run the cell below to confirm your choice of parameters and algorithm!

Training Time!

For evaluating our reward functions, algorithm selection, and hyperparameters, we plot training performance according to *fitness* and *undiscounted return*.

Each episode consists of a trajectory $\tau = [(s_0, a_0, s_1), (s_1, a_1, s_2), \dots]$.

Fitness is computed as the sum of states in which the agent is not hungry:

- Fitness := $\Sigma_{(s,a,s') \in \tau}$ $\mathbb{1}$(s[is_hungry] == False)

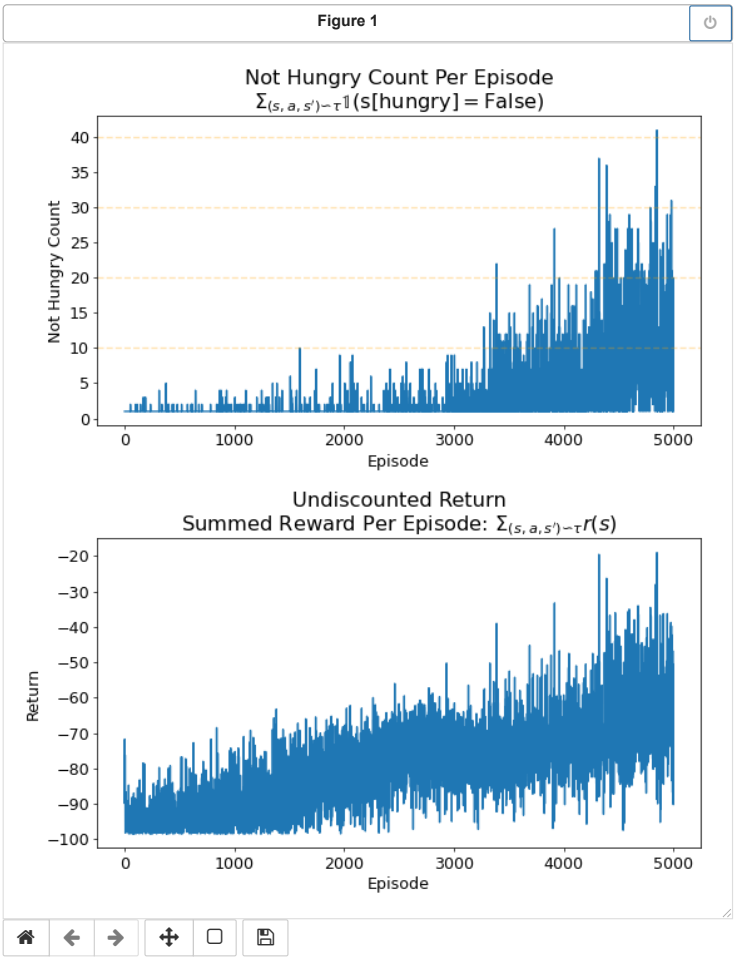
Undiscounted return is computed using the reward function you specified:

- Undiscounted Return := $\Sigma_{(s,a,s') \in \tau}$ r(s)

You may wish to go back and change one or more of your reward function, learning algorithm, or learning algorithm parameters.

In [15]:
```
%matplotlib notebook

train_agent(alg_and_reward_params=get_params(widget_ref=selectors),
            hyper_params=get_params(widget_ref=select_learning_alg_params),
            study_id=study_id)
```




```
100%|██████████████████████████████████| 5000/5000 [02:19<00:00, 35.90it/s]

Total updates:  62496
FINISHED TRAINING
```

In [17]:
```
view_training_runs()
```

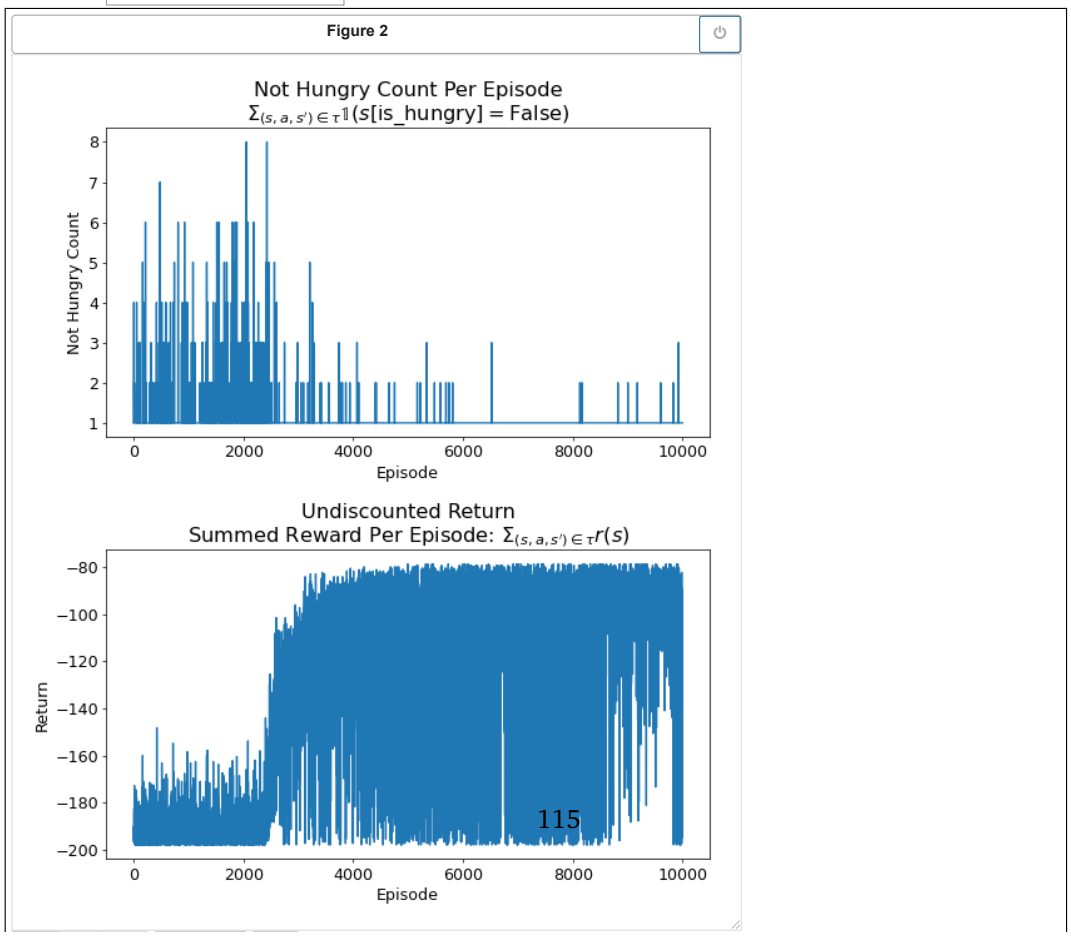
```
Trial 0---------------------------------------
     Algorithm : A2C
     Reward Function:
          r(hungry and thirsty): -1.00
          r(hungry and not thirsty): -0.40
          r(not hungry and thirsty): 0.45
          r(not hungry and not thirsty): 1.00
     Hyper-parameters:
          gamma     : 0.99000
          num_episodes: 10000.00000
          lr        : 0.00100
-----------------------------------------------


Trial 1---------------------------------------
     Algorithm : A2C
     Reward Function:
          r(hungry and thirsty): -0.25
          r(hungry and not thirsty): 0.00
          r(not hungry and thirsty): 0.25
          r(not hungry and not thirsty): 1.00
     Hyper-parameters:
          gamma     : 0.99000
          num_episodes: 5000.00000
          lr        : 0.00100
-----------------------------------------------


Trial 2---------------------------------------
     Algorithm : DQN
     Reward Function:
          r(hungry and thirsty): -0.50
          r(hungry and not thirsty): -0.25
          r(not hungry and thirsty): 0.25
          r(not hungry and not thirsty): 1.00
     Hyper-parameters:
          gamma     : 0.99000
          num_episodes: 5000.00000
          q_net_lr  : 0.00100
          update_steps: 64.00000
          batch_size: 64.00000
          epsilon_min: 0.15000
          epsilon_decay: 10000.00000
-----------------------------------------------
```

In [9]:
```
review_past_run()
```

Select trial: Trial: 0; Alg: A2C See Trial Metrics



```
In [18]: submit_agent()
```

Select trial: Trial: 2; Alg: DQN Submit

```
You selected trial: 2
Trial 2----------------------------------------
     Algorithm : DQN
     Reward Function:
          r(hungry and thirsty): -0.50
          r(hungry and not thirsty): -0.25
          r(not hungry and thirsty): 0.25
          r(not hungry and not thirsty): 1.00
     Hyper-parameters:
          gamma     : 0.99000
          num_episodes: 5000.00000
          q_net_lr  : 0.00100
          update_steps: 64.00000
          batch_size: 64.00000
          epsilon_min: 0.15000
          epsilon_decay: 10000.00000
------------------------------------------------
```

Object-Factored Models with Partially Observable State

Isaiah Brand*
CSAIL, MIT
Cambridge, MA, USA
ibrand@mit.edu

Michael Noseworthy*
CSAIL, MIT
Cambridge, MA, USA
mnosew@mit.edu

Sebastian Castro
CSAIL, MIT
Cambridge, MA, USA
scastro@csail.mit.edu

Nicholas Roy
CSAIL, MIT
Cambridge, MA, USA
nickroy@csail.mit.edu

Abstract

In a typical robot manipulation setting, the physical laws that govern object dynamics never change, but the set of objects the robot interacts with does change. To complicate matters, objects may have intrinsic properties that are not directly observable (e.g., center of mass or friction coefficients). In order to successfully manipulate previously unseen objects, the robot must efficiently adapt to their object-specific properties. In this work, we introduce a latent-variable model of object-factored dynamics. The model represents uncertainty about the dynamics using *deep ensembles* while capturing uncertainty about each object's intrinsic properties using object-specific latent variables. We show that this model allows a robot to rapidly generalize to new objects by using information theoretic active learning. Additionally, we highlight the benefits of the deep ensemble for robust performance in downstream tasks.

Keywords: Partial Observability, Latent Variable Models, Deep Ensembles, Robotic Manipulation

Acknowledgements

The authors would like to thank Caris Moses, Leslie Pack Kaelbling, and Tomás Lozano-Pérez for invaluable discussions relating to the ideas presented. We also thank the reviewers for their helpful feedback. This research was generously sponsored by the Honda Research Institute.

*Equal contribution.

1 Introduction

Predictive models are indispensable in the roboticist's toolkit. The ability to predict the outcome of an action allows a robot to plan robustly in a task-agnostic manner [3, 8]. For such models to be useful, they need to accurately capture the nuances of each object the robot interacts with. This is challenging due to the vast diversity of objects present in the world. Previous work has shown that factoring models to decouple object state from the global dynamics leads to effective generalization when these properties are fully observable [5, 8]. However, the assumption that object state is fully-observed does not generally hold; many objects have intrinsic properties that influence how they behave and that are not visually observable. For example, an object's *center of mass* will influence how it can be stacked, and a ball's *rolling resistance* will influence how far it rolls. When interacting with novel objects, we generally do not have labels for these properties (even at training time, these labels can be laborious to collect).

The key question we are interested in is: *how do we learn predictive models that support quick adaptation to objects with unobserved properties?* To do so, we propose to use an object-factored latent variable model that explicitly captures uncertainty about the object's unobserved properties *and* the global dynamics. The object-specific latent variables support information theoretic data acquisition in the learned latent space, allowing for rapid adaptation. The forward model, or dynamics function, is represented using a deep ensemble, allowing the model to capture uncertainty about complex dynamics. Both types of uncertainty can be incorporated when performing downstream tasks, leading to more robust uncertainty-aware planning.

We evaluate the proposed model in two domains: stacking blocks with unobserved centers of mass and throwing balls with unobserved rolling resistance. We demonstrate the utility of representing uncertainity at both the object and global levels. At the object level, the robot can use the learned latent space to quickly adapt to novel objects. At the dynamics level, the deep ensemble leads to more robust task performance when compared to a model that does not use ensembles.

2 Learning Phases for Object Manipulation

We consider a robot operating in a domain with K objects from the same class. Each object's state can be divided into observed and unobserved properties. The observed state, $\mathcal{X}_t = \{x_t^{(k)}\}_{k=1}^K$, can be time-varying, whereas the unobservable state, $\mathcal{Z} = \{z^{(k)}\}_{k=1}^K$, is assumed to be static. This factorization covers a wide range of object properties including friction coefficients, mass, and coefficient of restitution. Our objective is to learn a model that predicts some aspect of the environment, y_t, which we will refer to as the *outcome*, that can be useful for downstream tasks. Concretely, the goal is to estimate $p(y_t|\mathcal{X}_t, \mathcal{Z}, a_t)$, where $a_t \in \mathcal{A}$ is the action space of the robot.

We assume the robot's operation is divided into three distinct phases: *training*, *adaptation*, and *testing* (see Figure 1). In the *training* phase, the set of objects is fixed, and the robot has some finite amount of time to act in the environment without any task descriptions. In the *adaptation* phase, the robot is given a new object and a short amount of time to experiment and adapt. Finally, there is a *testing* phase, in which a robot is supplied with an external reward function, and interacts with the objects in order to maximize that reward. Sampling-based methods are used to find reward-maximizing actions and no further adaptation or learning is performed in this phase.

3 Object-Factored Latent Variable Model

We represent the predictive model using the probabilistic graphical model shown in Figure 1. The outcome, y_t, is generated via a stochastic process governed by global parameters, θ, unobserved object parameters, $\mathcal{Z}$, the current state, $\mathcal{X}_t$, and action, a_t. Due to the complexity of object dynamics, we represent θ using an ensemble [2] of domain-dependent neural network architectures. We also include a prior over object-specific latent variables, $p(\mathcal{Z})$. In the *training phase*, the primary goal is to learn about the global parameters, θ, that will generalize to the downstream *adaptation* and *testing phases*. Then, in the *adaptation phase*, we only need to infer the unobserved properties of novel objects, $\mathcal{Z}$.

3.1 Inference in Factored-Object Models

In the training and adaptation phases, the robot will have a collection of transitions: $\mathcal{D} = \{(\mathcal{X}_t, a_t, y_t)_i\}_{i=1}^N$. With this dataset, we would like to estimate a posterior distribution over the unobserved parameters: $p(\mathcal{Z}, \theta \mid \mathcal{D})$ during the training phase and $p(\mathcal{Z} \mid \mathcal{D})$ during the adaptation phase. A posterior that accurately represents *epistemic uncertainty* will allow us to gather data more efficiently and plan more robustly with models resulting from limited data. In the training phase, we assume the robot has access to a dataset, while in the adaptation phase, it must gather its own.

In general, computing the true posterior is intractable, so we take a *Variational Inference* (VI) approach, and compute an approximate posterior distribution, $q(\mathcal{Z}, \theta) = q_z(\mathcal{Z})q_\theta(\theta)$. We represent each $q(z^{(k)})$ as a diagonal Gaussian distribution.

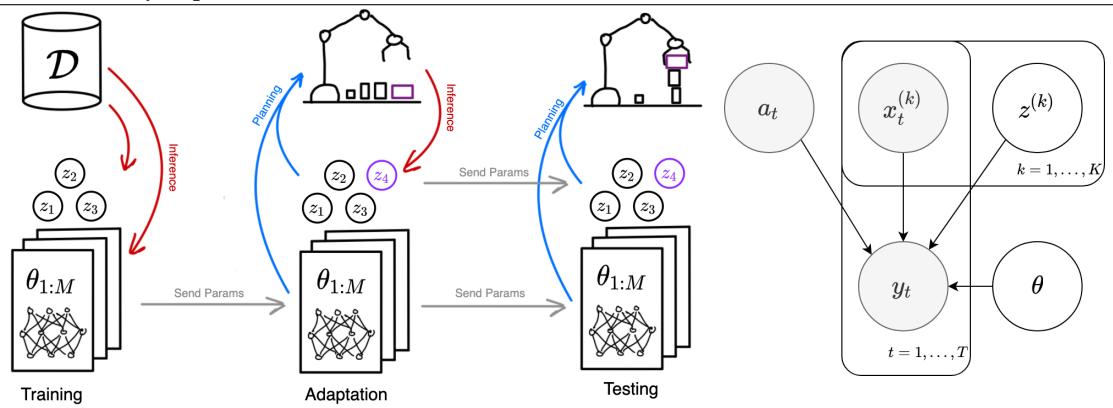


Figure 1: **Left.** Following a *training phase* to learn the global parameters, θ, the robot is given new objects (purple) in the *adaptation phase* before being evaluated in the *testing phase*. **Right.** We model this process using a probabilistic graphical model where the outcome of an action, y_t, depends on the state, $\mathcal{X}_t = \{x_t^{(k)}\}$, action, a_t, dynamics, θ, and each object's unobservable static properties, $\mathcal{Z} = \{z^{(k)}\}$.

Although VI in theory can apply to both $\mathcal{Z}$ and θ, recent work has proposed representing uncertainty over neural network parameters implicitly using an ensemble of M networks: $\theta = \theta_1, \ldots, \theta_M$ [7].

We propose an inference algorithm for generative models that uses ensembles to represent the uncertainty of the dynamics parameters while using VI for inference of the latent variables. The algorithm alternates between optimizing $\mathcal{Z}$ with respect to the ELBO and updating each model in the ensemble to maximize the likelihood of the data.

Optimizing $\mathcal{Z}$: To optimize the parameters of $q_z(\mathcal{Z})$, we maximize the ELBO with respect to the variational distribution parameters, while fixing the ensemble parameters.

We derive a batch-loss which we minimize via gradient descent on the parameters of q_z,

$$\mathcal{L}_z(B) = \frac{N}{|B|} \sum_{t \in B} \mathbb{E}_q[-\log p(y_t \mid \mathcal{X}_t, a_t, \mathcal{Z}, \theta)] + \frac{1}{|B|} \mathrm{D}_{\mathrm{KL}}(q_z(z) \parallel p(z)). \tag{1}$$

Optimizing θ: Unlike for $\mathcal{Z}$, we do not explicitly place a prior on θ. Instead, we can view the prior as being implicitly defined by the weight-initialization and stochasticity in SGD. Each ensemble model receives batches in a different order to further encourage ensemble diversity.

$$\mathcal{L}_{\theta_i}(B) = \frac{N}{|B|} \sum_{t \in B} \mathbb{E}_q[-\log p(y_t \mid \mathcal{X}_t, a_t, \mathcal{Z}, \theta_i)]. \tag{2}$$

During the adaptation phase, we only fit $q_z(\mathcal{Z})$ and freeze θ. This can be done using VI or by particle filtering methods.

3.2 Experimental Design

When presented with a new object, a robot will need to adapt its model in a self-supervised manner. Because data collection on a robotic platform is expensive, it is important to be efficient. Random exploration is unlikely to lead to transitions that elicit the effects of the unobservable object properties, so we employ an information-theoretic active learning strategy. The robot iteratively performs an experiment and uses the resulting label to update the posterior. Specifically, during the adaptation phase, we choose actions that maximize the information gain between the label and the object-specific latent variable: $I(\mathcal{Z}; y_t | a_t, \mathcal{X}_t, \mathcal{D})$. We compute the information gain in label space Houlsby et al. [6] and marginalize over the uncertainty in the ensemble when selecting actions. The resulting acquisition function is,

$$\underset{a_t}{\text{argmax}} \ \mathrm{H}(y_t \mid \mathcal{X}_t, a_t, \mathcal{D}) - \mathbb{E}_{\mathcal{Z} \sim q}\left[\mathrm{H}(y_t \mid \mathcal{X}_t, a_t, \mathcal{Z}, \mathcal{D})\right]. \tag{3}$$

4 Experimental Domains

To evaluate the effectiveness of our approach, we analyze and ablate components of our model in two domains where the objects have unobserved properties that affect their dynamics: *stacking* and *throwing*. In each domain, the agent first

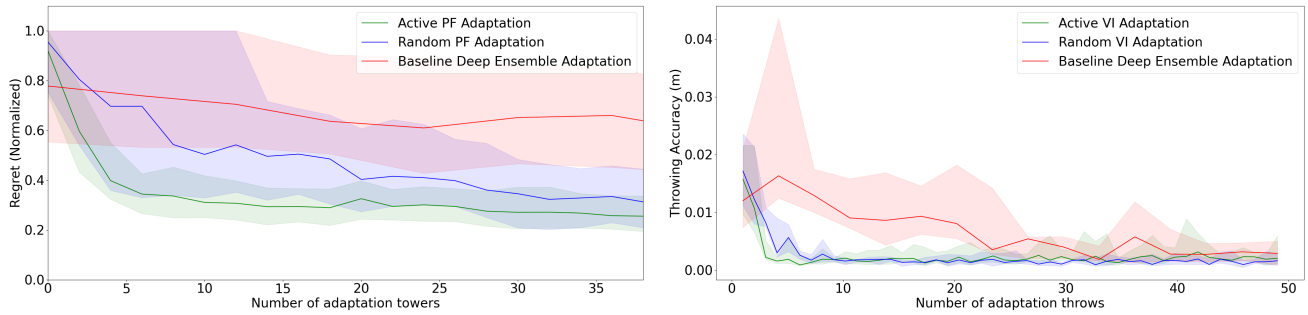


Figure 3: Task performance on each domain comparing adaptation strategies. Shaded areas are performance quartiles across 20 adaptation/testing runs. **Left.** Regret for the tower domain. **Right.** RMSE for the throwing domain.

fits the model parameters using a fixed dataset in the training phase, and is then given a novel object to adapt to before its performance is evaluated in the testing phase.

Stacking Domain In the stacking domain, a simulated robot arm learns to build towers out of adversarially weighted blocks. The observed state, $\mathcal{X}_t$ is the shape and color of each block. The object-level latent variables, $\mathcal{Z}$, will need to represent the mass and center of mass of each block which are not visually observable. The robot's action space consists of parameterized pick-and-place actions where the parameters include which block to pick and its final position. The outcome is a Bernoulli random variable, y_t, which describes whether the robot actually ends up in the state expected by the action or not (this is the notion of *feasibility* described in [8]). In our case, this corresponds to predicting if a tower is stable. The global model, θ, is represented using the same graph neural network architecture as in Noseworthy et al. [8]. The training phase uses 50 blocks and 2500 total towers.

To investigate how well the learned latent space captures center of mass, we design a downstream task that requires detailed knowledge of this property: *maximum overhang*. The robot must place the novel block on top of another block such that the distance between the base of the bottom block and the farthest edge of the top block is maximized. We disincentivize the robot from building unstable towers by giving the robot a large negative reward of $-1m$ if the tower falls. Thus, given our model which predicts stability, the expected reward, R, is,

$$R = -1 * (1 - \mathbb{E}_{z\sim q_z,\, \theta\sim q_\theta}\left[p(y_t|\mathcal{X}_t, a_t, \mathcal{Z}, \theta)\right]) + r * \mathbb{E}_{z\sim q_z,\, \theta\sim q_\theta}\left[p(y_t|\mathcal{X}_t, a_t, \mathcal{Z}, \theta)\right],$$

where r is the reward the robot would receive if the tower is stable. The robot is only tested on towers with two blocks as this most clearly elicits the effects of the latent space (the model is trained on towers with $2-5$ blocks).

Throwing Domain In the throwing domain, a simulated robot arm learns to throw a variety of balls with an obstacle present. Each ball varies in its radius and rolling resistance. However, both these properties are not observed, and the model will need to use the object-level latent space, $\mathcal{Z}$, to represent them. The action space of the robot consists of a throwing primitive parameterized by the release angle and angular velocity of the ball. The outcomes, $y_t \in \mathbb{R}$, capture how far the balls comes to rest along the x-axis. The dynamics function is represented as a standard multilayer perceptron. In the testing phase, the robot's goal is throw the ball such that it comes to rest at a desired distance. We report the RMSE averaged over many different goal distances.

5 Evaluation on Downstream Tasks

We now evaluate the robot's ability to rapidly generalize to new objects and its performance robustness in downstream tasks.

Rapid Adaptation To measure how quickly different methods can adapt to novel objects, we plot task performance after each data point was collected with a given acquisition method (see Figure 3). In the stacking domain (left), information-theoretic data acquisition consistently outperforms the random strategy, successfully adapting after only 5 placements. This is because figuring out the center of mass accurately requires several consecutive and carefully planned placements. On the other hand, in the throwing domain, both random and active achieve similar good performance. We believe this is due to the smoother relationship between the actions, latents, and outcome, meaning most throws are going to be similarly informative about the

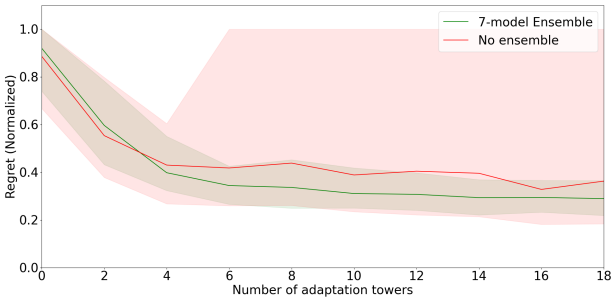


Figure 2: Task performance in the stacking domain when using a single neural network as opposed to a full ensemble.

latents. In both domains, we also compare to a baseline where we adapt a deep ensemble without latent variables at test time. Concretely, the baseline is a deep ensemble with the same network architecture as the dynamics ensemble in our object factored model. During the adaptation phase, active learning using ensembles is applied to generate experiments with the new object [2, 8]. The baseline fails to adapt quickly in each domain.

Ensemble Robustness The proposed model represents uncertainty at two levels: at the object level through $\mathcal{Z}$ and at the global level through θ. We show that the ensemble is important for having robust performance in noisy domains by ablating the ensemble and using a single network for θ in the stacking domain (see Figure 2). Without the ensemble, the model does not consistently perform well, often receiving a regret of 1 (indicating the tower fell over). On the other hand, the ensemble leads to consistently good performance, showing the utility of the uncertainty-aware representation for planning in downstream tasks.

6 Related Works

Several previous works have introduced Bayesian models for rapid adaptation. For example, Doshi-Velez and Konidaris [4] and Sæmundsson et al. [11] both use latent variables to parameterize a task and use Gaussian processes as the global dynamics. Closely related to our model, Perez et al. [9] and Belkhale et al. [1] use both deep ensembles and low-dimensional latent variables within their models. We extend this framework to show that learned latent space can be used for active learning and analyze the importance of the ensemble. Other work has focused on learning object dynamics with unobservable intrinsic object properties. In some works, the authors assume the global dynamics are known *a priori* which can inform the values of the object-specific properties [12]. Related work which does not assume the dynamics are known often rely on either a fixed dataset [13, 14] at adaption time or a task definition in order to infer the latent properties [4, 10, 11]. Instead, we propose to use active learning with respect to the model output which can be beneficial when the task is not yet known. Our approach can yield zero-shot performance on a new task, because all the experimentation was performed in a task-agnostic adaptation phase.

References

[1] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine. Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads. *IEEE Robotics and Automation Letters*, 6(2), 2021.

[2] W. H. Beluch, T. Genewein, A. Nurnberger, and J. M. Kohler. The Power of Ensembles for Active Learning in Image Classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[3] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, 2018.

[4] F. Doshi-Velez and G. Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2016.

[5] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia. Relational inductive bias for physical construction in humans and machines. In *the Annual Meeting of the Cognitive Science Society*, 2018.

[6] N. Houlsby, F. Huszar, Z. Ghahramani, and M. Lengyel. Bayesian Active Learning for Classification and Preference Learning, 2011.

[7] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2017.

[8] M. Noseworthy, C. Moses, I. Brand, S. Castro, L. Kaelbling, T. Lozano-Pérez, and N. Roy. Active learning of abstract plan feasibility. In *Proceedings of Robotics Science and Systems*, 2021.

[9] C. Perez, F. Such, and T. Karaletsos. Generalized Hidden Parameter MDPs Transferable Model-based RL in a Handful of Trials. In *Proceedings of The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[10] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[11] S. Sæmundsson, K. Hofmann, and M. Deisenroth. Meta reinforcement learning with latent variable gaussian processes. In *Proceedings of Uncertainty in Artificial Intelligence*, 2018.

[12] J. Wu, J. Lim, H. Zhang, J. Tenenbaum, and W. Freeman. Physics 101: Learning Physical Object Properties from Unlabeled Videos. In *Procedings of the British Machine Vision Conference 2016*, 2016.

[13] Z. Xu, J. Wu, A. Zeng, J. Tenenbaum, and S. Song. DensePhysNet: Learning Dense Physical Object Representations via Multi-step Dynamic Interactions. In *Proceedings of Robotics: Science and Systems*, 2019.

[14] D. Zheng, V. Luo, J. Wu, and J. Tenenbaum. Unsupervised Learning of Latent Physical Properties Using Perception-Prediction Networks. In *Proceedings of Uncertainty in Artificial Intelligence*, 2018.

Teaching categories to human semi-supervised learners

Franziska Bröker
Deparment of Computational Neuroscience
Max Planck Institute for Biological Cybernetics
Tübingen, Germany
Gatsby Computational Neuroscience Unit
UCL
London, UK
franziska.broeker@tuebingen.mpg.de

Brett Roads
Deparment of Experimental Psychology
UCL
London, UK
b.roads@ucl.ac.uk

Peter Dayan*
Deparment of Computational Neuroscience
Max Planck Institute for Biological Cybernetics
Tübingen, Germany
University of Tübingen
Tübingen, Germany
dayan@tue.mpg.de

Bradley Love *
Deparment of Experimental Psychology
UCL
London, UK
b.love@ucl.ac.uk

Abstract

Teaching involves a mixture of instruction, self-studying in the absence of a teacher and assessment that provides corrective feedback. Despite the mixture of supervised and unsupervised learning in the real-world, the literature on human learning has traditionally focused on one or the other, or reinforcement learning. By contrast, literature on semi-supervised learning is surprisingly recent, conflicting and sparse. Reports about the benefit of unsupervised information in category learning conflict across experimental designs, leading researchers to conclude that its effects on learning may be minimal at best. Here, we adopt a machine teaching approach to create a targeted test of the effects of unsupervised information in a simple categorization task. Taking two paradigmatic models of semi-supervised category learning (prototype and exemplar models), we infer that the sequential difficulty of the unsupervised items affects learning in the models due to self-reinforcement of beliefs. Critically, the models make opposite predictions as to whether unsupervised items should optimally be ordered from easy-to-hard or hard-to-easy, setting the stage for an empirical test. We find that hard-to-easy ordering leads to better task performance, consistent with the predictions of the prototype model. However, additional analyses show that the model predicts this result for reasons that are not consistent with human data, as it underestimates performance drops in the face of hard items. In sum, our machine teaching approach revealed novel evidence that ordering of unsupervised information affects category learning and highlighted shortcomings of existing semi-supervised categorization models. Future work will help understand semi-supervised learning principles and their connections with results on supervised easy-to-hard schedules and training set idealization. This has the potential to help improve teaching curricula.

Keywords: category learning, semi-supervised learning, machine teaching, computational modeling, online behavioural experiment

Acknowledgements

We thank Mihaly Banyai for reviewing the code associated with the models in this abstract. This work was supported by the Gatsby Charitable Foundation (FB); the Max Planck Society (FB, PD); the Alexander von Humboldt Foundation (PD); Wellcome Trust Investigator Award WT106931MA (BR, BCL), and Royal Society Wolfson Fellowship 183029 (BR, BCL).

*equal contributions

1 Central Question

We investigate whether unsupervised trials can help or harm category learning. Specifically, we test which semi-supervised sequences are optimally suited to help human learners acquire categories in a one-dimensional task.

2 Approach

To study how humans can be taught on semi-supervised inputs such that they optimally learn categorizations, we employed a machine teaching approach. Three semi-supervised category models have been proposed and evaluated against human data [Zhu et al., 2010]. Here, we employed exemplar and prototype models as candidate models for human learning because only they make appropriate conceptual assumptions for our task. To understand which semi-supervised sequences each model predicts to be most helpful to human learning, we optimized the order of presentation of unsupervised trials in training sequences using a simple genetic algorithm. This revealed opposite predictions as to whether learning is most helped if unsupervised trials are ordered from easy-to-hard (exemplar, Fig.2a) or hard-to-easy (prototype, Fig.2c) when interspersed with supervised trials. We then performed an empirical study on Amazon Mechanical Turk to test whether humans indeed conform to one of the models and improve their performance when trained on one of these ordered sequences. For this, we employed three experimental conditions in which unsupervised trials were ordered from easy-to-hard, hard-to-easy or were uniformly distributed (Fig.1c).

Our approach makes two important contributions: (a) The model-based approach to the design of optimal training schedules increases the chances of observing interpretable group differences – these are notoriously elusive in semi-supervised studies. (b) Computational models are put to a more comprehensive test via a range of extreme (best case and worst case) rather than just random input sequences.

3 Models

We predicted optimal training sequences under two category learning models that belong to well-studied model classes that make different assumptions about how category information is represented, manipulated and stored. Exemplar models assume that all items are stored individually and all contribute to category decisions. Prototype models assume that the information available in the observations is compressed and retained only as a representation of prototypical averages for each category. In their semi-supervised formulations proposed by Zhu et al. [2010], the traditionally supervised learning mechanism in the models is augmented with self-training by which models learn from their own predictions, in lieu of actual labels, when no supervision is available. In addition, models were simplified and reformulated in terms of machine classification algorithms. Most notably, the prototype model was formulated as a Bayesian Mixture of Gaussian model tracking not only prototype means but also their variance. The interplay of the specific model features leads to the different predictions about optimal orderings of unsupervised trials. Learning in the exemplar model is most helped if unsupervised trials are ordered from easy-to-hard (Fig.2a) while the prototype model prefers an ordering from hard-to-easy (Fig.2c). Important model features that contribute to these predictions are the inability of the exemplar model to correct wrongly predicted category labels committed to memory and the estimation of variances in the (Bayesian) prototype model.

4 Methods and Materials

A total number of 510 subjects completed the experiment via Amazon Mechanical Turk. Subjects were randomly assigned to one of the three experimental conditions. After automatic exclusion 309 subjects were analyzed (easy-to-hard: 103, uniform: 102, hard-to-easy: 104). Subjects completed a total of 154 trials in which they learned to categorize one-dimensional visual stimuli into two categories whilst only occasionally receiving corrective feedback (Fig.1 a and b). The experiment contained five different phases that were unsignalled to subjects: supervised initialization, unsupervised initialization, pre-test, main phase, and post-test (Fig.1 c). All phases, apart from the main phase, were identical between conditions. Stimulus sequences were randomly sampled for every individual subject.

5 Results

5.1 Empirical results

Our empirical results show that subjects in the hard-to-easy condition performed better on the post-test than in the easy-to-hard condition ($p = 0.022$; see Fig.2e). Further, we observed different performance trajectories between conditions over the course of the training phase. This was evident in subjects' change in accuracy between the main phase and post-test, which was significantly lower for the easy-to-hard condition than for the hard-to-easy and uniform condition

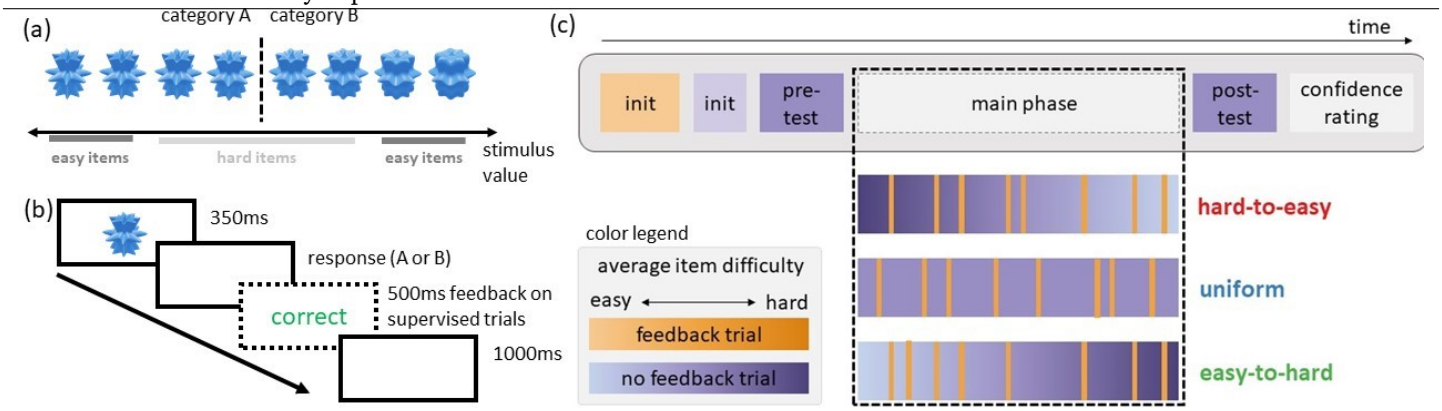


Figure 1: (a) Subjects performed a one-dimensional visual categorization task. Stimuli were 3D objects that varied in inflatedness. We define the 50% of items furthest away from the category boundary as easy items and the 50% of items closest to the boundary as hard items (only a subset of these items is shown here). (b) On each trial, subjects were presented with a stimulus to which they responded with a key press to indicate their categorization choice. On supervised trials, subjects then received corrective feedback. On unsupervised trials, they instead moved to the next trial directly. (c) The experiment consisted of five consecutive phases in which subjects received different amounts of corrective feedback: supervised and unsupervised initialization phases to anchor both categories on easy items far from the boundary were followed by a pre-test, the main semi-supervised training phase and a post-test. Subjects were also asked for one overall confidence rating at the end. The three experimental conditions involved exactly the same items and differed only in the ordering of easy and hard unsupervised trials (purple) which were randomly interspersed with supervised trials of all difficulties (orange) during the main training phase.

($p = 0.001$ and $p = 0.034$ respectively). Performance change on easy and hard items was also qualitatively different between groups (Fig.2f). In particular, it can be seen that improvement or worsening on easy and hard items is of similar magnitude within each condition. This suggests that sequence ordering affected performance across all levels of item difficulty rather than just the steepness of the category decision curve.

Additional analyses of performance change over the course of the main phase provide a measure of evidence that these signatures may be due to recency effects of exposure to hard trials, which appear to negatively affect performance on all item difficulties: at the beginning of the second half of the main phase, the easy-to-hard condition demonstrates a significant effect of learning, evident as higher performance on the same items that the hard-to-easy condition viewed earlier in learning (easy items: $p = 0.03$; hard items: $p = 0.007$). However, this learning effect vanishes in the face of increasingly many hard items towards the end of the main phase ($p > 0.15$ for both easy and hard items). That performance on easy and hard items appears to be equally affected suggests that the overall decline in performance in the easy-to-hard condition may be due to some form of confusion (for example because of noisy memory retrieval) rather than a pure learning effect.Furthermore, we find no evidence for motivational effects to have caused group differences: Neither subjects' self-assessment of their overall confidence in their performance (all $p > 0.1$) nor drop out rates (all $p > 0.3$) was significantly different between groups.

5.2 Modelling results

Comparing the empirical results to the theoretical predictions of the models (Fig.2a-d), it is clear that only the prototype model, but not the exemplar model, predicted the superior performance in the hard-to-easy condition at test. However, closer investigation of the prototype model's predictions about performance change revealed qualitative patterns at odds with those observed in the data. For example, the data shows that subjects improved or worsened equally on easy and hard items in each condition (Fig.2f). By contrast, the model predicts that subjects in the easy-to-hard condition improve significantly more on easy than on hard items (Fig. 2d). That is, the model could not account for the overall performance drop between main phase and post-test in the easy-to-hard condition which appears to have been introduced by the hard items at the end of training.

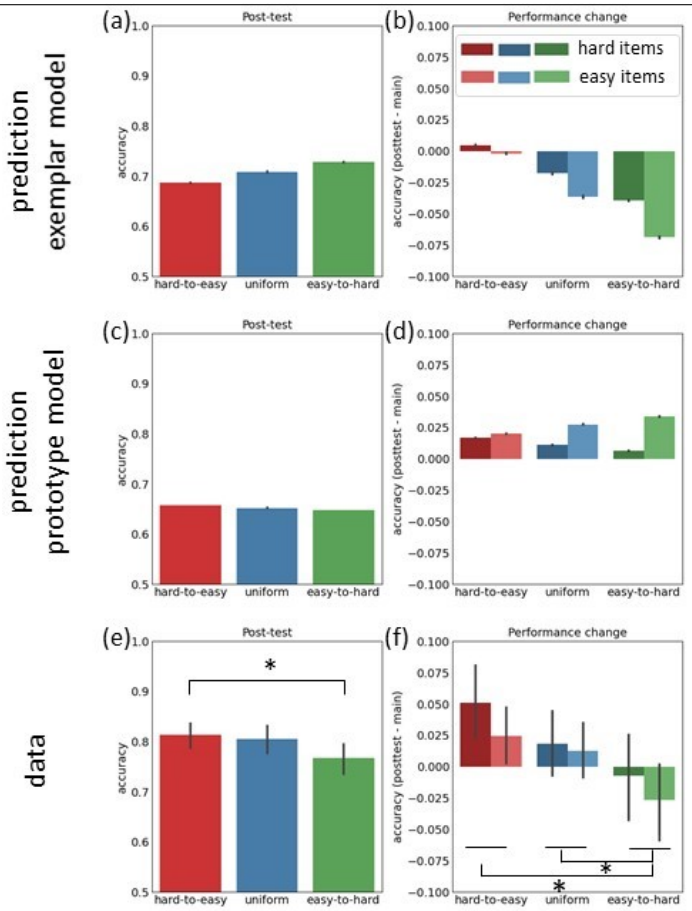


Figure 2: (a) The exemplar model predicts that subjects should perform best on the post-test in the easy-to-hard condition and worst in the hard-to-easy condition. (b) The exemplar model predicts that subjects' performance should drop from the main phase to post-test in all but the hard-to-easy condition. (c) The prototype model predicts the opposite post-test pattern compared to the exemplar model. (d) The prototype model predicts that subjects' performance should increase from the main phase to post-test (especially on easy items). (e) Subjects in the hard-to-easy condition performed significantly better on the post-test than the easy-to-hard condition. (f) Subjects' average change in accuracy between the main phase and post-test was significantly lower for the easy-to-hard condition than for the other two conditions. Qualitatively, the improvement or worsening on easy and hard items appears concurrent across conditions.

6 Conclusion

We employed a model-based study design that used machine teaching to generate empirical insights into the optimal teaching of human learners. We provide novel evidence that unsupervised exposure does affect category learning and that this can be caused by the ordering of unsupervised input alone. These insights are significant since previous work on semi-supervised category learning primarily focussed on distributional aspects of the input and reported conflicting results as to whether humans can learn from unsupervised trials at all. Our data also challenges existing semi-supervised learning models which were unable to account for behavioural signatures revealed in the human data. Thus, future modeling work will be needed to capture learning more accurately. This may also reveal important connections to supervised studies on easy-to-hard ordering and training set idealization which may account for the various results reported. Our approach at the intersection of machine and human teaching appears to be a promising direction that can help understand the underlying principles of semi-supervised learning better and can thus also help to improve teaching and learning in educational settings in the future.

References

Xiaojin Zhu, Bryan R Gibson, Kwang-Sung Jun, Timothy T Rogers, Joseph Harrison, and Chuck Kalish. Cognitive models of test-item effects in human category learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 1247–1254, 2010.

Task-Agnostic Continual Reinforcement Learning: In Praise of a Simple Baseline

Massimo Caccia*
Mila - Quebec AI Institute
Université de Montréal
Amazon Web Services

Jonas Mueller
Amazon Web Services

Taesup Kim
Amazon Web Services

Laurent Charlin
Mila - Quebec AI Institute
HEC Montréal
Canada CIFAR AI Chair

Rasool Fakoor
Amazon Web Services

Abstract

We study task-agnostic continual reinforcement learning (TACRL) in which standard RL challenges are compounded with *partial observability* stemming from task agnosticism, as well as additional difficulties of continual learning (CL), i.e., learning on a non-stationary sequence of tasks. Here we compare TACRL methods with their soft upper bounds prescribed by previous literature: multi-task learning (MTL) methods which do not have to deal with non-stationary data distributions, as well as task-aware methods, which are allowed to operate under *full observability*. We consider a previously unexplored and straightforward baseline for TACRL, replay-based recurrent RL (3RL), in which we augment an RL algorithm with recurrent mechanisms to address partial observability and experience replay mechanisms to address catastrophic forgetting in CL.

Studying empirical performance in a sequence of RL tasks, we find surprising occurrences of 3RL matching and overcoming the MTL and task-aware soft upper bounds. We lay out hypotheses that could explain this inflection point of continual and task-agnostic learning research. Our hypotheses are empirically tested in continuous control tasks via a large-scale study of the popular multi-task and continual learning benchmark Meta-World. By analyzing different training statistics including gradient conflict, we find evidence that 3RL's outperformance stems from its ability to quickly infer how new tasks relate with the previous ones, enabling forward transfer.

1 Introduction

Continual learning (CL) creates models and agents that can learn from a sequence of tasks. Continual learning agents promise to solve multiple tasks and adapt to new tasks without forgetting the previous one(s), a major limitation of standard learning agents [16, 50, 37, 30]. In many studies, the performance of CL agents is compared against *multi-task* (MTL) agents that are jointly trained on all available tasks. During learning and evaluation, these multi-task agents are typically provided with the identity of the current task (e.g. each datum is coupled with its task ID). The performance of multi-task agents is thought to provide a soft upper bound on the performance of continual learning agents since the latter are restricted to learn from tasks in a given sequential order that introduces new challenges, in particular *catastrophic forgetting* [37]. Moreover, continual-learning agents are often trained without knowing the task ID, a challenging setting motivated by practical constraints and known as *task-agnostic* CL [65, 20, 5, 4].

This paper considers continual learning agents that learn using reinforcement learning (RL), i.e. the setting of *continual* reinforcement learning (CRL) or *lifelong* reinforcement learning [23, 45, 46, 2]. We study the task-agnostic continual reinforcement learning (TACRL) setting in challenging robotic manipulation tasks from Meta-World (see top of Fig. 4).

We find two observations challenging common beliefs in CL. First, we surprisingly discover that TACRL agents endowed with a recurrent memory can outperform task-aware agents. We refer to this methodology as *replay-based recurrent reinforcement learning* (3RL) (see bottom of Fig. 4 for a visualization of its representations).

*corresponding author, email: massimo.p.caccia@gmail.com

We report a second surprising discovery in which 3RL reaches the performance of its multi-task upper bound (as well as other multi-task RL methods), despite only being exposed to the tasks in a sequential fashion. Conducting a large-scale empirical study[1] in which many RL methods are compared in different regimes, we explore several hypotheses to understand the factors underlying these results. Our study indicates that 3RL: 1) quickly infers how new tasks relate to the previous ones, enabling forward transfer as well as 2) learns representations of the underlying MDPs that reduce *task interference*, i.e., when the gradients of different tasks are in conflict [63].

These findings question the need for forgetting-alleviating and task-inference tools when we bring CL closer to some of its real-world applications, i.e., TACRL on a diverse sequence of challenging and inter-related tasks. While it is conventional to assume that task-agnostic continual RL is strictly more difficult than task-aware multi-task RL, this may not actually be the case for representative multi-task RL benchmarks like Meta-World. Despite being far more broadly applicable, TACRL methods may nonetheless be just as performant as their task-aware and multi-task counterparts. Note that the current manuscript is a shortened version of a longer one. We have placed the complete sections in the Appendix and have summarized them here.

2 Background Task-agnostic Continual Reinforcement Learning

TACRL agents operate in a POMDP special case, explained next, designed to study the *catastrophic forgetting* that plagues neural networks [37] when they learn on non-stationary data distributions, as well as *forward transfer* [58], i.e., a method's ability to leverage previously acquired knowledge to improve the learning of new tasks [35]. First, TACRL's environments assume that the agent does not have a causal effect on s^h. This assumption increases the tractability of the problem. It is referred to as an hidden-mode MDP (HM-MDP) [9]. Table 1 provides its mathematical description.

The following assumptions helps narrow down on the forgetting problem and knowledge accumulation abilities of neural networks. TACRL's assumes that s^h follows a non-backtracking chain. Specifically, the hidden states are locally stationary and are never revisited. Finally, TACRL's canonical evaluation reports the anytime performance of the methods on all tasks, which we will refer to as *global return*. In this manner, we can tell precisely which algorithm has accumulated the most knowledge about all hidden states at the end of its *life*. The hidden state is often referred to as *context*, but more importantly in CRL literature, it represents a *task*. As each context can be reformulated as a specific MDP, we treat tasks and MDP as interchangeable.

For the full section, including a review of MDP, POMDP, as well as the task-awareness and multi-task soft upper bounds, see App. A.

3 Methods

In this section, we explain how an RNN can be used for task-agnostic recurrent modeling as well as Experience Replay, CL's most durable basline. We then describe the baseline at the center of this work, replay-based recurrent RL (3RL).

Task-agnostic recurrent modeling (RNN). Recurrent neural networks are able to encode the history of past data [33, 56, 3, 14, 40]. Their encoding can implicitly identify different tasks (or MDP). Thus, we introduce RNNs as a history encoder where history is defined by $\{(s_i, a_i, r_i)\}_i^N$ and we utilize the hidden states z as additional input data for the actor $\pi_\phi(a|s, z)$ and critic $Q_\theta(s, a, z)$ networks. This allows us to train models without any explicit task information, and therefore we use this modeling especially for task-agnostic continual learning. More details about the RNN are provided at the end of the next subsection.

Experience Replay (ER) accumulates data from previous tasks in a buffer for retraining purposes, thus slowing down forgetting [46, 1, 8, 28]. Although simple, it is often a worthy adversary for CL methods. One limitation of replay is that, to approximate the data distribution of all tasks, its compute requirements scale linearly with the number of tasks. To alleviate this problem, we use a strategy that caps replay by oversampling the current task from the buffer explained in Alg. 1 L8-9.

Replay-based Recurrent RL (3RL) A general approach to TACRL is to combine ER—one of CL's most versatile baseline—with an RNN, one of RL's most straightforward approach to handling partial observability. We refer to this baseline as *replay-based reccurent RL* (3RL). As an episode unfolds, 3RL's RNN representations $z_t = \text{RNN}(\{(s_i, a_i, r_i)\}_{i=1}^{t-1})$ should predict the task with increasing accuracy, thus helping the actor $\pi_\theta(a|s, z)$ and critic $Q_\phi(s, a, z)$ in their respective approximations. We will see in Sec. 4, however, that the RNN delivers more than expected: it enables forward transfer by decomposing new tasks and placing them in the context of previous ones. We provide pseudocode for 3RL in Alg. 1,

[1] The codebase to reproduce the results will be made available upon publication.

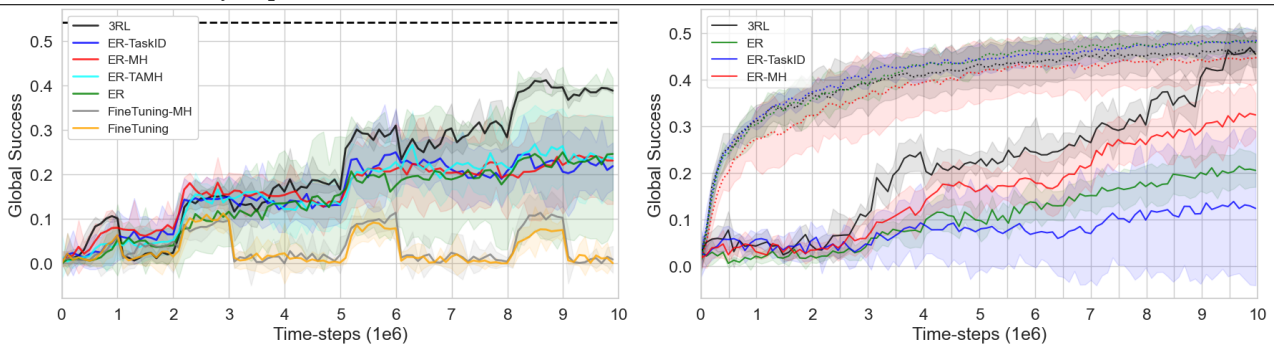


Figure 1: **3RL outperforms all baselines in both `CW10` (left) and `MW20` (right). Furthermore, 3RL reaches the performance of its multi-task analog (right)**. The horizontal line (left) is the reference performance of training independent models on all tasks. The dotted lines (right) represent the methods' analogous multi-task baselines. 3RL not only improves on its task-awareness soft upper bound, it also reaches its multi-task one.

which we kept agnostic to the base algorithm and not tied to episodic RL. Note that in our implementation, the actor and critics enjoy their own RNNs, as in [14, 40]: they are thus parameterize by θ and ϕ, respectively.

For the full section including a review of the base algorithm (soft-actor critic) and details about other baselines employed in the experiments, see App. B.

4 Empirical Findings

We now investigate some alluring behaviours we have come upon, namely that replay-based recurrent reinforcement learning (3RL), a task-agnostic continual reinforcement learning (TACRL) baseline, can outperform other task-agnostic but more importantly task-aware baselines, as well as match its MTL soft upper bound. Details about the benchmarks and experiments ares provided in App. E.

Task Agnosticism overcomes Task Awareness Our first experiments are conducted on `CW10` and `MW20` and are reported in Fig. 1. Interestingly, 3RL outperforms all other methods in both benchmarks. This is surprising for two reasons. First, at training time, the task-aware methods learn task-specific parameters to adapt to each individual task. Hence, they suffer less or no forgetting (in general). Second, at test time, the task-agnostic method has to first infer the task through exploration, at the expense of exploitation (see bottom of Fig. 4 for a visualization).

Continual Learning can Match Multi-Task Learning Multi-task learning is often used as a soft upper bound in evaluating CL methods in both supervised [1, 35] and reinforcement learning [46, 52, 58]. The main reason is that in the absence of additional constraints, multi-task learning (jointly training with data from all tasks in a stationary manner) does not suffer from the catastrophic forgetting that typically plagues neural networks trained on non-stationary data distributions.

3RL can reach this multi-task learning upper bound. In Fig. 1 we report, the results of the `MW20` experiments, but this time we focus the performance of each method's multi-task analog, i.e. their soft-upper bound (dotted line of the same color). 3RL, is the only approach that matches the performance of its MTRL equivalent. We believe it is the first time that a specific method achieves the same performance in a non-stationary task regime compared to the stationary one, amidst the introduced challenges like catastrophic forgetting.

For the remainder of the section, we investigate some hypotheses that might explain 3RL's alluring behavior. We first hypothesise that the RNN boosts performance because it is simply better at learning a single MDP (Hypothesis #1). This hypothesis is falsified in App. E. Next, we investigate the hypothesis that 3RL reduces parameter movement, as it is often a characteristic of successful continual-learning methods (Hypothesis #2). Similarly, this hypothesis is falsified in App. E. We then explore the hypothesis that the RNN correctly places the new tasks in the context of previous ones (Hypothesis #3). We discuss one last hypothesis in App. M.

Hypothesis #3: RNN correctly places the new tasks in the context of previous ones, enabling forward transfer and improving optimization As in real robotic use-cases, MW tasks share a set of low-level reward components like grasping, pushing, placing, and reaching, as well as set of object with varying joints, shapes, and connectivity. As the agent experiences a new task, the RNN could quickly infer how the new data distribution relates with the previous ones and provide to the actor and critics a useful representation. Assume the following toy example: task one's goal is to grasp

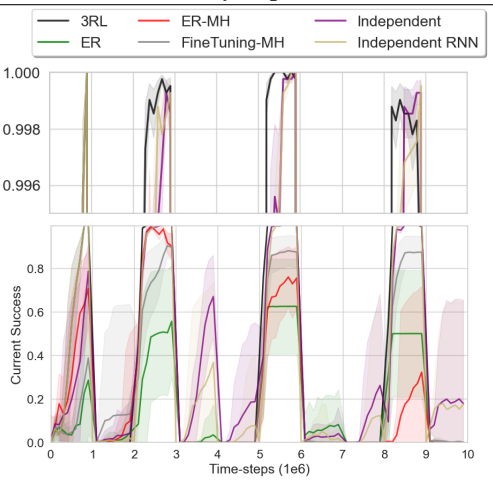


Figure 2: **3RL is the fastest learner.** Current success rate on CW10. The Independent method, which trains on each task individually, is still the best approach to maximise single-task performance. However, on the task that the continual-learning methods succeed at, 3RL is the fastest learner. In these cases, its outperformance over Independent and Independent RNN indicates that forward transfer is achieved.

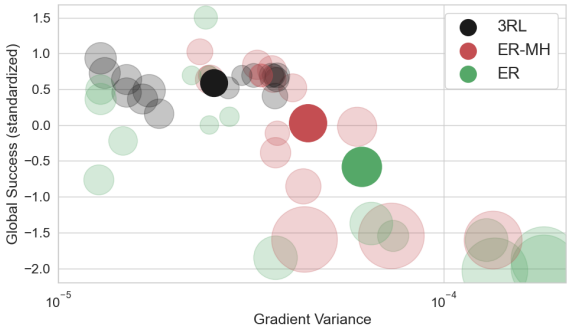


Figure 3: **3RL decreases gradient conflict leading to an increase in training stability and performance.** The global success and gradient as measured by the variance of the gradients are shown are plotted against each other. Training instability as measured by the variance of the Q-values throughout learning is represented by the markers' size, in a log-scale. Transparent markers depict seeds, whereas the opaque one the means. We observe a negative correlation between performance and gradient conflict (-0.75) as well as performance and training stability (-0.81), both significant under a 5% significance threshold. The hypothesis is that 3RL improves performance by reducing gradient conflict via dynamic task representations.

a door handle, and task two's to open a door. The RNN could infer from the state-action-reward trajectory that the second task is composed of two subtasks: the first one as well as a novel pulling one. Doing so would increase the policy learning's speed, or analogously enable forward transfer.

Now consider a third task in which the agent has to close a door. Again, the first part of the task consists in grasping the door handle. However, now the agent needs to subsequently push and not pull, has was required in task two. In this situation, task interference [63] would occur. Once more, if the RNN could dynamically infer from the context when pushing or pulling is required, it could modulate the actor and critics to have different behaviors in each tasks thus reducing the interference. Note that a similar task interference reduction should be achieved by task-aware methods. E.g., a multi-head component can enable a method to take different actions in similar states depending on the tasks, thus reducing the task interference.

Observing and quantifying that 3RL learns a representation space in which the new tasks are correctly *decomposed* and placed within the previous ones is challenging. Our initial strategy is to look for effects that should arise if this hypothesis was true (so observing the effect would confirm the hypothesis).

First, we take a look at the time required to adapt to new tasks: if 3RL correctly infers how new tasks relate to previous ones, it might be able to learn faster by re-purposing learned behaviors. Fig. 8 depicts the current performance of different methods throughout the learning of CW10. For reference, we provide the results of training separate models, which we refer to as Independent and Independent RNN. The challenges of Meta-World v2 compounded with the ones from learning multiple policies in a shared network, and handling an extra level of non-stationary, i.e. in the task distribution, leaves the continual learners only learning task 0, 2, 5, and 8. On those tasks (except the first one in which no forward transfer can be achieved) 3RL is the fastest continual learner. Interestingly, 3RL showcases some forward transfer by learning faster than the Independent methods on those tasks. This outperformance is more impressive when we remember that 3RL is spending most of its compute replaying old tasks, and supports Hypothesis #3.

Second, simultaneously optimizing for multiple tasks can lead to conflicting gradients or task interference [63]. To test for this effect, we use the variance of the gradients on the mini-batch throughout the training as a proxy of gradient conflict (see App. L for a discussion on why the gradient's variance is superior to the gradients' angle as a proxy for gradient conflict). In Fig. 9 we show the normalized global success metric plotted against the gradient variance. In line with our intuition, we do find that the RNN increases gradient agreement over baselines. As expected, adding a multi-head scheme can also help, to a lesser extent. We find a significant negative correlation of -0.75 between performance and gradient conflict. fig:stability also reports training stability, as measured by the standard deviation of Q-values throughout training (not to be confused with the parameter stability, at the center of Hypothesis #2, which measures how much the parameters move around during training). We find 3RL enjoys more stable training as well as a the significant negative correlation of -0.81 between performance and training stability.

We wrap up the hypothesis with some qualitative support for it. fig:fig1 showcases the RNN representations as training unfolds. If the RNN was merely performing task inference, we would observe the trajectories getting further from each other, not intersecting, and collapsing once the task is inferred. Contrarily, the different task trajectories constantly evolve and seem to intersect in particular ways. Although only qualitative, this observation supports the current hypothesis. A related work and conclusion are provided in App. N and App. O, respectively.

References

[1] R. Aljundi, L. Caccia, E. Belilovsky, M. Caccia, M. Lin, L. Charlin, and T. Tuytelaars. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019.

[2] H. B. Ammar, E. Eaton, P. Ruvolo, and M. Taylor. Online multi-task learning for policy gradient methods. In *International conference on machine learning*, pages 1206–1214, 2014.

[3] B. Bakker. Reinforcement learning with long short-term memory. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 1475–1482, Cambridge, MA, USA, 2001. MIT Press.

[4] G. Berseth, Z. Zhang, G. Zhang, C. Finn, and S. Levine. Comps: Continual meta policy search. *ArXiv*, abs/2112.04467, 2021.

[5] M. Caccia, P. Rodriguez, O. Ostapenko, F. Normandin, M. Lin, L. Page-Caccia, I. H. Laradji, I. Rish, A. Lacoste, D. Vázquez, et al. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[6] D. Calandriello, A. Lazaric, and M. Restelli. Sparse multi-task reinforcement learning. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in neural information processing systems 27*, pages 819–827. Curran Associates, Inc., 2014.

[7] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *European Conference on Computer Vision (ECCV)*, 2018.

[8] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. S. Torr, and M. Ranzato. Continual learning with tiny episodic memories. *ArXiv*, abs/1902.10486, 2019.

[9] S. P. Choi, D.-Y. Yeung, and N. L. Zhang. Hidden-mode markov decision processes for nonstationary sequential decision making. In *Sequence Learning*, pages 264–287. Springer, 2000.

[10] A. Cossu, A. Carta, V. Lomonaco, and D. Bacciu. Continual learning for recurrent neural networks: an empirical evaluation. *Neural Networks*, 143:607–627, 2021.

[11] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[12] B. Ehret, C. Henning, M. R. Cervera, A. Meulemans, J. Von Oswald, and B. F. Grewe. Continual learning in recurrent neural networks. *arXiv preprint arXiv:2006.12109*, 2020.

[13] R. Fakoor, P. Chaudhari, and A. J. Smola. P3o: Policy-on policy-off policy optimization. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115, pages 1017–1027. PMLR, 2020.

[14] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola. Meta-q-learning. In *International Conference on Learning Representations*, 2020.

[15] R. Fakoor and M. Huber. Improving tractability of pomdps by separation of decision and perceptual processes. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 593–598, 2012.

[16] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1861–1870. PMLR, 10–15 Jul 2018.

[19] R. Hadsell, D. Rao, A. Rusu, and R. Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24:1028–1040, 12 2020.

[20] X. He, J. Sygnowski, A. Galashov, A. A. Rusu, Y. W. Teh, and R. Pascanu. Task agnostic continual learning via meta learning. *ArXiv*, abs/1906.05201, 2019. 130

[21] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[23] K. Khetarpal, M. Riemer, I. Rish, and D. Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2017.

[25] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[26] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[27] A. Kumar, A. Gupta, and S. Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*, 2020.

[28] T. Lesort. Continual learning: Tackling catastrophic forgetting in deep neural networks with replay processes. 2020.

[29] T. Lesort, M. Caccia, and I. Rish. Understanding continual learning settings with data distribution drift analysis. *arXiv preprint arXiv:2104.01678*, 2021.

[30] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52 – 68, 2020.

[31] T. Lesort, A. Stoian, and D. Filliat. Regularization shortcomings for continual learning. *ArXiv*, abs/1912.03049, 2019.

[32] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[33] L.-J. Lin and T. M. Mitchell. Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on From Animals to Animats 2: Simulation of Adaptive Behavior: Simulation of Adaptive Behavior*, page 271–280, Cambridge, MA, USA, 1993. MIT Press.

[34] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. 1995.

[35] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[36] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. 2018.

[37] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[38] J. A. Mendez, B. Wang, and E. Eaton. Lifelong policy gradient learning of factored policies for faster training without forgetting. 2020.

[39] T. Nguyen and T. Obafemi-Ajayi. Recurrent network and multi-arm bandit methods for multi-task learning without task specification (abstract). In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[40] T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent model-free rl is a strong baseline for many pomdps. 2021.

[41] F. Normandin, F. Golemo, O. Ostapenko, P. Rodriguez, M. D. Riemer, J. Hurtado, K. Khetarpal, R. Lindeborg, L. Cecchi, T. Lesort, L. Charlin, I. Rish, and M. Caccia. Sequoia: A software framework to unify continual learning research. 2022.

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[43] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. USA, 1st edition, 1994.

[44] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv:1903.08254*, 2019.

[45] J. Ribeiro, F. S. Melo, and J. Dias. Multi-task learning and catastrophic forgetting in continual reinforcement learning. *arXiv preprint arXiv:1909.10008*, 2019.

[46] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[47] S. Sodhani, A. Zhang, and J. Pineau. Multi-task reinforcement learning with context-based representations. *arXiv preprint arXiv:2102.06177*, 2021.

[48] A. Y. Sorokin and M. S. Burtsev. Continual and multi-task reinforcement learning with shared episodic memory. *arXiv preprint arXiv:1905.02662*, 2019.

[49] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[50] S. Thrun and T. M. Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

[51] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[52] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, G. Cai, N. D. Rodríguez, and D. Filliat. Discorl: Continual reinforcement learning via policy distillation. *CoRR*, abs/1907.05855, 2019.

[53] G. M. van de Ven and A. S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[54] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. 2018.

[55] Z. Wang, C. Subakan, E. Tzinis, P. Smaragdis, and L. Charlin. Continual learning of new sound classes using generative replay. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 308–312. IEEE, 2019.

[56] S. D. Whitehead and L.-J. Lin. Reinforcement learning of non-markov decision processes. *Artificial Intelligence*, 73(1):271–306, 1995. Computational Research on Interaction and Agency, Part 2.

[57] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International conference on artificial neural networks*, pages 697–706. Springer, 2007.

[58] M. Wolczyk, M. Zajac, R. Pascanu, L. Kucinski, and P. Milos. Continual world: A robotic benchmark for continual reinforcement learning. *CoRR*, abs/2105.10919, 2021.

[59] M. Wolczyk, M. Zajac, R. Pascanu, L. Kucinski, and P. Milos. Continual world: A robotic benchmark for continual reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

[60] T. Wolf, J. Chaumond, and C. Delangue. Continuous learning in a hierarchical multiscale neural network. In *ACL*, 2018.

[61] T. Wu, M. Caccia, Z. Li, Y.-F. Li, G. Qi, and G. Haffari. Pretrained language model in continual learning: A comparative study. In *International Conference on Learning Representations*, 2021.

[62] R. Yang, H. Xu, Y. WU, and X. Wang. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems*, 33:4767–4777, 2020.

[63] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[64] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

[65] C. Zeno, I. Golan, E. Hoffer, and D. Soudry. Task agnostic continual learning using online variational bayes. 2019.

[66] Y. Zhang and Q. Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

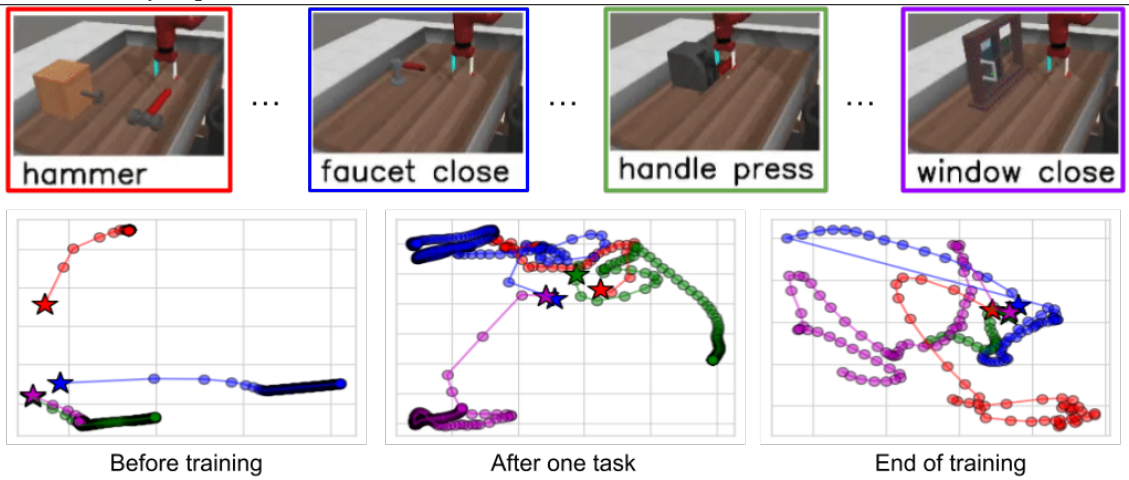


Figure 4: **The Continual-World[2]benchmark (top) as well as evolving RNN representations (bottom).** Continual World consists of 10 robotic manipulation environments (four of which are shown above) within the same state space and built on a common reward structure composed of shared components, i.e, reaching, grasping, placing and pushing. We explore the task-agnostic setting in which agents need a full trajectory (rather than a single state) for task identification. We performed a PCA analysis of 3RL's RNN representations at different stages of training. One episode is shown per task, and the initial task representation (at $t = 0$) is represented by a star. As training progresses, the model learns i) a task-invariant initialization, drawing the initial states closer together, and ii) richer, more diverse representations. Furthermore, the representations constantly evolve throughout the episodes, suggesting the RNN performs more than task inference: it provides useful local information to the policy and critic (see Hypothesis #3 in Sec. 4).

Appendix: Task-Agnostic Continual Reinforcement Learning: In Praise of a Simple Baseline

A Background & Task-agnostic Continual Reinforcement Learning

Here we formally define task-agnostic continual reinforcement learning (TACRL), and contrast it against multi-task RL as well as task-aware settings.

MDP. The RL problem is often formulated using a Markov decision process (MDP) [43]. An MDP is defined by the five-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$ with $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, $\mathcal{T}(s'|s,a)$ the transition probabilities, $r(s,a) \in \mathbb{R}$ the reward obtained by taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, and $\gamma \in [0,1)$ a scalar constant that discounts future rewards. In RL, the transition probabilities and the rewards are typically unknown and the objective is to learn a policy, $\pi(a|s)$ that maximizes the sum of discounted rewards $\mathcal{R}_t^\pi = \sum_{i=t}^{\infty} \gamma^{i-t} r_i = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$ generated by taking a series of actions $a_t \sim \pi(\cdot|s_t)$. The Q-value $Q^\pi(s,a)$ corresponding to policy π, is defined as the expected return starting at state s, taking a, and acting according to π thereafter:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_t\Big] = r(s,a) + \gamma \mathop{\mathbb{E}}_{s',a'}\Big[Q^\pi(s',a')\Big]$$

POMDP. In most real world applications, if not all, the full information about an environment or a task is not always available to the agent due to various factors such as limited and/or noisy sensors, different states with identical observations, object occlusion, etc. [34, 15]. For this class of problems in which environment states are not fully observable by the agent, partially-observable Markov decision processes (POMDPs) [22] are used to model the problem. A POMDP is defined by a seven-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{X}, \mathcal{O}, r, \gamma \rangle$ that can be interpreted as an MDP augmented with an observation space $\mathcal{X}$ and a observation-emission function $\mathcal{O}(x'|s)$. In a POMDP, an agent cannot directly infer the current state of the environment s_t from the current observation x_t. We split the state space into two distinct parts: the one that can can be directly unveiled from the observation x_t, which we refer to as s_t^o, and the remainder as the hidden state s_t^h, similarly to [40]. To infer the correct hidden state, the agent has to take its history into account: the policy thus becomes $\pi(a_t|s_{1:t}^o, a_{1:t-1}, r_{1:t-1})$.

[2]The figures depict the rendering of Meta-World, and not what the agent observes. The agent's observation space is mainly composed of object, targets and gripper position. Because of the randomness of those positions, the agents needs more than one observation to properly infer the hidden state.

An obvious choice to parameterize such a policy is with a recurrent neural network [33, 56, 3, 14, 40], as described in Sec. 3. Like MDPs, the objective in POMDP is to learn a policy that maximizes the expected return $\mathop{\mathbb{E}}_{s^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]|s^h\Big]$.

Task-agnostic Continual Reinforcement Learning (TACRL). TACRL agents operate in a POMDP special case, explained next, designed to study the *catastrophic forgetting* that plagues neural networks [37] when they learn on non-stationary data distributions, as well as *forward transfer* [58], i.e., a method's ability to leverage previously acquired knowledge to improve the learning of new tasks [35]. First, TACRL's environments assume that the agent does not have a causal effect on s^h. This assumption increases the tractability of the problem. It is referred to as an hidden-mode MDP (HM-MDP) [9]. Table 1 provides its mathematical description.

The following assumptions helps narrow down on the forgetting problem and knowledge accumulation abilities of neural networks. TACRL's assumes that s^h follows a non-backtracking chain. Specifically, the hidden states are locally stationary and are never revisited. Finally, TACRL's canonical evaluation reports the anytime performance of the methods on all tasks, which we will refer to as *global return*. In this manner, we can tell precisely which algorithm has accumulated the most knowledge about all hidden states at the end of its *life*. The hidden state is often referred to as *context*, but more importantly in CRL literature, it represents a *task*. As each context can be reformulated as a specific MDP, we treat tasks and MDP as interchangeable.

Awareness of the Task Being Faced In practical scenarios, deployed agents cannot always assume full observability, i.e. to have access to a *task label* or *ID* indicating which task they are solving or analogously which hidden state they are in. They might not even have the luxury of being "told" when the task changes (task boundary): agents might have to infer it themselves in a data-driven way. We call this characteristic *task agnosticism* [65]. Although impractical, CL research often treats *task-aware* methods, which observe the task label, as a soft upper bound to their task-agnostic counterpart [53, 65]. Augmented with task labels, the POMDP becomes fully observable, collapsing to an MDP problem.

Multi-task Learning (MTL) For neural network agents, catastrophic forgetting can be simply explained by the stationary data distribution assumption of stochastic gradient descent being violated, such that the network parameters become specific to data from the most recent task. Thus it is generally preferable to train on data from all tasks jointly as in MTL [66]. However this may not be possible in many settings, and thus CL is typically viewed as a more broadly applicable methodology that is expected to perform worse than MTL [46, 7].

For RL specifically, multi-task RL (MTRL) often refers to scenarios with families of similar tasks (i.e. MDPs) where the goal is to learn a policy (which can be contextualized on each task's ID) that maximizes returns across *all* the tasks [62, 6, 26]. While seemingly similar to CRL, the key difference is that MTRL assumes data from all tasks are readily available during training and each task can be visited as often as needed. These are often impractical requirements, which CRL methods are not limited by. Table 1 summarizes the settings we have discussed in this section.

	T	π	Objective	Evaluation
MDP [49]	$p(s_{t+1}\|s_t,a_t)$	$\pi(a_t\|s_t)$	$\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]$	-
POMDP [22]	$p(s^h_{t+1},s^o_{t+1}\|s^h_t,s^o_t,a_t)$	$\pi(a_t\|s^o_{1:t},a_{1:t-1},r_{1:t-1})$	$\mathop{\mathbb{E}}_{s^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$	-
HM-MDP [9]	$p(s^o_{t+1}\|s^h_{t+1},s^o_t,a_t)p(s^h_{t+1}\|s^h_t)$	$\pi(a_t\|s^o_{1:t},a_{1:t-1},r_{1:t-1})$	$\mathop{\mathbb{E}}_{s^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$	-
Task-agnostic CRL	$p(s^o_{t+1}\|s^h_{t+1},s^o_t,a_t)p(s^h_{t+1}\|s^h_t)$	$\pi(a_t\|s^o_{1:t},a_{1:t-1},r_{1:t-1})$	$\mathop{\mathbb{E}}_{s^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$	$\mathop{\mathbb{E}}_{\tilde{s}^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$
Task-Aware CRL	$p(s^o_{t+1}\|s^h_{t+1},s^o_t,a_t)p(s^h_{t+1}\|s^h_t)$	$\pi(a_t\|s^h_t,s^o_t)$	$\mathop{\mathbb{E}}_{s^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$	$\mathop{\mathbb{E}}_{\tilde{s}^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$
Multi-task RL	$p(s^o_{t+1}\|s^h_{t+1},s^o_t,a_t)p(s^h_{t+1})$	$\pi(a_t\|s^h_t,s^o_t)$	$\mathop{\mathbb{E}}_{\tilde{s}^h}\Big[\mathop{\mathbb{E}}_{\pi}\big[\sum_{t=0}^{\infty}\gamma^t r_t\big]\|s^h\Big]$	-

Table 1: Summarizing table of the settings relevant to TACRL. For readability purposes, $\tilde{s}^h$ denotes the stationary distribution of s^h. The Evaluation column if left blank when it is equivalent to the Objective one.

B Methods

In this section, we detail the base algorithm and different model architectures used for assembling different continual and multi-task learning baselines.

B.1 Algorithms

We use off-policy RL approaches which have two advantages for (task-agnostic) continual learning. First, they are more sample efficient than online-policy ones [17, 13]. Learning from lower-data regimes is important for CRL as a task is

likely to only be seen once as data comes in stream. Second, task-agnostic CRL most likely requires some sort of replay function [52, 31]. This is in contrast to task-aware methods which can, at the expense of computational efficiency, *freeze-and-grow*, e.g. PackNet [36], to incur no forgetting. Off-policy methods, by decoupling the learning policy from the acting policy, support replaying of past data. In short, off-policy learning is the approach of choice in CRL.[3]

Base algorithm The Soft Actor-Critic (SAC) [18] is an off-policy actor-critic algorithm for continuous actions. SAC adopts a maximum entropy framework that learns a stochastic policy which maximizes the expected return and also encourages the policy to contain some randomness. To accomplish this, SAC utilizes an actor/policy network π_ϕ and critic/Q network Q_θ, parameterized by ϕ and θ respectively. Q-values are learnt by minimizing one-step temporal difference (TD) error by sampling previously collected data from the replay buffer [32]. For more details on SAC, please look at App. C.

B.2 Models

We consider various architectures to handle multi-task learning (MTL) as well as continual learning (CL) in both task-aware and task-agnostic setting.

Task ID modeling (TaskID). We assume that a model such as SAC can become task adaptive by providing task information to the networks. Task information such as task ID (e.g. one-hot representation), can be fed into the critics and actor networks as an additional input: $Q_\theta(s, a, k)$ and $\pi_\phi(a|s, k)$ where k is the task ID. We refer to this baseline as Task ID modeling (TaskID)This method is applicable to both multi-task learning and continual learning.

Multi-head modeling (MH). For multi-task learning (which is always task-aware), the standard SAC is typically extended to have multiple heads [62, 64, 59, 63], where each head is responsible for a single distinctive task, i.e. $Q_\Theta = \{Q_{\theta_k}\}_k^K$ and $\pi_\Phi = \{\pi_{\phi_k}\}_k^K$ where K denotes total number of tasks. MH is also applicable to all reinforcement learning algorithms. That way, the networks can be split into 2 parts: (1) a shared state representation network (feature extractor) and (2) multiple prediction networks (heads). This architecture can also be used for task-aware CL, where a new head is newly attached (initialized) when an unseen task is detected during learning.

We also use this architecture in task-agnostic setting for both MTL and CL. Specifically, the number of heads is fixed a priori (we fix it to the number of total tasks) and the most confident actor head, w.r.t. the entropy of the policy, and most optimistic critic head are chosen. Task-agnostic multi-head (TAMH) can help us fraction the potential MH gains over the base algorithm: if MH and TAMH can improve performance, some of MH gains can be explained by its extra capacity instead of the additional task information.

Task-agnostic recurrent modeling (RNN). Recurrent neural networks are able to encode the history of past data [33, 56, 3, 14, 40]. Their encoding can implicitly identify different tasks (or MDP). Thus, we introduce RNNs as a history encoder where history is defined by $\{(s_i, a_i, r_i)\}_i^N$ and we utilize the hidden states z as additional input data for the actor $\pi_\phi(a|s, z)$ and critic $Q_\theta(s, a, z)$ networks. This allows us to train models without any explicit task information, and therefore we use this modeling especially for task-agnostic continual learning. More details about the RNN are provided at the end of the next subsection.

B.3 Baselines

FineTuning is a simple approach to a CL problem. It learns each incoming task without any mechanism to prevent forgetting. Its performance on past tasks indicates how much forgetting is incurred in a specific CL scenario.

Experience Replay (ER) accumulates data from previous tasks in a buffer for retraining purposes, thus slowing down forgetting [46, 1, 8, 28]. Although simple, it is often a worthy adversary for CL methods. One limitation of replay is that, to approximate the data distribution of all tasks, its compute requirements scale linearly with the number of tasks. To alleviate this problem, we use a strategy that caps replay by oversampling the current task from the buffer explained in Alg. 1 L8-9.

Multi-task (MTL) trains on all tasks simultaneously and so it does not suffer from the challenges arising from learning on a non-stationary task distribution. It serves as a soft upper bound for CL methods.

Independent learns a set of separate models for each task whereby eliminating the CL challenges as well as the MTL ones, e.g., learning with conflicting gradients [63].

[3] Note that the findings of this paper are not limited to off-policy methods, in fact our 3RL model can be extended to any on-policy method as long as it utilizes a replay buffer [13] Having the capability to support a replay buffer is more important than being on-policy or off-policy.

The aforementioned baselines are mixed-and-matched with the architectural choices to form different baselines, e.g. MTL with TaskID (MTL-TaskID) or FineTuning with MH (FineTuning-MH). At the core of this work lies a particular combination, explained next.

Replay-based Recurrent RL (3RL) A general approach to TACRL is to combine ER—one of CL's most versatile baseline—with an RNN, one of RL's most straightforward approach to handling partial observability. We refer to this baseline as *replay-based reccurent RL* (3RL). As an episode unfolds, 3RL's RNN representations $z_t = \text{RNN}(\{(s_i, a_i, r_i)\}_{i=1}^{t-1})$ should predict the task with increasing accuracy, thus helping the actor $\pi_\theta(a|s, z)$ and critic $Q_\phi(s, a, z)$ in their respective approximations. We will see in Sec. 4, however, that the RNN delivers more than expected: it enables forward transfer by decomposing new tasks and placing them in the context of previous ones. We provide pseudocode for 3RL in Alg. 1, which we kept agnostic to the base algorithm and not tied to episodic RL. Note that in our implementation, the actor and critics enjoy their own RNNs, as in [14, 40]: they are thus parameterize by θ and ϕ, respectively.

Algorithm 1: 3RL in TACRL

Environment: a set of K MPDs, allowed timesteps T
Input: initial parameters θ, empty FIFO replay buffers $\{\mathcal{D}^i\}_i^K$, replay cap β, batch size b, history length h

for *task* k *in* K **do**
  set environment to n^{th} MDP
  for *times-steps* t *in* T **do**
    /* Sampling stage */
    compute dynamic task representation $z_t = \text{RNN}_\theta(\{(s_i, a_i, r_i)\}_{i=t-h-1}^{t-1})$
    observe state s_t and execute action $a_t \sim \pi_\theta(\cdot|s_t, z_t)$
    observe reward r_t and next state s_{t+1}
    store (s_t, a_t, r_t, s_{t+1}) in buffer $\mathcal{D}^k$
    /* Updating stage */
    sample a batch B of $b \times min(\frac{1}{n}, 1-\beta)$ trajectories from the current replay buffer $\mathcal{D}^k$
    append to B a batch of $b \times min(\frac{n-1}{n}, \beta)$ trajectories from the previous buffers $\{\mathcal{D}^i\}_i^{k-1}$
    Compute loss on B and accordingly update parameters θ with one step of gradient descent

C Soft-Actor Critic

The Soft Actor-Critic (SAC) [18] is an off-policy actor-critic algorithm for continuous actions. SAC adopts a maximum entropy framework that learns a stochastic policy which not only maximizes the expected return but also encourages the policy to contain some randomness. To accomplish this, SAC utilizes an actor/policy network (i.e. π_ϕ) and critic/Q network (i.e. Q_θ), parameterized by ϕ and θ respectively. Q-values are learnt by minimizing one-step temporal difference (TD) error by sampling previously collected data from the replay buffer [32] denoted by $\mathcal{D}$.

$$\mathcal{J}_Q(\theta) = \mathop{\mathbb{E}}_{s,a}\Big[\Big(Q_\theta(s,a) - y(s,a)\Big)^2\Big],\ a' \sim \pi_\phi(\cdot|s') \tag{1}$$

where $y(s, a)$ is defined as follows:

$$y(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{s',a'}\Big[Q_{\hat{\theta}}(s',a') - \alpha \log(a'|s')\Big]$$

And then, the policy is updated by maximizing the likelihood of actions with higher Q-values:

$$\mathcal{J}_\pi(\phi) = \mathop{\mathbb{E}}_{s,\hat{a}}\Big[Q_\theta(s,\hat{a}) - \alpha \log \pi_\phi(\hat{a}|s)\Big],\ \hat{a} \sim \pi_\phi(\cdot|s) \tag{2}$$

where $(s, a, s') \sim \mathcal{D}$ (in both (1) and (2)) and α is entropy coefficient. Note that although SAC is used in this paper, other off-policy methods for continuous control can be equally utilized for CRL. SAC is selected here as it has a straightforward implementation and few hyper-parameters.

D Baselines Definitions

FineTuning-MH is FineTuning with task-specific heads. For each new task, it spawns and attaches an additional output head to the actor and critics. Since each head is trained on a single task, this baseline allows to decompose forgetting

happening in the representation of the model (trunk) compared to forgetting in the last prediction layer (head). It is a task-aware method.

ER-TaskID is a variant of ER that is provided with task labels as inputs (i.e. each observation also contains a task label). It is a task-aware method that has the ability to learn a task representation in the first layer(s) of the model.

ER-MH is ER strategy which spawns tasks-specific heads [58], similar to FineTuning-MH. ER-MH is often the hardest to beat task-aware baseline []. ER-TaskID and ER-MH use two different strategies for modelling task labels. Whereas, MH uses $|h| \times |A|$ task-specific parameters (head) taskID only uses $|h|$, with $|h|$ the size of the network's hidden space (assuming the hidden spaces at each layer have the same size) and $|A|$ the number of actions available to the agent.

ER-TAMH (task-agnostic multi-head) is similar to ER-MH, but the task-specific prediction heads are chosen in a task-agnostic way. Specifically, the number of heads is fixed a priori (in the experiments we fix it to the number of total tasks) and the most confident actor head, w.r.t. the entropy of the policy, and most optimistic critic head are chosen. ER-TAMH has the potential to outperform ER, another task-agnostic baseline, if it can correctly infer the tasks from the observations.

MTL is our backbone algorithm, namely SAC, trained via multi-task learning. It is the analog of ER.

MTL-TaskID is MTL, but the task label is provided to the actor and critic. It is the analog of ER-TaskID and is a standard method, e.g. [17].

MTL-MH is MTL with a task-specific prediction network. It is the analog of ER-MH and is also standard, e.g. [63, 17, 64].

MTL-TAMH is similar to MTL-MH, but the task-specific prediction heads are chosen in the same way as in ER-TAMH.

MTL-RNN is similar to MTL, but the actor and critic are mounted with an RNN. It is the analog of 3RL.

E Empirical Findings

We now investigate some alluring behaviours we have come upon, namely that replay-based recurrent reinforcement learning (3RL), a task-agnostic continual reinforcement learning (TACRL) baseline, can outperform other task-agnostic but more importantly task-aware baselines, as well as match its MTL soft upper bound.

Benchmarks The benchmark at the center of our empirical study is Meta-World [64], which has become the canonical evaluation protocol for multi-task reinforcement learning (MTRL) [63, 62, 27, 47]. Meta-World offers a suite of 50 distinct robotic manipulation environments. What differentiates Meta-World from previous MTRL and meta-reinforcement learning benchmarks [44] is the broadness of its task distribution. Specifically, the different manipulation tasks are instantiated in the same state and action space[4] and share a reward structure, i.e., the reward functions are combinations of reaching, grasping, and pushing different objects with varying shapes, joints and connectivity. Meta-World is thus fertile ground for algorithms to transfer skills across tasks, while representing the types of tasks likely relevant for real-world RL applications (see Fig. 4 for a rendering of some of the environments). Consequently, its adoption in CRL is rapidly increasing [58, 38, 4].

In this work, we study `CW10`, a benchmark introduced in [58] with a particular focus on *forward transfer*, namely, by comparing a method's ability to outperform one trained from scratch on new tasks. `CW10` is composed of a particular subset of Meta-World conductive for forward transfer and prescribes 1M steps per task, where a step corresponds to a sample collection and an update.

We also study a new benchmark composed of the 20 first alphabetical tasks of Meta-World which we will use to explore more challenging regime: the task sequence is twice as long and data and compute are constrained to half, i.e., 500k steps are allowed per task. We refer to this benchmark as `MW20`.

In terms of metrics, the reported global and current success are the average success on all tasks and average success on the task that the agent is currently learning, respectively.

Experimental Details We use the hyperparameters prescribed by Meta-World for their Multi-task SAC (MTL-SAC) method. We ensured the performance of our SAC implementation on the `MT10`, one of Meta-World's prescribed MTRL benchmark, matches theirs (see App. G). For more details on the hyperparameters and training, see App. F. We test the methods using 8 seeds and report 90% confidence intervals as the shaded area of the figures.

As explained in Alg. 1, we have employed an scheme that oversamples recently gathered data scheme. Whenever we mention oversampling, we mean that the ER algorithm never spends more than 80% of its compute budget replaying old tasks. Without this feature, an ER algorithm would, for example, spend 90% of its compute budget when learning the 10^{th} task.

[4] the fixed action space is an important distinction with traditional incremental supervised learning

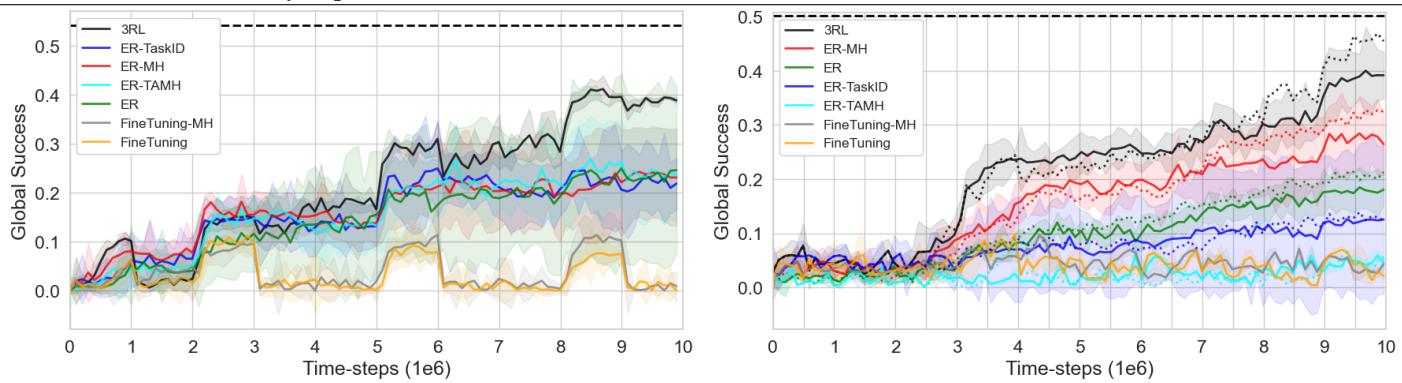


Figure 5: **3RL outperforms all baselines in both CW10 (left) and MW20 (right).** The horizontal line is the reference performance of training independent models on all tasks. The dotted lines (right plot) represent the methods' performance when they oversample recently collected data. 3RL outperforms all other task-agnostic and more interestingly task-aware baselines. As a side note, ER is high variance as it attempts to solve the POMDP directly without having any explicit or implicit mechanism to do so.

Task Agnosticism overcomes Task Awareness Our first experiments are conducted on CW10 and MW20 and are reported in Fig. 5. Interestingly, 3RL outperforms all other methods in both benchmarks. This is surprising for two reasons. First, at training time, the task-aware methods learn task-specific parameters to adapt to each individual task. Hence, they suffer less or no forgetting (in general). Second, at test time, the task-agnostic method has to first infer the task through exploration, at the expense of exploitation (see bottom of Fig. 4 for a visualization).

Of course, one could add an RNN to a task-aware method. We intend to compare the effect of learning task-specific parameters compared to learning a common task inference network (the RNN). Nevertheless, a combination of the RNN with ER-MH which was unfruitful (see App. I).

To ensure that these results are not a consequence of the methods having different number of parameters, we learned the CW10 benchmark with bigger networks and found the performance to drop across all methods (see App. J). Further, in our experiments ER-MH is the method with the most parameters and it is outperformed by 3RL.

Continual Learning can Match Multi-Task Learning Multi-task learning is often used as a soft upper bound in evaluating CL methods in both supervised [1, 35, 11] and reinforcement learning [46, 52, 58]. The main reason is that in the absence of additional constraints, multi-task learning (jointly training with data from all tasks in a stationary manner) does not suffer from the catastrophic forgetting that typically plagues neural networks trained on non-stationary data distributions.

3RL can reach this multi-task learning upper bound. In Fig. 6 we report, for the second time, the results of the MW20 experiments. This time, we focus on methods that oversample the current task and more importantly, we report the performance of each method's multi-task analog, i.e. their soft-upper bound (dashed line of the same color). 3RL, is the only approach that matches the performance of its MTRL equivalent. We believe it is the first time that a specific method achieves the same performance in a non-stationary task regime compared to the stationary one, amidst the introduced challenges like catastrophic forgetting.

For the remainder of the section, we investigate some hypotheses that might explain 3RL's alluring behavior. We first hypothesise that the RNN boosts performance because it is simply better at learning a single MDP (Hypothesis #1). Next, we investigate the hypothesis that 3RL reduces parameter movement, as it is often a characteristic of successful continual-learning methods (Hypothesis #2). We then explore the hypothesis that the RNN correctly places the new tasks in the context of previous ones (Hypothesis #3). We discuss one last hypothesis in App. M.

Hypothesis #1: RNN individually improves the single-task performance A simple explanation is that the RNN enhances SAC's ability to learn each task independently, perhaps providing a different inductive bias beneficial to each individual task. To test this hypothesis, we run the CW10 benchmark this time with each task learned separately, which we refer to as the Independent baselines. Note that this hypothesis is unlikely since the agent observes the complete state which is enough to act optimally (i.e. the environments are MDPs not POMDPs). Unsurprisingly, the RNN does not appear to improve performance in STL settings and we discard this hypothesis. Appendix H provides a complete analysis of STL results.

Hypothesis #2: RNN increases parameter stability, thus decreasing forgetting The plasticity-stability tradeoff is at the heart of continual learning: plasticity eases the learning of new tasks. Naive learning methods assume stationary

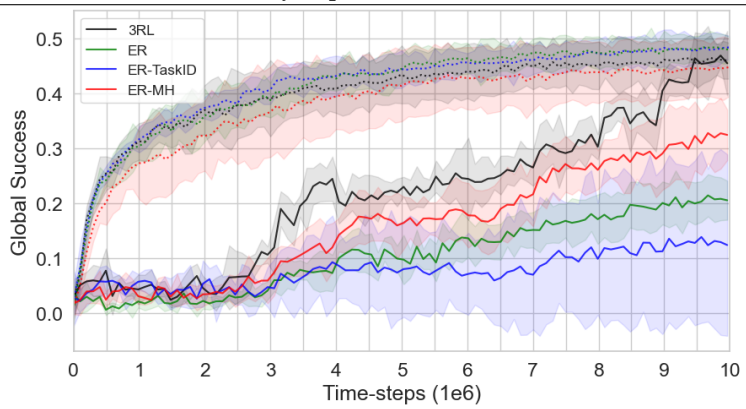


Figure 6: **3RL reaches its MTRL soft-upper bound**. MW20 in solid lines vs MTRL-MW20 in dotted lines. 3RL methods matches its soft-upper bound MTL analog as well as the other MTRL baselines. In contrast, other baselines' performance are drastically hindered by the non-stationary task distribution.

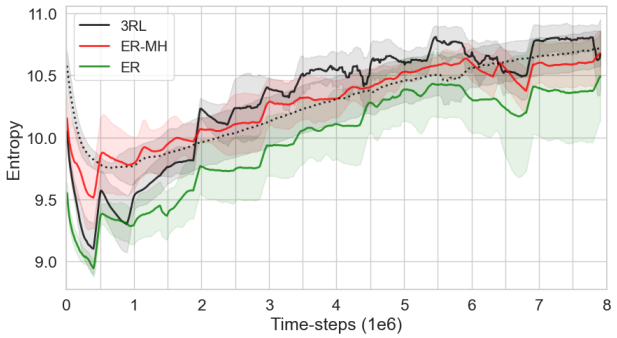


Figure 7: **3RL doesn't achieve superior continual learning performance through increased parameter stability**. We show the evolution of the methods' entropy in the parameters updates. We include MTL-RNN (dotted line) as a ref. We do not observe an increase in parameter stability: on the contrary, all methods, increasingly update more weights as new tasks (or data) come in.

data and so are too plastic in non-stationary regimes leading to catastrophic forgetting. To increase stability, multiple methods enforce [36] or regularize for [25] *parameter stability*, i.e., the tendency of a parameter to stay within its initial value while new knowledge is incorporated. Carefully tuned task-aware methods, e.g. PackNet [36] in [58], have the ability to prevent forgetting.[5]

Considering the above, we ask: could 3RL implicitly increase parameter stability? To test this hypothesis we measure the entropy of the weight changes throughout an epoch of learning defined by all updates in between an episode collection. Details about this experiment are found in App. K.

Fig. 7 shows the entropy of 3RL, ER, and ER-MH. We use these baselines since the gap between ER and its upper bound is the largest and the ER-MH gap is in between ER's and 3RL's. We find strong evidence to reject our hypothesis. After an initial increase in parameter stability, weight movement increases as training proceeds across all methods and even spikes when a new task is introduced (every 500K steps). MTL-RNN follows the same general pattern as the ER methods.

Hypothesis #3: RNN correctly places the new tasks in the context of previous ones, enabling forward transfer and improving optimization As in real robotic use-cases, MW tasks share a set of low-level reward components like grasping, pushing, placing, and reaching, as well as set of object with varying joints, shapes, and connectivity. As the agent experiences a new task, the RNN could quickly infer how the new data distribution relates with the previous ones and provide to the actor and critics a useful representation. Assume the following toy example: task one's goal is to grasp a door handle, and task two's to open a door. The RNN could infer from the state-action-reward trajectory that the second task is composed of two subtasks: the first one as well as a novel pulling one. Doing so would increase the policy learning's speed, or analogously enable forward transfer.

Now consider a third task in which the agent has to close a door. Again, the first part of the task consists in grasping the door handle. However, now the agent needs to subsequently push and not pull, has was required in task two. In this situation, task interference [63] would occur. Once more, if the RNN could dynamically infer from the context when pushing or pulling is required, it could modulate the actor and critics to have different behaviors in each tasks thus reducing the interference. Note that a similar task interference reduction should be achieved by task-aware methods. E.g., a multi-head component can enable a method to take different actions in similar states depending on the tasks, thus reducing the task interference.

Observing and quantifying that 3RL learns a representation space in which the new tasks are correctly *decomposed* and placed within the previous ones is challenging. Our initial strategy is to look for effects that should arise if this hypothesis was true (so observing the effect would confirm the hypothesis).

First, we take a look at the time required to adapt to new tasks: if 3RL correctly infers how new tasks relate to previous ones, it might be able to learn faster by re-purposing learned behaviors. Fig. 8 depicts the current performance of different methods throughout the learning of CW10. For reference, we provide the results of training separate models, which we refer to as Independent and Independent RNN.

[5]The observation that PackNet outperforms *an* MTRL baseline in [58] is different from our stronger observation that a *single* method, namely 3RL, achieves the same performance in CRL than in MTRL.

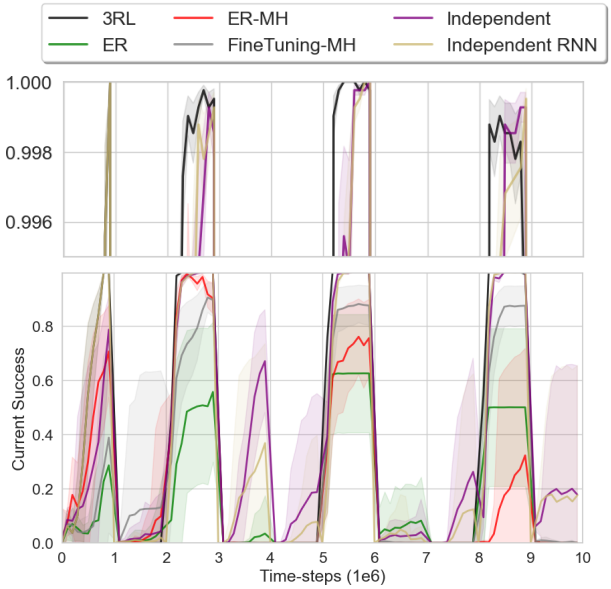


Figure 8: **3RL is the fastest learner.** Current success rate on CW10. The Independent method, which trains on each task individually, is still the best approach to maximise single-task performance. However, on the task that the continual-learning methods succeed at, 3RL is the fastest learner. In these cases, its outperformance over Independent and Independent RNN indicates that forward transfer is achieved.

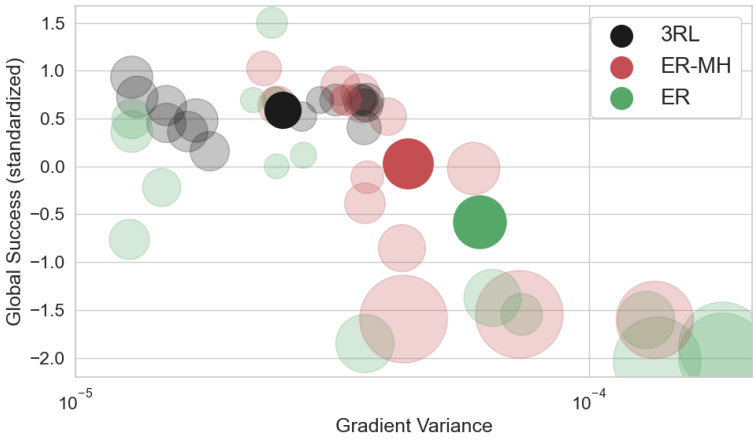


Figure 9: **3RL decreases gradient conflict leading to an increase in training stability and performance.** The global success and gradient as measured by the variance of the gradients are shown are plotted against each other. Training instability as measured by the variance of the Q-values throughout learning is represented by the markers' size, in a log-scale. Transparent markers depict seeds, whereas the opaque one the means. We observe a negative correlation between performance and gradient conflict (-0.75) as well as performance and training stability (-0.81), both significant under a 5% significance threshold. The hypothesis is that 3RL improves performance by reducing gradient conflict via dynamic task representations.

The challenges of Meta-World v2 compounded with the ones from learning multiple policies in a shared network, and handling an extra level of non-stationary, i.e. in the task distribution, leaves the continual learners only learning task 0, 2, 5, and 8. On those tasks (except the first one in which no forward transfer can be achieved) 3RL is the fastest continual learner. Interestingly, 3RL showcases some forward transfer by learning faster than the Independent methods on those tasks. This outperformance is more impressive when we remember that 3RL is spending most of its compute replaying old tasks. We thus find some support for Hypothesis #3.

Second, simultaneously optimizing for multiple tasks can lead to conflicting gradients or task interference [63]. To test for this effect, we use the variance of the gradients on the mini-batch throughout the training as a proxy of gradient conflict (see App. L for a discussion on why the gradient's variance is superior to the gradients' angle as a proxy for gradient conflict). In Fig. 9 we show the normalized global success metric plotted against the gradient variance. In line with our intuition, we do find that the RNN increases gradient agreement over baselines. As expected, adding a multi-head scheme can also help, to a lesser extent. We find a significant negative correlation of -0.75 between performance and gradient conflict. Fig. 9 also reports training stability, as measured by the standard deviation of Q-values throughout training (not to be confused with the parameter stability, at the center of Hypothesis #2, which measures how much the parameters move around during training). We find 3RL enjoys more stable training as well as a the significant negative correlation of -0.81 between performance and training stability. Note that The plausibility of Hypothesis #3 is thus further increased.

We wrap up the hypothesis with some qualitative support for it. Fig. 4 showcases the RNN representations as training unfolds. If the RNN was merely performing task inference, we would observe the trajectories getting further from each other, not intersecting, and collapsing once the task is inferred. Contrarily, the different task trajectories constantly evolve and seem to intersect in particular ways. Although only qualitative, this observation supports the current hypothesis.

F Experimental Details

We've used the SAC hyperparameters prescribed by Meta-World [64]. Specifically, we used a learning rate of 1×10^{-3}; mini-batch size of 1028; actor and critics are 2-layer MLPs with hidden sizes of [400, 400]; episode length of 500 (except 200 in CW10; ReLU activation function, soft target interpolation parameter of 5×10^{-3}.

We used automatic entropy tuning except in the MTRL experiments, where we found it to be detrimental. Because their MT-SAC implementation learns a task-specific entropy term, we think this is the reason why they do not observe the same behavior.

As prescribed, we use a minimum buffer batch size of 1500 when doing 10 tasks and 7500 when doing more. We also use a burn in period of 10,000 time-steps, now prescribed by Continual World [59].

Finally, after struggling with some deadly triad [54] problems in CRL and MTRL, we decided to clip the gradients' norm at 1, which turned out an effective solution.

F.1 Computing Resources

All experiments were performed on Amazon EC2's P2 instances which incorporates up to 16 NVIDIA Tesla K80 Accelerators and is equipped with Intel Xeon 2.30GHz cpu family.

All experiments included in the paper can be reproduced by running 43 method/setting configurations with 8 seeds, each running for 4.2 days on average.

F.2 Software and Libraries

In the codebase we've used to run the experiments, we have leverage some important libraries and software. We used Mujoco [51] and Meta-World [64] to run the benchmarks. We used Sequoia [41] to assemble the particular CRL benchmarks, including `CW10`. We used Pytorch [42] to design the neural networks.

G Validating our SAC implementation on MT10

In Fig. 10 we validate our SAC implementation on Meta-World v2's MT10.

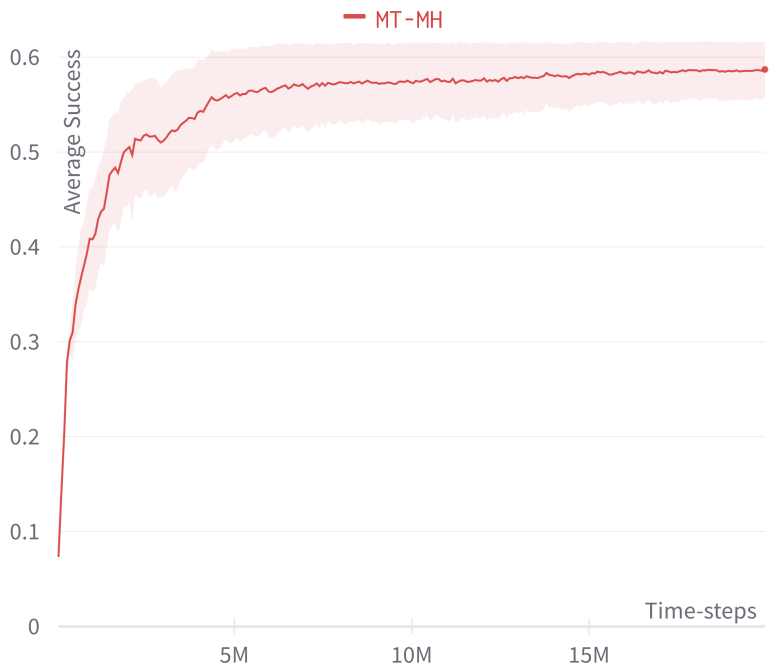


Figure 10: **MT10 experiment**. We repeat the popular MT10 benchmark with our MT-MH implementation. After 20M time-steps, the algorithm reaches a success rate of 58%. This is in line with Meta-World reported results. In Figure 15 of their Appendix, their MT-SAC is trained for 200M time-steps the first 20M time-steps are aligned with our curve. We further note that our CW10 result might seem weak vis à vis the reported ones in [59]. This is explained by [59] using Meta-World-v1 instead of the more recent Meta-World-v2.

H Extended CW10 Single-task results

Results for single-task experiments are shown in Fig. 11 and Fig. 12.

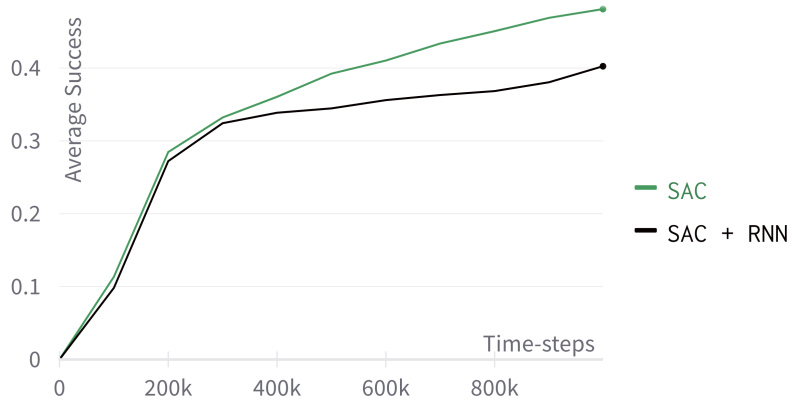


Figure 11: **Single-task learning CW10 experiments**. Average success on all task trained independently. In this regime, the RNN doesn't help.

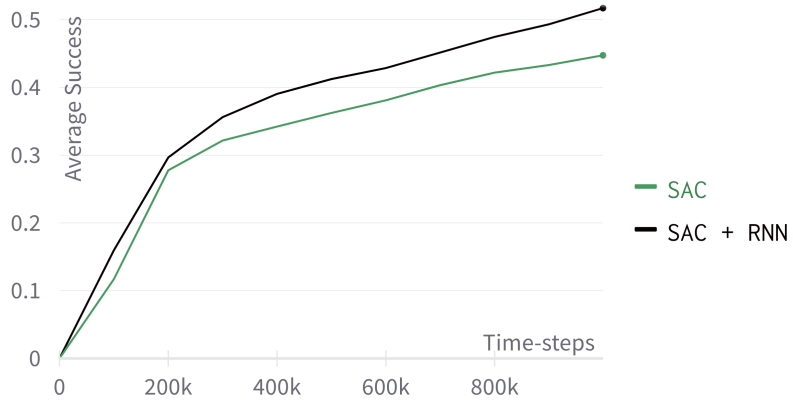


Figure 12: **Single-task learning CW10 experiments without gradient clipping**. We enstored gradient clipping in CRL and MTRL to alleviate the deadly triad problem, a problem we did not find in single-task learning (STL). For completeness, we reran the STL experiments without gradient clipping. We found the performance of the RNN to dramatically increase. Note that this is not the same algorithms used in the CRL and MTRL experiments because of the gradient clipping discrepancy.

I Task-aware meets task-agnostic

In Fig. 13 we show that combining the RNN with MH is not a good proposition in `CW10`.

J Larger networks don't improve performance

In Fig. 14 we report that increasing the neural net capacity doesn't increase performance.

K Gradient Entropy Experiment

To assess parameter stability, we look at the entropy of the parameters for each epochs. Because the episode are of size 500, the epoch corresponds to 500 updates. To remove the effect of ADAM [24], our optimizer, we approximate the parameters' movement by summing up their absolute gradients throughout the epoch. To approximate the sparsity of

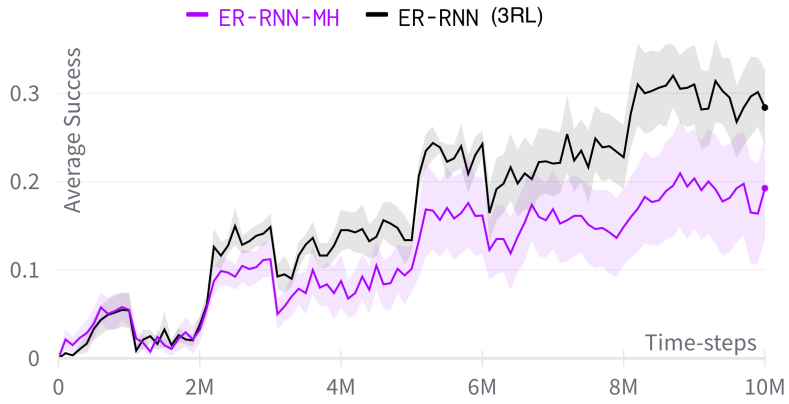


Figure 13: **CW10 experiment combining the RNN and MH**. Combining the best task-agnostic (3RL) and task-aware (ER-MH) CRL methods did not prove useful. Note that this experiment was ran before we enstored gradient clipping, which explains why the performance is lower than previously reported.

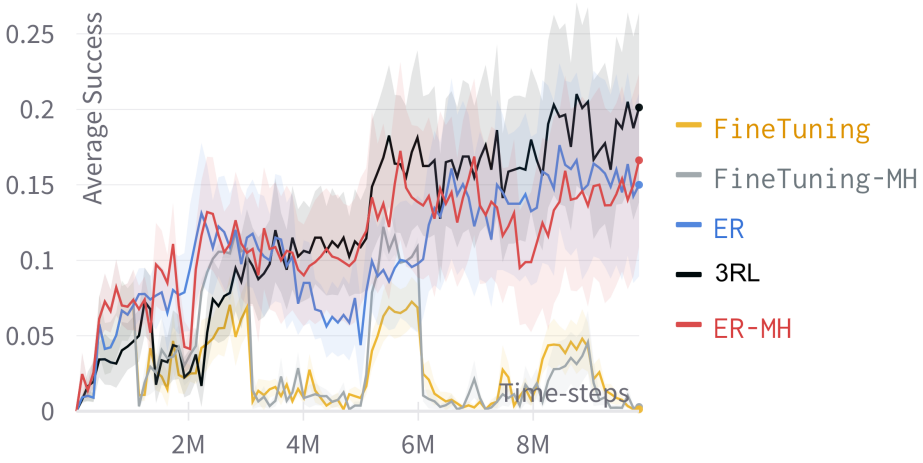


Figure 14: **CW10 experiment with larger neural networks.**. We repeated the CW10 experiment, this time with larger neural networks. Specifically, we added a third layer to the actor and critics. Its size is the same as the previous two, i.e. 400. The extra parameters have hindered the performance of all baselines.

the updates, we report the entropy of the absolute gradient sum. For example, a maximum entropy would indicate all parameters are moving equally. If the entropy drops, it means the algorithm is applying sparser updates to the model, similarly to PackNet.

L Gradient conflict through time

One might ask why we use the gradients' variance as a proxy for gradient conflict. Indeed, [63] measures the conflict between two tasks via the angle between their gradients: the tasks conflict if the angle is obtuse. However, we argue that this thinking is inadequate because it does not consider the magnitude of the gradients.

Assume a 1D optimization problem. In case 1, assume that the first task's gradient is 0.01 and the second's -0.01. In case 2, assume the gradients are now 0.01 and 0.5. Measuring the conflict via the angle would lead us to think that the tasks are in conflict in case 1 and are not in case 2. This is, however, not the case. The agreement is much higher in case 1: both tasks agree that they should not move too far from the current parameter. In case 2, although the two tasks agree on the direction of the step, they do not agree about the curvature of the loss landscape. The update step will thus be too small and too big for the two tasks. We thus think standard deviation is a better proxy for gradient conflict. We have updated the manuscript to defend our position better. 143

We've included the evolution of gradient conflict in the actor and critics in Fig. 15.

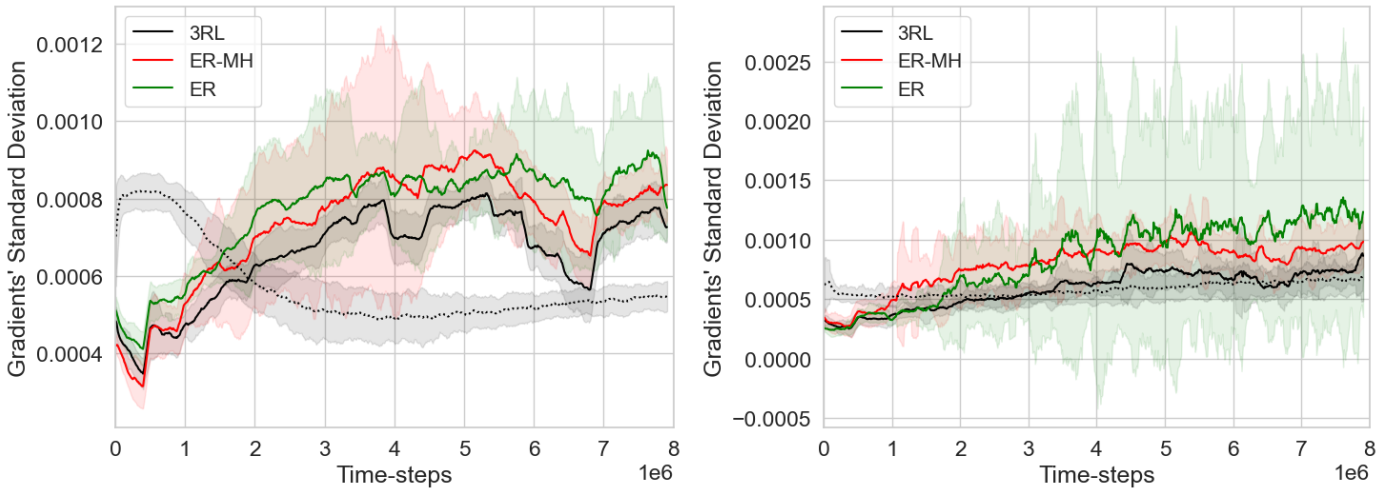


Figure 15: **Gradient variance analysis on CW20**. Comparison of the normalized standard deviation of the gradients for the actor (left) and critics (right) in for different CRL methods. For reference, we included MTL-RNN as the dotted line. The gradient alignment's rank is perfectly correlated with the performance rank.

M Hypothesis #4: we are operating in regimes where additional task-specific parameter can hurt performance

The task-aware methods learn task-specific parameters which might require more training data. In Figure 5 we compare the relative performance the methods as we increase compute and data from 500k time-steps to 1M. We observe that increasing the data/compute narrows the gap between the 3RL and ER-MH. This increases the plausibility of hypothesis #4.

However, some observations suggest we should look elsewhere. First, in `CW10` (Figure Fig. 5) another, another high-data regime, the task-aware methods do not shine. Second, ER-TaskID, which has fewer extra parameters compared to ER-MH, does not work as well in general, and doesn't improve when data/compute is increased two-fold.

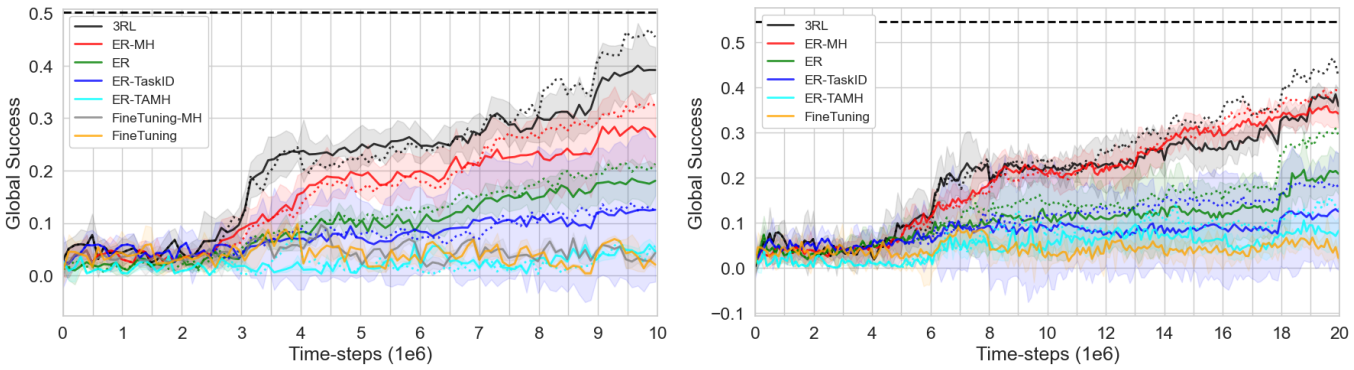


Figure 16: **Increased data/compute experiment.** We show the methods' performance on MW20 with 500k steps (left) and 1M steps (right). Hypothesis #4 is inconclusive.

N Related Work

To study CRL in realistic settings, [59] introduce the Continual World benchmark and discover that many CRL methods that reduce forgetting lose their transfer capabilities in the process, i.e. that policies learned from scratch generally learn new tasks faster than continual learners. Previous works study CL and compare task-agnostic methods to their upper bounds [65, 53] as well as CL methods compare to their multi-task upper bound [45, 46, 2]. Refer to [23] for an in-depth review of continual RL as well as [29, 19] for a CL in general.

RNN were used in the context of continual supervised learning in the context of language modeling [60, 61] as well as in audio [12, 55]. We refer to [10] for a in-depth review of RNN in continual supervised learning.

As in our work, RNN models have been effectively used as policy networks for reinforcement learning, especially in POMDPs where they can effectively aggregate information about states encountered over time [57, 14, 40, 21]. RNN were used in the context of MTRL [39]. Closer to our work, [48] leverages RNN in a task-aware way to tackle a continual RL problem. To the best of our knowledge, RNN have not been employed within TACRL nor combined with Experience Replay in the context of CRL.

O Conclusion

We have shown that adding a recurrent memory to task-agnostic continual reinforcement learning allows TACRL methods like 3RL to match their multi-task upper bound and even outperform similar task-aware methods. Our large experiments suggest that 3RL manages to decompose the given task distribution into finer-grained subtasks that recur between different tasks, and that 3RL learns representations of the underlying MDP that reduce task interference.

Our findings question the conventional assumption that TACRL is strictly more difficult than task-aware multi-task RL. Despite being far more broadly applicable, TACRL methods like 3RL may nonetheless be just as performant as their task-aware and multi-task counterparts. The need for the forgetting-alleviating and task-inference tools developed by the CL community is now questioned, at least in CRL.

Habituation reflects optimal exploration over noisy perceptual samples

Anjie Cao
Department of Psychology
Stanford University
Stanford, CA 94305
anjiecao@stanford.edu

Gal Raz
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139
galraz@mit.edu

Rebecca Saxe
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139
saxe@mit.edu

Michael Frank
Department of Psychology
Stanford University
Stanford, CA 94305
mcfrank@stanford.edu

Abstract

From birth, humans constantly make decisions about what to look at and for how long. Yet the mechanism behind such decision-making remains poorly understood. Here we present the Rational Action, Noisy Choice for Habituation (RANCH) model. RANCH is a rational learning model that takes noisy perceptual samples from stimuli and makes sampling decisions based on Expected Information Gain (EIG). The model captures key patterns of looking time documented in develop- mental research: habituation and dishabituation. We evaluated the model with adult looking time collected from a paradigm analogous to the infant habituation paradigm. We compared RANCH with baseline models (no learning model, no perceptual noise model) and models with alternative linking hypotheses (surprisal, KL divergence). We showed that 1) learning and perceptual noise are critical assumptions of the model, and 2) Surprisal and KL are good proxies for EIG under the current learning context.

Keywords: decision making; learning; bayesian modeling; cognitive development

1 Introduction

From trying to find our way through a busy street to swiping through TikTok, people are constantly making the decision of whether to keep looking or to look at something else. Even the youngest infants decide whether to keep looking at what is in front of them or move on. How can we explain this decision-making process? Our goal in the current paper is to provide a model of the basic decision: whether to keep looking at a stimulus. To do so, we model looking as rational active selection of noisy perceptual samples for learning.

Developmental researchers have long capitalized on infants' ability to control their attention, making inferences about learning and mental representations from changes in looking duration (e.g. Baillargeon, Spelke, & Wasserman, 1985). In a typical experiment, infants decrease their looking duration upon seeing the same stimulus repeatedly (habituation) but recover interest when seeing a novel stimulus (dishabituation). While these phenomena are well-documented, the mechanisms underlying them remain poorly understood, even though assumptions about habituation and dishabituation underpin many other claims about infants' cognitive repertoire (Carey, 2009).

Here we attempt to provide a model of looking behaviors as arising from optimal decision-making over noisy perceptual representations (Callaway, Rangel, & Griffiths, 2021). We present the Rational Action, Noisy Choice for Habituation (RANCH) model. RANCH works by accumulating noisy samples and choosing at each moment whether to continue to look at the current stimulus or to look away to the rest of the environment. The architecture of the RANCH model allows us to investigate different information-theoretic linking hypotheses as informing choice, including EIG, surprisal, and KL. We make a preliminary evaluation of the RANCH model using adult looking time data collected from a self-paced habituation paradigm that captures habituation, dishabituation, and how these phenomena are modified by stimulus complexity. We begin by presenting our experiment, since it frames the learning task for our model.

2 Experiment

To reproduce the key looking time patterns from infant habituation experiments in adult participants, we chose a learning context in which participants learn about the stimuli as they look at visually presented exemplars for as long as they like, with no explicit task. Our initial data come from adults for two reasons. First, adult data are suitable for establishing quantitative links between models and human behaviors, since infants' looking time data tend to have small sample sizes and are therefore limited in their quantitative details (Frank et al., 2017). Second, adult data allow us to test the hypothesis that similar rational choice processes underlie infant and adult behavior under similar learning contexts.

2.1 Methods

The experiment was a web-based, self-paced visual presentation task. Participants were instructed to look at a sequence of animated creatures (created with Spore, a game developed by Maxis in 2008) at their own pace and answer some questions throughout. On each trial, an animated creature showed up on the screen. Participants could press the down arrow to go to the next trial whenever they wanted to, after a minimum viewing time of 500 ms.

Each block consisted of six trials. Unbeknownst to the participants, each trial within the block was either a background trial or a deviant trial. One creature was assigned to be the 'background' for each block, and was presented five or six times. If the block contained a deviant trial, then a new, unique, creature was presented on that trial. The deviant trial could appear at either the second, the fourth, or the sixth trial in the block, or not at all. The creatures presented in the deviant trials and background trials were matched for complexity. Each participant saw eight blocks in total, four with simple creatures and four with complex creatures, in random order across participants.

We recruited 449 participants on Prolific. To test whether behavior was related to task demands, participants were randomly assigned to one of three attention check conditions, differing in the questions asked following each block.Participants were excluded if they showed irregular reaction times or their responses in the attention check tasks indicated low engagement with the experiment. The final sample included 380 participants.

2.2 Results

There were no task effects so we averaged all results across three conditions. We ran a linear mixed effects model with maximal random effect structure. The predictors included in the model were a three-way interaction term between the trial number (modeled as an exponential decay; Kail, 1991), the type of trial (background vs. deviant) and the complexity of the stimuli (simple vs. complex). The model failed to converge, so we pruned the model following the pre-registered procedure. The final model included per-subject random intercepts. All predictors except for the three-way interaction were significant in the model (all $p < .001$), providing a quantitative confirmation that our paradigm successfully captured the key looking time patterns: habituation, dishabituation, and complexity.

3 Model

We next tested whether we could capture these behavioral results using the RANCH model. RANCH treats the learning problem that participants face in our experiment as a form of Bayesian concept learning (Goodman, Tenenbaum, Feldman, & Griffiths, 2008). In this setting, multiple noisy samples inform the learner's hypothesis about a probabilistic concept represented by a set of binary features.

3.1 Model definition

In our setting, the goal is to learn a concept θ, which is a set of probabilities over independent binary features $\theta_{1,2,..,n}$, where n is the number of features. θ in turn generates exemplars y: instantiations of $\bar{\theta}$, where each feature $y_{1,2,..,n}$ is present or absent. The weights on each feature θ_i are sampled from a Beta prior, and individual exemplars y_i are distributed as a binomial with parameter θ_i, forming a conjugate Beta-Bernoulli distribution. Since the features are independent, this relationship holds for the entire concept θ.

To model the timecourse of attention, RANCH does not observe exemplars directly. Instead, it can observe repeated noisy samples $\bar{z}$ from each exemplar. For any sample z from an exemplar y there is a small probability ϵ that the observation is flipped and the feature is seen to be present when it was actually absent or vice versa. ϵ is assumed to be unknown but to have a Beta prior; in practice, we integrate over all possible values of ϵ. Therefore, by making noisy observations $\bar{z}$, RANCH obtains information about the true identity of the exemplar y, and by extension, about the concept $\bar{\theta}$. By Bayes' rule:

$$P(\theta|\bar{z}) \quad = p(\bar{z}|y)p(y|\theta)p(\theta)/p(\bar{z}) \tag{1}$$

To compute approximate posterior probability distributions during inference, we used a discrete grid approximation with a step size of .001 over both θ and ϵ.

Upon observing a sample, RANCH then decides whether to keep sampling or not. We chose EIG from the next sample as the main linking hypothesis between the learned posterior and sampling choice.

RANCH computes EIG by iterating through each possible next observation and weighing the information gain from each observation by its posterior predictive probability $p(z|\theta)$. We defined information gain as the KL between the hypothetical posterior after observing a future sample z_{t+1} and the current posterior (Baldi & Itti, 2010):

$$EIG(z_{t+1}) = \sum_{z_{t+1} \in [0,1]} p(z_{t+1}|\theta_t) * D_{KL}(\theta_{t+1}||p(\theta_t)) \tag{2}$$

Finally, to get actual sampling behavior from the model, it has to convert EIG into a binary decision about whether to continue looking at the current sample, or to advance to the next trial. The model does so via a Luce choice between the EIG from the next sample and a constant "environmental EIG" that is assumed to be the amount of information to be gained via looking away from the stimulus.

$$p(lookaway) = \frac{EIG(env)}{EIG(z_{t+1}) + EIG(env)} \tag{3}$$

3.2 Alternative models

We are also interested in what aspects of the model are necessary to produce the phenomena. Specifically, we tested two assumptions of the model: 1) the model makes decision based on learning, and 2) perception is noisy. We implemented two lesioned baseline models. The No Learning model made random sampling decisions by drawing $p(lookaway)$ from a uniform distribution between 0 and 1 at every time step. The No Noise model omitted the noisy sampling aspect of RANCH.

We also studied the behavior of RANCH using two other linking hypotheses, surprisal, and KL divergence. We implemented these by replacing $EIG(zt+1)$ in Equation 2 with either surprisal or KL. Both have been used in previous attempts to model infant looking behavior (Kidd, Piantadosi, & Aslin, 2012; Poli, Serino, Mars, & Hunnius, 2020), and to approximate EIG in the reinforcement learning literature (Kim, Sano, De Freitas, Haber, & Yamins, 2020).

3.3 Simulations

To model the behavioral experiment, we first represented the stimuli as binary-valued vectors indicating the presence (1) or absence (0) of each feature. All stimulus vectors were chosen to be length 6 to provide sufficient representational

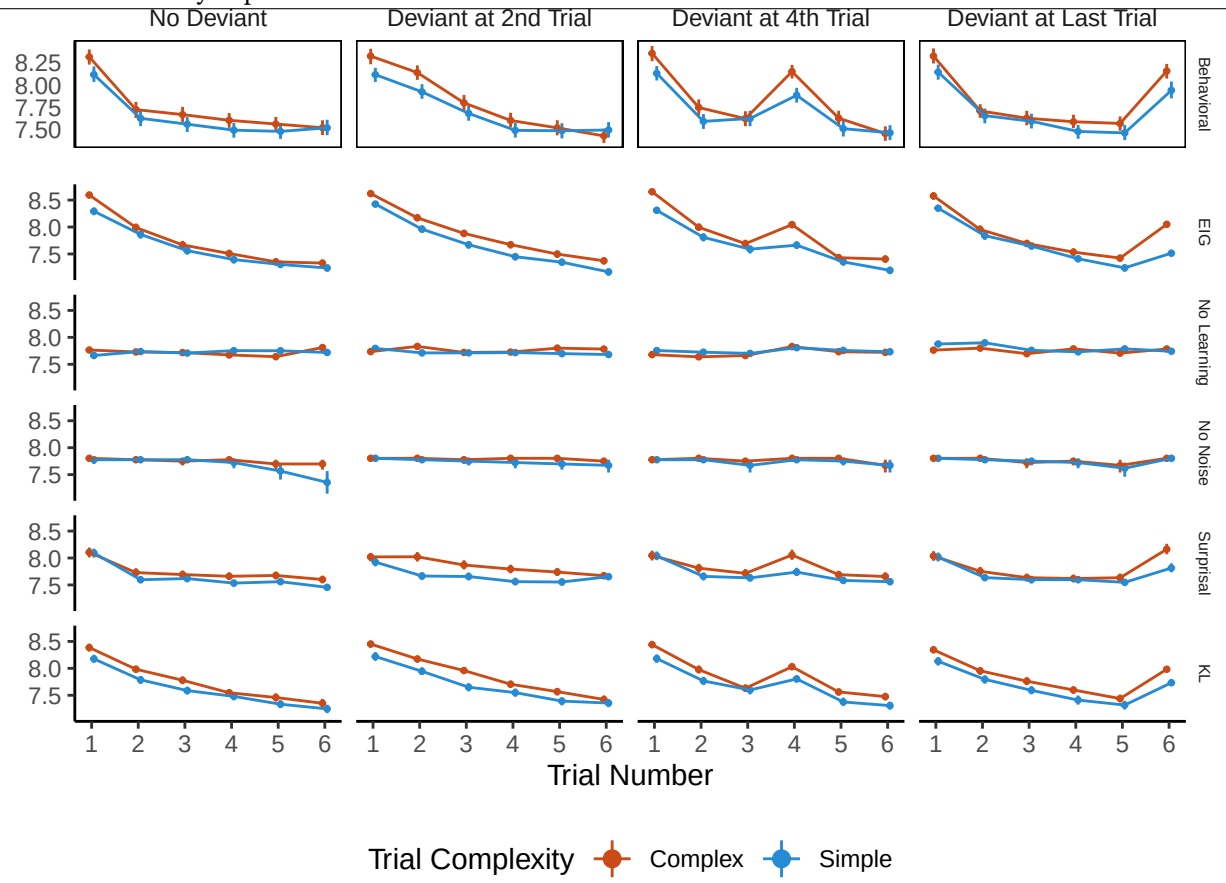


Figure 1: For comparison, all model results were linearly scaled to match the behavioral data. All results were log-transformed.

flexibility. Complex stimuli were represented as having three 1s and simple stimuli were represented as having one 1, with the rest of the features set to 0. Individual stimuli were then assembled into sequences to reflect the stimuli sequences in the behavioral experiment. For a particular sequence, we constructed the deviant stimulus based on the background stimulus to make sure that they were always maximally different and had the same number of features present.

Since the model makes stochastic choices about how many samples to take from each stimulus, behavior varies substantially across runs. Thus, we conducted 500 runs for each stimuli sequence and parameter value to obtain a reasonably precise estimate of the model's behavior.

3.4 Parameter estimation

We performed an iterative grid search in parameter space. We constrained our parameter space on the beta priors over features to have shape parameters $\alpha_\theta > \beta_\theta$, which describe the prior beliefs as "more likely to see the absence of a feature than the presence of a feature." We then searched for the priors over the concept (θ), the noise parameter that decides how likely a feature would be misperceived (ϵ), and the constant EIG from the environment ($EIG(env)$). The prior over the noise parameter was fixed for all searches ($\alpha_\epsilon = 1; \beta_\epsilon = 10$). We selected the parameters that achieved the highest correlation with the behavioral data averaged across participants and blocks ($\alpha_\theta = 1$, $\beta_\theta = 4$, $\epsilon = 0.065$, $EIG(env) = 0.01$). No parameter regimes showed qualitatively different patterns, though the magnitude of dishabituation was strongly dependent on the priors over θ; a test of the generality of these specific parameter values is left for future work.

3.5 Results

The simulation results are summarised in Figure 1. RANCH exhibited the main phenomena of interest, showing habituation, dishabituation, and complexity effects. We also quantitatively explored the model by fitting the model results to the behavioral data. Overall RANCH achieved a good fit across the three linking hypotheses (EIG: Pearson's r: = 0.92 [0.84, 0.96], RMSE = 0.19 [0.16, 0.24]; Surprisal: Pearson's r = 0.92 [0.85, 0.95], RMSE = 0.13 [0.11, 0.16]; KL: Pearson's r

= 0.93 [0.88, 0.96], RMSE = 0.12 [0.1, 0.15]), but in contrast, the No Learning (Pearson's r = 0.21 [-0.09, 0.46]; RMSE: 0.27 [0.23, 0.34]) and No Noise model (Pearson's r = 0.5 [0.36, 0.65]; RMSE: 0.25 [0.21, 0.31]) fit the data poorly.

4 General discussion

The current work aims to provide a computational model that can explain key phenomena observed in typical infant looking time paradigms: habituation, dishabituation, and how these are modified by stimulus complexity. RANCH assumes a rational learner that takes noisy perceptual samples from stimuli and makes sampling decisions based on EIG. We evaluated the model with adult looking time data collected from a paradigm that mirrors classic infant looking time paradigms, in which participants are learning about multi-feature concepts, and found that RANCH could successfully reproduce the patterns observed in behavioral data. By contrasting the model results with our baseline models, we showed that habituation, dishabituation, and complexity effects only arise in a learning model that takes into account the noisy nature of perception. Moreover, we found that, in the current learning context, other information theoretic quantities (surprisal and KL) are good proxies for the optimal linking hypothesis, EIG.

The similarity between model fits among models with different linking hypotheses highlights the significance of learning contexts. Our results should not be interpreted as evidence showing that the three linking hypotheses are indistinguishable across all learning contexts. Previous work has shown that adopting surprisal as learning policy can lead to undesirable behaviors in artificial agents (e.g. "the white noise problem," Oudeyer, Kaplan, & Hafner, 2007). Moreover, the two alternative linking hypotheses are backward-looking metrics that utilize heuristics about the past to make decisions. This characteristic could constrain their application to situations in which the environment is stable and the cost of sampling is low. Since adult exploration is sensitive to environmental complexity, a forward-looking metric like EIG might be particularly suitable to predict behaviors in a more dynamic learning context (Dubey & Griffiths, 2020; Vogelstein et al., 2022).

Our ultimate goal is to provide a rational learner model that can account for information-seeking behaviors reflected in infants' looking time. Here we have shown that a simple model of learning from sampling can reproduce habituation, dishabituation, and complexity effects. Moving forward, we aim to capture and explain more contentious phenomena documented in the infant looking time literature such as familiarity preferences and age effects (Hunter & Ames, 1988). Our ongoing work with infants will eventually enable us to evaluate our model with developmental data. When combined with adult results, the data and model will provide insights into the general mechanisms through which learners decide what to look at, and when to stop looking.

References

Baillargeon, R., Spelke, E. S., & Wasserman, S. (1985). Object permanence in five-month-old infants. *Cognition*, *20*(3), 191–208.
Baldi, P., & Itti, L. (2010). Of bits and wows: A bayesian theory of surprise with applications to attention. *Neural Networks*, *23*(5), 649–666.
Callaway, F., Rangel, A., & Griffiths, T. L. (2021). Fixation patterns in simple choice reflect optimal information sampling. *PLoS computational biology*, *17*(3), e1008863.
Carey, S. (2009). *The origin of concepts*. Oxford University Press.
Dubey, R., & Griffiths, T. L. (2020). Reconciling novelty and complexity through a rational analysis of curiosity. *Psychological Review*, *127*(3), 455.
Frank, M. C., Bergelson, E., Bergmann, C., Cristia, A., Floccia, C., Gervain, J., … others (2017). A collaborative approach to infant research: Promoting reproducibility, best practices, and theory-building. *Infancy*, *22*(4), 421–435.
Goodman, N. D., Tenenbaum, J. B., Feldman, J., & Griffiths, T. L. (2008). A rational analysis of rule-based concept learning. *Cognitive science*, *32*(1), 108–154.
Hunter, M. A., & Ames, E. W. (1988). A multifactor model of infant preferences for novel and familiar stimuli. *Advances in infancy research*.
Kail, R. (1991). Development of processing speed in childhood and adolescence. *Advances in child development and behavior*, *23*, 151–185.
Kidd, C., Piantadosi, S. T., & Aslin, R. N. (2012). The goldilocks effect: Human infants allocate attention to visual sequences that are neither too simple nor too complex. *PloS one*, *7*(5), e36399.
Kim, K., Sano, M., De Freitas, J., Haber, N., & Yamins, D. (2020). Active world model learning with progress curiosity. In *International conference on machine learning* (pp. 5306–5315).
Oudeyer, P.-Y., Kaplan, F., & Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, *11*(2), 265–286.
Poli, F., Serino, G., Mars, R., & Hunnius, S. (2020). Infants tailor their attention to maximize learning. *Science advances*, *6*(39), eabb5053.
Vogelstein, J. T., Verstynen, T., Kording, K. P., Isik, L., Krakauer, J. W., Etienne-Cummings, R., … others (2022). Prospective learning: Back to the future. *arXiv preprint arXiv:2201.07372*.

REAL-X – Robot open-Ended Autonomous Learning Architectures: challenges and solutions

Emilio Cartoni
Institute of Cognitive Sciences and Technologies
National Research Council
Rome, Italy
emilio.cartoni@istc.cnr.it

Davide Montella
Institute of Cognitive Sciences and Technologies
National Research Council
Rome, Italy
davidelmontella@gmail.com

Jochen Triesch
Frankfurt Institute of Advanced Studies
Frankfurt, Germany
triesch@fias.uni-frankfurt.de

Gianluca Baldassarre
Institute of Cognitive Sciences and Technologies
National Research Council
Rome, Italy
gianluca.baldassarre@istc.cnr.it

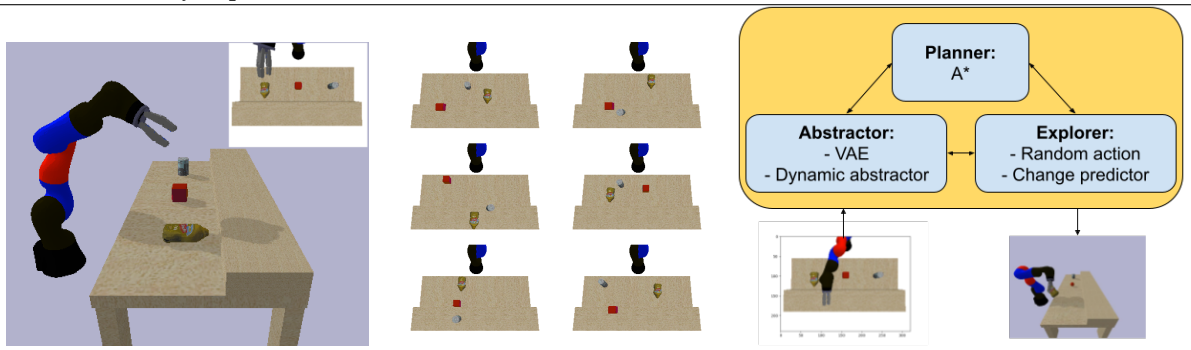
Abstract

Open-ended learning, an important research area of developmental robotics and machine learning, aims to build robots and machines able to incrementally and autonomously learn knowledge and skills based on intrinsic motivations and goal self-generation. In this work, we first highlight the challenges posed by the benchmark 'REAL', a robot competition fostering the development and comparison of robot architectures for truly open-ended learning. The benchmark requires to control a simulated camera-arm robot that undergoes two phases: (a) in the 'intrinsic phase': the robot autonomously interacts with some objects for a long time to acquire knowledge and skills; (b) in the 'extrinsic phase': the robot is tested with a set of goals unknown in the intrinsic phase to measure the quality of the autonomously acquired knowledge. The benchmark involves a number of challenges that are commonly faced in isolation, in particular the decision of which actions to use for exploration, the learning of the very concept of object based on pixels, the autonomous generation of tasks/goals, the generalisation to new conditions, and the autonomous learning of sensorimotor skills. The main contribution is the presentation of a set of robot architectures, called 'REAL-X', that are able to solve the different challenges posed by the benchmark and that are introduced progressively by releasing initial simplifications. The tests of the architectures show that the REAL benchmark is a useful means to clarify and face the challenges posed by open-ended learning in their hardest form. The REAL-X architectures succeed to achieve a good performance in the demanding conditions posed by the benchmark by using an intrinsic-motivation mechanism to foster the selection of actions during the intrinsic phase, and a novel mechanism that dynamically increases the level of abstraction used for planning during the extrinsic phase.

Keywords: Autonomous robot, simulation, open-ended learning, competition, benchmark, intrinsic motivation, planning.

Acknowledgements

We thank Francesco Mannella for developing the REAL environment; Simone Asci for developing and testing earlier versions of the change predictor. We also thank Hewlett Packard Enterprise for providing the hardware to run several of the simulations whose results are reported here. This research has received funding from the European Union Horizon 2020 Research and Innovation Programme under Grant Agreement No 713010, Project *Goal-Robots – Goal-based Open-ended Autonomous Learning Robots* (http://www.goal-robots.eu/), and from the Johanna Quandt Foundation.



(a) REAL Environment (b) Examples of extrinsic goals (c) REAL-X architecture

1 Introduction

Learning processes of humans and other animals with a sophisticated cognition are driven not only by biological and social needs but also by drives, such as *intrinsic motivations*, directed to the acquisition of knowledge and skills only later expressing a biological or social utility [6, 1]. Several algorithms have been proposed to reproduce intrinsically motivated learning in artificial intelligent machines and robots undergoing *open-ended learning* – the autonomous cumulative acquisition of knowledge and skills without relying on pre-wired reward functions, tasks, or goals [2, 11, 8]. Open-ended learning has been studied within the field of *developmental robotics* [12, 7], that has proposed various architectures for robot open-ended learning based on *intrinsic motivations* (e.g., [5, 15, 13, 16, 14]). More recently, also machine learning has entered the field, in particular by building on reinforcement learning algorithms [17, 4]. Notwithstanding recent advancements, we still do not have robots able to undergo a truly open-ended learning process as the learning progress of the current systems stops beyond a certain extent. One way to promote the development of such systems is the proposal of benchmarks and competitions that facilitate the comparison of alternative approaches and models. For this reason, we have proposed a competition called (*REAL 2021 – Robot open-Ended Autonomous Learning*[1]) now at its third edition. The competition pivots on a benchmark encompassing the major challenges of robotic open-ended learning [10]). The competition is based on two phases [10, 3]. A first *intrinsic phase* involves a very long period of learning where the robot has to acquire knowledge and skills and in which no external guidance is available (no supervision, rewards, tasks, goals, etc.. In a second *extrinsic phase* the robot has to accomplish a number of extrinsic goals unknown during the first phase and its average performance is measured. This is the central element of the benchmark as it allows an *objective measure of the quality of the knowledge that the robot acquired autonomously in the intrinsic phase*, independently of the algorithm used in such phase. This is important as it furnishes a standard to compare different autonomous learning algorithms used in the intrinsic phase. The benchmark poses extremely hard challenges, as revealed by the previous editions of the competition [10]. The first challenge is that the robot perceives only pixels, joint angles, and touch-sensor readings and so needs to autonomously understand from scratch the very concept of object. Second, the robot does not have any information on how to control its several degrees of freedom to produce 'relevant' effects on the environment. Moreover, these challenges are made particularly hard as they are strongly interdependent but they have to be faced at the same time. In this work, we propose a set of architectures, called *REAL-X*, that encompass a number of solutions to face the different challenges posed by the REAL benchmark. The 'X' in the name refers to the fact that the architecture has been developed in different versions, as we first faced simplified versions of the REAL challenge with a basic version of the architecture, and then we progressively removed the simplifications and faced them by endowing the architecture with enhanced components. In future work, we plan to use the REAL benchmark and the architecture to further develop these solutions and compare and integrate them with other relevant models and mechanisms from the literature.

2 Methods

Scenario of the open-ended learning benchmark The competition scenario is inspired by an assistant-robot scenario where the robot should help to tidy up a kitchen (*'kitchen scenario'*, see Figure 1a). The robot is a 7-DoF Kuka arm coupled with a 2 DoF gripper. It stands in front of a table, which has a shelf and 1 to 3 objects: a cube, a tomato can, and a mustard bottle. The perceptual space comprises the images from a fixed top-view camera, the proprioception of joint angles, and the gripper touch sensors; in a simplified version of the benchmark the robot directly perceives the positions of the objects. In the REAL-X architectures discussed below, the touch sensors will not be used. In a first *intrinsic phase*, the robot *autonomously* interacts with the environment for a long time (about 20 hours of simulated time) during which it should acquire as much knowledge and skills as possible to best solve the tasks in a second 'extrinsic phase'. During the *extrinsic phase*, the acquired knowledge and skills are evaluated with tasks (here 'goals', intended as desired world states) unknown by the robot during the intrinsic phase. Each goal has to be reached within a short

[1] https://eval.ai/web/challenges/challenge-page/2134/overview

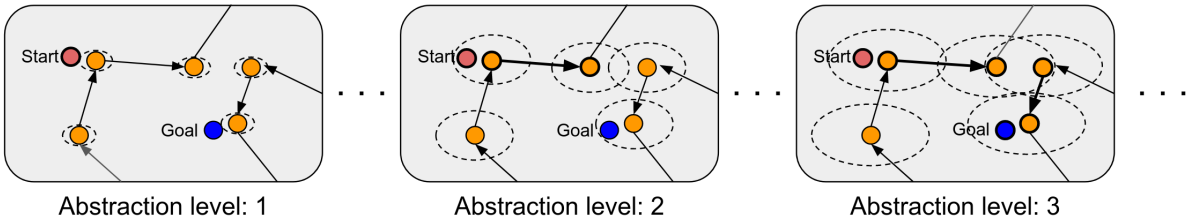


Figure 2: Dynamic abstraction functioning. Yellow circles: states (x-y object position or latent variables) experienced during the intrinsic phase. Red circle: current state. Blue circle: goal state. Arrows: experienced actions connecting states. Dashed ovals: boundary within which a certain state is considered as the same as the other states within the boundary. Bold circles and arrows: plan portion that the algorithms managed to find up to a certain level of abstraction.

timeframe (50 seconds of simulated time). Each goal features a different desired object configuration and also a different starting configuration involving one or more objects, positioned along the table or on the shelf. Figure 1b shows some examples of extrinsic goals. Crucially, in the extrinsic test the robot can still learn, but this is of little help given the limited time available to solve each task, so *the performance in the extrinsic phase can be considered an objective measure of the system's capacity to autonomously acquire knowledge during the intrinsic phase*. The extrinsic-phase performance is scored as the average performance on all 50 goals, with each goal being scored as a function of how much distant was each object final position compared to the goal position (1 if the object is exactly there, less if it is more distant - see also [9]).

Objective function of the open-ended learning benchmark The overall objective of the robot participating in the competition is to find, *during the intrinsic phase*, the parameter vector θ^* that maximizes the expected reward collected *during the extrinsic phase*: $\theta^* = \arg\max_\theta E_{g\sim\tau(g)} \left(E_{\pi(a|s,g,\theta)} R(g)\right)$ where $R(g)$ are the total rewards obtained for goal g used in the extrinsic phase to evaluate the system, $g \sim \tau(g)$ is the distribution of possible tasks that can be posed in the environment, $\pi(a|s,g,\theta)$ is the control policy, dependent on parameters θ, that the robot uses to select actions a in response to state s and the currently pursued goal g. *The crucial feature of the benchmark is that the parameters θ must be learned during the intrinsic phase but are tested with goals g during the extrinsic phase, and these goals are unknown during the intrinsic phase.*

Simplifications We designed different REAL-X architectures to face different versions of the benchmark. These versions feature different simplifications that were progressively released to move towards the full hard challenge. *Perception* A first simplification gave the robot the x, y, z positions of the objects. When this simplification was removed, the robot instead received a raw camera image of 320x240 RGB pixels. *Motor control* This simplification gave the robot a parameterised macro-action to act on the objects. The macro-action first moved the (closed) end effector close to the table plane, then along a segment trajectory parallel to the plane, and then back to a home position. The parameters of the macro-actions were the two (x, y) and final (x', y') extremes of the movement segment on the plane. The use of the macro-action greatly facilitated hitting/pushing the objects during exploration. When this simplification was removed, the robot was directly controlled in the joint space. *Number of objects* We currently report only results with 1 object.

REAL-X architectures. We first describe the simpler architecture, *REAL-D*, then introduce two reduced architectures used for comparison (REAL-R, REAL-T) and finally the architectures enhanced with components to face the more complex benchmark conditions (REAL-LD, REAL-ILD; see Table 1). These conditions are called $REAL\_X$ for ease of reference (note _ in place of $-$). REAL-D is formed by three main components (Figure 1c). **Explorer**. This component guides exploration during the intrinsic phase and aims to maximise the acquisition of motor skills. Each action lasts 1000 steps and starts and ends in a 'home' position involving the arm and gripper straight upward. The explorer produces different actions by randomly generating their parameters. For each executed action the component stores the acquired knowledge as an *action triplet* (s, a, s') containing: (a) the action *precondition* state s, encoding either the image or the objects' positions; (b) the *action parameters* a; (c) the action *outcome* state s'. **Abstractor**. We introduced a 'Dynamic Abstraction' (DA) mechanism, sketched in Figure 2 that defines a number of abstraction levels starting from the (s, a, s') triplets experienced. To define when a certain state s_i can be considered equivalent to an s or s' state of a triplet, we define a series of increasing thresholds that define different abstraction levels. The output of the DA is an $L \times V$ matrix, where L is the number of desired abstraction levels (here 200), and V is the number of state variables to consider (e.g. the x-y position of an object; or the latent variables of an auto-encoder used to abstract images). For each level of abstraction, the DA gives V thresholds, one for each variable. This $L \times V$ matrix is used later by the planner to decide which states are considered equivalent to each other, or different, at a given level of the abstraction.To compute the levels of abstraction, the absolute differences $|s - s'|$ of each action triplet are first computed and then ranked, independently for each variable. From these ranked differences, 200 differences, one for each abstraction level, are then selected for each variable, starting from the lowest rank up to the maximum rank, at equal intervals. The lowest level of abstraction thus involves V thresholds representing the minimum differences found, for each state variable, between all the starting-outcome state couples of the triplets; the maximum level of abstraction uses instead the largest differences found. This process allows the DA to work in an unsupervised fashion for any domain and requires only a single parameter establishing how many abstraction levels are desired. **Planner**. The planning used was A* with a maximum depth of 10 actions and a heuristic

based on the outcome-goal distance computed as the L-1 norm. The planner searches a plan by starting from the lowest abstraction level from the DA and then raises it in case of failure until success. If the planner reaches the maximum level of abstraction without finding a plan the goal is deemed non reachable. **REAL-T: Threshold** To test the utility of the DA we also tested a version of the system, REAL-T, where we manually set a *fixed* abstraction threshold. **REAL-R: Random** We also compared all systems with REAL-R, a system facing the extrinsic goals by producing random actions, to have a baseline performance in all conditions. **REAL-LD: Latent variables and Dynamic abstractor**. REAL-LD was used to face the conditions with raw RGB images. The Abstractor was enhanced with an additional abstraction process: the precondition and outcome images were processed with a background filtering algorithm and then transformed into latent variables using a Variational Autoencoder (VAE). The latent variables were then used as the input for the subsequent DA and Planning processes. REAL-LD was tested in the REAL_OM and REAL_IJ conditions involving not only raw pixels but also the direct control of joints. To face this condition, the macro-action was substituted with an action composed by random via-points in the joints space. **REAL-ILD: Intrinsic motivation exploration, Latent variables, Dynamic abstractor** REAL-ILD worked as REAL-LD but its Exploration component was enhanced with a mechanism increasing the likelihood that the actions generated for exploration touched the objects. The mechanism is based on a neural-network predictor that takes as input the current state (latent variables) and a planned action, and predicts if the action will lead to a *significant change* of the state, for example because an object has been moved. The predictor is trained periodically during exploration based on the information acquired that far. Before each action performance in the intrinsic phase, the system generates up to 1,000 actions and evaluates them with the predictor; it then performs the first action predicted to cause a change or a random one if no such action is found.

3 Results

Table 1 reports the scores of all REAL-X architectures tested with increasingly difficult benchmark conditions. A video of the performance of REAL-LD is available on `https://youtu.be/kl26SyGAy_M` (REAL_IM condition) and on `https://youtu.be/nriO73Sftq0` (REAL_IJ condition).

REAL-X models	REAL_X benchmark conditions		
	REAL_OM	REAL_IM	REAL_IJ
REAL-R	$0.021 \pm .002$	$0.021 \pm .002$	$0.038 \pm .002$
REAL-T	$0.212 \pm .006$*		
REAL-D	$0.216 \pm .004$	NA	
REAL-LD		$0.222 \pm .005$	$0.139 \pm .007$
REAL-ILD		$0.234 \pm .007$	$0.149 \pm .006$

Table 1: Performance of the different architecture configurations, named REAL-X, in different conditions, named as REAL_X. OM: object positions, macro actions. IM: raw Images, macro actions. IJ raw Images, joint control. *: Best result after running with different thresholds. NA: the Planner was not able to return a plan before running out of memory.

REAL-D vs. REAL-R and REAL-T: test in the REAL_OM condition REAL-D, tested in the REAL_OM condition, achieves a score of 0.216 while the random model REAL-R achieves a performance of 0.021. The role of the Dynamic abstractor for such result can be seen by comparing it with the performance of REAL-T. We tried different values of the REAL-T threshold, and only the best one led to a performance similar to REAL-D, 0.212. This shows that the DA is able to automatically find a good level of abstraction. To investigate how the DA works, Figure 3 shows the number of effective actions available to the Planner (A*) for each level of abstraction. We consider an action to be 'effective' if its precondition is considered different from the outcome at the given abstraction level. At level 0, all actions are different, so the planner has 15,000 actions available (all those found in the intrinsic phase). At abstraction level 0, the Planner usually cannot find an action to start from or an action that brings to a a state as the goal state. As the abstraction level is increased to higher levels, the minimum distance to consider two states as different rises, so an increasing number of actions become non effective. We can see that up to about the abstraction level 171 the distance is very small, less than 1 cm: the actions with such a short distance are only due to the noise of the perceived object position and the object was not actually moved. In the simulations, the Planner quickly filters out all these first abstraction levels where no solution is found and then starts to find workable solutions at the abstraction level 170 or higher. The combination of the Planner with the DA thus succeed to find the right abstraction level with which to plan without needing preset thresholds.

REAL-LD: test in the REAL_IM condition REAL-LD in the REAL_IM condition achieves an average score of 0.222, even slightly higher than the score of REAL-D in the simpler REAL_OM condition (0.216). This confirms that converting the images into a latent space with the VAE, and then planning within it, performs as well as when using object positions. **REAL-LD: test in the REAL_IJ condition** REAL-LD was tested in the REAL_IJ condition to evaluate its ability to handle more general actions defined by an arbitrary sequence of joint via points. These joint actions produce non-coordinated arm movements, which are however still able to hit and move the object. REAL-LD is capable of extracting from these non-coordinated actions the effective ones to reach its goals, with a performance of 0.139 that is lower than REAL_OM and REAL_IM conditions but still higher than the random model (0.038). These tests thus showed that the REAL-X architecture can be used without predefined macro-actions. **REAL-ILD: test in the REAL_IM and REAL_IJ condition** Figure 3 shows the effectiveness of the change predictor during the intrinsic phase. In the previous architectures, random action sampling led to hitting the object with only 1 out of 6 actions. REAL-ILD gradually filters out the ineffective actions before executing them, so the chances to actually hit the object rise through the phase up to about 273 hits every

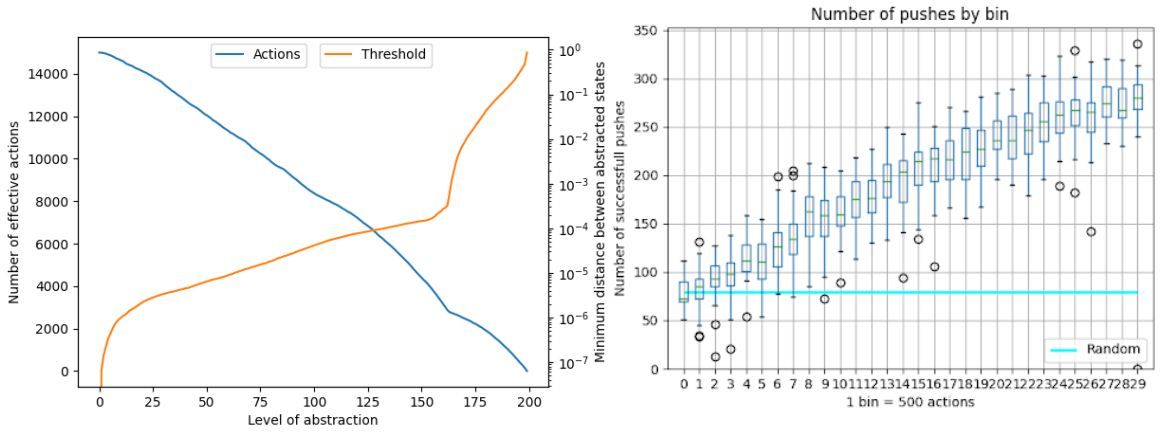


Figure 3: Left: REAL-D 'effective' actions at different abstraction levels; Right: REAL-ILD progressive increase of successful actions through exploration

500 actions – about half of the times. As shown in Table 1, in the REAL_IM condition the higher number of actions discovered allows REAL-ILD to achieve a higher performance than REAL-LD, 0.234 vs. 0.222. In the REAL_IJ condition, these figures become 0.149 vs. 0.139. This performance is only marginally higher so we investigated how the REAL-X performance scales with the number of actions. We found that the performance of REAL-LD is not really limited by the number of experienced actions as it already reaches a plateau with an 8ML-step intrinsic phase (8,000 actions), with most of the performance acquired during the first 2,000 actions. These results suggest that to improve the performance exploration should not simply provide 'more actions' but rather finding actions that have a higher precision and actions to bring the object onto the shelf. On the other hand, the change predictor can still be very useful when the performance is limited by the small number of actions collected as it improved the speed of finding effective actions up to three-fold. This can be especially relevant when using real robots for which exploration can be very time consuming.

4 Conclusions

Open-ended learning has still not produced robots able of truly open-ended learning. The REAL competition represents a benchmark usable to overcome this challenge as it allows a rigorous comparison of different intrinsically motivated learning models. This is done by measuring how the models perform with a set of representative goals drawn in the environment. In this work we presented several versions of a robot architecture, REAL-X, that can be used to face increasingly complex versions of the benchmark. The architecture solves the benchmark challenges by abstracting the images to identify objects, by selecting promising actions for exploration, and by dynamically abstracting over state representations to support planning. The performance of the architecture variants was well above the chance level both when some simplifications were kept and in the most challenging conditions. However, the performance was still far from the maximum, thus indicating that the REAL benchmark still presents interesting open challenges to be solved.

References: **[1]** G. Baldassarre. "What are intrinsic motivations? A biological perspective". In: *Proceedings of ICDL*. Frankfurt am Main, Germany, 24–27/08/2011. 2011. **[2]** G. Baldassarre and M. Mirolli. *Intrinsically Motivated Learning in Natural and Artificial Systems*. Berlin: Springer, 2013. **[3]** G. Baldassarre et al. "An embodied agent learning affordances with intrinsic motivations and solving extrinsic tasks with attention and one-step planning". In: *Frontiers in Neurorobotics* 13.45 (2019). **[4]** A. G. Barto and S. Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *Discrete event dynamic systems* 13.1-2 (2003). **[5]** A. G. Barto, S. Singh, and N. Chentanez. "Intrinsically motivated learning of hierarchical collections of skills". In: *Proceedings of the 3rd ICDL*. 2004. **[6]** D. E. Berlyne. *Conflict, arousal, and curiosity*. McGraw-Hill Book Company, 1960. **[7]** A. Cangelosi and M. Schlesinger. *Developmental robotics: From babies to robots*. MIT Press, 2015. **[8]** E. Cartoni and G. Baldassarre. "Autonomous discovery of the goal space to learn a parameterized skill". In: *arXiv preprint arXiv:1805.07547* (2018). **[9]** E. Cartoni et al. *An open-ended learning architecture to face the REAL 2020 simulated robot competition*. arXiv preprint arXiv:2011.13880v1. 2020. **[10]** E. Cartoni et al. "REAL-2019: Robot open-Ended Autonomous Learning competition". In: *Proceedings of Machine Learning Research* 1 (2020). **[11]** S. Doncieux et al. "Open-Ended Learning: A Conceptual Framework Based on Representational Redescription." In: *Frontiers in neurorobotics* 12.59 (2018). **[12]** M. Lungarella et al. "Developmental robotics: a survey". In: *Connection science* 15 (2003). **[13]** P.-Y. Oudeyer, F. Kaplan, and V. Hafner. "Intrinsic motivation systems for autonomous mental development". In: *IEEE transactions on evolutionary computation* 11.6 (2007). **[14]** V. G. Santucci, G. Baldassarre, and M. Mirolli. "Cumulative learning through intrinsic reinforcements". In: *Evolution, Complexity and Artificial Life*. Springer, 2014. **[15]** M. Schembri, M. Mirolli, and G. Baldassarre. "Evolving childhood's length and learning parameters in an intrinsically motivated reinforcement learning robot". In: *Proceedings of EpiRob2007*. Ed. by L. Berthouze et al. 2007. **[16]** J. Schmidhuber. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)". In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010). **[17]** R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

An Analysis of Measure-Valued Derivatives for Policy Gradients

João Carvalho
Department of Computer Science
Technische Universität Darmstadt, Germany
joao@robot-learning.de

Jan Peters
Department of Computer Science
Technische Universität Darmstadt, Germany
jan.peters@tu-darmstadt.de

Abstract

Reinforcement learning methods for robotics are increasingly successful due to the constant development of better policy gradient techniques. A *precise* (low variance) and *accurate* (low bias) gradient estimator is crucial to face increasingly complex tasks. Traditional policy gradient algorithms use the likelihood-ratio trick, which is known to produce unbiased but high variance estimates. More modern approaches exploit the reparametrization trick, which gives lower variance gradient estimates but requires differentiable value function approximators. In this work, we study a different type of stochastic gradient estimator - the Measure-Valued Derivative. This estimator is unbiased, has low variance, and can be used with differentiable and non-differentiable function approximators. We empirically evaluate this estimator in the actor-critic policy gradient setting and show that it can reach comparable performance with methods based on the likelihood-ratio or reparametrization tricks, both in low and high-dimensional action spaces. With this work, we want to show that the Measure-Valued Derivative estimator can be a useful alternative to other policy gradient estimators.

Keywords: Reinforcement Learning, Policy Gradients,
Measure-Valued Derivatives

An extended version of this work can be found at https://arxiv.org/pdf/2107.09359.pdf.

1 Introduction

Complex robotics tasks, such as locomotion and manipulation, can be formulated as Reinforcement Learning (RL) problems in continuous state and action spaces [1]. In these settings, the RL objective is commonly an expectation dependent on a parameterized policy, whose parameters are optimized via gradient ascent with the policy gradient theorem [2]. As this gradient cannot always be obtained in closed-form, there are three main approaches to obtain an unbiased estimate: the likelihood-ratio, also called Score-function (SF); the Pathwise Derivative, commonly known as the Reparametrization trick (Rep-trick); and the Measure-Valued Derivative (MVD) [3]. The SF has been use extensively in policy gradients, but is known to produce high-variance estimates. REINFORCE is an example of a SF based method [4]. Generally, the Rep-trick [5] is a low variance gradient estimator, but it requires the optimized function to be differentiable, which excludes non-differentiable approximators, such as regression trees . Soft Actor-Critic (SAC) [6] is an example of an algorithm that uses this estimator. MVD is a third method to compute unbiased gradient estimates, which is not commonly used in the Machine Learning (ML) community. It generally has low variance and unlike the SF avoids computing an extra baseline for variance reduction. Few works have studied its application to RL [7, 8]. In this work, we analyze the properties of MVDs in actor-critic policy gradient algorithms in the Linear-Quadratic Regulator (LQR) problem, for which we know the true policy gradient, and with an off-policy algorithm for complex environments with high-dimensional spaces. The results suggest that this estimator can be an alternative to compute policy gradients in continuous action spaces.

2 Background

2.1 Reinforcement Learning and Policy Gradients

Let a Markov Decision Process (MDP) be defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma, \mu_0)$, where $\mathcal{S}$ is a continuous state space $\boldsymbol{s} \in \mathcal{S}$, $\mathcal{A}$ is a continuous action space $\boldsymbol{a} \in \mathcal{A}$, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a transition probability function, with $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ the density of transitioning to state $\boldsymbol{s}'$ when taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor, and $\mu_0 : \mathcal{S} \to \mathbb{R}$ the initial state distribution. A policy π defines a (stochastic) mapping from states to actions. The discounted state distribution under a policy π is given by $d^{\pi}_{\gamma}(\boldsymbol{s}) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P(\boldsymbol{s}_t = \boldsymbol{s}|\boldsymbol{s}_0, \pi)$, where $\boldsymbol{s}_0$ is the initial state, and $P : \mathcal{S} \to \mathbb{R}$ the probability of being in state $\boldsymbol{s}$ at time step t. The state-action value function is the discounted sum of rewards collected from a state-action pair following the policy π, $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)|\boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right]$, and the state value function is its expectation w.r.t. the policy $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi}\left[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})\right]$. The advantage function is the difference between the two $A^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - V^{\pi}(\boldsymbol{s})$. The goal of a RL agent is to maximize the expected sum of discounted rewards from any initial state $J(\pi) = \mathbb{E}_{\boldsymbol{s}_0\sim\mu_0}\left[V^{\pi}(\boldsymbol{s}_0)\right]$. In continuous action spaces, it is common to optimize J using gradient ascent with the policy gradient theorem [2] $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\boldsymbol{s}\sim\mu^{\pi}_{\gamma}}\left[\int \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\theta}) Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a}\right]$. Note that the integral over the action space is not an expectation.

2.2 Monte Carlo Gradient Estimators

The objective function of several problems in ML, e.g. Variational Inference (VI), is often posed as an expectation of a function f w.r.t. a distribution p parameterized by $\boldsymbol{\omega}$, $J(\boldsymbol{\omega}) = \mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\omega})}\left[f(\boldsymbol{x})\right] = \int p(\boldsymbol{x}; \boldsymbol{\omega}) f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$, where $f : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function of $\boldsymbol{x} \in \mathbb{R}^n$, $p : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the distribution of $\boldsymbol{x}$, and $\boldsymbol{\omega} \in \mathbb{R}^m$ are the parameters encoding the distribution - also known as distributional parameters. For instance, in amortized VI [5] , f is the Evidence Lower Bound (ELBO), and if p is a multivariate Gaussian that approximates the posterior distribution, $\boldsymbol{\omega}$ aggregates the mean and covariance. A gradient ascent algorithm is a common choice to find the parameters that maximize $J(\boldsymbol{\omega})$, for which we need to compute $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \int \nabla_{\boldsymbol{\omega}} p(\boldsymbol{x}; \boldsymbol{\omega}) f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$. Since the derivative of a distribution is in general not a distribution itself, the integral is not an expectation and thus not solvable directly by Monte Carlo (MC) sampling. If f also depends on $\boldsymbol{\omega}$, the product rule is applied. Let a MC estimation of the gradient be obtained as $\widehat{\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})} \approx 1/M \sum_{i=1}^{M} \hat{\boldsymbol{g}}_i$, with $\hat{\boldsymbol{g}}_i \in \mathbb{R}^m$ an unbiased gradient estimate. There are three known ways to build an unbiased estimator of $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$ - Rep-trick, SF and MVD. Due to space constraints, we refer the reader to [3] for more details on the Rep-trick and SF estimators.

Measure-Valued Derivative (MVD) [9] Even though in general the derivative of a distribution w.r.t. its parameters is not a distribution, it is a difference between two distributions up to a normalizing constant [9]. The main idea behind MVD is to write the derivative w.r.t. a single distributional parameter $\omega_k \in \mathbb{R}$ as a difference between two distributions, i.e. $\nabla_{\omega_k} p(\boldsymbol{x}; \boldsymbol{\omega}) = c_{\omega_k}\left(p^{+}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega}) - p^{-}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega})\right)$, where c_{ω_k} is a normalizing constant that can depend on ω_k, and $p^{+}_{\omega_k}$ and $p^{-}_{\omega_k}$ are two distributions referred as the positive and negative components, respectively. The MVD is the triplet $\left(c_{\omega_k}, p^{+}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega}), p^{-}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega})\right)$. For common distributions, such as the Gaussian, Poisson, or Gamma, decompositions have been analytically derived [3]. The MVD decomposition gives $\nabla_{\omega_k} J(\boldsymbol{\omega}) = \int c_{\omega_k}\left(p^{+}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega}) - p^{-}_{\omega_k}(\boldsymbol{x}; \boldsymbol{\omega})\right) f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$. I.e., the derivative is the difference of two expectations scaled with the normalization constant. For example, to compute the derivative of a one-dimensional Gaussian $\mathcal{N}(x; \mu, \sigma)$ w.r.t. σ, we can sample the positive part from a Double-sided Maxwell and the negative one from a Gaussian.

The gradient w.r.t. all parameters is the concatenation of the derivatives w.r.t. all individual parameters ω_k, which results in $\mathcal{O}(2|\boldsymbol{\omega}|)$ queries to f to compute the full gradient. In contrast, the SF and Rep-trick have $\mathcal{O}(1)$ complexity. The variance of this estimator is $\mathbb{V}_{p(\boldsymbol{x};\boldsymbol{\omega})}\left[\hat{g}_k^{\text{MVD}}\right] = \mathbb{V}_{p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})}\left[f(\boldsymbol{x})\right] + \mathbb{V}_{p^-_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})}\left[f(\boldsymbol{x})\right] - 2\text{Cov}_{p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})p^-_{\omega_k}(\boldsymbol{x}';\boldsymbol{\omega})}\left[f(\boldsymbol{x}), f(\boldsymbol{x}')\right]$ [3], which depends on the chosen decomposition and how correlated are the function evaluations at the positive and negative samples. If p is a multivariate distribution with independent dimensions it factorizes as $p(\boldsymbol{x};\boldsymbol{\omega}) = \prod_{i=1}^{n} p(x_i;\boldsymbol{\xi}_i)$, with $x_i \in \mathbb{R}$ and $\boldsymbol{\xi}_i$ a subset of $\boldsymbol{\omega}$ (if $p(\boldsymbol{x};\boldsymbol{\omega})$ is a Gaussian with diagonal covariance, then $\boldsymbol{\xi}_i = \{\mu_i, \sigma_i\}$). The derivative w.r.t. one parameter $\omega_k \in \boldsymbol{\xi}_k$ is given by $\partial/\partial\omega_k p(\boldsymbol{x};\boldsymbol{\omega}) = c_{\omega_k}\left(p^+_{\omega_k}(x_k;\boldsymbol{\xi}_k) - p^-_{\omega_k}(x_k;\boldsymbol{\xi}_k)\right)\prod_{i=1,i\neq k}^{n} p(x_i;\boldsymbol{\xi}_i) = c_{\omega_k}\left(p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega}) - p^-_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})\right)$, where $p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})$ is the original multivariate distribution with the k-th component replaced by the positive part of the decomposition of the univariate marginal $p(x_k;\boldsymbol{\xi}_k)$ w.r.t. ω_k (the negative component is analogous).

Illustrative Example Fig. 1 shows the optimization of the expectation of common test functions w.r.t. a 2-dimensional Gaussian distribution with diagonal covariance, using gradient ascent. All estimators use 8 MC samples per gradient update. The SF uses an optimal baseline for variance reduction. Rep-trick and MVD consistently move towards the global maximum, while the SF shows unstable behaviors.

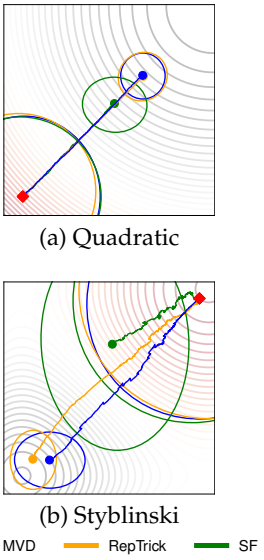


Figure 1: Sample run of the unbiased gradient estimators in test functions. Lines show the means of the distributions, and ellipses one standard deviation at the beginning (diamond) and end (circle).

3 Actor-Critic Policy Gradients with Measure-Valued Derivatives

Given the properties of MVDs, especially its low variance, we propose to analyze them in actor-critic policy gradients, which can be seen as computing an unbiased stochastic gradient estimate of the expected return. Let $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega} = g(\boldsymbol{s};\boldsymbol{\theta}))$ be a stochastic policy, where $\boldsymbol{\omega}$ are distributional parameters resulting from applying g with parameters $\boldsymbol{\theta}$ to the state $\boldsymbol{s}$, e.g. neural networks that output the mean and covariance of a Gaussian distribution. Since the gradient of g w.r.t. $\boldsymbol{\theta}$ can be easily computed if g is a continuous deterministic function, we only consider the gradient w.r.t. $\boldsymbol{\omega}$. In Table 1 we write the policy gradient theorem for a single parameter ω_k with the three different estimators. The MVD formulation of the policy gradient shows that to compute the gradient of one distributional parameter we can sample from the discounted state distribution by interacting with the environment, and then evaluate the Q-function at actions sampled from the positive and negative components of the policy decomposition conditioned on the sampled states, $\pi^+_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})$ and $\pi^-_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})$, respectively. Importantly, the Q-function estimate is the one from π and not from π^+ or π^-. The function approximator for Q can be a differentiable one, such as a neural network, or a non-differentiable one, such as a regression tree . For the SF estimator, the Q-function is replaced by the advantage function , which keeps the estimator unbiased but has lower variance.

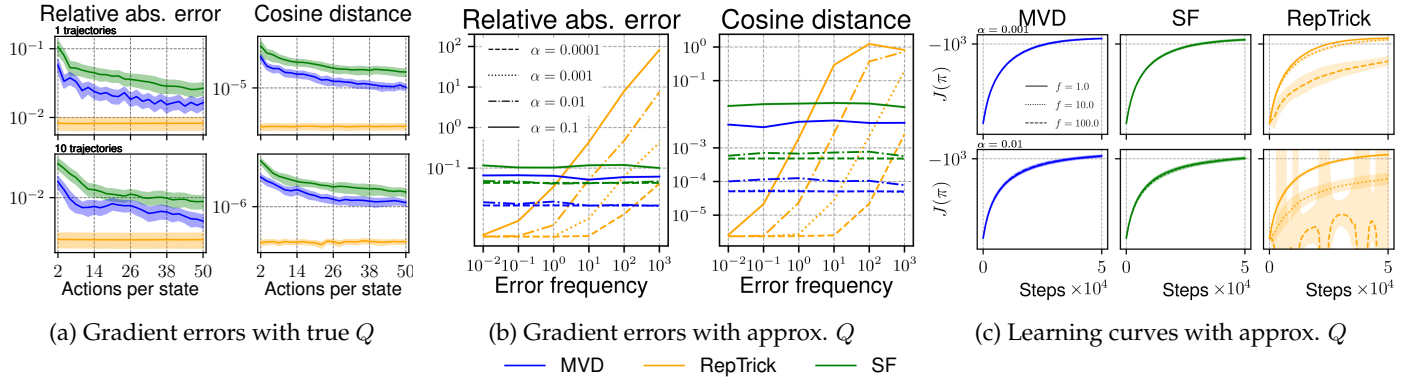
The MVD formulation assumes we can query the Q-function for the same state with multiple actions. While previous work [7] assumed access to a simulator that can estimate the return for different actions starting from the same state, e.g. with MC rollouts, this scenario is not generally applicable to RL, where the agent cannot easily reset to an arbitrary state, especially in real-world environments. Hence, we assume a critic approximator is available, e.g. fitted from samples.

Score-Function	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s}\sim\mu^\pi_\gamma,\, \boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[\nabla_{\omega_k}\log\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega})Q^\pi(\boldsymbol{s},\boldsymbol{a})\right]$
Reparametrization Trick	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s}\sim\mu^\pi_\gamma,\, \boldsymbol{\varepsilon}\sim p_{\boldsymbol{\varepsilon}}}\left[\nabla_{\boldsymbol{a}} Q^\pi(\boldsymbol{s},\boldsymbol{a}=h(\boldsymbol{s},\boldsymbol{\varepsilon};\boldsymbol{\omega}))\nabla_{\omega_k} h(\boldsymbol{s},\boldsymbol{\varepsilon};\boldsymbol{\omega})\right]$
Measure-Valued Derivative	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s}\sim\mu^\pi_\gamma}\left[c_{\omega_k}\left(\mathbb{E}_{\boldsymbol{a}\sim\pi^+_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[Q^\pi(\boldsymbol{s},\boldsymbol{a})\right] - \mathbb{E}_{\boldsymbol{a}\sim\pi^-_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[Q^\pi(\boldsymbol{s},\boldsymbol{a})\right]\right)\right]$

Table 1: Policy gradient theorem with the different unbiased stochastic gradient estimators.

3.1 Gradient Analysis in the Linear Quadratic Regulator

We consider a discounted infinite-horizon discrete-time LQR with a linear Gaussian stochastic policy $\pi(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a}|-\mathbf{K}\boldsymbol{s},\boldsymbol{\Sigma})$, where $\mathbf{K} \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|}$ is a learnable feedback gain matrix, and $\boldsymbol{\Sigma} \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{A}|}$ a fixed covariance. Since the value function is known and can be computed by numerically solving the Algebraic Riccati Equation, as well as the gradient w.r.t. $\mathbf{K}$, the LQR is a good baseline for policy gradient algorithms. We construct an LQR with $|\mathcal{S}| = 2$ states and $|\mathcal{A}| = 1$ action, with dynamics such that the uncontrolled system is unstable, and select randomly a suboptimal initial gain $\mathbf{K}_{\text{init}}$ such that the closed-loop is stable. The initial state is uniformly drawn and fixed for all rollouts. We compare the policy gradient of the expected return w.r.t. $\mathbf{K}_{\text{init}}$ with the estimators from Table 1. For the SF, we replace the Q-function with the advantage function for variance reduction. The expectation of the discounted on-policy state distribution is sampled by interaction with the environment, but the expectation over actions uses the LQR's true critics Q and V, to represent an idealized scenario.



(a) Gradient errors with true Q

(b) Gradient errors with approx. Q

(c) Learning curves with approx. Q

Figure 2: Gradient analysis for a LQR with 2 states and 1 action. a) Gradient errors in magnitude and direction for 1 and 10 trajectories (rows) and sampled actions. b) Gradient errors with increasing error noise and frequency. The different linestyles correspond to different error amplitudes α. The results are estimated using 10 trajectories and $20|\mathcal{A}|$ actions per state. c) Learning curves for different error amplitudes (rows) and frequencies. The gradients are estimated using 1 trajectory and $2|\mathcal{A}|$ actions per state. Some configurations lead to indistinguishable results, and thus plots appear superposed. The solid lines depict the mean and the shaded area the 95% confidence interval of 25 random seeds.

We study two sources of errors - the *relative absolute error* $(|\|\hat{\boldsymbol{g}}\| - \|\boldsymbol{g}\||)/\|\boldsymbol{g}\|$ that relates the magnitude of the estimated $\hat{\boldsymbol{g}}$ and true gradient $\boldsymbol{g}$, and the *cosine distance* $1 - \hat{\boldsymbol{g}}^\intercal \boldsymbol{g}/(\|\hat{\boldsymbol{g}}\|\|\boldsymbol{g}\|)$ that is the direction error. An ideal estimator has both errors close to zero. The errors are analyzed along two dimensions – the number of trajectories and the number of actions sampled to solve the action expectations. The results from Fig. 2a show that, as expected, with the number of trajectories fixed, increasing the number of sampled actions decreases the estimators' errors. The Rep-trick achieves the best results in both magnitude and direction in all environments, and the MVD is slightly better than the SF. In our experiments we observed this holds for higher dimensions as well. This reveals that even though the SF complexity is $\mathcal{O}(1)$, in practice we need roughly the same number of samples as MVD to obtain a low error gradient estimate. Knowing that the Q-function is quadratic in the action space, the results are in line with the example from Fig. 1a where a quadratic function was optimized and all estimators performed equally well. Next, we consider a scenario where the true Q-function has to be estimated. For that we model the approximator with a local approximation error on top of the true value as $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha Q(\boldsymbol{s}, \boldsymbol{a}) \cos(2\pi f \boldsymbol{p}^\intercal \boldsymbol{a} + \phi)$, i.e. we add noise proportional to the true estimate, where α represents the fraction of the true Q amplitude, f is the error frequency, $\boldsymbol{p}$ is a random vector sampled from a symmetric Dirichlet distribution, and $\phi \sim \mathcal{U}[0, 2\pi]$ a phase shift. $\boldsymbol{p}$ and ϕ introduce randomness to remove correlation between action dimensions. Fig. 2b shows the gradient errors in magnitude and direction as a function of the error frequency and amplitude. The error frequency does not affect the gradient estimation of SF or MVD, as can be seen by the horizontal blue and green lines. On the contrary the Rep-trick is heavily affected by both. This result is in line with the theory, since under a Gaussian distribution the variance of the Rep-trick is upper-bounded by the Lipschitz constant of the derivative of Q [3]. Fig. 2c shows the expected discounted return per steps taken in the environment with different amplitudes and frequencies of errors. The SF and MVD do not suffer from errors in the estimation and converge towards the optimal policy even in the presence of high-frequency error terms. The Rep-trick on the other hand either shows slower convergence or fails to converge due to the poor gradient estimates. From these experiments we observe that even though the Rep-trick provides the most precise and accurate gradient under a true value function, this does not always hold for approximated functions, especially when there is an action correlated error.

3.2 MVDs in Off-Policy Policy Gradient for Deep Reinforcement Learning

Next we illustrate how MVDs can be used in a deep RL algorithm such as SAC, whose surrogate objective is $J_\pi(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s}\sim d^\beta, \boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[Q^\pi(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}) - \alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\omega})\right] = \mathbb{E}_{\boldsymbol{s}\sim d^\beta, \boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega})\right]$, where d^β is an off-policy state distribution, Q is a neural network parameterized by $\boldsymbol{\phi}$, α weighs the entropy regularization term, and $\boldsymbol{\omega}$ are the distributional parameters of π. SAC estimates the gradient w.r.t. $\boldsymbol{\omega}$ using the Rep-trick. Instead we propose to compute it using MVD and name this modification as SAC-MVD, whose gradient w.r.t. a single parameter ω_k becomes $\nabla_{\omega_k} J_\pi(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s}\sim d^\beta}\left[c_{\omega_k}\left(\mathbb{E}_{\boldsymbol{a}\sim\pi^+_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega})\right] - \mathbb{E}_{\boldsymbol{a}\sim\pi^-_{\omega_k}(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega})\right]\right) - \mathbb{E}_{\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s};\boldsymbol{\omega})}\left[\nabla_{\boldsymbol{\omega}} \alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\omega})\right]\right]$, where $\pi^+_{\omega_k}$ and $\pi^-_{\omega_k}$ are the positive and negative components of the MVD decomposition.

We benchmark SAC with the different gradient estimators in high-dimensional continuous control tasks from the PyBullet simulator with: SAC-MVD with one MC sample; SAC-SF - a version of SAC using the SF; SAC-SF-extra-samples - same as SAC-SF but with the same number of queries as SAC-MVD per gradient estimate; SAC-extra-samples - same as SAC but with the same number of gradient estimates as SAC-SF-extra-samples; DDPG [10] and TD3 [11] - two off-policy algorithms. The average reward curves obtained during training are shown in Fig. 3. SAC with Rep-trick and SAC-MVD

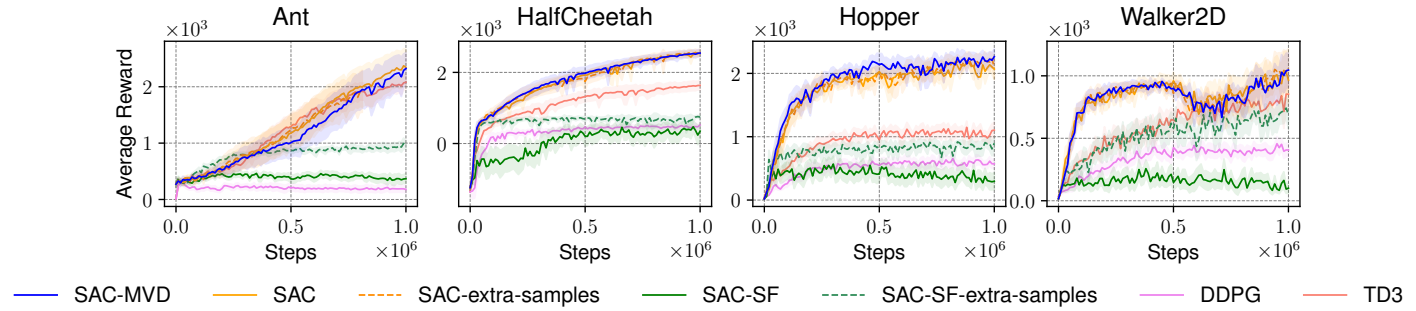


Figure 3: Policy evaluation results during training on different tasks in deep RL. The lines depict the mean of the average reward per samples collected and the shaded area the 95% confidence interval of 25 random seeds.

show similar performance, and increasing the number of MC samples does not improve the overall results (SAC-extra-samples performs equally well). Even though SAC-MVD needs more forward passes of the Q-function, SAC needs to backpropagate through it, and so we found the computation time to be similar in our implementation. The results of our experiments reveal that the Rep-trick is not fundamental for the performance of SAC, and suggest that the superior performances of this algorithm depend on other aspects, such as the entropy regularization, the state-dependent covariance and the squashed Gaussian policy. Additionally, it is worth noticing that MVDs are applicable to function classes where the Rep-trick is not, which allows to explore other approximator classes while using the benefits of SAC.

4 Conclusion

We presented MVDs as an alternative to the SF and Rep-trick estimators for actor-critic policy gradient algorithms. The empirical results showed that methods based on the MVD are a viable alternative to the other two, and differently from [7], we avoided resetting the environment to a specific state, showing how MVDs are applicable to the general RL framework. In the simple LQR environment and with an oracle critic, the MVD performs better than the SF and worse than the Rep-trick. However, with an action dependent error, the MVD and SF estimates are not affected by the error frequency, while the Rep-trick is sensitive to it. In tasks with high-dimensional action spaces we obtain comparable results with the Rep-trick using only one gradient estimate, which shows that it is not crucial for the SAC algorithm. Furthermore, unlike in SAC, MVDs do not require a differentiable Q-function, allowing the use of other types of function approximators. In future work we will investigate how to reduce the computational complexity of MVDs by computing derivatives along important dimensions and using a convex combination the estimators, which still remains unbiased [3].

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, USA: A Bradford Book, 2018.

[2] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 1999, pp. 1057–1063.

[3] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *Journal of Machine Learning Research*, vol. 21, no. 132, pp. 1–62, 2020.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, 1992.

[5] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 1856–1865.

[7] S. Bhatt, A. Koppel, and V. Krishnamurthy, "Policy gradient using weak derivatives for reinforcement learning," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 5531–5537.

[8] V. Krishnamurthy and F. V. Abad, "Real-time reinforcement learning of constrained markov decision processes with weak derivatives," 2011.

[9] G. C. Pflug, "Sampling derivatives of probabilities," *Computing*, vol. 42, no. 4, pp. 315–328, 1989.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations, (ICLR)*, 2016.

[11] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 1587–1596.

Probing the function of the dorsal striatum in rats navigating a two-step task using model-free learning

Yifeng Cheng and Eric Garr
Department of Psychological and Brain Sciences
Johns Hopkins University
ycheng62@jhu.edu
egarr1@jhu.edu

Cecelia Shuai
Department of Psychological and Brain Sciences
Johns Hopkins University
xshuai3@jhu.edu

Nicholas Malloy
Department of Psychological and Brain Sciences
Johns Hopkins University
nmalloy5@jhu.edu

Patricia H. Janak
Department of Psychological and Brain Sciences
Solomon H. Snyder Department of Neuroscience
Johns Hopkins University
patricia.janak@jhu.edu

Abstract

The two-step task has classically been used to distinguish between model-free and model-based learning and decision-making. Here, we designed a two-step task for rats that amended the original design to more faithfully detect model-based behavior, should it arise. We find that rats are overwhelmingly model-free in their choice behavior. The best-fitting model was a combination of simple response rules that do not require the computation of intermediate variables like value or probability. Rats also experienced optogenetic inhibition in the dorsomedial and dorsolateral striatum during a subset of trials at the onset of two epochs: the second step state immediately following choice execution, and during the time of trial outcome. Research is ongoing, but for now the effect of optogenetic inhibition is confined to the former epoch—experiencing laser stimulation in the dorsolateral striatum during a common state-action transition increased the probability of repeating the action that led to that transition.

Keywords: rat, model-free, model-based, striatum

Acknowledgements

We are grateful for the support of NIH grants R01 DA035943-09 and R01 AA026306-04 awarded to PHJ .

1 Introduction

There is a degree to which decisions are based on the detailed anticipation of future consequences versus cached values that reflect a summary of past experience. These differing decision-making strategies can be quantified in the computations of model-based and model-free reinforcement learning, respectively. In humans, there is a rich literature probing the determining conditions of these learning and decision-making strategies using the classic two-step task[1]. The use of the two-step task is vastly more limited in rodents, with mixed findings. In rats, behavior is either dominated by model-based learning or a combination of model-based and model-free learning[2,3]. In mice, behavior was initially reported as a mix of model-based and model-free learning strategies[4], but with more careful analysis was shown to be best explained by simple state-action rules that tend to produce behavior that appears model-based, but is actually model-free[5]. This latter finding comes from a study where the researchers introduced specific contingency reversals in the state transition structure of the task—something that has not been tried with rats. This change in the task structure is based on the recognition that simple state-action rules can flourish under the original implementation of the two-step task—giving rise to behavior that appears model-based when it truly is not[6]. It is, therefore, important to understand how this specific change in the task structure alters the tendency of rats to engage in model-based planning.

Model-based and model-free learning are thought to rely on separate parts of the brain. Specifically, the dorsomedial and dorsolateral striatum (DMS and DLS), when lesioned or pharmacologically inactivated, are necessary for behaviors analogous to model-based and model-free decision-making, respectively[7,8]. These findings come from studies in which rodents are trained to perform an action on loosely structured free-operant schedules and then tested following reward devaluation via satiety. However, no study to date has investigated whether the DMS and DLS also serve similar roles in the more structured two-step task. We therefore sought to quantify rat behavior on the updated two-step task while optogenetically inhibiting neural activity in the DMS and DLS within the same subjects during specific portions of task performance.

2 Methods

Male and female rats (n = 8) were trained in a two-step task. Rats were water restricted and trained in a sound-proof operant chamber. On the north wall was a reward magazine flanked by two levers, and on the south wall was a center magazine flanked by two nose ports (Figure 1A). Rats were gradually shaped to perform the task in six phased. First, rats were trained to collect reward (.1 ml of 10% sucrose in tap water) delivered randomly into the reward magazine. Second, rats were subsequently trained to lever press on a fixed-ratio 1 schedule until they earned 100 total rewards. Third, rats were trained to poke in the center magazine on the south wall to trigger lever insertion on the north wall, which allowed for a single press for one reward. Fourth, a simplified two-step task was initiated. Trials began with illumination of the south wall center magazine. Upon entry into this magazine, one of the side nose ports illuminated and required the rat to nose-poke (first-step choice). Once the rat did this, a tone began to play with the pitch indicating the side of lever insertion (second-step state). One nose port was always more commonly associated with one lever, and the specific mapping between a specific nose port and a given lever followed static 0.8/0.2 transition probabilities. After pressing the lever, the reward was delivered with 100% certainty. A clicker sound accompanied the reward. Fifth, the reward probabilities associated with two levers were probabilistic drawn from the set [0.8, 0.2] and reversed after 20-35 trials (Figure 1B). If no reward was delivered, a 1s white noise was presented. Sixth, and lastly, we introduced reversals into the transition matrix which map the first step choice to second-step state. These reversals occurred every 80-100 trials, and the two types of reversal were be separated by at least 20 trials (Figure 1C). Rats were trained on the final stage of the task for 30 daily sessions. We collected last 15 sessions for data analysis.

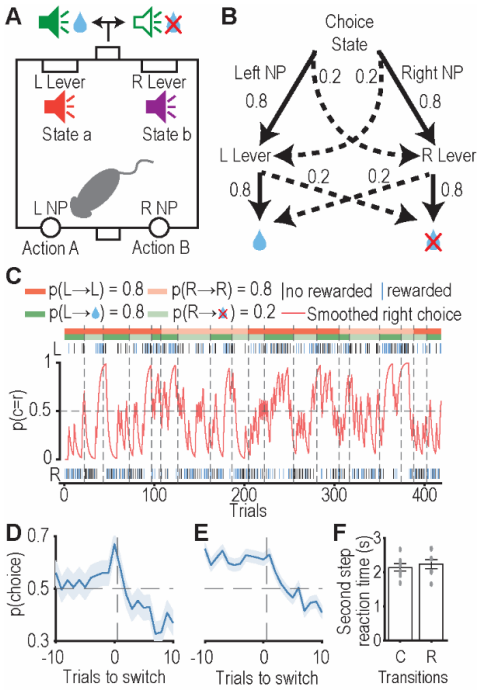


Figure 1. (**A**) Diagram of apparatus. L, left; R, right; NP, nose port. (**B**) The diagram outlines the transitions probabilities between action, state, and outcomes for one of four possible block types. (**C**) Example session. Red line represents the exponential moving averaged choices (red line; tau = 5 trials). Top raster shows choice on left and bottom raster showed choice on right. Rewarded choices are labeled in blue, otherwise in black. (**D,E**) Pre-reversal correct choice probability around state transition probability reversal (D) and reward probability reversal (E). Blue line is averaged probability across all subject and all sessions. Shaded area indicates SEM. (**F**) Reaction time from first step choice (nose poke) to second step response (lever press) with common and rare transitions. The gray dots represent individual subjects. Error bars show cross-subject SEM. *n* = 8 for D-F.

Next, a virally encoded inhibitory opsin was unilaterally infused into different hemisphere of the DMS and the DLS within the same rats. Optical fibers (300 μm, NA 0.37) were implanted into the DMS and the DLS accordingly. After surgery, rats continued to train for another 6 weeks (~30 sessions) to allow viral expression in striatal neurons. We conducted brief (20Hz, 500 ms) and probabilistic (10% of trials) optical inhibition via green laser activation aligned with the time immediately after the first-step choice.

3 Results

On each trial, we defined the "correct choice" as the first step choice leading to a common transition and high reward probability. Rats tracked the correct choice and adapted their choice after reversal (Figure 1D, 1E). Reaction times to reach and press the second-step lever were slightly slower with rare transition but not significantly different compared to those with common transitions (Figure 1F).

We next analyzed how trial outcome and state transition affected choice behavior. Stay probability resembled typical model-free behavior, with rats choosing to repeat the first-step action that led to reward on the previous trial and avoid the action which led to reward omission, regardless of state transition (Figure 2A). This result was confirmed using two types of logistic regression. First, choice probability was modeled as a function of the previous trial's transition (common vs. rare), outcome (reward vs. omission), and their interaction (Figure 2B). The outcome and transition variables served as the only predictors with significantly positive loadings. Second, choice probability was modeled as a function of four different trial types (reward-common, reward-rare, omission-common, omission-rare) up to five trials back (Figure 2C).

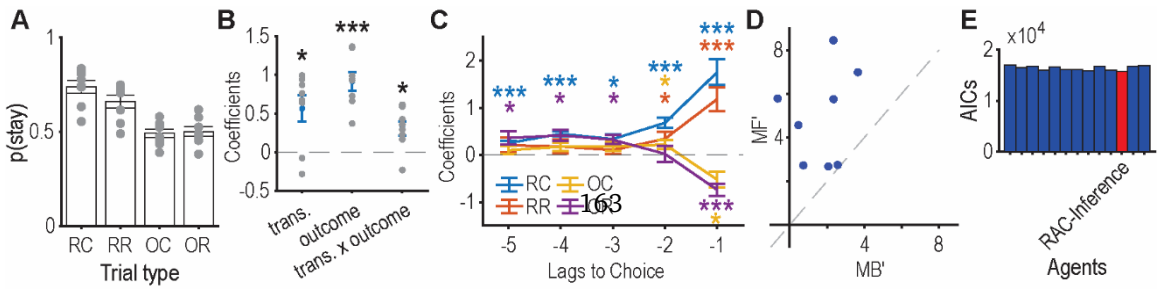


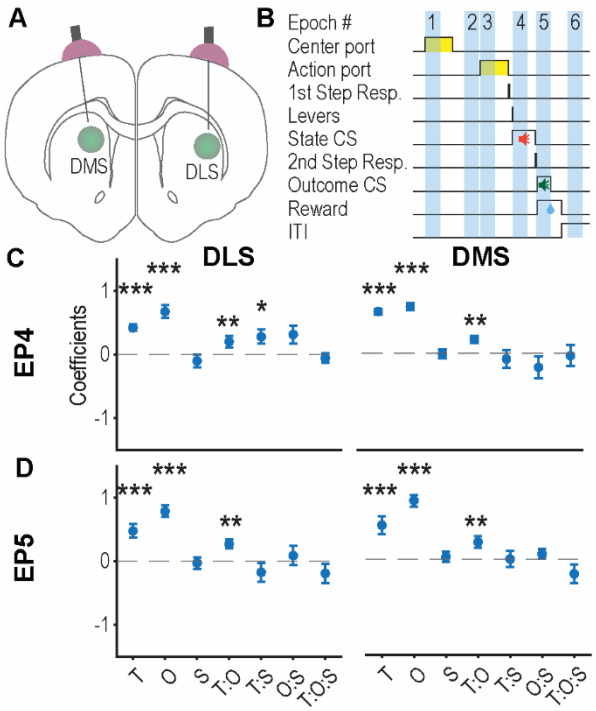
Figure 2. (**A**) Stay probability analysis showing the probability of repeating the same choice given that the last trial was rewarded (R) or not (O) and the transition was common (C) or rare (R). Gray dots represent individual subjects. (**B**) Logistic regression predicting choice as a function of last trial transition, outcome, and the interaction between them. Gray dots represent individual subjects. One-sample *t* test. **p<0.01,***p<0.001. (**C**) Five-trials back logistic regression predicts the probability of a right choice as a function of previous outcome type and transition type. One-sample *t* test. *p<0.05, **p<0.01, ***p<0.001. (**D**) Model-based (MB) and model-free (MF) indices computed from (C). Each dot represents an individual rat. (**E**) RL model comparison on the behavioral dataset using AIC scores. The winning model is highlighted in red. From left to right the models include model-based (MB), Q1, Q0, reward-as-cue (RAC), Inference, Q1 + MB, Q0+MB, RAC+MB, RAC+Q1, RAC+Q0, RAC+Inference, win-stay-lose-shift, random selection with motor bias. Error bars show cross-subject SEM for A-C. n = 8 for A-E.

Choice was mainly a function of the previous trial's outcome. To quantify the degree of model-free versus model-based behavior per rat, we computed indices according to the following equations using the five trials-back regression coefficients (β):

$$MF = \sum_{\tau=1}^{5}[\beta_{RC}(\tau) + \beta_{RR}(\tau)] - \sum_{\tau=1}^{5}[\beta_{OC}(\tau) + \beta_{OR}(\tau)]$$

$$MB = \sum_{\tau=1}^{5}[\beta_{RC}(\tau) - \beta_{RR}(\tau)] - \sum_{\tau=1}^{5}[\beta_{OC}(\tau) - \beta_{OR}(\tau)]$$

All rats were MF dominant (Figure 2D). Given that there are many varieties of model-free learning strategies, we fit various models to behavior using maximum likelihood. The model that best fit choice data was a mixture of reward-as-cue (RAC) and inference (Figure 2F). RAC involves using the identity of the previous trial's outcome and second-step state to update the value of the first-step actions, much like a stimulus-response mapping. Inference involves inferring a latent state representing which second-step lever is better or worse in terms of reward probability, and then choosing the first-step action based on the belief that the rat is in that latent state. This hybrid model was the winning model. RAC received, on average, a higher weight compared to inference at a mixture of 86.2% and 13.8%, respectively.



Optogenetic inhibition was applied to the DLS and DMS via green laser stimulation in separate sessions immediately after the first-step choice and simultaneous with the onset of the second-step state or outcome time (Figure 3A, 3B). We added laser stimulation as a regressor in addition to transition and outcome to predict nose poke choices. The effect of stimulation and its interaction with transition, outcome, and both transition and outcome did not reach significance except in one case—the transition x stimulation interaction was significant when targeting the DLS during the onset of the second step state (Figure 3C, left). Trials with a common transition and stimulation increased the stay probability compared to no stimulation (p < .05). There was no effect of stimulation on choice following rare transitions (p > .05).

Figure 3. (**A**) Coronal section showing locations of laser stimulation for optogenetic inhibition. (**B**) Stimulation epochs. Data is presented only from epoch 4. (**C, D**) Regression coefficients quantifying the effect of previous trial transition, outcome, stimulation, and their interactions on subsequent trial choice in epoch 4 (**C**) or epoch 5 (**D**), and their interactions (T:O, T:S, O:S, and T:O:S) on subsequent trial choice. One-sample *t* test. *p<0.05, **p<0.01, ***p < 0.001. *n* = 7 for C and 8 for D.

4 Discussion

Rats were trained to choose between two actions that probabilistically led to one of two states to earn probabilistic rewards. Unlike the classic two-step task, the probabilities governing the transitions between the first-step actions and second-step states switched occasionally over the course of a single session. We found that rats were overwhelmingly model-free in their choice behavior. We identified the reinforcement learning model that best fit behavior as a mixture between reward-as-cue and latent state inference. This hybrid model is the same model that also best fit mouse data using a similar task structure[4]. We wish to note that these models are simple rules that do not require representation of intermediate variables like values or probabilities.

Why did rats fail to use a model-based strategy? Previous research has shown that offering a single action instead of a choice at the second step incentivizes model-based planning in terms of proportion of trials rewarded[9]. However, given the added complexity of adding reversals in the action-state transitions, the time to engage in model-based computation may decrease the number of rewards per unit time. This is a difficult tradeoff. On the one hand, it has been argued that adding in this contingency reversal is what allows the detection of true model-based planning[5]. On the other hand, these reversals could tax subjects' cognition to the point of degrading the worth of model-based planning. To test this idea, we plan to remove this contingency reversal after the optogenetic portion of the experiment is complete. We also plan to complete more optogenetic manipulations by expanding the number of stimulation epochs (see Figure 3B).

References

[1]Gillan, C., Kosinski, M., Whelan, R., Phelps, E., & Daw, N. (2016). DOI: 10.7554/eLife.11305.001

[2]Miller, K.J., Botvinick, M.M., & Brody, C.D. (2017). DOI: 10.1038/nn.4613

[3]Groman, S.M., Massi, B., Mathias, S.R., Curry, D.W., Lee, D., & Taylor, J.R. (2019). DOI: 10.1523/JNEUROSCI.2219-18.2018

[4]Akam, T., Rodrigues-Vaz, I., Marcelo, I., Zhang, X., Pereira, M., Oliveira, R. F., Dayan, P., & Costs, R. M. (2021). DOI: 10.1016/j.neuron.2020.10.013

[5]Blanco-Pozo, M., Akam, T., & Walton, M. (2021). DOI: 10.1101/2021.06.25.449995

[6]Akam, T., Costa, R., & Dayan, P. (2015). DOI: 10.1371/journal.pcbi.1004648

[7]Yin, H., Osltund, S., Knowlton, B., & Balleine, B. (2005). DOI: 10.1111/j.1460-9568.2005.04218.x

[8]Yin, H., Knowlton, B., & Balleine, B. (2004). DOI: 10.1111/j.1460-9568.2004.03095.x

[9]Kool, W., Cushman, F. A., & Gershman, S. J. (2016). DOI: 10.1371/journal.pcbi.1005090

Faster Learning with a Team of Reinforcement Learning Agents

Stephen Chung
University of Massachusetts Amherst
minghaychung@umass.edu

Andrew G. Barto
University of Massachusetts Amherst
barto@cs.umass.edu

Abstract

Though backpropagation underlies nearly all deep learning algorithms, it is generally regarded as being biologically implausible. An alternative way of training an artificial neural network is through making each unit stochastic and treating each unit as a reinforcement learning agent, and thus the network is considered as a team of agents. As such, all units can learn via REINFORCE, a local learning rule modulated by a global reward signal that is more consistent with biologically observed forms of synaptic plasticity. However, this learning method suffers from high variance and thus the slow learning. The high variance stems from the lack of effective structural credit assignment. This paper reviews two recently proposed algorithms to facilitate structural credit assignment when all units learn via REINFORCE, namely *MAP Propagation* and *Weight Maximization*. In MAP Propagation an energy function of the network is minimized before applying REINFORCE, such that the activities of hidden units are more consistent with the activities of output units. In Weight Maximization the global reward signal to each hidden unit is replaced with the change in the squared L^2 norm of the vector of the unit's outgoing weights, such that each hidden unit is trying to maximize the norm of its outgoing weights instead of the external reward. Experiments show that both algorithms can learn significantly faster than a network of units learning via REINFORCE, and have a comparable speed to backpropagation when applied in standard reinforcement learning tasks. In contrast to backpropagation, both algorithms retain certain biologically plausible properties of REINFORCE, such as having local learning rules and the ability to be computed asynchronously. Therefore these algorithms may offer insights for understanding possible mechanisms of structural credit assignment in biological neural systems.

Keywords: backpropagation, synaptic plasticity, structural credit assignment, multi-agent reinforcement learning, hierarchical reinforcement learning

Overview

The *error backpropagation algorithm* (backprop) efficiently computes the gradient of an objective function with respect to parameters by iterating backward from the last layer of a multi-layer artificial neural network (ANN). However, backprop is generally regarded as being biologically implausible [1]. First, the learning rule given by backprop is non-local, as it relies on a feedback signal backpropagating from the downstream units, while biologically-observed synaptic plasticity depends mostly on local information (e.g. spike-timing-dependent plasticity (STDP) [2]) and possibly some global signals, e.g. reward-modulated spike-timing-dependent plasticity (R-STDP) [2]. Second, to compute the feedback signal, backprop requires synaptic symmetry in the forward and backward paths, which has not been observed in biological systems. Third, backprop cannot be implemented asynchronously across units since it requires the network to alternate precisely between the feedforward and feedback phase, and the feedback signal has to be backpropagated layer-by-layer. Nonetheless, recent work has shown that these issues may be overcome. For example, the feedback signal in backprop may be approximated locally by the difference in neural activities across time [3], and synaptic symmetry may not be necessary for backprop due to the 'feedback alignment' phenomenon [4], but asynchronous computation remains a major obstacle to biological plausibility.

Alternatively, each unit in an ANN can be made stochastic (i.e. each unit's activation value is sampled from some distributions instead of being computed deterministically) and learn via the REINFORCE learning rule [5], a special case of the associative reward-penalty $(A_{R-\lambda P})$ algorithm [6] that preceded REINFORCE. This learning method has also been called 'node perturbation' [3], and is generally considered more biologically plausible than backprop. REINFORCE, when applied to Bernoulli-logistic units, gives a three-factor learning rule which depends on a reward signal in addition to a unit's input and output signals. This three-factor learning rule is closely related to R-STDP, which depends on pre-synaptic and post-synaptic activity plus a neuromodulatory input [7]. Since the only non-local information required by the REINFORCE learning rule is the globally broadcasted reward signal, the three aforementioned issues of backprop regarding biological plausibility do not exist when all units learn via REINFORCE.

Another interpretation of training all units by REINFORCE relates to viewing each unit as a reinforcement learning (RL) agent, with each agent trying to maximize the same reward signal from the environment. We can thus view an ANN as a *team of agents* playing a cooperative game, a scenario where all agents receive the same reward; agents here refer to RL agents [8]. Such a team of agents is also known as *coagent network* [9]. The idea of solving a task by a team of agents has a long history, and we refer readers to [8, Chapter 15] for the related work. However, due to the weak correlation between the team's reward signal and the action of an agent in the team, this learning method is associated with high variance and thus the slow learning. The high variance stems from the lack of effective structural credit assignment.

This paper reviews two recently proposed algorithms, namely *MAP Propagation* [10] and *Weight Maximization* [11], that facilitate structural credit assignment when training all units by REINFORCE. MAP propagation replaces the hidden units' outputs with their maximum a posteriori (MAP) estimates conditioned on the state and the selected action, or equivalently, minimizes an energy function of the network, before each unit applies REINFORCE. For a network of normally distributed units (that is, an ANN with i.i.d Gaussian noise added to the activation value of each unit), by minimizing the energy function of the network, the parameter update given by REINFORCE and backprop with the reparametrization trick become the same, thus establishing a connection between REINFORCE and backprop. In Weight Maximization we replace the global reward signal to each hidden unit with the change in the squared L^2 norm of its *outgoing weight*, which is defined as the vector of the weights by which the unit's outputs influence other units in the network. With the replaced reward signals, each hidden unit in the network is trying to maximize the norm of its outgoing weight. We prove that Weight Maximization approximately follows the gradient of reward in expectation, showing that every hidden unit maximizing the norm of its outgoing weight also approximately maximizes the external reward.

Our experiments show that ANNs trained with MAP Propagation or Weight Maximization can learn much faster than REINFORCE, such that the learning speed is comparable to backprop on standard RL tasks[1]. The experimental results are shown in Fig 1. It is worth noting that both MAP Propagation and Weight Maximization only assume a scalar reward signal from the environment, so they are more suitable for RL tasks instead of supervised learning (SL) tasks. It remains to be investigated how to best incorporate the knowledge of optimal network outputs into both algorithms for competitive performance in SL tasks.

MAP Propagation and Weight Maximization retain certain properties relevant to biological plausibility of REINFORCE, but neither algorithms can be implemented asynchronously. Alternative versions of MAP Propagation and Weight Maximization that allow asynchronous computation have been proposed. In *asynchronous MAP Propagation* (to be published) the feedforward phase is merged with the energy minimization phase to remove the requirement of alternating between phases. In *Weight Maximization with traces* [11] eligibility traces are employed to remove the need of waiting for downstream units to finish updating. A comparison of different biologically inspired algorithms according to biological

[1]To be precise, REINFORCE here refers to the case of all units implementing the REINFORCE learning rule, and backprop here refers to the case of output units implementing the REINFORCE learning rule and hidden units implementing backprop.

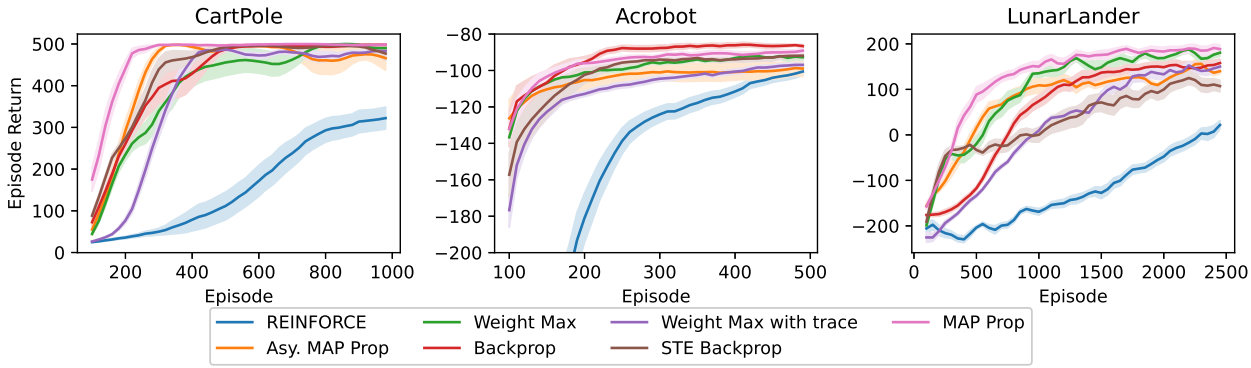


Figure 1: Episode returns in different RL tasks. Results are averaged over 10 independent runs, and shaded areas represent standard deviation over the runs. Curves are smoothed with a running average of 100 episodes. All networks have 64 and 32 units on the first and the second hidden layer respectively. REINFORCE, Weight Max and straight-through estimator (STE) backprop [12] use Bernoulli-logistic units; MAP Prop uses normally distributed units, and backprop uses Rectified Linear Units (ReLU). Hyper-parameters are selected based on manual tuning to optimize the average episode returns.

	Algorithm Type	Local learning rule	No symmetric feedback connections	Asynchronous computation across units
REINFORCE [5]	RL	✓	✓	✓
MAP Propagation [10]	RL	✓	✗	✗
Asy. MAP Propagation	RL	✓	✗	✓
Weight Max. [11]	RL	✓	✓	✗
Weight Max. with traces [11]	RL	✓	✓	✓
Backprop	SL	✗	✗	✗
Equilibrium Propagation [13]	SL	✓	✗	✗
Target Propagation [14]	SL	✓	✓	✗

Table 1: Comparison of biologically inspired algorithms on properties relevant to biological plausibility. Algorithm type: RL (Reinforcement Learning) - the algorithm only assumes the knowledge of a scalar reward signal from the environment; SL (Supervised Learning) - the algorithm assumes the knowledge of optimal network outputs from the environment. We call a learning rule local if it only depends on local variables (incoming weights, outgoing weights, activation values of the unit, incoming units, and outgoing units) and possibly a global scalar variable (e.g. reward signal in R-STDP). Backprop requires an error signal propagating from the outgoing units, so it is not local. Note that backprop can be combined with REINFORCE to train hidden units in RL tasks as in most deep RL algorithms.

plausibility properties is summarized in Table 1. Nonetheless, other properties relevant to biological plausibility of these algorithms remain to be investigated. For example, it is not yet clear if there exists a mechanism that allows changes in the efficacies of a neuron's outgoing synapses to influence changes in its incoming synapses.

From the computational perspective, MAP Propagation and Weight Maximization offer different advantages and disadvantages compared to backprop. Experiments showed that an ANN trained with MAP Propagation is less likely than bcakprop to become stuck in local optima in the MountainCar task [10], suggesting that stochastic activities of all units may lead to more effective exploration. However, each iteration of MAP Propagation is computationally more costly compared to each iteration of backprop due to the energy minimization phase. For Weight Maximization, experiments showed that it performs well when applied to discrete units such as Bernoulli-logistic units but performs poorly when applied on continuous units. Nonetheless, the ability to efficiently train discrete units and to localize optimization to each network unit may open the door to other RL methods for training ANNs.

Algorithms

For simplicity, we consider an MDP with only immediate reward[2] or, equivalently, the contextual bandit problem, defined by a tuple $(\mathcal{S}, \mathcal{A}, R, d_0)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of action, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,

[2]The algorithms in this paper can be applied in general MDPs by replacing the reward with the sum of discounted reward (i.e. return) or TD error, and can be applied in supervised learning tasks by replacing the reward with the negative loss.

and $d_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution. Denoting the state, action, and reward by S, A, and R respectively, $\Pr(S = s) = d_0(s)$ and $\mathbb{E}[R|S = s, A = a] = R(s, a)$. We are interested in learning the policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ such that selecting actions according to $\Pr(A = a|S = s) = \pi(s, a)$ maximizes the expected reward $\mathbb{E}[R|\pi]$.

Here we restrict attention to policies computed by a multi-layer ANN consisting of L layers of stochastic units. Let H^l denote the vector of activation values of layer l. To simplify notation, we also let $H^0 := S$ and $H^L := A$. For all $1 \leq l \leq L$, the distribution of H^l conditional on H^{l-1} is given by $\Pr(H^l_t = h^l|H^{l-1}_t = h^{l-1}; W^l) = \pi_l(h^{l-1}, h^l; W^l)$, where W^l is the parameter of layer l. We further assume[3] W^l is a matrix and $\pi_l(h^{l-1}, h^l; W^l) = \prod_i f(h^l_i, W^l_{:,i} \cdot h^{l-1})$ for some differentiable function f, where $\cdot$ denotes the dot product and $W^l_{:,i}$ denotes the i^{th} column of matrix W^l. Note that H^l can be discrete-valued or continuously-valued depending on the choice of π_l or the MDP. To sample an action A from the network, we iteratively sample $H^l \sim \pi_l(H^{l-1}, \cdot; W^l)$ from $l = 1$ to L.

The gradient of reward with respect to W^l (where $l \in \{1, 2, ..., L\}$ in all discussion below unless stated otherwise) can be estimated by REINFORCE [5] :

$$\nabla_{W^l} \mathbb{E}[R|\pi] = \mathbb{E}[R\nabla_{W^l} \log \pi(S, A)|\pi] = \mathbb{E}[R\nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l)|\pi]. \tag{1}$$

Therefore, to perform gradient ascent on the reward, we can update parameters by adding $\alpha R\nabla_{W^l} \log \pi(S, A)$ to W^l, where α is the step size. $\nabla_{W^l} \log \pi(S, A)$ can be computed by backprop for deterministic and differentiable ANNs, or backprop with the reparametrization trick for some stochastic and continuous ANNs (e.g. a network of normally distributed units). For stochastic and discrete ANNs (e.g. a network of Bernoulli-logistic units), $\nabla_{W^l} \log \pi(S, A)$ cannot be computed by backprop or backprop with the reparametrization trick. But the second equality tells us that we can also update parameters by applying REINFORCE locally to all units:

$$W^l \leftarrow W^l + \alpha R\nabla_{W^l} \log \pi_l(H^{l-1}, H^l; W^l). \tag{2}$$

This learning rule can applied in any stochastic ANNs. However, the parameter update in this learning rule has a large variance since a single reward R is used to evaluate the collective actions of all units. In the following we discuss two algorithms that help reduce this variance.

MAP Propagation - Let define an *energy function* by $E(h^1, h^2, ..., h^L; s) := -\log \Pr(H^1 = h^1, H^2 = h^2, ..., H^L = h^L|S = s)$. That is, the energy function measures the lack of compatibility between hidden units and output units. MAP Propagation minimizes the energy function of the network w.r.t. activation values of hidden units, before applying REINFORCE:

$$W^l \leftarrow W^l + \alpha R\nabla_{W^l} \log \pi_l(\hat{h}^{l-1}, \hat{h}^l; W^l), \tag{3}$$

where $\hat{h}^1, \hat{h}^2, ..., \hat{h}^{L-1} = \arg\min_{h^1, h^2, ..., h^{L-1}} E(h^1, h^2, ..., h^{L-1}, H^L; S)$ and $\hat{h}^L = H^L$. In other words, we try to make the activation values of hidden units to be more compatible with the selected action before applying REINFORCE. This energy minimization is equivalent to replacing the activation values of hidden units by their most probable activation values conditioned on the state and the selected action (i.e. MAP inference).

The remaining question is how to minimize the energy function. For continuous-valued units, we can perform gradient descent on the energy function w.r.t. activation values of hidden units, and the update rule can be shown to be local (since $\nabla_{h^l} E(h^1, h^2, ..., h^L; s)$ does not depend on h^k for any $k < l - 1$ or $k > l + 1$). In other words, MAP Propagation only requires units on adjacent layers to communicate via their activation values instead of separate error signals as in backprop. For discrete-valued units, we can use hill climbing methods or iteratively compute the minima for each unit.

Weight Maximization - Let define the *outgoing weight* of hidden unit i on layer l by the vector $W^{l+1}_{i,:}$ (the i^{th} row of matrix W^{l+1}), i.e. the weights connecting from that unit to units on the next layer. Weight Maximization uses a different approach from MAP Propagation. In learning rule (2), all units receive the same reward signal R. It is natural to ask whether there exists a more specific signal that evaluates the activation value of each unit, such that a more precise structural credit assignment is possible. Inspired by this idea, Weight Maximization replaces the external reward signal R to each hidden unit by the change in the squared L^2 norm of its outgoing weight. The motivation of using the change in the norm of a unit's outgoing weight as a reward signal is based on the idea that the norm of a unit's outgoing weight roughly reflects the contribution of the unit in the network. For example, if the hidden unit is useful in guiding action, then the output unit will learn a large weight associated with it. Conversely, if the hidden unit is outputting random noise, then the output unit will learn a zero weight associated with it.

Assuming that the next layer is also learning, the outgoing weight $W^{l+1}_{i,:}$ changes on every step, and we denote this change (before multiplying the step size α) by $\Delta W^{l+1}_{i,:}$. Thus, the change in the squared L^2 norm of the outgoing weight

[3] This assumption says that each unit adds the activation values of all units in a layer below multiplied by some weights to determine the distribution of its activation values. Note that this assumption is not necessary for MAP Propagation.

can be expressed as ($\cdot$ denotes the dot product):

$$||W_{i,:}^{l+1} + \alpha \Delta W_{i,:}^{l+1}||_2^2 - ||W_{i,:}^{l+1}||_2^2 = 2\alpha \Delta W_{i,:}^{l+1} \cdot W_{i,:}^{l+1} + \mathcal{O}(\alpha^2). \tag{4}$$

We propose to ignore $\mathcal{O}(\alpha^2)$ since the step size is usually very small, and thus we define the reward signal to unit i on layer l by:

$$R_i^l := \begin{cases} 2\alpha \Delta W_{i,:}^{l+1} \cdot W_{i,:}^{l+1} & \text{for } l \in \{1, 2, ..., L-1\}, \\ R & \text{for } l = L. \end{cases} \tag{5}$$

The learning rule is the same as (2) but with the new reward signal ($W_{:,i}^l$ denotes the i^{th} column of matrix W^l):

$$W^l \leftarrow W^l + \alpha \Delta W^l, \text{where } \Delta W_{:,i}^l = R_i^l \nabla_{W_{:,i}^l} \log \pi_l(H^{l-1}, H^l; W^l). \tag{6}$$

(5) and (6) together defines the learning rule for all units in Weight Maximization. It can be proved that the update rule is still approximately following the gradient of rewards in expectation.

However, (5) and (6) require iterating backward from the top layer since units need to wait for the upper layer to finish learning to compute the reward R_i^l. To remove this iteration requirement, we consider the case where all units are learning at the same time step. Then, it can be seen that the reward signal to a unit on layer l is delayed by $L - l$ time steps as there are $L - l$ upper layers and each layer requires one time step to update. The problem of delayed reward is well studied in RL, and one prominent and biologically plausible solution is eligibility traces. By using eligibility traces, the algorithm can be implemented in parallel for all layers, and there are no distinct feedforward and feedback phases for the whole network (see paper [11] for details of Weight Maximization with eligibility traces).

Conclusion

It has long been proposed that ANNs can be trained via each unit implementing an RL algorithm and learning from the same global reward signal. We review two algorithms that improve the learning speed of this learning method. By showing the possibility of efficient structural credit assignment, we hope this work reinvigorates interest in developing new learning methods for ANNs that employ RL, and that are both biologically plausible and fast enough to be useful in ANN applications.

References

[1] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[2] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[3] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020.

[4] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, no. 1, pp. 1–10, 2016.

[5] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[6] A. G. Barto, "Learning by statistical cooperation of self-interested neuron-like computing elements," *Human Neurobiology*, vol. 4, no. 4, pp. 229–256, 1985.

[7] J. Wickens, "Striatal dopamine in motor activation and reward-mediated learning: steps towards a unifying model," *Journal of Neural Transmission/General Section JNT*, vol. 80, no. 1, pp. 9–31, 1990.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[9] P. S. Thomas, "Policy gradient coagent networks," in *Advances in Neural Information Processing Systems*, pp. 1944–1952, 2011.

[10] S. Chung, "Map propagation algorithm: Faster learning with a team of reinforcement learning agents," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[11] S. Chung, "Learning by competition of self-interested reinforcement learning agents," in *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022. To appear.

[12] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[13] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[14] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515, Springer, 2015.

Deep Reinforcement Learning with Weighted Q-Learning

Andrea Cini
IDSIA, Università della Svizzera italiana
cinia@usi.ch

Carlo D'Eramo
IAS, TU Darmstadt

Jan Peters
IAS, TU Darmstadt
Max Planck Institute for Intelligent Systems

Cesare Alippi
IDSIA, Università della Svizzera italiana
Politecnico di Milano

Abstract

Reinforcement learning algorithms based on Q-learning are driving Deep Reinforcement Learning (DRL) research towards solving complex problems and achieving super-human performance on many of them. Nevertheless, Q-Learning is known to be positively biased since it learns by using the maximum over noisy estimates of expected values. Systematic overestimation of the action values coupled with the inherently high variance of DRL methods can lead to incrementally accumulate errors, causing learning algorithms to diverge. Ideally, we would like DRL agents to take into account their own uncertainty about the optimality of each action, and be able to exploit it to make more informed estimations of the expected return. In this regard, Weighted Q-Learning (WQL) effectively reduces bias and shows remarkable results in stochastic environments. WQL uses a weighted sum of the estimated action values, where the weights correspond to the probability of each action value being the maximum; however, the computation of these probabilities is only practical in the tabular setting. In this work, we provide methodological advances to benefit from the WQL properties in DRL, by using neural networks trained with Dropout as an effective approximation of deep Gaussian processes. In particular, we adopt the Concrete Dropout variant to obtain calibrated estimates of epistemic uncertainty in DRL. The estimator, then, is obtained by taking several stochastic forward passes through the action-value network and computing the weights in a Monte Carlo fashion. Such weights are Bayesian estimates of the probability of each action value corresponding to the maximum w.r.t. a posterior probability distribution estimated by Dropout. We show how our novel *Deep Weighted Q-Learning* algorithm reduces the bias w.r.t. relevant baselines and provides empirical evidence of its advantages on representative benchmarks.

Keywords: Deep Reinforcement Learning, Q-learning, Overestimation bias, Maximum expected value

1 Introduction

Temporal difference (TD) and off-policy learning are the constitutional elements of modern Reinforcement Learning (RL). TD allows agents to bootstrap their current knowledge to learn from a new observation as soon as it is available. Off-policy learning gives the means for exploration and enables experience replay. Q-Learning [18] implements both paradigms. Overestimation of the maximum action value is a well-known problem that hinders Q-Learning performance, leading to suboptimal policies and unstable learning. This overoptimism can be particularly harmful in stochastic environments and when using function approximation [14], notably also in the Deep Reinforcement Learning (DRL) settings [17]. Among possible solutions, the Double Q-Learning [15] algorithm and its DRL variant – Double DQN – tackle the overestimation problem by disentangling the choice of the target action and its evaluation. The resulting estimator, while achieving superior performance in many problems, is negatively biased. In fact, overly pessimistic estimates might undervalue a good course of action and are thus problematic in their own way. In this regard, Weighted Q-Learning (WQL) [4] is among several methods to reduce the bias and shows remarkable empirical results. WQL is based upon the Weighted Estimator (WE), which weights each estimate of the action values, based on an estimated probability of each particular action value being the maximum. However, doing so in high-dimensional environments is not trivial due to the additional challenges imposed by using function approximation. Our objective is, then, to devise an approach to scale WQL to the DRL settings. We do so by using neural networks trained with Dropout as an effective, computationally cheap, Bayesian inference technique [5]. We combine, in a novel way, the dropout uncertainty estimates with the Weighted Q-Learning algorithm, extending it to the DRL settings. The proposed *Deep Weighted Q-Learning* algorithm, or *Weighted DQN* (WDQN), leverages an approximated posterior distribution on Q-networks to reduce the bias of deep Q-learning. WDQN bias is neither always positive, nor negative, but depends on the state and the problem at hand. WDQN only requires minor modifications to the baseline algorithm, and its computational overhead is negligible on specialized hardware. In Section 2, we define the problem settings and discuss some related works, then, in Section 3 we present our approach. We present empirical results in Section 4 and draw our conclusions in Section 5.

2 Preliminaries

A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a Markovian transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1]$ is a discount factor. A sequential decision maker ought to estimate, for each state s, the optimal value $Q^*(s, a)$ of each action a, i.e., the expected cumulative discounted reward obtained by taking action a in s and following the optimal policy π^* afterwards.

(Deep) Q-Learning A classical approach for solving finite MDPs is the Q-Learning algorithm, an off-policy value-based RL algorithm, based on TD. The popular Deep Q-Network algorithm (DQN) [10] is a variant of Q-Learning designed to stabilize off-policy learning with deep neural networks in high-dimensional state spaces. The two most relevant architectural changes to standard Q-Learning introduced by DQN are the adoption of a replay memory, to learn offline from experience, and the use of a target network, to reduce correlation between the current model estimate and the bootstrapped target value. In practice, a DQN agent interacts with the environment, stores in the replay buffer performed actions and corresponding observations, and learns the Q-values online, training a neural network. The network, with parameters $\boldsymbol{\theta}$, is trained by sampling mini-batches from the memory and using a target network whose parameters $\boldsymbol{\theta}^-$ are updated to match those of the online model every C steps. The model is trained to minimize the loss

$$L(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\langle s_i, a_i, r_i, s_i' \rangle \sim m} \left[\left(y_i^{DQN} - Q(s_i, a_i; \boldsymbol{\theta}) \right)^2 \right], \qquad y_i^{DQN} = r_i + \gamma \max_a Q(s_i', a; \boldsymbol{\theta}^-), \tag{1}$$

where m is a uniform distribution over the transitions stored in the replay buffer and y_i^{DQN} is the DQN target. Among the many studied improvements and extensions of the baseline DQN algorithm, Double DQN (DDQN) [17] reduces the overestimation bias of DQN with a simple modification of the minimized loss. In particular, DDQN uses the target network to decouple action selection and evaluation, and estimates the target value as

$$y_i^{DDQN} = r_i + \gamma Q(s_i', \operatorname*{argmax}_a Q\left(s_i', a; \boldsymbol{\theta}\right); \boldsymbol{\theta}^-) .$$

DDQN improves on DQN, converging to a more accurate approximation of the value function, while maintaining the same model complexity and adding minimal computational overhead.

Estimation biases in Q-Learning Choosing a target value for the Q-Learning update rule can be seen as an instance of the Maximum Expected Value (MEV) estimation problem for a set of random variables, here the action values. Q-Learning uses the Maximum Estimator (ME) to estimate the maximum expected return and exploits it for policy improvement. It is well known that ME is a positively biased estimator of MEV [16]. Double Q-Learning [15], on the other hand, learns two value functions in parallel and uses an update scheme based on the Double Estimator (DE). It is shown that DE is a negatively biased estimator of MEV, which helps to avoid catastrophic overestimates of the Q-values. In practice, as also

shown by Lan et al. [8], the overestimation bias of Q-Learning is not always harmful and may also be convenient when the action values are significantly different among each other. Conversely, the underestimation of Double Q-Learning is effective when all the action values are very similar. Unfortunately, prior knowledge about the environment is not always available, and it would be desirable to have an estimator which is not always positively or negatively polarized. Clearly, besides Double Q-Learning several other methods exist to mitigate the overestimation problem (e.g., [9, 1, 8]), here we mainly focus on Weighted Q-learning and on the two most used algorithms, namely standard Q-Learning and its DE variant.

Weighted Q-Learning D'Eramo et al. [4] propose the Weighted Q-Learning (WQL) algorithm, a variant of Q-Learning based on therein introduced Weighted Estimator (WE). WE estimates MEV as the weighted sum of the random variables sample means, weighted according to their probability of corresponding to the maximum. Intuitively, the amount of uncertainty, i.e., the entropy of the WE weights, will depend on the nature of the problem, the number of samples and the variance of the mean estimator (critical when using function approximation). WE bias is bounded by the biases of ME and DE [4]. The target value of WQL can be computed as

$$y_t^{WQL} = r_t + \gamma \sum_{a \in \mathcal{A}} w_a^{s_{t+1}} Q(s_{t+1}, a) \qquad w_a^s = P\left(a = \underset{a'}{\operatorname{argmax}}\, Q(s, a')\right). \tag{2}$$

where $w_a^{s_{t+1}}$ are the WE weights and correspond to the probability of each action value being the maximum. The weights of WQL are estimated in the tabular setting, assuming the sample means to be normally distributed.

3 Deep Weighted Q-Learning

Algorithm 1 Weighted DQN

Input: Q-network parameters $\boldsymbol{\theta}$, dropout rates $p_1, \dots, p_L$, a policy π, replay memory $\mathcal{D}$
$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
Initialize memory $\mathcal{D}$.
for step $t = 0, \dots$ **do**
  Select an action a_i according to some policy π given the distribution over action-value functions $Q(s, \cdot\,; \boldsymbol{\theta}, \boldsymbol{\omega})$
  Execute a_t and add $\langle s_t, a_t, r_t, s_{t+i} \rangle$ to $\mathcal{D}$
  Sample a mini-batch of transitions $\{\langle s_i, a_i, r_i, s_i' \rangle, i = 1, \dots, M\}$ from $\mathcal{D}$
  for $i = 1, \dots, M$ **do** ▷ can be done in parallel
    Take K samples from $Q(s_i, \cdot\,; \boldsymbol{\theta}^-, \boldsymbol{\omega})$ by performing K stochastic forward passes
    Use the samples to compute the WDQN weights (Eq. 6) and targets (Eq. 7)
  end for
  Perform an SGD step using y^{WDQN} as the target value.
  Eventually update $\boldsymbol{\theta}^-$
end for

A natural way to extend the WQL algorithm to the DRL settings is to consider the uncertainty over the model parameters by using a Bayesian approach. Dropout [13] is a regularization technique used to train large neural networks by randomly dropping units during learning. In recent years, dropout has been analyzed from a Bayesian perspective [5], and interpreted as a variational approximation of a posterior distribution over the parameters of the neural network. In particular, Gal and Ghahramani [5] show how a neural network trained with dropout and weight decay can be seen as an approximation of a deep Gaussian process. A single stochastic forward pass through a neural network trained with Dropout can, in fact, be interpreted as taking a sample from the model's predictive distribution. This inference technique, known as *Monte Carlo dropout*, can be efficiently parallelized on modern GPUs. This approach has found several applications in RL, e.g., as a practical approach to perform Thompson Sampling [5] and to estimate uncertainty in model-based RL [6]. Here we focus on the problem of action evaluation, and we show how to use approximate Bayesian inference to evaluate WE by introducing a novel approach to exploit uncertainty estimates in DRL. Our method is grounded in theory and simple to implement.

Weighted DQN Let $Q(\cdot\,, \cdot\,; \boldsymbol{\theta}, \boldsymbol{\omega})$ be a neural network with weights $\boldsymbol{\theta}$ trained with a Gaussian prior and Dropout Variational Inference to learn the optimal action-value function of a certain MDP. We indicate with $\boldsymbol{\omega}$ the set of random variables that represents the dropout masks, with $\boldsymbol{\omega}_i$ the i-th realization of the random variables and with $\boldsymbol{\Omega}$ their joint distribution:

$$\boldsymbol{\omega} = \{\omega_{lk} : l = 1, \dots, L, k = 1, \dots, K_l\}, \qquad \omega_{lk} \sim \text{Bernoulli}(p_l), \qquad \boldsymbol{\omega}_i \sim \boldsymbol{\Omega}(p_1, \dots, p_L), \tag{3}$$

where L is the number of weight layers of the network and K_l is the number of units in layer l. Consider a sample $q(s, a)$ of the MDP return, obtained taking action a in s and following the optimal policy afterwards. Following the Gaussian

process interpretation of Dropout of Gal and Ghahramani [5], we can approximate the likelihood of this *observation* as a Gaussian such that

$$q(s,a) \sim \mathcal{N}\left(Q(s,a;\boldsymbol{\theta},\boldsymbol{\omega});\tau^{-1}\right), \tag{4}$$

where τ is the model precision. We can approximate the predictive mean of the process, and the expectation over the posterior distribution of the Q-value estimates, as the average of T stochastic forward passes through the network:

$$\mathbb{E}\left[Q(s,a;\boldsymbol{\theta},\boldsymbol{\omega})\right] \approx \hat{Q}_T(s,a;\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} Q(s,a;\boldsymbol{\theta},\boldsymbol{\omega}_t). \tag{5}$$

$\hat{Q}_T(s,a;\boldsymbol{\theta})$ is the estimate of the action values associated with state s and action a. We can estimate the probability required to calculate WE similarly, by again exploiting a Monte Carlo estimator. Given an action a, the probability that a corresponds to the maximum expected action value can be approximated as the number of times in which, given T samples, the sampled action value of a is the maximum over the number of samples

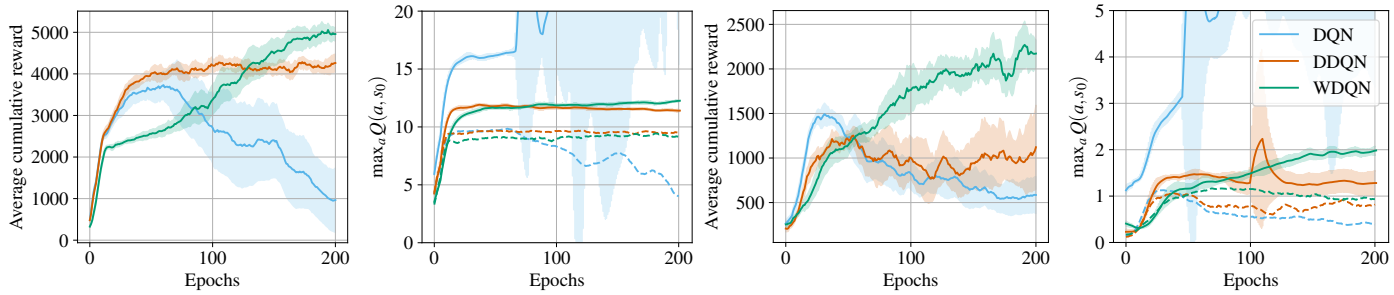
$$w_a^s(\boldsymbol{\theta}) = P\left(a = \operatorname*{argmax}_{a'} Q(s,a';\boldsymbol{\theta},\boldsymbol{\omega})\right) \approx \frac{1}{T}\sum_{t=1}^{T}\left[\!\left[a = \operatorname*{argmax}_{a'} Q(s,a';\boldsymbol{\theta},\boldsymbol{\omega}_t)\right]\!\right], \tag{6}$$

where $[\![\ldots]\!]$ are the Iverson brackets ($[\![P]\!]$ is 1 if P is true, 0 otherwise). The weights can be efficiently inferred in parallel with no impact in computational time. We can define the WE target given the target Q-network estimates by using the obtained weights as:

$$y_i^{WDQN} = r_i + \gamma \sum_{a\in\mathcal{A}} w_a^{s_i'}(\boldsymbol{\theta}^-)\hat{Q}_T(s_i',a;\boldsymbol{\theta}^-). \tag{7}$$

For completeness, the complete WDQN algorithm is reported in Algorithm 1. Dropout probabilities are variational parameters and influence the quality of the approximation. Ideally, they should be tuned to maximize the log-likelihood of the observations using a validation method. This is clearly not possible in RL where the available samples and the underlying distribution generating them, change as the policy improves. In fact, using dropout with a fixed probability might lead to poor uncertainty estimates [11, 12, 7]. Concrete Dropout [7] mitigates this problem by using a differentiable continuous relaxation of the Bernoulli distribution and learning the dropout rate from data. In practice, then, we substitute the standard Dropout with its Concrete variant in our WDQN implementation.

4 Experiments



(a) Asterix. (b) Wizard Of Wor.

Figure 1: Learning curves on Atari games. The curves on the left of each subfigure show the evaluation scores. The curves on the right, conversely, report the estimate of the expected return w.r.t. the starting screen of the game; the dashed curves indicate the real discounted return for each agent. Each epoch corresponds to 1M frames. The results shown here are the average of 10 independent runs and the shaded areas represent 95% confidence intervals. The curves are smoothed using a moving average of 10 epochs to improve readability.

In this section, we compare WDQN against the standard DQN algorithm and its Double DQN variant in the Arcade Learning Environment (ALE) [2]. In particular, we choose environments where DQN is known to overestimate action values and exhibit unstable learning [17]. We use the same neural network and hyperparameters of Mnih et al. [10], for WDQN we use Concrete Dropout only in the fully connected layer after the convolutional block. We found WDQN to be robust to the number of dropout samples used to compute the WE (e.g., $T \geq 30$), while being more sensitive to the Concrete Dropout regularization coefficient. For more in-depth results and discussion, we refer to our journal paper [3].

Figure 1 shows the result of the comparison in terms of the average reward and prediction accuracy of WDQN against DQN and DDQN. Here, WDQN stabilizes learning in both scenarios, but it also outperforms DDQN. Conversely, DDQN

manages to stabilize the learning performance in Asterix, but, in our setting, is not able to yield satisfactory performance in Wizard of Wor differently from what observed by Van Hasselt et al. [17], probably due to minor implementation differences. However, it is worth mentioning that DDQN would most likely improve in this scenario by reducing the frequency of updating the target network's parameters. WDQN, on the other hand, manages to control the bias and the learning instability in both cases, achieving a comparable performance and prediction accuracy w.r.t. DDQN in Asterix and outperforming the other two baselines on Wizard of Wor. Again, we refer to the journal paper for additional results.

5 Conclusion and future works

We present WDQN, a new value-based Deep Reinforcement Learning algorithm that extends the Weighted Q-Learning algorithm to work in environments with a high-dimensional state representation. WDQN is a principled and robust method to exploit uncertainty in DRL to accurately estimate the maximum action value in a given state. Our results corroborate the findings of previous works [5, 7], confirming that dropout can be used successfully for approximate Bayesian inference in DRL. Future works may explore the combination of WDQN with other orthogonal DQN extensions and may attempt to adapt the WDQN approach to other techniques modeling uncertainty in deep neural networks.

References

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[3] Carlo D'Eramo, Andrea Cini, Alessandro Nuara, Matteo Pirotta, Cesare Alippi, Jan Peters, and Marcello Restelli. Gaussian approximation for bias reduction in Q-learning. *Journal of Machine Learning Research*, 2021.

[4] Carlo D'Eramo, Marcello Restelli, and Alessandro Nuara. Estimating maximum expected value through Gaussian approximation. In *International Conference on Machine Learning*, pages 1032–1040, 2016.

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[6] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML 2016*, 2016.

[7] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete Dropout. In *Advances in Neural Information Processing Systems 30*, pages 3581–3590. Curran Associates, Inc., 2017.

[8] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin Q-learning: Controlling the estimation bias of Q-learning. In *International Conference on Learning Representations*, 2020.

[9] Donghun Lee, Boris Defourny, and Warren B Powell. Bias-corrected Q-learning to control max-operator bias in Q-learning. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 93–99. IEEE, 2013.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[11] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[12] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31*, pages 8617–8629. Curran Associates, Inc., 2018.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[14] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263. Lawrence Erlbaum, 1993.

[15] Hado Van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, 2010.

[16] Hado Van Hasselt. Estimating the maximum expected value: an analysis of (nested) cross-validation and the maximum sample average. *arXiv preprint arXiv:1302.7175*, 2013.

[17] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[18] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

A hierarchy of distributed task representations in the anterior cingulate cortex

Thomas R. Colin, Iris Ikink, Clay B. Holroyd
Ghent University
Department of Psychology
Ghent, Belgium
thomas.colin@ugent.be

Abstract

Current conceptualizations of the anterior cingulate cortex (ACC) alternatively emphasize its distributed or hierarchical characteristics, but it is not clear how these characteristics can be integrated into a unified account. We explore this issue by taking a modeling approach that builds off of prior work by incorporating aspects of both hierarchical and distributed representations into a single model. Using representational similarity analysis, we compare task representations in the brains of human subjects with those obtained from recurrent neural network models trained via supervised learning, when both complete the same task. We introduce hierarchy first by means of explicit goal units, and second by inducing a gradient of abstraction (from action representations to goal representations) in the hidden layer of the neural network. Both of these hierarchical models capture more rostral areas of ACC compared to a model devoid of hierarchical features, and the abstraction gradient of the second model appears to mirror an abstraction gradient in ACC. These results suggest that the ACC represents tasks in a distributed manner, along a rostro-caudal hierarchical gradient, such that caudal areas close to the pre-supplementary motor area represent actions, and rostral areas represent goals. These results are consistent with the ACC playing a key role in hierarchical decision-making by way of distributed yet hierarchically organized representations.

Keywords: Anterior cingulate cortex, hierarchy, fMRI, representational similarity analysis

Acknowledgements

This work is part of a project that has received funding from the European Research Council (ERC) under the EU's Horizon 2020 Research and Innovation Programme (grant agreement no. 787307).

1 Background: Anterior Cingulate Cortex and Hierarchy

The function of the anterior cingulate cortex (ACC) remains mysterious. It is involved in a wide range of different processes including conflict monitoring, attention, reward prediction and prediction errors, self-control, and strategic decisions; lesions of the ACC can trigger both apathy and impulsiveness, whereas stimulation can cause subjects to experience a "will to persevere" [5]. We have proposed that the ACC is involved in hierarchical decision making [9], leveraging principles of modularity [12] and compositionality [10]. This is supported by imaging evidence of a rostro-caudal gradient of abstraction in the dorso-lateral prefrontal cortex (DLPFC) [1], mirroring a parallel topographical hierarchical organization in the ACC of rats [7] and humans [15]. Yet hierarchical models often make use of discrete representations of goals or options, whereas ACC appears to encode representations of behavior across ensemble activity using a distributed code [13, 14].

To investigate task representations in ACC, in previous work [8] we asked participants to complete an adapted "activity of daily living" task while undergoing fMRI. Separately, we trained a recurrent neural network (RNN) model on the same task. Using representational similarity analysis (RSA), activation patterns in the hidden layer of the RNN were compared to activation patterns in the brain of the participants. Neural representations of participants resembled those of the model most closely within the ACC, consistent with the hypothesis that ACC encodes distributed, goal-directed representations of action-sequences. However, crucially, that RNN did not incorporate hierarchical features, even though hierarchy is central to current thinking about prefrontal cortex and ACC in particular. To address this open question, we revisited this study by extending the model with goal units and enforcing an "abstraction gradient" along the hidden layer of the model. We predicted that this would enable the model to simulate more rostral areas of the ACC.

2 Methods

2.1 The tea-coffee task

Participants were told that they were baristas preparing tea or coffee for customers. While lying in an MRI scanner, they were instructed to choose randomly which beverage to prepare on each trial, "as if flipping a coin". The preparation of a beverage always took six action steps. During each step they saw three images (e.g., during step 1: coffee beans, tea leaves, or pepper) from which they should select one according to specific rules (see Figure 1). For example, if they selected 'coffee' at step 1 and 'cream' at step 2, then they should select 'water' at step 4, and 'serve coffee' at step 6. After preparing the beverage correctly (resp. incorrectly) they would see as feedback a happy (resp. sad) smiley representing the customer. Since participants always saw the same three images during each action step, they needed to remember contextual information. For example: were they preparing coffee or tea, and which ingredient was added first?

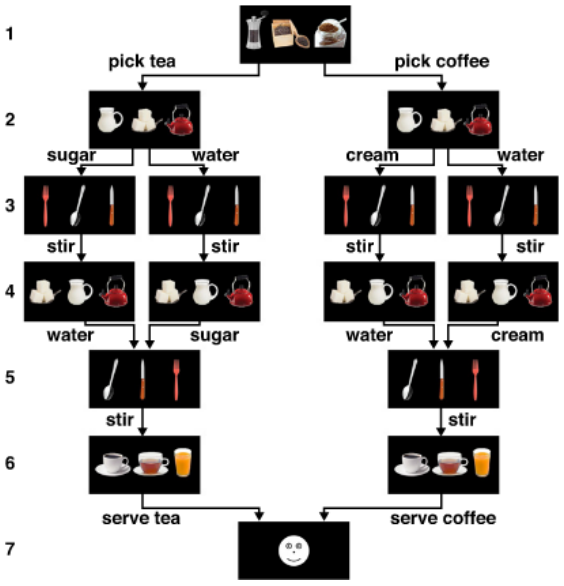


Figure 1: Correct sequences for the tea-coffee task.

The task included had 4 sequences of 6 action-steps each. The three images displayed during each action step were shown to participants for 1s, followed by a black screen for 3.5 +/- 1s (with 7 possible jitters at .33s increments). If no response was given before the three images of the next step appeared, the words "too slow" appeared and the trial was rendered incorrect. Feedback (happy/sad smiley) was also shown for 1s, followed by a 3s inter-trial interval. Full details of the task are given in [8].

2.2 Neural Network Modeling

We extended our original RNN model of sequential task execution in two steps that gradually increased the degree of hierarchy incorporated in it; the progressive impact of these changes on ACC representations was examined using representational dissimilarity matrices (RDMs) produced from these models. An RDM reflects how dissimilar activity patterns are for – in this case – each pair of the 24 unique steps of the coffee-tea task. For instance, the activity patterns across a specific group of artificial neurons during a 'stir' action may be more similar to the activity patterns for a different 'stir' action than to those for an 'add water' action. RDMs are built using one specific source of data (e.g., activation values of the units in the hidden layer in an RNN) but can be compared to another, qualitatively different source of data (e.g., fMRI voxels in the brain). Strong correlations between RDMs indicate similar representations.

The one-hot input[1] at each time-step corresponded to the previous action output, modeling short-term memory of actions. 100 instances of each model were trained by stochastic gradient descent using backpropagation through time, at a learning rate of 0.1, for 5000 iterations. An RDM was then constructed, per network, by extracting the vector of activations of the hidden layer at each action-step, and computing the 1 - Spearman rank-order correlation between each pair of these vectors. For each model, the final RDM was the mean of the RDMs from the 100 trained instances of that model.

Model 1: Elman Network. In Elman networks [6] (Figure 2) the hidden layer is copied to a "context" layer, whose units serve as fully connected inputs to the hidden layer on the next time-step. This model replicates [8] with a minor change in implementation (we use cross-entropy instead of mean squared error for the loss function). The 15 units of the hidden layer used a sigmoid activation function, whereas the output layer used softmax activation, corresponding to a standard cross-entropy classification loss ($L = -\sum_{t=1}^{6}\sum_{a=1}^{7} y_{t,a}\log(p_{t,a})$, where t is the action-step, a is one of 7 outputs ("add grounds", "stir", etc.), $y_{t,a}$ equals 1 if a is the target action at t and 0 otherwise, and $p_{t,a}$ is the network's actual output.

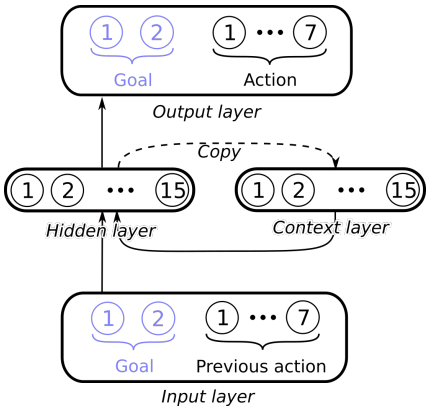


Figure 2: Elman network (black) and goal network additions (blue), inspired by [2] and [4] respectively. Layers linked by a plain arrow are fully connected.

Model 2: Goal Network. In a second model, we augmented the Elman network with two goal units, 'tea' and 'coffee'. This architecture is an extension of model 1 where goal units constitute both an input (after a winner-take-all operation on the previous output) and an output. The loss was the sum of the cross-entropy losses for goals and for actions. By training the network to output goals and enabling it to use these goals to select actions, we forced the hidden layer to represent goal information. Note that although the complete network model includes explicit goals, only the activations of the hidden layer are compared to brain activity.

Model 3: Abstraction Gradient Network. Unlike virtual artificial neurons, biological ones incur wiring costs dependent on the length of axons and the speed of communication along them. The brain must arbitrate a trade-off between these costs and cognitive performance [3]. We enforced this in model 3 by way of the loss function. We assumed that the action units corresponded to premotor cortex and that goal units matched abstract representations in other rostral areas; such that the hidden layer, corresponding to the ACC, spanned the distance between them. If the wiring costs are proportional to the length and weights of axons, the additional loss is $k(\sum_{i=1}^{n}\sum_{j=1}^{n}|i-j|w_{ij} + \sum_{i=1}^{n}\sum_{a=1} iw_{ia} + \sum_{i=1}^{n}\sum_{g=1}|n-i|w_{ig})$, where $w_{i,j}$ denotes the weight of the recurrent connection between the ith and jth units of the hidden layer (out of a total of n units), $w_{i,a}$ (resp. $w_{i,g}$) denotes the weight of a connection between the ith unit of the hidden layer and output action unit a (resp. goal g), and k is a constant (here set to .001). The first term corresponds to the loss incurred by recurrent connections, and the second and third term to loss incurred by connections to the action and goal outputs respectively. In all other respects, this model was trained like model 1 and model 2. In order to conduct the RSA, we generated two separate RDMs for the 'goal' hidden layer units ("left" side of the hidden layer) and the 'action' units ("right" side).

2.3 fMRI methods and representational similarity analysis (RSA)

This study re-analyzed existing fMRI data from [8]: 18 participants performed 6 practice trials, then 72 trials split over four runs. The General Linear Model included 24 regressors of interest; one for each unique action step in the task at the moment a response was given. They were convolved with the canonical hemodynamic response function, while the 6 motion parameters were entered as nuisance regressors. The fMRI time series were high-pass filtered (cut-off 128s) and a first-order autoregressive model was used to correct for temporal autocorrelation.

To investigate the representational similarity between activity patterns in the brain and activity patterns in the hidden layer of the three different RNNs, we conducted the RSA using a searchlight approach [11]. We expected to identify primarily the ACC. For the brain, we created a 27-element vector of BOLD signal activations for each 3×3×3 voxel cube. This was done for each action-step of the task, such that 24 vectors of activity patterns were extracted per person, per voxel cube. Then, for each voxel cube an RDM was built by computing the 1 - Spearman rank-order correlation between the activity patterns across all pairwise combinations of the 24 action steps (i.e., a 24×24 matrix). Thus, for the brain an RDM was created per voxel cube and per participant, whereas each RNN model only yielded one RDM (or two, in the case of the abstract gradient network model).

Secondly, the RNN- and brain-related RDMs were correlated with each other. For each participant, we computed the Spearman correlation between the upper triangle of the RDM of one of the RNNs versus those of the brain per voxel cube, such that a correlation value was returned for each voxel cube in the brain volume. Each correlation reflects how similar the activity patterns around this voxel are to that of the tested RNN. Correlation-values were z-transformed

[1] A one-hot vector has one element equal to 1 and all others to 0.

within participants, such that larger values indicate stronger correlations to the RNN compared to other voxels cube in the brain [8]. Furthermore, data were normalized into MNI space. Lastly, a simple t-test was run across participants per voxel cube, to test which voxel cubes showed a significant correlation with that of the tested RNN. Results were family-wise error (FWE) cluster-corrected, using a primary voxel-wise threshold of $p < .001$.

3 Results and discussion

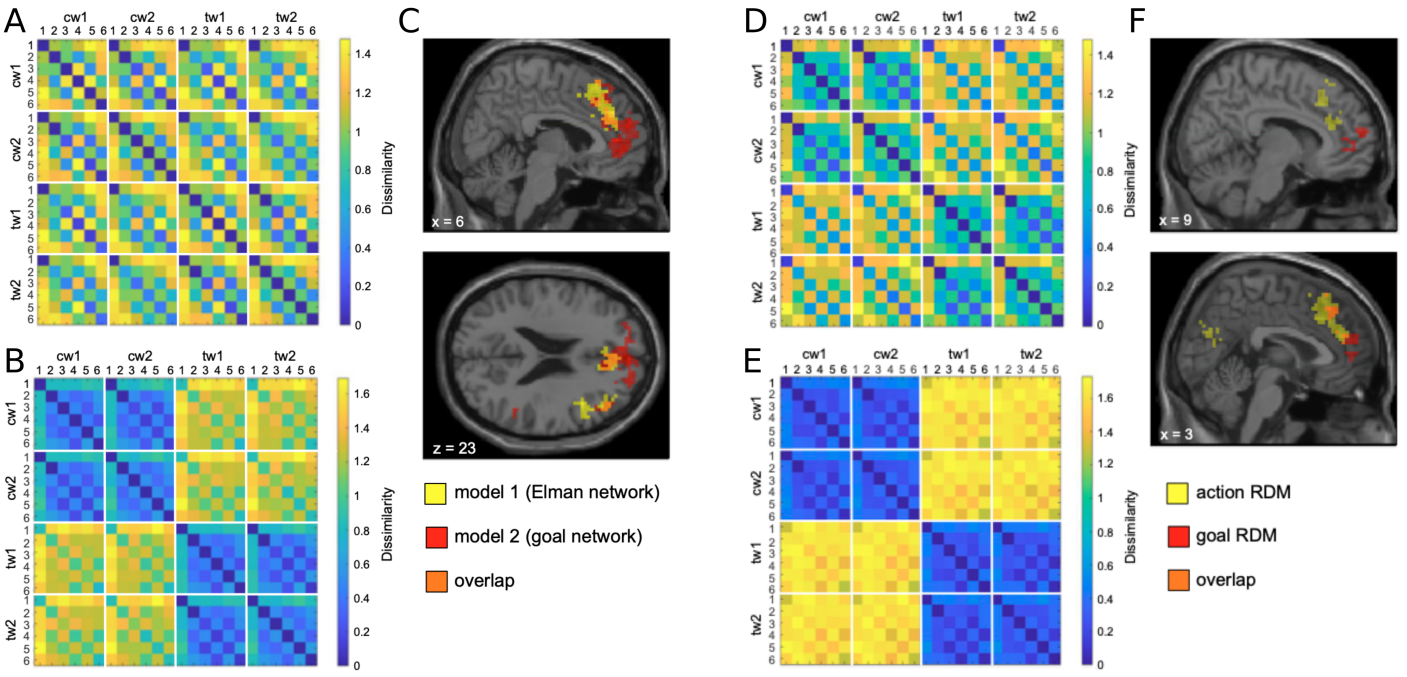


Figure 3: **Left:** RDMs of the context layer of (A) the Elman network, replicating [8], and (B) the goal network. The RDM of the goal network shows stronger similarity across coffee steps and across tea steps, separating the RDM in 4 "blocks". (C) Results of the searchlight analysis for both models. The inclusion of goal units yields a larger cluster that includes more anterior regions such as rostral ACC, consistent with a rostral-caudal abstraction gradient.
Right: RDMs of the context layer of the abstraction gradient model, using (D) the 'action' units and (E) the 'goal' units from the hidden layer. The RDM based on the goal units visibly separates between coffee and tea more than the RDM based on action units. (F) Results of the searchlight analysis for the abstraction gradient model. The rostral-caudal abstraction gradient is visible: actions are mostly represented in the caudal part of the ACC, while goals are mostly represented in the rostral part of the ACC.
All coordinates are given in MNI space, and clusters are displayed using an uncorrected p-value of .001. fMRI visualization done using xjView toolbox (https://www.alivelearn.net/xjview).

Unsurprisingly, our analysis using model 1 (the Elman network) replicated the results of [8]. Several interpretable patterns can be seen in the neural network RDM (Figure 3A), for example that the "stir" actions (actions 3 and 5 of all sequences) are represented similarly within and across sequences; an extensive analysis can be found in [8]. When comparing the neural network RDM to the fMRI data, the largest identified cluster corresponded to an ACC cluster with MNI coordinates and a peak t-value that were identical to those found in [8] (see Figure 4a in [8]).

Model 2, the goal network, was based on model 1, but was supplemented by goal units corresponding to "tea" and "coffee". As can be seen in Figure 3B, this change caused the network to separate representations of tea sequences from those of coffee sequences, while nevertheless preserving much of the patterns that were visible in Figure 3A. Comparing the neural network RDM to the fMRI RDMs in the searchlight analysis, the largest cluster was again clearly the ACC cluster, and comprised more voxels compared to model 1 (798 instead of 351). More specifically, as Figure 3C indicates, the cluster identified by the goal network showed partial overlap with that of [8], but extended to ventral and rostral ACC, consistent with a rostral-caudal abstraction gradient.

Model 3, the abstraction gradient network, simulated wiring costs to produce a gradient of abstraction in the hidden layer. Thus, in a single neural network, units were increasingly prone to represent goal-level information and to communicate with other goal-focused units as they were closer to one side of the layer. We produced an action RDM (Figure 3D) and a goal RDM (Figure 3E). As predicted, the largest identified cluster was found in the ACC for both the action and goal units (Figure 3F). Further, the action units identified the caudal regions more strongly while the goal units identified the rostral regions more strongly (including rostral ACC), indicating presence of a rostral-caudal abstraction gradient that mirrors the hierarchical representations encoded within the RNN model.

Table 1 shows an overview of the three largest neural clusters that were identified by each of the three RNNs. We identify some perceptual and motor areas presumably due to superficial task features, as well as insula, which often co-activates with ACC; but the largest clusters for all models were located in the ACC, and matched our topographical predictions. Specifically, the goal network demonstrated a larger cluster when the network specialized per goal, including additional voxels located more rostrally, suggesting that the ACC represents sequential tasks in a hierarchical manner. Crucial support for this idea comes from the abstraction gradient network, which showed a rostral-caudal gradient depending on the level of hierarchy of the task (i.e., low-level actions being represented more strongly in the caudal ACC; high-level goals represented more strongly in the rostral ACC).

RNN model	Brain region	x	y	z	Peak t-value	Cluster size
1) Elman network	ACC	6	41	32	7.18	351
	IFG	42	14	23	5.96	151
	anterior insula	48	20	2	6.39	109
2) Goal network	ACC/dlPFC/SMA	6	44	23	8.21	798
	IFG/insula	42	26	-1	6.19	146
	dlPFC/MFG	42	38	26	5.56	90
3) Abstraction gradient						
- goal units	(r)ACC/aPFC	18	50	17	6.51	224
	SMG/FEF	3	29	47	5.93	47
	insula/IFG	36	26	-7	5.21	28
- action units	(c)ACC/pre-SMA	3	32	41	6.23	271
	V1/Par-Occ sulcus	0	-76	29	6.45	136
	insula/IFG	42	23	-4	5.46	87

Table 1: Overview of the three largest clusters per model yielded by the searchlight analysis, using MNI coordinates. All clusters survived family-wise error (FWE) cluster-correction ($p_{\mathrm{FWE}} < .05$) using a primary voxel-wise threshold of $p < .001$. ACC = anterior cingulate cortex; IFG = inferior frontal gyrus; dlPFC = dorsolateral prefrontal cortex; MFG = middle frontal gyrus; SMA = supplementary motor area; aPFC = anterior prefrontal cortex; FEF = frontal eye fields; V1 = primary visual cortex.

This study replicated and extended the work of [8], finding evidence that the ACC not only tracks the execution of extended action sequences via distributed patterns of activity that evolve over time, but that these patterns appear to have a hierarchical component as well. By applying two different artificial RNNs with hierarchical structure, we found that their hierarchical representations of the coffee-tea task were most similar to how the ACC represented the same task. Crucially, by explicitly adding goal units (model 2), a larger ACC cluster was identified, that extended more rostrally compared to the original Elman network. The use of an abstraction gradient and separating goal versus action units (model 3) resulted in two ACC clusters with action units mostly identifying caudal ACC and goal units mostly identifying rostral ACC. These findings suggest that the ACC encodes extended action sequences in a hierarchical fashion along a rostral-caudal gradient (as suggested by e.g. [1], [8] and [7], and in line with findings from [15]).

The present work focused on finding evidence of distributed yet hierarchical representations, rather than identifying their function. In ongoing computational work, we investigate the properties of the models using a more difficult task, combined with a range of interventions on the model. Preliminary results suggest that our implementation of goal units offers no benefit for discovering temporally extended behavior patterns; but could constitute a hub for cognitive control, e.g. representing structured behaviors that deviate from habit, or increasing the robustness of these representations under stress.

References

[1] D. Badre and D. E. Nee. Frontal cortex and the hierarchical control of behavior. *Trends in cognitive sciences*, 22(2):170–188, 2018.

[2] M. Botvinick and D. C. Plaut. Doing without schema hierarchies: a recurrent connectionist approach to normal and impaired routine sequential action. *Psychological review*, 111(2):395, 2004.

[3] E. Bullmore and O. Sporns. The economy of brain network organization. *Nature Reviews Neuroscience*, 13(5):336–349, 2012.

[4] R. P. Cooper, N. Ruh, and D. Mareschal. The goal circuit model: A hierarchical multi-route model of the acquisition and control of routine sequential action in humans. *Cognitive Science*, 38(2):244–274, 2014.

[5] R. B. Ebitz and B. Y. Hayden. Dorsal anterior cingulate: a Rorschach test for cognitive neuroscience. *Nature neuroscience*, 19(10):1278–1279, 2016.

[6] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[7] C. B. Holroyd and S. M. McClure. Hierarchical control over effortful behavior by rodent medial frontal cortex: A computational model. *Psychological review*, 122(1):54, 2015.

[8] C. B. Holroyd, J. J. F. Ribas-Fernandes, D. Shahnazian, M. Silvetti, and T. Verguts. Human midcingulate cortex encodes distributed representations of task progress. *Proceedings of the National Academy of Sciences*, 115(25):6398–6403, 2018.

[9] C. B. Holroyd and T. Verguts. The best laid plans: Computational principles of anterior cingulate cortex. *Trends in Cognitive Sciences*, 25(4):316–329, 2021.

[10] R. A. Jacobs, M. I. Jordan, and A. G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive science*, 15(2):219–250, 1991.

[11] N. Kriegeskorte, M. Mur, and P. A. Bandettini. Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 2008.

[12] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[13] L. Ma, J. M. Hyman, A. J. Lindsay, A. G. Phillips, and J. K. Seamans. Differences in the emergent coding properties of cortical and striatal ensembles. *Nature neuroscience*, 17(8):1100–1106, 2014.

[14] D. Shahnazian and C. B. Holroyd. Distributed representations of action sequences in anterior cingulate cortex: A recurrent neural network approach. *Psychonomic Bulletin & Review*, 25(1):302–321, 2018.

[15] V. Venkatraman, A. G. Rosati, A. A. Taren, and S. A. Huettel. Resolving response, decision, and strategic control: evidence for a functional topography in dorsomedial prefrontal cortex. *Journal of Neuroscience*, 29(42):13158–13164, 2009.

Learning how to Interact with a Complex Interface using Hierarchical Reinforcement Learning

Gheorghe Comanici
DeepMind
Montreal, Canada
gcomanici@deepmind.com

Amelia Glaese
DeepMind
Montreal, Canada
glamia@deepmind.com

Anita Gergely
DeepMind
Montreal, Canada
agergely@deepmind.com

Daniel Toyama
DeepMind
Montreal, Canada
kenjitoyama@deepmind.com

Zafarali Ahmed
DeepMind
Montreal, Canada
zaf@deepmind.com

Tyler Jackson
DeepMind
Montreal, Canada
tbjackson@deepmind.com

Philippe Hamel
DeepMind
Montreal, Canada
hamelphi@deepmind.com

Doina Precup
DeepMind
Montreal, Canada
dprecup@deepmind.com

Abstract

Hierarchical Reinforcement Learning (HRL) allows interactive agents to decompose complex problems into a hierarchy of sub-tasks. Higher-level tasks can invoke the solutions of lower-level tasks as if they were primitive actions. In this work, we study the utility of hierarchical decompositions for learning an appropriate way to interact with a complex interface. Specifically, we train HRL agents that can interface with applications in a simulated Android device. We introduce a Hierarchical Distributed Deep Reinforcement Learning architecture that learns (1) subtasks corresponding to simple finger gestures, and (2) how to combine these gestures to solve several Android tasks. Our approach relies on goal conditioning and can be used more generally to convert any base RL agent into an HRL agent. We use the AndroidEnv environment to evaluate our approach. For the experiments, the HRL agent uses a distributed version of the popular DQN algorithm to train different components of the hierarchy. While the native action space is completely intractable for simple DQN agents, our architecture can be used to establish an effective way to interact with different tasks, significantly improving the performance of the same DQN agent over different levels of abstraction.

Keywords: Android, Hierarchical Reinforcement Learning, Generalized Value Functions

1 Introduction

As we scale up Reinforcement Learning (RL) agents to tackle large varieties of problems in domains that are commonly controlled by humans, these agents need to consider how to acquire and reuse diverse knowledge about the world [14, 6, 2, 13]. AndroidEnv is an open-sourced domain that poses such a challenge: general purpose agents need to control a universal touchscreen interface and tackle a wide variety of tasks in Android applications; these are developed for human users, hence they leverage human abilities to reuse knowledge and and build intuitions through constant interaction with the platform [19]. Controlling AndroidEnv is purposely designed to match real devices: agents observe screen pixels and control finger positioning in real-time; the environment runs in its own timeline and does not wait for the agent to deliberate over its choices; actions are executed asynchronously; agents can interact with any Android application.

One of the main driving principles for Hierarchical Reinforcement Learning (HRL) is the explicit decomposition of RL problems into a hierarchy of subtasks such that higher-level parent-tasks invoke low-level child tasks as if they were primitive actions. The space of all possible decompositions is complex and hard to work with, albeit extensive research shows that proper inductive biases can facilitate the search for useful decompositions (e.g. diffusion models [9], bottleneck states [10, 17], intrinsic goals [8], language [5], empowerment [15]). We introduce an HRL agent that acquires simple finger gesture skills and successfully reuses this knowledge in several diverse AndroidEnv tasks. To demonstrate the generality of the approach, we use the framework of General Value Functions (GVFs) [18] to capture domain knowledge about gestures for AndroidEnv. GVFs have been proposed in prior work as a way to capture diverse knowledge about the world in the form of long-term predictions associated with agent experience. GVFs can be learned incrementally using off-policy methods, and can be used to capture knowledge at different time-scales and levels of abstraction [20, 12].

Our main contribution is a novel Hierarchical Distributed Deep Reinforcement Learning architecture for AndroidEnv. The architecture first builds a goal-conditioned deep model [16] for GVFs that capture knowledge about simple finger gestures then it learns how to combine corresponding skills to solve several tasks from Android applications. Instead of using general RL agents to solve a complex problem directly, the architecture first decomposes it into a three-level hierarchy of sub-tasks: the lowest level (level 0) interacts with the screen to complete gestures (taps, swipes and flings), the next level provides the target gesture (e,g. where to tap, direction of a swipe), the final level decides which gesture amongst the three to execute to maximize per-step rewards. The same general RL agent is then used to solve decision making processes corresponding to each of the levels in the hierarchy. We demonstrate that even though the native action space is intractable for the baseline distributed DQN agent [11], the same agent becomes much more efficient when used to solve sub-tasks and to make abstract choices at higher levels in the hierarchy.

2 The architecture

General Value Functions (GVFs). Sutton et al. [18] introduced a unified way to express long-term predictions for signals that are independent of task-specific rewards, under policies that are different from the agent's behavior, and under flexible state-dependent discounting schemes. GVFs are associated with tuples $\langle \gamma, C, \pi \rangle$, where $\gamma : \mathcal{S} \to [0, 1]$ is known as a *continuation function*, defined over all states $\mathcal{S}$ of an MDP, $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the cumulant function over MDP transitions, and $\pi : \mathcal{S} \to \mathcal{D}(\mathcal{A})$ is a policy that generates an action distribution for each MDP state. The corresponding prediction is denoted by $v_{\pi,\gamma,C}$ and it is the expected cumulant-based return: $v_{\pi,\gamma,C}(s) = \mathbb{E}[\sum_{t=0}^{\infty}(\prod_{i=1}^{t}\gamma(S_i))C_i \mid S_0 = s, A_{0:\infty} \sim \pi]$. We use $q_{\pi,\gamma,C}(s, a)$ for predictions that are conditioned both on the initial state $S_0 = s$ and action $A_0 = a$. Discounted expected returns area appealing because they all obey some form of a *Bellman equation* which greatly facilitates estimation and are used to derive tractable objective functions for optimization based algorithms, such as gradient descent over deep networks. For simplicity, we describe below the Bellman equation for the optimal cumulant-based q-value, $q^*_{\gamma,C}(s, a) = \sum_{s' \in S} p(s'|s, a) \left[C(s, a, s') + \gamma(s) \max_{a'} q^*_{\gamma,C}(s', a')\right]$.

Hierarchy of GVFs. We present a general approach to implement hierarchical decompositions of complex problems into a multi-layered hierarchy of sub-tasks, where each level is trained to maximize GVFs: given a fixed cumulant-continuation pair (C, γ), agents maintain estimates for the value of the corresponding optimal policy, i.e. $q^*_{\gamma,C}(s, a) = \max_\pi q_{\pi,\gamma,C}(s, a)$. Instead of solving the problem with a single RL agent operating on the "raw" action space of an environment, we prioritize modularity and comprehension to build a hierarchy of "problems" that are solved by independent agents, working at different levels of space and temporal abstraction. A hierarchical decomposition on levels 0 to N works under the assumption that each level i operates over a set of control GVFs, $\Omega_i := \{(C_i, \gamma_i)\}_{i=1}^{M}$ and, at each timestep, the corresponding RL agent follows the policy maximizing one of these GVFs. The selection of the active GVF at every timestep comes as a signal $\omega = (C, \gamma) \in \Omega_i$ from the level $i + 1$. For all levels, except for the lowest level 0, the corresponding agent selects an *abstract* action a_i by maximizing $q^*_{\gamma,C}(s, a_i)$, and propagates it down as a GVF selection for level $i - 1$. In other words, the level is always maximizing one of the many signals that it is designed to predict. Lastly, temporal abstraction can be achieved within this framework by using the continuation function γ of the selected GVF to determine the temporal extent of its execution. See Figure 1 for the concrete three-level hierarchy we used in our work.

The main advantage of the hierarchical decomposition is that RL agents operating at different levels can be designed in isolation and perhaps can be trained either at different stages or using completely different techniques. For example, one could select among a finite set of abstract actions in level 1, while a continuous control agent interacts with an environment that operates with a continuous (or relatively large) action space.

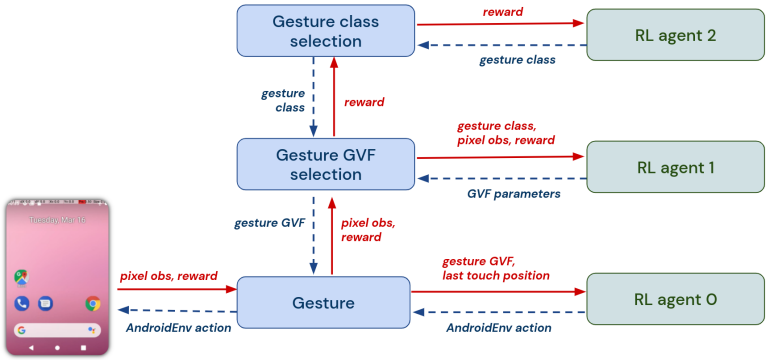


Figure 1: **Gesture Hierarchy**. The architecture used for the Android applications is based on a 3-layer hierarchy: (1) The lowest level operates over GVFs corresponding to all supported gestures; (2) The middle layer selects a gesture GVF given the latest pixel image in AndroidEnv and its agent is trained to maximize the return associated with the task that the agent is trained on; and (3) The top layer selects a single gesture class for the task and the agent is trained to maximize the average per step reward. All levels are operated by distributed DQN agents.

Distributed Hierarchies. Distributed computing architectures for Deep Reinforcement Learning have been shown to play an important role in scaling up these algorithms to relatively challenging domains [4, 7]. In particular, these allow for asynchronous learning, and, when working with simulated environments, asynchronous acting. The modular hierarchical decomposition that we describe in this section is well suited for distributed architectures, as different levels operate with RL agents that are potentially independent of each other. Albeit these levels are tied during the execution of a policy due to the hierarchical signal processing procedure, learning is not: each level can maintain its own training dataset and perform learning updates on separate machines. Since AndroidEnv runs in real-time and the underlying simulation cannot be sped up, multiple actors run in parallel to generate sufficient experience for all learners.

3 Experimental implementation

We present results on a selection of AndroidEnv tasks. For our experiments, we used the Acme framework [3] and its Distributed TensorFlow implementation of the DQN agent [11], configured for runs on Atari games, available at Acme's Github Repository.[1] To be able to readily use agents designed for Atari games, we simplified the AndroidEnv interface by (1) down-sampling the input images to a 120 x 80 resolution, and (2) restricting taps and swipes to 54 locations on the screen, corresponding to a 9 by 6 discretization of the Android touch-screen. Moreover, the agent's input has further knowledge of any completed tap, swipe, or fling operation, as well as the most recent finger touch location. For more details on implementation, network architecture, and default hyper parameter settings, please refer to the Acme open-source code. Details on the set of AndroidEnv tasks for which we report results are available on AndroidEnv's Github Repository.[2] Figures 2 and 3 provide a summary of the observed empirical results. The rest of this section provides a detailed description of the hierarchy used to obtain these results.

Level 0: gesture execution. The lowest level in the hierarchy is designed to execute gestures by operating on a set of GVFs composed of tap, swipe, and fling gestures. To fully define these GVFs, level 0 maintains a sequence of all touch positions in a trajectory, denoted by $(\mathbf{p}_0, \mathbf{p}_1 \cdots, \mathbf{p}_t)$, with all $\mathbf{p}_i$ either positions on the screen for tap actions or $\mathbf{p}_i = \mathbf{0}$ for lift actions. For example, to capture a swipe gesture from location $\mathbf{q}_1$ to $\mathbf{q}_2$ we use a cumulant $C_{\mathbf{q}_1,\mathbf{q}_2}(\mathbf{p}_0, \mathbf{p}_1 \cdots, \mathbf{p}_t) = 1$ if $\exists i < t$ such that $[\mathbf{p}_i, \mathbf{p}_{i+1}, \cdots, \mathbf{p}_{t-1}, \mathbf{p}_t] = [\mathbf{0}, \mathbf{q}_1, \mathbf{p}_{i+2}, \cdots, \mathbf{p}_{t-2}, \mathbf{q}_2, \mathbf{0}]$ with $\mathbf{p}_j \neq \mathbf{0}, \forall i < j < t$, $C_{\mathbf{q}_1,\mathbf{q}_2} = 0$ otherwise, and $\gamma_{\mathbf{q}_1,\mathbf{q}_2} = 1 - C_{\mathbf{q}_1,\mathbf{q}_2}$. In all experiments, we use tap locations and swipe start/end locations based on the 9 by 6 discretization described above, resulting in $54x54$ swipe GVFs and 54 tap GVFs. We additionally define 8 fling GVFs corresponding to N, NE, E, SE, S, SW, W and NW cardinal directions.

	Random	DQN	Hierarchy	Human
Apple Flinger	150	0.0±0.0	1899±276	3000
Blockinger	0.0	0.0±0.0	44414±2369	-
Catch	-0.5	-0.72± 0.13	0.96±0.15	1
Classic 2048	1000	1126± 105	1615±161	6000
Dodge	0.0	0.0±0.0	0.3±0.09	1
Floodit Easy	2	0.0±0.0	8.40±0.28	7
Nostalgic Racer	310	196±40	372±41	1000
Vector Pinball	9000	7478±1471	33240±8028	13000

Figure 3: Summary of results at the end of training, compared to human performance and return under a random policy.

[1] https://github.com/deepmind/acme/tree/master/acme/agents/tf/dqn
[2] https://github.com/deepmind/android_env

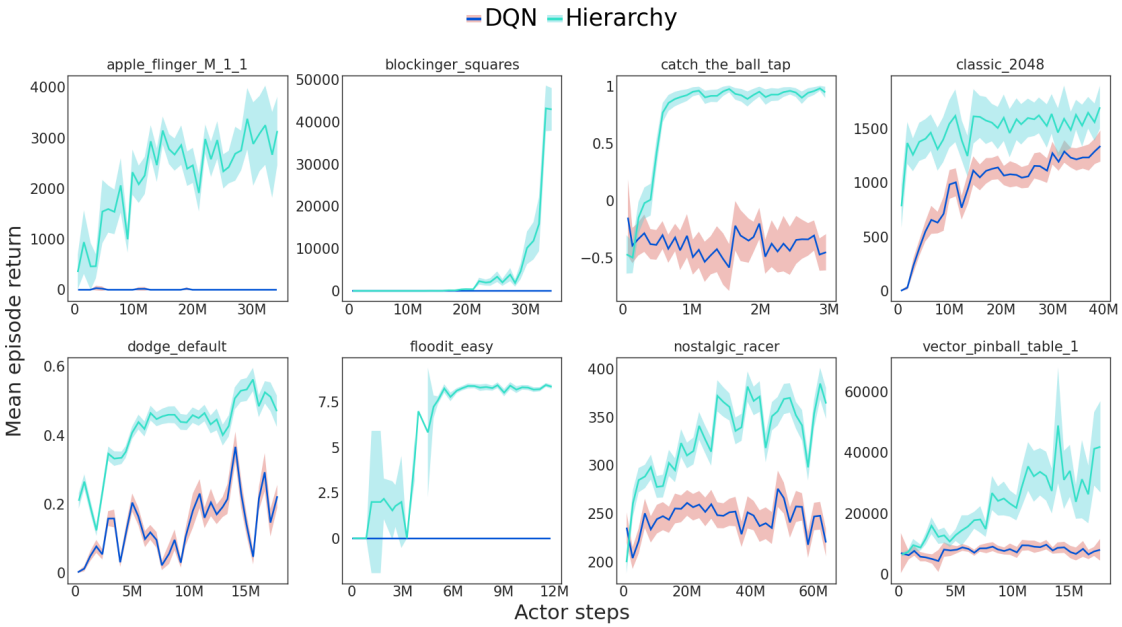


Figure 2: **Empirical results**. We tested our agents on a number of AndroidEnv tasks of different levels and with varying complexity in the action interface. We report results on tasks where at least one of the agents was able to improve its behavior. For tasks such as `classic_2048` and `nostalgic_racer`, using any fling or tap gesture, correspondingly, incurs significant changes in the score outcome. On the other hand, for tasks such as `apple_flinger_M_1_1`, `blockinger_squares`, and `floodit_easy`, the agent can only operate by direct interaction with specific buttons or objects and rewards are very sparse, making all of these tasks intractable for most agents.

As shown in Figure 1, the signal from above fully define individual gestures: $\omega_0 \in \Omega_0$ contains both a gesture class and a gesture parameter, e.g. $\omega_0 = (\text{swipe}, \mathbf{q}_1, \mathbf{q}_2)$ for a swipe from $\mathbf{q}_1$ to $\mathbf{q}_2$. To train the corresponding agent, we concatenate one-hot encodings for the gesture class, gesture parameters, and the last tap location. Each class of gestures was trained separately, hence the execution at this level is based on 3 separate networks. Lastly, we also apply Hindsight Experience Replay (HER) [1] for improved data-efficiency: we always select a single GVF during acting, but we compute cumulants and continuations for all GVFs as to relabel the training data and use it to train predictions corresponding to all GVFs for which a cumulant $C = 1$ is observed. All GVFs were trained with random agents at levels above (explained below) and, in all, we used approximately 10^7 actor steps to train this level, a cost that was paid only *once*, as the same model was reused by all agents training the higher levels in specific Android applications.

Level 1: gesture GVF selection. The second level in the hierarchy uses pixel input data coming from interaction with Android apps to select among all gesture GVFs, which in turn is executed by the lowest level. The level uses the pixel input and reward and the gesture class selection from the upper level to train the corresponding RL agent. The latter combines these signals to generate a parameter, e.g. tap location, for the GVF that should be executed at the lowest level. The GVF selection policy is trained using a DQN agent training a joint network for all gesture GVFs. Since the set of swipe GVFs is quite large, i.e. 54 x 54, the Q-value network is designed to output two sets of value estimates: one for the selection of the first parameter out of 54, and another one for the selection of the second parameter.

Level 2: gesture class selection. The third level is trained to select among gesture classes {tap, swipe, fling}. The corresponding agent is trained to maximize the average per step reward over the entire episode. This level receives only the environment reward as input and returns one of the three gesture classes. We use the same agent as for the other two layers for training. Since the problem is substantially simpler at this level of abstraction, we used a tabular Q-value representation for the average reward estimations associated with each gesture class.

4 Discussion

The results we presented provide strong evidence that task-independent knowledge about the Android action interface, e.g. *finger gestures*, can be used to derive useful hierarchical decompositions. We introduced a flexible and modular signal processing distributed architecture that effectively generates streams of training data for separate reinforcement learning agents, operating at different levels of abstractions, e.g. selecting a class of GVFs, selecting specific GVFs, executing

GVFs. The architecture was used to convert a simple DQN agent into a hierarchy of similar DQN agents, all operating on Android applications, but there is no restriction to this particular choice of agent or environment. Moreover, the hierarchical architecture is not restricted to learning knowledge that is related to finger gestures. In fact, we anticipate even stronger results when the agent is learning abstractions that correspond to more conceptual knowledge on the AndroidEnv platform, e.g. predicting and controlling object movement, menu navigation, affordable interactions with other apps or internet services, discovering common functionalities. Lastly, we believe that the most promising avenue is to allow agents to discover their own collection of GVFs as well as the most appropriate level of abstraction of the knowledge they can capture.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.

[2] Paul Cisek and John F Kalaska. Neural mechanisms for interacting with a world full of action choices. *Annual review of neuroscience*, 33:269–298, 2010.

[3] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[4] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018.

[5] Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *NeurIPS*, 2019.

[6] Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

[7] Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *ICLR*, 2019.

[8] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016.

[9] Marlos C. Machado, Marc G. Bellemare, and Michael H. Bowling. A laplacian framework for option discovery in reinforcement learning. In *ICML*, 2017.

[10] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *ECML*, volume 2430 of *Lecture Notes in Computer Science*, 2002.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[12] Joseph Modayil, Adam White, and Richard S. Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behaviour*, 22(2):146–160, 2014.

[13] Giovanni Pezzulo and Paul Cisek. Navigating the affordance landscape: feedback control as a process model of behavior and cognition. *Trends in cognitive sciences*, 20(6):414–424, 2016.

[14] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

[15] Christoph Salge, Cornelius Glackin, and Daniel Polani. Empowerment–an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer, 2014.

[16] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML*, 2015.

[17] Özgür Simsek and Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML*, 2004.

[18] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*. IFAAMAS, 2011.

[19] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android, 2021.

[20] Adam White. *Developing a predictive approach to knowledge*. PhD thesis, University of Alberta, 2015.

Fair E^3: Efficient Welfare-Centric Fair Reinforcement Learning

Cyrus Cousins
Brown University
Department of Computer Science
Providence, RI
cyrus_cousins@brown.edu

Kavosh Asadi
Amazon

Michael L. Littman
Brown University
Department of Computer Science
Providence, RI

Abstract

As the negative societal consequences of machine learning systems run amok have become increasingly apparent, fair machine learning methods have seen increased attention for tasks like facial recognition, medical care and diagnosis, and employment hiring decisions. Despite this positive trend, most attention on the theory side has been focused on fair supervised and unsupervised settings, whereas second-order impact of machine learning applications, such as the runaway positive feedback loops in settings like predictive policing, are more naturally posed in the setting of *reinforcement learning*. We propose a novel, welfare-centric, fair reinforcement-learning setting, in which the agent enjoys *vector-valued* reward from a set of beneficiaries. Given a *welfare function* $\mathcal{W}(\dots)$, the task is to select a policy π that is favorable to all beneficiaries, in the sense that it optimizes the welfare of the value of the beneficiaries from state s_0, i.e., $\operatorname{argmax}_\pi \mathcal{W}\big(V_1^\pi(s_0), V_2^\pi(s_0), \dots, V_g^\pi(s_0)\big)$. We show that, in this setting, both per-beneficiary exploration and per-beneficiary policy optimization are insufficient to identify the welfare-optimal policy. Whether an individual action is a mistake depends on the context of *subsequent actions*, therefore the standard PAC-MDP framework does not readily generalize to fair reinforcement learning. Consequently, we develop a stronger learning model, wherein at each timestep an agent either takes an *exploration action* or outputs an *exploitation policy*. We require that each exploitation policy be ε-welfare optimal, and the number of exploration steps be polynomial in all relevant parameters. We reduce PAC-MDP learning to this framework, showing that our framework is sufficiently challenging so as to be interesting, and define the fair E^3 learner to operate under this model, thus demonstrating that fair reinforcement learning is tractable.

Keywords: Fairness, Welfare, Vector-Valued MDPs, Learning, Planning

1 Introduction

Fair machine learning (ML) methods have recently seen increasing attention for tasks like facial recognition [Lohr, 2018] and employment hiring decisions [Kleinberg et al., 2018]. Despite this positive trend, most attention on the theory side has been focused on fair supervised [Agarwal et al., 2018] and unsupervised [Chierichetti et al., 2018] ML, whereas second-order impact of ML models, such as the runaway feedback loops in settings like predictive policing [Ensign et al., 2018]are more naturally posed in reinforcement learning (RL) settings. We apply ideas from welfare-centric supervised learning [Cousins, 2021, 2022] to reinforcement learning (RL) settings; in particular, we assume an agent receives a vector-valued reward signal from a set of *beneficiaries*, each representing, e.g., different racial, gender, religious groups, and the task is to learn a single policy that treats beneficiaries fairly.

We argue it is not the role of the algorithm designer to dictate *what fairness means* in the sense of *how to compromise between beneficiaries*, but rather to optimize for a *given fairness notion* (ideally one agreed upon by society, government, political philosophers, and other interested parties), as encapsulated by a metric of *societal welfare*. In supervised learning, doing so is relatively straightforward, as we generally maximize the welfare of *expected per-beneficiary utility*, so in RL, we take utility to be the standard *geometrically discounted value function* w.r.t. each beneficiary's reward. In general, optimizing welfare is referred to as the *social planner's problem*, so in a sense our work addresses this problem in the context of RL.

While optimizing the welfare of beneficiary value functions is a well specified goal for a *planning algorithm*, or when studying the asymptotic behavior of a learning algorithm, when considering the process of fair learning in an *unknown MDP*, it is imperative that we also ask the right questions as to *how quickly* we can learn and if we can guarantee that our learning algorithm behaves fairly in all but a finite number of steps. To this end, we generalize the PAC-MDP framework to a novel *adversarial fair MDP learning* framework, which represents a substantially more difficult learning task. Nevertheless, we show that an algorithm inspired by the classic E^3 algorithm [Kearns and Singh, 2002], which we call fair E^3, is capable of adversarial fair MDP learning.

In fair learning settings, quantifying whether an action is fair is substantially more difficult than quantifying whether an action is optimal to a single agent, because fairness depends on the context of how the agent behaves overall (i.e., tradeoffs between beneficiaries should be balanced). Consequently, in our model, the agent must output fair policies when it is capable of doing so. When it is not, the agent can output only exploration actions, and our concept of learnability requires that the agent with high probability always outputs ε-optimal fair policies, while taking only a bounded number of exploration actions over its infinite lifetime. We allow an adversary to move the agent arbitrarily after it outputs a policy. At any step, the adversary is allowed to select a new welfare function, representing changing societal ideals of how fairness should work, and the agent is expected to output either an exploration action, or a policy optimizing said welfare function. Of course, the adversary is free to leave the agent's position and welfare function unchanged with no degradation in the upper-bounds we prove.

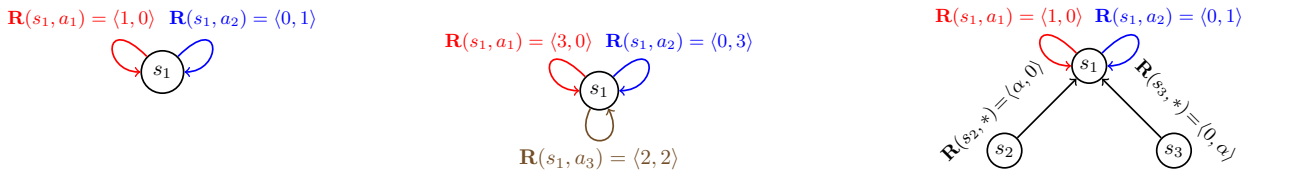
Contributions

1. We frame the traditionally egocentric challenge of reinforcement learning as a social problem, where the actions taken by an agent impact a set of *beneficiaries*.
2. Using ideas from vector-valued reinforcement learning, econometrics, and social welfare theory, we establish *welfare optimal* policies over the value functions of the set of beneficiaries. We focus on finite-state fully-observable MDPs with bounded reward and γ-geometric value-discounting, but our philosophy and methodology can be generalized.
3. We introduce a learning framework in which the agent only observes the consequences of its actions during exploration, and that an adversary can freely move the agent whenever the agent outputs an exploitation policy. During exploitation, the agent is expected to correctly return a near-welfare optimal policy from the current state with high probability, and is also expected to exploit in all but a finite number of exploratory steps. This specific decoupling of exploration and exploitation conditions is particularly conducive to studying the multi-beneficiary RL setting, in which the optimal policy may in general depend on the starting state.
4. We show that our learning framework, which we refer to as the adversarial fair MDP learning framework, is stronger (more difficult) than the well-known probably-approximately correct (PAC) MDP reinforcement learning framework.
5. In section 3, we present the *fair* E^3 algorithm, and prove that it is an adversarial fair MDP learner.

2 Illuminating Examples

We first consider a few simple examples (visualized in fig. 1) that illustrate that intuition from the standard scalar-reward RL setting can be misleading. In these examples, we consider only the relatively straightforward *egalitarian welfare* (i.e., minimum utility, $\mathcal{W}_{\text{Egal}}(a, b) = \min(a, b)$) function, on stateless (single-state) MDPs with deterministic rewards. Despite this barren setting, we find that the standard RL algorithms exhibit misbehavior in learning.

Example 1 (Symmetric 2-Arm Bandit; fig. 1a)**.** *We first consider a 2-arm bandit, with reward* $\mathbf{R}(a_1) = \langle 1, 0\rangle$*, and* $\mathbf{R}(a_2) = \langle 0, 1\rangle$*. Here, the unique optimal Markovian policy is* $\pi^*_{\mathcal{W}} = \langle \frac{1}{2}, \frac{1}{2}\rangle$*.* 187



(a) Symmetric 2-Arm Bandit (b) Compromise 3-Arm Bandit (c) Asymmetric Start Bandit MDP

Figure 1: Small MDPs that exhibit surprising behavior under multi-beneficiary objectives.

There are several surprises here:

1. The (unique) optimal policy is *stationary*, but not *deterministic*, i.e., it is *stochastic*.
2. Policy iteration iteratively selects the greedy welfare-optimal policy, i.e., selects the policy

$$\pi^{(i)} \leftarrow \operatorname*{argmax}_{\pi} \mathcal{W}\Big(\mathbb{E}_{\pi}[\mathbf{R}_1(s_0, a) + \gamma \boldsymbol{V}_1^{\pi^{(i-1)}}(s_1)], \ldots, \mathbb{E}_{\pi}[\mathbf{R}_g(s_0, a) + \gamma \boldsymbol{V}_g^{\pi^{(i-1)}}(s_1)]\Big) \ .$$

This would be optimal under the (false) assumption that the *value function remains fixed*, and iis *convergent* for linear (value) MDP objectives. However, policy iteration for the egalitarian welfare objective, initiated at either deterministic policy, *oscillates* between $\pi(s) = \langle 1, 0 \rangle$ and $\pi(s) = \langle 0, 1 \rangle$. which leads to repeated *overcorrections* for initial policy unfairness, hence the oscillatory behavior.

We now consider an extension, which has a third option, which is not preferable to either beneficiary, but is more effective as a compromise than any mixture of the first two options.

Example 2 (Compromise 3-Arm Bandit; fig. 1b)**.** *We next consider a 3-arm bandit, with reward* $\mathbf{R}(a_1) = (3, 0)$, $\mathbf{R}(a_2) = (0, 3)$, *and* $\mathbf{R}(a_3) = (2, 2)$. *The optimal policies for beneficiary 1, beneficiary 2, and the welfare objective,* $\pi_1 = \langle 1, 0, 0 \rangle$, $\pi_2 = \langle 0, 1, 0 \rangle$, *and* $\pi_{\mathcal{W}} = \langle 0, 0, 1 \rangle$, *respectively.*

This is perhaps not hugely surprising, but it is notable nonetheless, as it starkly illustrates just how different a welfare optimal policy may be from any individual beneficiary's optimal policy. In particular, despite there being unique optimal stationary policies π_1^*, π_2^*, and $\pi_{\mathcal{W}}^*$, clearly, $\pi_{\mathcal{W}}^*$ is not a linear combination of π_1^* and π_2^*. Furthermore, at every state, the probability of selecting any action never exceeds 0 in more than one of these optimal policies. This also has implications for how the MDP is explored; for instance if beneficiaries one and two are independently allowed to run a UCB-style algorithm, neither will even bother to fully explore a_3, thus even together they don't collect enough information for welfare-optimal planning. We thus conclude that not only the planning, but also the exploration aspect of RL must explicitly consider welfare objectives.

We now extend the 2-armed bandit example by adding two additional states, which represent disparate starting conditions for the two beneficiaries.

Example 3 (Symmetric 2-Arm Bandit, with Asymmetric Starting Conditions; fig. 1c)**.** *This MDP has three states, but once* s_1 *is reached, it is never left, thus becoming a 2-armed bandit.*

From s_1, neither beneficiary is privileged, but from s_2, beneficiary 1 begins by receiving α utility, and from s_3, beneficiary 2 begins by receiving α utility. To make things fair, we need to select $\pi(s_1)$ to benefit the underprivileged group, i.e., starting from s_2, have $\pi(s_1)$ choose action 1 with probability x such that $\frac{\gamma}{1-\gamma}x + \alpha = \frac{\gamma}{1-\gamma}(1-x)$, thus $2\frac{\gamma}{1-\gamma}x = \frac{\gamma}{1-\gamma} - \alpha \implies x = \frac{1}{2} - \frac{1-\gamma}{2\gamma}\alpha$. This makes sense: larger α need to subtract a larger privilege correction term from $\frac{1}{2}$, and after a certain point, negative x (which is of course impossible) would be required to compensate for disparate starting conditions.

3 The Fair E^3 Algorithm

We now introduce the *fair* E^3 *algorithm* and bound its sample complexity, thus showing that it is a adversarial fair adversarial MDP learner. We provide pseudocode in Algorithm 1. The key to understanding the algorithm is that the state space is divided into three sets: the unknown set $\mathcal{S}_{\text{unk}}$, the outer-known set $\mathcal{S}_{\text{out}}$, and the inner-known set $\mathcal{S}_{\text{inn}}$. Initially, all states are unexplored, placed in $\mathcal{S}_{\text{unk}}$. After visiting a state and taking all actions sufficiently many times using balanced wandering [Kearns and Singh, 2002], it is placed in either $\mathcal{S}_{\text{out}}$ if with nonnegligible probability $\geq E$ it is possible to reach $\mathcal{S}_{\text{unk}}$ in $\leq T$ steps, or $\mathcal{S}_{\text{inn}}$ if it is not possible to reach the unknown set with nonnegligible probability. Note that E and T will be determined so as to ensure adversarial fair MDP learnability.

Algorithm 1 Fair E^3 and the Adversarial Fair MDP Setting

$\sim$ Fair E^3 Agent Code $\sim$

procedure AGENTINIT($\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, \boldsymbol{T}, \gamma \rangle, \epsilon, \delta, R_{\max}$)
  $T \leftarrow \lceil \frac{1}{1-\gamma} \ln \frac{3R_{\max}}{\varepsilon(1-\gamma)} \rceil$; $t \leftarrow 0$ ▷ Init. escape time; timer
  $E \leftarrow \frac{\varepsilon}{2TR_{\max}}$ ▷ Compute escape probability threshold
  $\mathrm{M} \leftarrow \left\lceil \ln\left(\frac{|\mathcal{S}||\mathcal{A}|\,(2^{|\mathcal{S}|}-2+g)}{\delta}\right) \max\left(\frac{1}{2\beta^2}, \frac{R_{\max}^2}{2\alpha^2}\right) \right\rceil$
  $\forall s \in \mathcal{S}, a \in \mathcal{A} : m_{s,a} \leftarrow 0$ ▷ Per-(s,a) visitation counters
  $\mathcal{S}_{\text{unk}} \leftarrow \mathcal{S}$; $\mathcal{S}_{\text{out}} \leftarrow \emptyset$; $\mathcal{S}_{\text{inn}} \leftarrow \emptyset$ ▷ Init. all states to unknown
  $\hat{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, \hat{\mathbf{R}}, \hat{\boldsymbol{T}}, \gamma \rangle \leftarrow \langle \mathcal{S}, \mathcal{A}, s \mapsto \mathbf{0}, s \mapsto \mathbb{1}_s, \gamma \rangle$
end procedure

procedure AGENTSTEP($s, \mathcal{W}(\cdot)$)
  if $s' \in \mathcal{S}_{\text{unk}}$ **then** ▷ Successful escape
    $t \leftarrow 0$
    return $a_{\text{xpr}} \leftarrow \underset{a \in \mathcal{A}}{\text{argmin}}\, m_{s,a}$ ▷ Balanced walk step
  end if
  if $t > 0$ **then** ▷ Ongoing attempt to escape to $\mathcal{S}_{\text{unk}}$
    $t \leftarrow t - 1$
    return $a_{\text{xpr}} \leftarrow \pi_{\text{esc}}(s, t)$ ▷ Explore a from escape policy π
  end if
  if $s \in \mathcal{S}_{\text{inn}}$ **then** ▷ Return exploit policy
    return $\pi_{\text{xpt}} \leftarrow \underset{\pi \in \Pi_{\mathcal{M}}}{\text{argmax}}\, \mathcal{W}\left(\hat{V}_1^{\pi}(s), \hat{V}_2^{\pi}(s), \ldots, \hat{V}_g^{\pi}(s)\right)$
  else ▷ $s \in \mathcal{S}_{\text{out}}$, begin escape attempt
    $t \leftarrow T$
    return $a_{\text{xpr}} \leftarrow \pi_{esc}(s, t)$
  end if
end procedure

procedure AGENTSARSUPDATE($s, a, \boldsymbol{r}, s'$)
  if $s \in \mathcal{S}_{\text{unk}}$ **then**
    $m_{s,a} \leftarrow m_{s,a} + 1$ ▷ Increment visitation count
    $(\boldsymbol{E}_{s,a,m_{s,a},\mathrm{S}}, \boldsymbol{E}_{s,a,m_{s,a},\mathrm{R}}) \leftarrow (s', \boldsymbol{r})$ ▷ Add to XP buffer
    if $\min_{a \in \mathcal{A}} m_{s,a} = \mathrm{M}$ **then** ▷ State s is learned
      $\forall a, s: \ \hat{\boldsymbol{T}}_{s,a,s'} \leftarrow \frac{1}{\mathrm{M}} \sum_{i=1}^{\mathrm{M}} \mathbb{1}_{s'}(\boldsymbol{E}_{s,a,i,\mathrm{S}})$
      $\forall a: \quad \hat{\boldsymbol{R}}_{s,a} \leftarrow \frac{1}{\mathrm{M}} \sum_{i=1}^{\mathrm{M}} (\boldsymbol{E}_{s,a,i,\mathrm{R}})$
      $\pi_{esc} \leftarrow \underset{\pi \in \Pi_T}{\text{argmax}} \sum_{s \in \mathcal{S}} \mathbb{P}\left(\bigvee_{i=1}^{T} s_i \in \mathcal{S}_{\text{unk}} \middle| \pi, s_1 = s\right)$
      $\mathcal{S}_{\text{unk}} \leftarrow \mathcal{S}_{\text{unk}} \setminus \{s\}$
      $\mathcal{S}_{\text{out}} \leftarrow \left\{ s \in \mathcal{S} \setminus \mathcal{S}_{\text{unk}} \middle| \mathbb{P}\left(\bigvee_{i=1}^{T} s_i \in \mathcal{S}_{\text{unk}} \middle| \pi_{esc}, s_1 = s\right) \geq E \right\}$
      $\mathcal{S}_{\text{inn}} \leftarrow \mathcal{S} \setminus (\mathcal{S}_{\text{unk}} \cup \mathcal{S}_{\text{out}})$
    end if
  end if
end procedure

$\sim$ Fair-Adversarial-MDP Interaction Loop $\sim$

procedure AGENTENVIRONMENTINTERACT($\mathcal{M}, \epsilon, \delta, R_{\max}$)
  AGENTINIT($\mathcal{M}, \epsilon, \delta, R_{\max}$)
  $s \leftarrow$ ADVERSARY ▷ Adversarially select initial state
  while True **do**
    $\mathcal{W}(\cdot) \leftarrow$ ADVERSARY ▷ Adversarial welfare $\mathcal{W}(\cdot)$
    $z \leftarrow$ AGENTSTEP($s, \mathcal{W}(\cdot)$)
    if $z \in \mathcal{A}$ **then** ▷ Explore Action
      $s', \boldsymbol{r} \leftarrow \mathcal{M}(s, z)$
      AGENTSARSUPDATE($s, a, \boldsymbol{r}, s'$)
      $s \leftarrow s'$
    else if $z \in \Pi_{\mathcal{M}}$ **then** ▷ Exploit Policy
      $s \leftarrow$ ADVERSARY ▷ Adversarial subsequent state
    end if
  end while
end procedure

As in the classic E^3 algorithm, within $\mathcal{S}_{\text{inn}}$, if all Chernoff bounds hold simultaneously, the value functions of the empirical MDP approximate the value functions of the true MDP, and furthermore, due to a Lipschitz assumption on welfare functions, optimizing welfare in the empirical MDP ε-optimizes welfare in the true MDP. Therefore, at each time step, if the agent is in $\mathcal{S}_{\text{inn}}$, it outputs a near optimal policy, otherwise if it is in the outer-known set, it begins an escape attempt, which either proceeds for T steps or until a state in $\mathcal{S}_{\text{unk}}$ is reached; if the agent is in $\mathcal{S}_{\text{unk}}$, it executes an action learning more about the unknown state, possibly moving it into a known set.

Definition 1. *Let* $\mathrm{TVD}(x, y)$ *denote the* total variation distance *between probability distributions* x, y. *An* (α, β) *uniform approximation* $\mathcal{M}' = \langle S, A, T', R', \gamma \rangle$ *of a vector-reward MDP* $\mathcal{M} = \langle S, A, T, R, \gamma \rangle$ *is an MDP that satisfies:*

1. $\forall (s,a) \quad \mathrm{TVD}\left(T'(\cdot|s,a), T(\cdot|s,a)\right) \leq \alpha$, *&*
2. $\forall (s,a) \quad |R'(s,a) - R(s,a)| \leq \beta$.

Lemma 1. *Suppose MDP* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, \boldsymbol{T}, \gamma \rangle$, *& let*

$$m \doteq \mathrm{M}(|\mathcal{S}|, |\mathcal{A}|, g, R_{\max}, \alpha, \beta, \delta) \doteq \left\lceil \ln\left(\frac{|\mathcal{S}||\mathcal{A}|\,(2^{|\mathcal{S}|}-2+g)}{\delta}\right) \max\left(\frac{1}{2\beta^2}, \frac{R_{\max}^2}{2\alpha^2}\right) \right\rceil .$$

Now if $\hat{\mathcal{M}}$ *is estimated from taking each (state, action) pair at least* m *times, then, with probability at least* $1 - \delta$, $\hat{\mathcal{M}}$ *is an* α-β *uniform approximation of* $\mathcal{M}$.

Theorem 1. *Algorithm 1* ε-δ *fair-adversarial-MDP learns* $\mathcal{M}$ *for all* $\delta \leq \frac{e}{2}$ *with sample complexity*

$$m_{\mathcal{M}}(\ldots) \leq \left\lceil \frac{|\mathcal{S}||\mathcal{A}| \cdot \mathrm{M}(|\mathcal{S}|, |\mathcal{A}|, g, R_{\max}, \alpha, \frac{\beta}{\lambda}, \frac{\delta}{2}) + 3 \ln \frac{3}{\delta}}{T} \right\rceil \in \mathrm{Poly}\left(|\mathcal{S}|, |\mathcal{A}|, g, R_{\max}, \lambda, \varepsilon, \frac{1}{\delta}, \frac{1}{\gamma}\right) .$$

Essentially, fair E^3 differs very little from E^3, as it will always seek to explore any state that is reachable with nonnegligible probability. However, it must explore each state more times than standard E^3 to account for learning *vector-valued rewards*, and the *exploitation* aspect changes greatly, as it must output *policies* that are nearly *welfare-optimal*, rather than just *actions*.

4 Discussion

We now note that in sensitive contexts, the decision to learn a policy from scratch is rather radical, and many suboptimal actions will likely be taken during the learning process. However, this isn't specific to fairness, but is rather an inherent problem in reinforcement learning in sensitive settings. In medical contexts, Thomas et al. [2019] learn starting from a *reference policy*, and seek to improve the policy while guaranteeing the learned policy is *no worse than* the reference. While this is laudable and reasonable in high-risk or sensitive settings, when *fairness between groups* is a concern, it is inherently a conservative approach (i.e., one which reinforces the status quo; in some sense comparison to reference is an *argumentum ad traditionem*), whereas starting *ex-nihilo* solely depends on the *structure of the MDP* and the *learning algorithm*, rather than existing societal bias which may be encoded in a reference policy.

Still, we note that suboptimal exploration actions taken during exploration could adversely affect one group or another in an unfair way, and in practice this is extremely important and should be monitored and controlled for. However, unlike in some bandit settings, where the cost of exploration may be unfairly borne by one group or another, in our algorithm, the escape policy and balanced walk actions are both completely independent of the reward structure in the MDP, and are thus inherently fairness agnostic. We note also that the number of suboptimal actions can be further reduced by a more careful analysis; for instance the sample complexity of learning transition matrices is much smaller when they are *sparse*, admit a *factoring*, or destination distributions are *far from uniform*, and the simple complexity of learning rewards is much smaller when the variance of rewards is also considered, or when per-beneficiary rewards are not-independent. We are hopeful hopeful that future work will lead to adversarial fair MDP learners that makes fewer suboptimal actions over the course of learning (i.e., have improved sample complexity).

In Conclusion We motivate and define a welfare-centric concept of fair reinforcement learning. Naïve approaches, like planning via policy iteration, and turn-based exploration strategies (as in multi-task learning settings) do not yield successful fair RL agents, even asymptotically. However, we show that under mild regularity conditions on the welfare function, it is possible to learn in the adversarial fair MDP framework while making polynomially many mistakes (algorithm 1, theorem 1). Our method adopts the classic E^3 algorithm, which is an appropriate fit, as its exploration strategy is actually independent of the reward function. As a result, the only change we require is attempting each action more often during exploration to account for the larger number of parameters that must be learned.

References

A. Agarwal, A. Beygelzimer, M. Dudík, J. Langford, and H. Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69. PMLR, 2018.

F. Chierichetti, R. Kumar, S. Lattanzi, and S. Vassilvitskii. Fair clustering through fairlets. *arXiv preprint arXiv:1802.05733*, 2018.

C. Cousins. An axiomatic theory of provably-fair welfare-centric machine learning. In *Advances in Neural Information Processing Systems*, 2021.

C. Cousins. Uncertainty and the social planner's problem: Why sample complexity matters. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022.

D. Ensign, S. A. Friedler, S. Neville, C. Scheidegger, and S. Venkatasubramanian. Runaway feedback loops in predictive policing. In *Conference on Fairness, Accountability and Transparency*, pages 160–171. PMLR, 2018.

M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2):209–232, 2002.

J. Kleinberg, J. Ludwig, S. Mullainathan, and A. Rambachan. Algorithmic fairness. In *AEA papers and proceedings*, volume 108, pages 22–27, 2018.

S. Lohr. Facial recognition is accurate, if you're a white guy. *New York Times*, 9(8):283, 2018.

P. S. Thomas, B. Castro da Silva, A. G. Barto, S. Giguere, Y. Brun, and E. Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

Adaptive Tree Backup Algorithms for Temporal-Difference Reinforcement Learning

Brett Daley*
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
daley.br@northeastern.edu

Isaac Chan*
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
chan.is@northeastern.edu

Abstract

Q(σ) is a recently proposed temporal-difference learning method that interpolates between learning from expected backups and sampled backups. It has been shown that intermediate values for the interpolation parameter $\sigma \in [0, 1]$ perform better in practice, and therefore it is commonly believed that σ functions as a bias-variance trade-off parameter to achieve these improvements. In our work, we disprove this notion, showing that the choice of $\sigma = 0$ minimizes variance without increasing bias. This indicates that σ must have some other effect on learning that is not fully understood. As an alternative, we hypothesize the existence of a new trade-off: larger σ-values help overcome poor initializations of the value function, at the expense of higher statistical variance. To automatically balance these considerations, we propose Adaptive Tree Backup (ATB) methods, whose weighted backups evolve as the agent gains experience. Our experiments demonstrate that adaptive strategies can be more effective than relying on fixed or time-annealed σ-values.

Keywords: Reinforcement Learning, Temporal-Difference Learning, Adaptive Methods, Tree Backup, Expected Sarsa

*Equal contribution.

1 Introduction

Unifying temporal-difference (TD), dynamic programming (DP), and Monte Carlo (MC) methods has helped illuminate trade-offs in reinforcement learning and has lead to the development of better algorithms. Much work has focused on the *length* of backups as a unifying axis between TD and MC methods, in which these two classes can be understood as opposite extremes of the TD(λ) algorithm (Sutton, 1988; Sutton and Barto, 1998). In this view, the decay parameter $\lambda \in [0, 1]$ serves not only as a conceptual link between these distinct methods, but also as a mechanism for managing the bias-variance trade-off (Kearns and Singh, 2000), where intermediate values of λ often deliver the best empirical performance (Sutton and Barto, 1998).

Recent research has begun to explore an orthogonal unification between TD and DP methods by varying the *width* of backups. This concept was formally introduced by Sutton and Barto (2018) in the Q(σ) algorithm; the method interpolates linearly, via a parameter $\sigma \in [0, 1]$, between Expected Sarsa ($\sigma = 0$) (Sutton and Barto, 1998) and Sarsa ($\sigma = 1$) (Rummery and Niranjan, 1994). As such, Q(σ) can be seen as a spectrum of algorithms whose backup widths lie somewhere between those of a sample backup and a full (expected) backup. De Asis et al. (2018) conducted further theoretical and empirical analysis of Q(σ), arriving at two major hypotheses: σ is a bias-variance trade-off parameter, and dynamically adapting σ can lead to faster convergence. These hypotheses appeared to be corroborated by their experiments, where intermediate σ-values performed well, and exponentially decaying σ during training performed even better.

In our work, we re-examine these two hypotheses regarding Q(σ), and ultimately show that they are incorrect. In particular, we prove that the choice of $\sigma = 0$ minimizes the variance of Q(σ) without increasing the bias. It follows that the choice of σ does not constitute a bias-variance trade-off, since both bias and variance can be simultaneously optimized without competition. Furthermore, we prove that the contraction rate and fixed point of the Q(σ) operator are independent of σ, suggesting again that the choice of $\sigma = 0$ is preferable for its variance-reduction effect. These findings are intriguing given the empirical results obtained by De Asis et al. (2018), which did not demonstrate a clear superiority of $\sigma = 0$. This apparent discrepancy between theory and practice suggests that other factors are responsible for the observed benefits of Q(σ), and that our understanding of the algorithm must be revised accordingly.

We therefore advance a new hypothesis to explain the effectiveness of intermediate σ-values: conducting partial backups ($\sigma \neq 0$) helps escape poor initializations of the value function Q by reducing the chance that previously unvisited state-action pairs are considered during the backup. The value estimates of these pairs could be arbitrarily incorrect, slowing initial learning; however, as training progresses and the majority of state-action pairs have been visited, $\sigma = 0$ starts to become more favorable for its variance reduction. This explanation matches the intuition of the time-decayed schedule utilized by De Asis et al. (2018), in which $\sigma = 1$ is desirable early in training and $\sigma = 0$ is desirable later.

This new hypothesis suggests that backup width should be adjusted based on the number of visitations to each state-action pair, with more emphasis placed on frequently sampled pairs. We therefore develop a new family of TD algorithms that we call Adaptive Tree Backup (ATB) methods, which generalizes Tree Backup (Precup et al., 2000) to arbitrary weightings of the value estimates. We highlight two instances of ATB methods (*Count-Based* and *Policy-Based*) that we believe to be promising. The proposed methods have the advantage of eliminating the hyperparameter σ, as they adapt automatically to the data experienced by the agent. Finally, our experiments demonstrate that Policy-Based ATB outperforms the exponential-decay schedule for σ introduced by De Asis et al. (2018).

2 Background

We model the environment as a Markov Decision Process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, P, R)$: $\mathcal{S}$ is a finite set of environment states, $\mathcal{A}$ is a finite set of actions, P is the transition function, and R is the reward function. An agent interacts with the environment by selecting an action $a_t \in \mathcal{A}$ in state s_t with probability $\pi(a_t|s_t)$, receiving a reward $r_t \coloneqq R(s_t, a_t)$ and changing the state to s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$. Given the agent's policy π, the objective is to estimate the expected discounted return $Q^\pi(s_t, a_t) \coloneqq \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$, where $\gamma \in [0, 1]$, since acting greedily with respect to Q^π is known to produce a better policy (Sutton and Barto, 1998).

Q(σ) (Sutton and Barto, 2018) is a method that iteratively improves its estimated value function Q according to

$$Q(s_t, a_t) \leftarrow (1 - \alpha_t)Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \left(\sigma Q(s_{t+1}, a_{t+1}) + (1 - \sigma) \sum_{a' \in \mathcal{A}} \pi(a'|s_{t+1}) Q(s_{t+1}, a') \right) \right), \tag{1}$$

where $\sigma \in [0, 1]$ is a user-specified hyperparameter. This hyperparameter interpolates between the exact on-policy expectation (Expected Sarsa, $\sigma = 0$) and noisy sample estimates of it (Sarsa, $\sigma = 1$), and has been hypothesized to serve as a bias-variance trade-off parameter that improves the contraction rate of the algorithm (De Asis et al., 2018). In the subsequent section, we analyze the theoretical properties of the Q(σ) update to investigate the validity of this hypothesis.

3 Analysis of Q(σ)

We investigate the effects that σ has on the Q(σ) algorithm in terms of convergence, contraction rate, bias, and variance. Our analysis dispels two misconceptions about Q(σ): that dynamically adjusting σ can achieve a faster contraction rate, and that σ functions as a bias-variance trade-off parameter.

192

3.1 Convergence and Contraction Rate

For notational conciseness when proving convergence, we represent the Q(σ) update (1) as an operator $T_\pi^\sigma \colon \mathbb{R}^n \to \mathbb{R}^n$, where $n \coloneqq |\mathcal{S} \times \mathcal{A}|$. The Bellman operator T_π for a policy π is defined as the affine function $Q \mapsto R + \gamma P_\pi Q$, where $(P_\pi Q)(s,a) \coloneqq \sum_{s',a'} P(s'|s,a)\pi(a'|s')Q(a'|s')$. The Bellman operator is a contraction mapping (when $\gamma < 1$) that admits the unique fixed point $Q^\pi = \sum_{k=0}^{\infty}(\gamma P_\pi)^k R$, the unique solution to $T_\pi Q = Q$ (Bellman, 1966).

For policy evaluation, the operator T_π^σ is considered convergent if and only if $\lim_{k\to\infty}(T_\pi^\sigma)^k Q = Q^\pi$ for every $Q \in \mathbb{R}^n$, where $(T_\pi^\sigma)^k Q \coloneqq (T_\pi^\sigma)^{k-1}(T_\pi^\sigma Q)$. Let $w \in \mathbb{R}^n$ be zero-mean noise that represents the randomness due to the environment and policy. Since Q(σ) interpolates linearly between Expected Sarsa and Sarsa, we can express its operator as

$$T_\pi^\sigma Q \coloneqq (1-\sigma)T_\pi Q + \sigma(T_\pi Q + w) = T_\pi Q + \sigma w. \tag{2}$$

We can therefore write update (1) in vector notation as

$$Q_{t+1} = (1-\alpha_t)Q_t + \alpha_t(T_\pi Q_t + \sigma w_t). \tag{3}$$

Theorem 1. *Assume $\sum_{t=0}^{\infty}\alpha_t = \infty$ and $\sum_{t=0}^{\infty}\alpha_t^2 < \infty$. For $\gamma < 1$, the sequence Q_t defined by (3) converges to Q^π almost surely.*

Proof. Iteration (3) is already in a form where Proposition 4.4 of Bertsekas and Tsitsiklis (1996) is applicable; T_π is a contraction mapping with respect to the maximum norm, and σw_t is zero-mean noise. Furthermore, $\mathbb{E}[(\sigma w_t)^2] \le E[w_t^2]$ because $\sigma \in [0,1]$, and $E[w_t^2]$ is bounded by a constant since $\mathcal{S}$ and $\mathcal{A}$ are finite sets. Consequently, under the assumed stepsize conditions, the sequence Q_t converges to Q^π (the fixed point of T_π) almost surely. □

From our derived Q(σ) operator (2), we see that σ primarily functions as a noise attenuation parameter; however, it does not affect the expected update, which is inherently related to the Bellman operator. It follows that the contraction rate and fixed point of Q(σ) are identical to those of the Bellman operator, and that these properties are independent of σ.

3.2 Variance

Theorem 1 suggests that small values of σ have a beneficial variance-reduction effect, with no detrimental effects on convergence; however, the abstract noise process w_t makes it difficult to quantify this effect. In this section, we derive the exact variance of Q(σ) by extending existing results for Sarsa and Expected Sarsa from van Seijen et al. (2009).

Theorem 2. *Let v_t and $\hat{v}_t$ be the variances of Expected Sarsa and Sarsa, respectively. The variance of Q(σ) is given by the expression*

$$\mathrm{Var}(v_t) + \sigma^2(\mathrm{Var}(\hat{v}_t) - \mathrm{Var}(v_t)), \tag{4}$$

and this quantity is minimized when $\sigma = 0$.

Proof. From the definition of Q(σ) as a linear interpolation, we can deduce that its variance has the form

$$\sigma^2 \mathrm{Var}(\hat{v}_t) + (1-\sigma)^2 \mathrm{Var}(v_t) + 2\sigma(1-\sigma)\mathrm{Cov}(\hat{v}_t, v_t). \tag{5}$$

The Sarsa and Expected Sarsa updates share the same expected value (van Seijen et al., 2009); let it be denoted by μ_t. We show algebraically that $\mathrm{Cov}(\hat{v}_t, v_t) = \mathrm{Var}(v_t)$ by comparing the exact calculation $\mathbb{E}[(\hat{v}_t - \mu_t)(v_t - \mu_t)]$ to the expression for $\mathrm{Var}(v_t)$ derived by van Seijen et al. (2009). Therefore, expression (5) reduces to

$$\sigma^2 \mathrm{Var}(\hat{v}_t) + [(1-\sigma)^2 + 2\sigma(1-\sigma)]\mathrm{Var}(v_t) = \mathrm{Var}(v_t) + \sigma^2(\mathrm{Var}(\hat{v}_t) - \mathrm{Var}(v_t)),$$

which is the desired result in expression (4). From van Seijen et al. (2009), $\mathrm{Var}(\hat{v}_t) - \mathrm{Var}(v_t) \ge 0$, and hence expression (4) is minimized when $\sigma = 0$. □

Theorem 2 shows that the variance of Q(σ) is minimal when $\sigma = 0$ and increases monotonically, which is interesting because it supports the view of σ as a noise attenuation parameter from Theorem 1. Furthermore, since the expected update is unaffected by the choice of σ, it cannot represent a bias-variance trade-off; the variance can be minimized with no apparent drawback. Our theory indicates that there must be a different explanation for the empirical success of Q(σ).

4 Adaptive Tree Backup (ATB) Algorithms

Q(σ) does not address the standard bias-variance trade-off for temporal-difference learning, since the choice of $\sigma = 0$ minimizes variance without impacting bias. Why, then, did De Asis et al. (2018) observe empirical improvement when employing an intermediate σ-value? We conjecture that Expected Sarsa ($\sigma = 0$) struggles to overcome initialization error early in training (during which the majority of the state-action space is unexplored). Until every action has been sampled at least once in a given state, Expected Sarsa mixes one or more nonsensical value estimates into its full backup, potentially resulting in severe inaccuracies. In contrast, a sample method like Sarsa ($\sigma = 1$) is likely to visit previously taken actions (simply by definition of a probabilistic policy), thereby bootstrapping sooner from more accurate estimates, but also with additional noise due to sampling. This phenomenon helps explain why the time-decayed schedule for σ

proposed by De Asis et al. (2018) is effective; excluding incorrect value estimates ($\sigma = 1$) is important when most state-action pairs have not been visited, but minimizing variance ($\sigma = 0$) becomes more important once all estimates are relatively accurate.

If our hypothesis is correct, then methods that dynamically increase the effective backup width based on state-action pair visitations should generally perform better than Q(σ) with a fixed σ-value or time-annealed schedule. We call these Adaptive Tree Backup (ATB) methods, as they generalize Tree Backup (Precup et al., 2000) to arbitrary backup weightings. For a given transition $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, ATB algorithms can be written in the generic form

$$Q(s_t, a_t) \leftarrow (1 - \alpha_t)Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \sum_{a' \in \mathcal{A}} c_t(s_{t+1}, a')Q(s_{t+1}, a') \right), \quad \text{subject to} \sum_{a \in \mathcal{A}} c_t(s, a) = 1, \forall\, s \in \mathcal{S}, \tag{6}$$

where each $c_t(s, a)$ is a nonnegative (possibly random) coefficient. Note that if $c_t(s, a) = (1-\sigma)\pi(a|s)+\sigma\mathbb{I}_{a=a_{t+1}}$, then we recover Q($\sigma$) exactly. However, the generality of the coefficients $c_t(s, a)$ permits myriad possibilities beyond the limited, one-dimensional spectrum spanned by Q(σ).

The backup associated with update (6) admits the correct fixed point Q^π only if $\mathbb{E}[c_t(s, a)] = \pi(a|s)$. Even so, if we ensure that $\mathbb{E}[c_t(s, a)] \to \pi(a|s)$ as $t \to \infty$, then the ATB method will eventually converge to Q^π according to Theorem 1. We subsequently discuss two promising variants that do this.

4.1 Count-Based ATB

To match the intuition that an effective ATB strategy should ignore unvisited state-action pairs, we develop a method that weights state-action pairs according to their relative frequency of appearance. This can be accomplished by tracking a count $n(s, a)$ for each state-action pair (s, a) and then normalizing it as a proportion. This variant, which we call *Count-Based* ATB, defines the backup coefficients as

$$c_t(s, a) = \frac{n(s, a)}{\sum_{a' \in \mathcal{A}} n(s, a')}. \tag{7}$$

By the law of large numbers, $c_t(s, a) \to \pi(a|s)$ after infinitely many visitations to state s, and Theorem 1 with $\sigma = 0$ applies at the limit. Beyond convergence to Q^π, there are several reasons why this particular ATB formulation is appealing. First, the count-based backup amounts to a maximum-likelihood 1-step estimate of $Q(s, a)$, and in this sense is the "most reasonable" backup to perform based on previously observed experiences. Furthermore, frequency counts implicitly emphasize value estimates that are more reliable, while excluding those that have not changed since initialization. This means that Count-Based ATB automatically transitions from Sarsa to Expected Sarsa, with a different effective rate for each state, which could not be achieved easily with Q(σ) and does not require a user-specified hyperparameter schedule.

Unfortunately, these benefits are not without some drawbacks. Although the algorithm theoretically converges to Q^π in the limit, it is extremely unlikely that $c(s, a) = \pi(a|s)$ after any finite amount of training. This means that Count-Based ATB may tend towards a fixed point other than Q^π in practice, which is undesirable even if it does represent a maximum-likelihood estimate according to past experience. Another problem—which is much more significant—occurs when the agent switches its policy from π to a different policy π' during training. At this point, all of the previously collected counts $n(s, a)$ will no longer reflect the correct on-policy distribution. The agent would either need to reset the counts to zero (thus discarding useful information), or wait many timesteps for the counts to accurately reflect the new policy π'. Both of these options are inefficient, motivating us to search for a better alternative in the next subsection.

4.2 Policy-Based ATB

We can resolve the issues of Count-Based ATB by directly utilizing knowledge of the policy π, as do Q(σ), Expected Sarsa, and Tree Backup. If some state-action pair (s, a) is visited for the first time, then we know that the eventual value of $c_t(s, a)$ will be $\pi(a|s)$ according to Count-Based ATB. We can greatly accelerate convergence by immediately assigning $c_t(s, a) = \pi(a|s)$ after this first visitation. Let u denote the unit step function such that $u(n) = 1$ if $n > 0$ and $u(n) = 0$ otherwise. The backup coefficients for this new variant, which we call *Policy-Based* ATB, become

$$c_t(s, a) = \frac{u(n(s, a))}{\sum_{a' \in \mathcal{A}} u(n(s, a'))}\pi(a|s). \tag{8}$$

This definition clearly solves the two problems of Count-Based ATB. Unvisited state-action pairs are still ignored, as desired, but after each pair has been sampled at least once, the algorithm becomes equivalent to Expected Sarsa and starts converging to Q^π. This effectively bypasses the infinite-visitation requirement of Count-Based ATB. We can also see that the weighted backup instantly reflects changes to the policy, thanks to the explicit dependency on π in definition (8). For these reasons, we expect Policy-Based ATB to learn significantly faster than Count-Based ATB in a tabular setting; however, Policy-Based ATB may be harder to combine with function approximation in high-dimensional MDPs, where the need for generalization makes it difficult to determine whether a state should be considered visited or unvisited. Additionally, the reliance on an explicit policy model $\pi(a|s)$ may be restrictive in some settings. We therefore foresee useful roles for both ATB variants.

5 Experiments and Conclusion

To test whether adaptive strategies can outperform fixed σ-values and the handcrafted schedule proposed by De Asis et al. (2018), we compare our two ATB variants against Q(σ) in a 19-state deterministic random walk environment (Sutton and Barto, 2018) and an 11-state stochastic gridworld environment (Russell and Norvig, 2010). For all methods, we fix $\alpha_t = 0.4$ and $\gamma = 1$. We train the agents for 200 episodes and plot the root-mean-square (RMS) error of the estimated value function Q relative to Q^π (Figure 1). We plot the error after each episode, averaging over 50 independent trials, where the shaded regions represent 99% confidence intervals. Policy-Based ATB performs strongly, learning significantly faster than all of the methods except Q(0), and even surpassing the handcrafted, dynamic σ schedule of De Asis et al. (2018)—an impressive feat for an automatic method. Count-Based ATB outperforms Q(1), but its asymptotic performance in the deterministic random walk deteriorates due to the fixed-point bias we discussed earlier; however, this seems to have a less detrimental effect in the stochastic gridworld. Surprisingly, in contrast to the results of De Asis et al. (2018), we find that $\sigma = 0$ is the best choice of fixed σ-value in both environments.

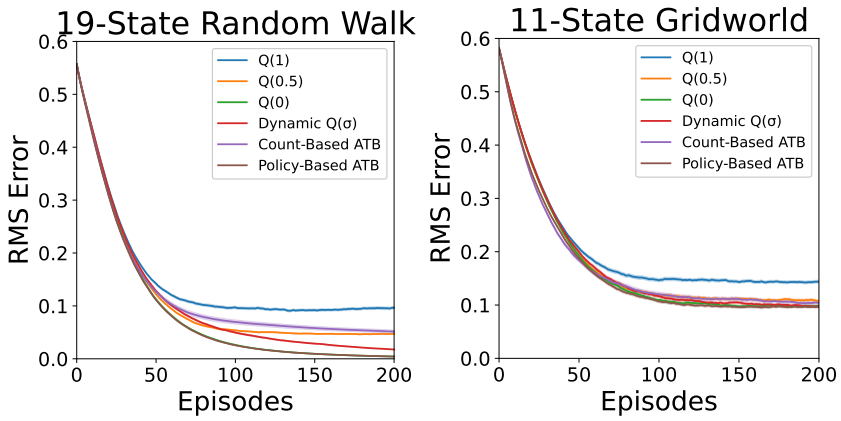


Figure 1: Comparison of our ATB methods against Q(σ) with fixed σ-values and the dynamic schedule proposed by De Asis et al. (2018).

The empirical success of our adaptive methods—particularly Policy-Based ATB—has important implications. First, it shows that data-driven adaptive strategies for adjusting the backup width can be more effective than Q(σ) with a fixed σ-value or a pre-specified schedule for annealing σ. In addition, since both of our methods gradually expand the effective backup width as new state-action pairs are visited, they further strengthen our hypothesis that initialization error in the value estimates is an important consideration when determining backup width. This also helps formalize the intuition of De Asis et al. (2018) that backup width should increase over time. Finally, these results show that $\sigma = 0$ is a good choice in practice, corroborating our theoretical predictions that it minimizes variance without negatively impacting the bias or contraction rate of Q(σ). Although σ does not represent a bias-variance trade-off in temporal-difference learning, it is clear that σ does play an important role in learning, and that analyzing methods in terms of how well their backups can balance initialization error with variance is a promising pathway to further algorithmic improvements.

References

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Kristopher De Asis, J Hernandez-Garcia, G Holland, and Richard Sutton. Multi-step reinforcement learning: A unifying algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Michael J Kearns and Satinder P Singh. Bias-variance error bounds for temporal difference updates. In *COLT*, pages 142–147, 2000.

Doina Precup, Richard S Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*, page 759–766, 2000.

G. A. Rummery and M. Niranjan. On-line Q-Learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1st edition, 1998.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Harm van Seijen, Hado van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of Expected Sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, 2009.

Inverse Policy Evaluation for Value-based Decision Making

Alan Chan[*]
Department of Computer Science
Université de Montréal
Montréal, QC
alan.chan@mila.quebec

Kristopher De Asis*
Department of Computing Science
University of Alberta
Edmonton, AB
kldeasis@ualberta.ca

Richard S. Sutton
Department of Computing Science
University of Alberta
Edmonton, AB
rsutton@ualberta.ca

Abstract

Value-based methods for control often involve approximate value iteration (e.g., Q-learning), and behaving greedily with respect to the resulting value estimates with some degree of entropy to ensure the state-space is sufficiently explored. Such a greedy policy is an improvement over the policy of which the current values reflect. As learning progresses, value-iteration may produce value functions that do not correspond with *any* policy. This is especially relevant with function-approximation, when the true value function cannot be perfectly represented. This raises questions about what such inaccurate value functions represent, and whether there exists alternative ways to derive value-based behavior. In this work, we explore the use of *Inverse Policy Evaluation*, the process of solving for a likely policy given a value function. We derive a simple, incremental algorithm for the procedure, analyze how inaccuracies in the value function manifest in the corresponding policy, and provide empirical results emphasizing key properties of the mapping from value functions to policies. This sets up a framework for matching *proposed returns*, as opposed to explicit maximization. Coupling this procedure with value-iteration, this provides a novel approach for deriving behavior from a value function which also tends toward an optimal policy, but appears to account for estimation error in the resulting behavior.

Keywords: reinforcement Learning, value-based control, exploration

Acknowledgements

The authors thank the entire Reinforcement Learning and Artificial Intelligence research group for providing the environment to nurture and support this research. We gratefully acknowledge funding from Alberta Innovates – Technology Futures, the Alberta Machine Intelligence Institute, Google Deepmind, and from the Natural Sciences and Engineering Research Council of Canada.

[*] = Equal contribution.

1 Value-based Reinforcement Learning

Reinforcement learning (RL) formalizes the sequential decision-making problem with the Markov Decision Process (MDP) framework [14, 17]. At each discrete time step t, an agent observes the current state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of states in an MDP. Given S_t, an agent selects action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(s)$ is the set of available actions in state s. The environment then returns a reward $R_{t+1} \in \mathbb{R}$, and an observation of the next state $S_{t+1} \in$, sampled from the environment's transition dynamics: $p(s', r|s, a) = \Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$. An agent selects actions according to a policy $\pi(a|s) = \Pr(A_t = a|S_t = s)$, and its goal is to find an *optimal policy* π^* which from each state, maximizes the expected discounted sum of future rewards with discount factor γ (denoted the expected *return*).

Value-based methods estimate *value functions* which quantifies a policy's performance. In control, an *action-value function* is approximated, representing the expected return from starting in state s, taking action a, and following policy π thereafter:

$$q_\pi(s, a) = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, A_t = a\Big] \tag{1}$$

Computing a policy's value function is known as *policy evaluation*. Value-based methods then rely on *policy improvement*, where greedifying with respect to another policy's value function will produce an improved policy. *Policy iteration* [4] interleaves policy evaluation and improvement until an optimal policy is found. In contrast, *policy gradient* methods explicitly parameterize a policy, and updates the parameters to maximize an objective [18, 17].

Q-learning [22] is a popular value-based method which ties to directly estimate the value function of the optimal policy. Value functions can be expressed in terms of successor states' values through their *Bellman equations*. For q_π, we have:

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)\Big(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\Big) \tag{2}$$

When π is greedy with respect to value estimates, the latter term becomes the maximum action-value in the successor state, and gives the *Bellman optimality equation*, the recursive relationship for the optimal action-value function q_*. Repeat evaluation of the Bellman optimality equation across the state-space is known as *value-iteration*, and is akin to policy iteration with partial policy evaluation. Q-learning uses stochastic samples of p to perform approximate value-iteration:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\Big) \tag{3}$$

with step size α. Convergence of Q to q_* for every state-action pair requires that all state-action pairs are visited infinitely often [5]. Even if convergence is not guaranteed, as is the case with function approximation [3, 19, 1], it is still necessary to *explore* different parts of state-space to obtain a reasonable estimate of q_* to inform decision making.

This exploration requirement is often satisfied with ϵ-greedy action selection [11, 17], where an agent behaves greedily with respect to its current estimates with probability $1-\epsilon$, and behaves uniform randomly otherwise. However, there are practical concerns with ϵ-greedy behavior: 1) reliance on random exploration is likely inefficient in large state spaces [9]; 2) due to estimation error or representational capacities, the greedy action may be a poor choice [7]; 3) needing to specify ϵ as a fixed value or an annealing schedule; and 4) the non-smoothness of the behavior policy with respect to changes in the value function can result in non-convergence [12, 13, 21].

Several alternatives to ϵ-greedy have been explored, e.g., Boltzmann policies [2], conservative policy iteration [8, 20]. Many of them directly work with the policy that an agent aims to *eventually* evaluate, i.e., some modification of the greedy policy. No work to our knowledge has explicitly considered the policy which corresponds with the *current* value estimates. Such an approach would preferably take estimation error into account, e.g., due to insufficient exploration or function approximation errors, rather than assume the current values are accurate. Given $Q = q_*$, the policy that gives zero Bellman error *is* an optimal policy by the uniqueness of the solution of the Bellman optimality equation [5]. Should Q be inaccurate, the policy should be suboptimal in a way that's directly related to the errors.

2 Inverse Policy Evaluation

Inverse Policy Evaluation (IPE) aims to derive a behavior policy that is consistent with a value function in the following sense:

$$\pi \in \operatorname*{argmin}_{\pi \in \Pi} \|Q(s, a) - \mathbb{E}_{s', a'}[r(s, a) + \gamma Q(s', a')]\| \tag{4}$$

where $s' \sim p(\cdot|s, a)$, $a' \sim \pi(a'|s')$, $r(s, a)$ giving the expected immediate reward for s, a, and $\|\cdot\|$ being some fixed norm over state-action pairs. Let us solidify the intuition that the solution takes function approximation error into account.

Proposition 1. *Assume that we are trying to estimate q_π with Q, for some q_π. Denote the solution of Equation 4 by π_{IPE}, the Bellman operator of Equation 2 as $\mathcal{T}$, and let $\|\cdot\|$ denote any norm under which Bellman operators are contraction mappings (e.g., infinity norm). We have the following bound.*

$$\|q_\pi - q_{\pi_{IPE}}\| \leq \frac{1}{1-\gamma}\big((1+\gamma)\|q_\pi - Q\| + \|\mathcal{T}^{\pi_{IPE}}Q - Q\|\big)$$

The first norm on the right-hand side measures the estimation error of Q, and the second norm on the right-hand side is exactly the objective in Equation 4. From this, the return generated by π_{IPE} is close to the return generated by π, proportional to how close Q is to q_π. We now consider how changes in value estimates affect π_{IPE}.

Proposition 2. *Suppose Q_1, Q_2 are two approximate action-value functions. Let π_i denote the IPE solution of Q_i. Then*

$$\|q_{\pi_1} - q_{\pi_2}\| \leq \frac{1}{1-\gamma}((1+\gamma)\|Q_1 - Q_2\| + \|\mathcal{T}^{\pi_1}Q_1 - Q_1\| + \|\mathcal{T}^{\pi_2}Q_2 - Q_2\|)$$

This smoothness result for IPE contrasts ϵ-greedy policies, which are known to be non-smooth with respect to changes in action-value estimates [12, 13]. So IPE can produce an estimation-error-aware policy, but how do we compute this? With the ℓ_2 norm and policy π_θ smoothly parameterized by θ, we can approach π_{IPE} through the gradient-based update:

$$\begin{aligned}\delta &= R_{t+1} + \gamma \sum_{a'} \pi_\theta(a'|S_{t+1})Q(S_{t+1}, a') - Q(S_t, A_t)\\ \theta_{t+1} &\leftarrow \theta_t - \alpha 2\delta\gamma \sum_{a'} \nabla_\theta \pi_\theta(a'|S_{t+1})Q(S_{t+1}, a')\end{aligned} \tag{5}$$

where δ is the conventional *temporal difference (TD) error* [16]. Of note, the update is remarkably similar to the *all-actions policy gradient* update [18, 17]. One can interpret Equation 5 as a policy gradient update for *matching proposed returns* of an approximate value function Q, with δ changing signs to ensure that the return is matched, rather than maximized.

It may be the case that Q does not correspond with *any* policy, relating to *delusional bias* [6, 10]. Nevertheless, the resulting policy will still minimize Equation 4, which by Proposition 1 would be close in a sense of achieving similar returns as a nearby "valid" value function (i.e., has a corresponding policy).

Should we couple IPE with value-iteration, the proposed returns will tend toward those achieved by the optimal policy, and IPE will approach an optimal policy. This results in an actor-critic-like procedure, but with less cyclic dependencies. Actor-critic methods typically perform a cycle of: evaluating the current behavior policy, improving the policy with the current evaluation. In contrast, value-iteration tries to compute the largest possible returns (and not those of the current policy), that the remaining dependency is on the distribution of transitions experienced by the current policy.

3 Empirical Evaluation

Here we run some experiments to validate some key intuitions behind IPE.

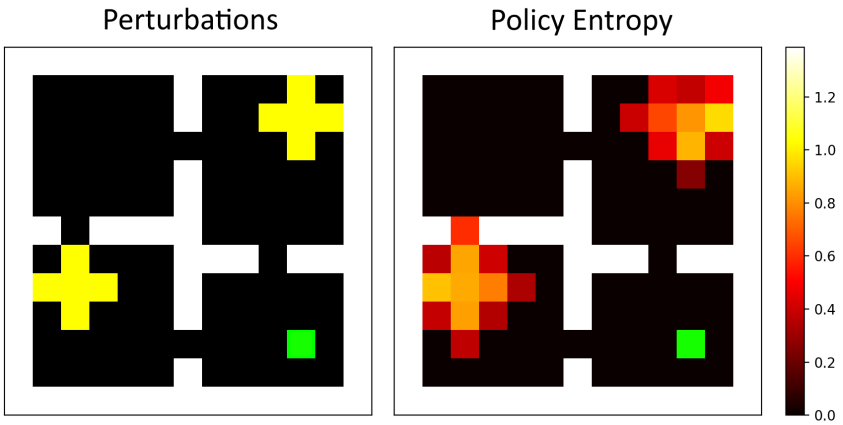


Figure 1: Resulting π_{IPE} entropy after perturbing q_* in the Four Rooms domain. Green denotes the goal state.
198

3.1 Stochasticity from Estimation Error

Using the Four Rooms domain [15], we look at how estimation error impacts the stochasticity of π_{IPE}. First, we compute q_* for the domain, and initialize π_{IPE} to an optimal policy (breaking ties consistently). We then perturb q_* in select state-action pairs by samples from $\sim \mathcal{N}(0, 0.1)$, and run IPE to convergence. Averaged over 1000 runs, Figure 1 visualizes the policy's entropy in each state of the environment and how they relate to the perturbations in the value function.

It can be seen that the increase in policy entropy is directly related to the perturbed state-action pairs, affecting both the perturbed areas as well as states immediately transitioning to them. This validates that the policy becomes suboptimal in a way that's related to the errors. It further motivates the possible use of IPE to adapt exploration, as the stochasticity tends to explore areas with larger errors. Viewing the policy entropy as a level of confidence in the accuracy of value estimates, it may also inform parameter selection, as having the confidence expressed in entropy is convenient for hyper-parameters with probabilistic interpretations (e.g., λ in TD(λ), ϵ in ϵ-greedy). Along these lines, we consider an ϵ-greedy Q-learning agent which additionally performs IPE updates on the experienced transitions, and adapts ϵ to match the entropy of the estimated π_{IPE} in the current state. We denote this approach ϵ-IPE.

3.2 Balancing Reward and Estimation Error

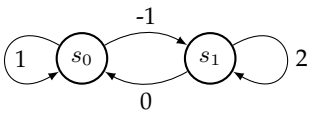


Figure 2: The switch-stay MDP. All transitions are deterministic and the agent starts in state s_0.

Here we use a simple 2-state MDP detailed in Figure 2. We compare four Q-learning agents, each differing in how behavior is derived: (1) ϵ-greedy with fixed ϵ, (2) ϵ-greedy with annealing ϵ, (3) following π_{IPE}, and (4) ϵ-IPE. We swept over ϵ for (1), the number of steps to linearly anneal ϵ from 1.0 to 0.1 for (2), and the IPE step size, α_π, for (3) and (4). Each setting performed 1000 runs of 500 steps, and Figure 3 shows the average reward over the 500 steps, as well as the final root-mean-squared error (RMSE) in the approximated optimal value function.

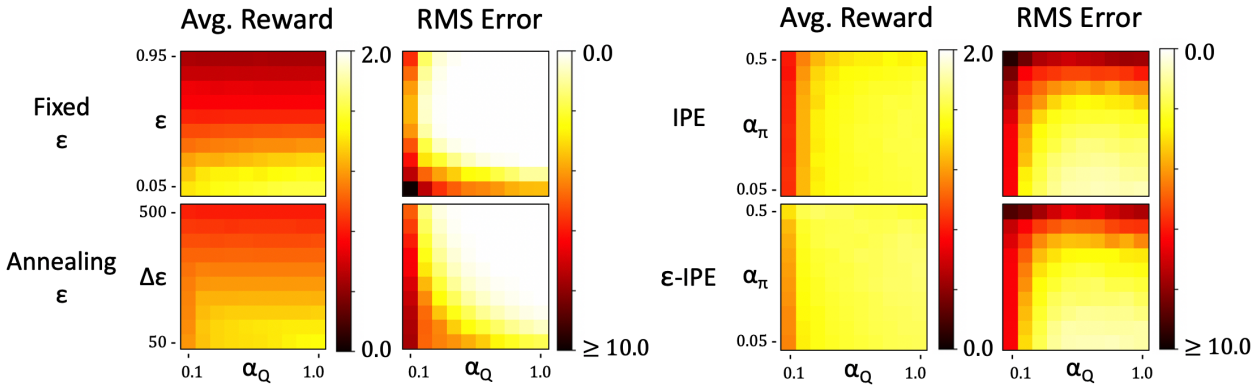


Figure 3: Hyperparameter sensitivities on the switch-stay MDP. 1000 runs of 500 steps were performed for each behavior policy's hyperparameter configuration.

In Figure 3, Q-learning with ϵ-greedy exhibited a negative correlation between the average reward and the value function RMSE. To attain high average reward, the agent tends to settle for an *inaccurate* value function, and vice-versa. Such a relation is what one might intuitively expect from tending toward either extreme of the exploration-exploitation trade-off.

On the other hand, IPE and ϵ-IPE exhibit a positive correlation between value function accuracy and the average reward obtained (with respect to the behavior policy's parameter). Given the large overlap between the regions of high average reward and low RMSE, there seems to be, without careful parameter tuning, a natural adequate balance of (1) exploration needed to learn an accurate value function and (2) exploitation of value estimates to achieve a large expected return.

4 Discussion and Conclusions

Our results suggest that IPE, when combined with value-iteration, provides a novel, viable way to derive a sensible behavior policy for value-based control. While greedy policies generally lead to (immediate) larger returns [8], we highlight how such greedification is still present in the use of value-iteration. We further emphasize how stochasticity related to estimation error can lead to visitation distributions which explore more in regions with larger estimation errors (and vice-versa), and be a useful metric of confidence in value estimates to inform parameter selection.

A key limitation in this work is the focus on the tabular setting. We expect the intuitions and results surrounding estimation errors to still be relevant, as examining behavior under inaccurate value functions (or perturbations of the optimal values) can be seen as simulating the inability to perfectly represent the value function. That said, extensive evaluation with function approximation (both linear and non-linear) are warranted to assess the approach's scalability.

References

[1] J. Achiam, E. Knight, and P. Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[2] K. Asadi and M. L. Littman. An alternative softmax operator for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 243–252. JMLR. org, 2017.

[3] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[4] D. P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

[6] R. Dadashi, A. A. Taïga, N. L. Roux, D. Schuurmans, and M. G. Bellemare. The value function polytope in reinforcement learning. *arXiv preprint arXiv:1901.11524*, 2019.

[7] H. V. Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[8] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[9] D. Korenkevych, A. R. Mahmood, G. Vasan, and J. Bergstra. Autoregressive policies for continuous control deep reinforcement learning. *arXiv preprint arXiv:1903.11524*, 2019.

[10] T. Lu, D. Schuurmans, and C. Boutilier. Non-delusional q-learning and value-iteration. In *Advances in neural information processing systems*, pages 9949–9959, 2018.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] T. J. Perkins and M. D. Pendrith. On the existence of fixed points for q-learning and sarsa in partially observable domains. In *ICML*, pages 490–497, 2002.

[13] T. J. Perkins and D. Precup. A convergent form of approximate policy iteration. In *Advances in neural information processing systems*, pages 1627–1634, 2003.

[14] M. L. Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[15] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[16] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[19] J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[20] N. Vieillard, O. Pietquin, and M. Geist. Deep conservative policy iteration. *arXiv preprint arXiv:1906.09784*, 2019.

[21] P. Wagner. Optimistic policy iteration and natural actor-critic: A unifying view and a non-optimality result. In *Advances in Neural Information Processing Systems*, pages 1592–1600, 2013.

[22] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Safety through Intrinsically Motivated Imitation Learning

Henrique Donâncio
Normandie Université, INSA Rouen
LITIS
Rouen, France
henrique.donancio@insa-rouen.fr

Laurent Vercouter
Normandie Université, INSA Rouen
LITIS
Rouen, France
laurent.vercouter@insa-rouen.fr

Abstract

Deep Reinforcement Learning methods require a large amount of data to achieve good performance. This scenario can be more complex, handling real-world domains with high-dimensional state space. However, historical interactions with the environment can boost the learning process. Considering this, we propose in this work an imitation learning strategy that uses previously collected data as a baseline for density-based action selection. Then, we augment the reward according to the state likelihood under some distribution of states given by the demonstrations. The idea is to avoid exhaustive exploration by restricting state-action pairs and encourage policy convergence for states that lie in regions with high density. The adopted scenario is the pump scheduling for a water distribution system where real-world data and a simulator are available. The empirical results show that our strategy can produce policies that outperform the behavioral policy and offline methods, and the proposed reward functions lead to competitive performance compared to the real-world operation.

Keywords: Learning from Demonstrations, Deep Reinforcement Learning, Pump Scheduling Optimization

Acknowledgements

We would like to thank Harald Roclawski (TU Kaiserslautern), Aloysio P. M. Saliba (UFMG), Benjamin Dewals (ULiège), Anika Theis (TU Kaiserslautern), Thomas Pirard (ULiège), and Thomas Krätzig (Dr. Kraetzig) for their comments and suggestions regarding this work. The authors acknowledge FAPEMIG, Federal Ministry of Education and Research of Germany, Agence Nationale de la Recherce de France and Fonds de la Recherce Scientifique Belge, for funding this research by the project IoT.H2O (ANR-18-IC4W-0003) on the IC4Water JPI call.

1 Introduction

This work proposes the *Safety through Intrinsically Motivated Imitation Learning (SIMIL)* strategy that uses the distribution of historical interactions (*demonstrations*) as a guideline for action selection. The approach works as follows: given a current state, action selection depends on choosing the one that occurs most frequently in the most similar states found in the demonstrations. Later, we augment the immediate reward with an intrinsic motivation [7] according to the state likelihood under some distribution of states. The underlying idea is to constrain the policy to state-action pairs found in expert demonstrations using k-Nearest Neighbors (k-NN) to avoid exhaustive exploration. Also, we encourage states that lie in high-density regions under the demonstrations distribution using Kernel Density Estimation (KDE) [1].

We apply this imitation learning strategy in a scenario of pumping scheduling for water distribution systems (WDS). For that, it is available a dataset of three years of data collected in timesteps of one minute from a real-world operation. The pump scheduling is the process to decide when, and in some cases at which speed, the pump(s) should operate regarding the forecasting of the water demand. Yet, some requirements must be satisfied, including safety constraints of water level in the tanks and pressure in the network's nodes. Some works have addressed these questions through several methods, including linear optimization, evolutionary and branch-and-bound algorithms, and recently Deep RL [8, 9, 10, 11]. This work uses a Deep Q-Networks (DQN) [2]-based approach to handle the pump scheduling problem. The contributions presented in this work are the following:

- A formulation of the pumping scheduling problem using Partially Observable Markov Decision Process (POMDP) is presented, with definitions of system states/observations, actions, and reward function. These definitions allow the system to operate by achieving the constraints and minimizing the associated costs;
- An imitation learning strategy using real-world/offline data. The empirical results demonstrated that the obtained policies achieved competitive average cumulative rewards compared with fully-offline training.
- To evaluate the proposed scheduling, we compare the results with the real-world water distribution system regarding the electricity consumed, pumps use distribution and the tank level profile. The results showed that our approach achieved competitive performance with real-world operation.

2 Modeling the pump scheduling problem

In water distribution systems, pump scheduling is a decision process about when operating pumps to supply water while limiting electricity consumption. Therefore some constraints must be respected, including a minimum pressure within the network, safety water level in the tanks, and avoiding frequent switches in pump operation to protect the assets. To that end, distinct strategies can be used according to the particularities of the system. For instance, pumping water in off-peak hours when the price of electricity has different tariffs throughout the day or reducing the tank level in periods of low consumption to preserve the water quality, and so on.

The water distribution system used here is located in Worms, Germany, and supplies water for about 120000 citizens[1]. The composition of this system is one station with four pumps (NP1, NP2, NP3, NP4), with distinct settings and fixed speed (ON/OFF). The flow Q through those pumps is proportional to the electricity consumption kW, being NP1 $>$ NP2 $>$ NP3 $>$ NP4. In other words, using pump NP1 supplies more water in the network than pump NP2 but also corresponds to higher electricity consumption. Also, two storage tanks with different capacities are placed and provide water for the end consumers. Among the constraints and requirements established in the operation settings for this system are the following:

- It is desirable to avoid frequent switches and distribute pump operations to protect the assets;
- It is imposed a boundary condition of the tank level and, once achieved, the minimum pressure is guaranteed;
- It is desirable to provide water exchange in the tank during one day of operation to keep the water quality.

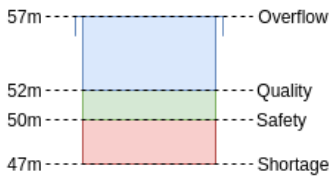


Figure 1: The tank constraints.

[1]The dataset has been provided by the IoT.H2O project (IC4WATER JPI funding)

Figure 1 shows the constraints defined for the tank levels. The tank is located 47m above the pumps and has a 10m length. Thus, the tank levels considered are in the range of [47, 57]m. We assume only one tank once that the second has the level *stable* along with the operation. The specialists consider a safety operation guarantee with at least 3m filled with water. Besides this, the system does not have sensors measuring the water's quality. Thus, to *ensure* the exchange and preserve the water's quality, we assume that in one operation day, the level must decrease below half of the total capacity. Finally, the upper boundary constraint overlaps the physical limit.

As with many real-world tasks, the scenario of pumping scheduling is partially observable. In other words, the agent has a noisy or incomplete observation of the environment. For example, some state features are noisy since they are collected by sensors. A POMDP is an extension of MDP that considers uncertainty regarding the current state of the environment. Formally, the POMDP can be defined as [3]:

$$POMDP = < S, A, P, R, \Omega, O, \gamma > \tag{1}$$

The **States** S and the **Observations** Ω are interchangeable in the context of this work as adopted in [12] and represented by:

- The water level in the tank and water consumption;
- The previous action performed (currently being applied);
- The cumulative time that the pumps operated in a horizon length of 24 hours, the month and time t;
- A binary value called water quality indicating whether on the current day of operation the system has reached a certain minimum in the tank level.

Actions A are defined by the set of binary values that represent if some pump is operating (value 1) or not (value 0) once the pumps have fixed speed. At each timestep, only one pump is running or none of them.

Finally, two **Reward** functions are designed to choose the most *efficient* pump at a given time t, as well respect the boundary conditions of the tank level, preserve the water quality, and make use of different pumps. The immediate rewards are defined by the Equations 2 and 3:

$$r_t = e^{1/(-Q_t/kW_t)} - B * \psi + \log(1/(P + \omega)) \tag{2}$$

$$r_t = -e^{(-1/kW_t)} - B * \psi + \log(1/(P + \omega)) \tag{3}$$

where at the time t, Q_t is the flow rate through the active pump, and kW_t is the respective electricity consumption; B is the achievement of lower/upper restrictions of the water level in the tank. These lower/upper values are defined by specialists in the system and in case of not achievement, $B = 1$ in case of overflow and $B = abs(level\_of\_the\_tank_t - boundary\_condition) \in (0, 1]$ in case of (near) shortage, being $\psi = 10$, otherwise B = 0. Also, B has an exception, being -1 strictly for the timestep when the tank level reaches the water quality condition. P is a penalty that increases with accumulated pump run time. The penalty P increases +1 at each timestep of cumulative operating time, and for the Equation 3 it also hold for the action *(NOP)*. In the case of switching to a pump that has already been running throughout the day, ω equals 30 for the respective timestep of the switch, otherwise 1. If no pumps are running, neither $-e^{(-1/kW_t)}$ nor $e^{1/(-Q_t/kW_t)}$ are considered.

3 Safety through Intrinsically Motivated Imitation Learning

The imitation learning strategy *Safety through Intrinsically Motivated Imitation Learning (SIMIL)* present in this work assumes that offline data is available and online data collection is feasible. The underlying idea is to use the offline dataset distribution as a model to constrain the action selection and enhance sample efficiency while encouraging the policy's convergence to states that lie in high-density regions under the same prior distribution.

The imitation learning strategy works as a follows: given a current state s_t and demonstrations $\mathcal{D}$, select the action a mostly applied in the *k-most* similar states to s_t in $\mathcal{D}$. For that, we make use of k-Nearest Neighbors (k-NN), where the parameter *k* can be chosen such that minimizes the distance $min \sum_{\mathcal{D}} d(\tau, \tau^{\mathcal{D}})$, regarding trajectories $\tau^{\mathcal{D}} \in \mathcal{D}$. The objective is to keep new transitions tied to the previously collected data, mitigating overestimation facing unseen state-action pairs. Finally, a reward bonus $\rho\eta(s_t)$ is added to the immediate reward according to the *Kernel Density Estimation* (KDE) for s_t through Equation 4, being ρ the importance factor for the bonus. Thus, we encourage policy convergence to states with high density under prior dataset distribution. 203

$$\eta(s_t) = \frac{1}{N}\sum_{i=1}^{N} K\left(\frac{s_t - s_i^{\mathcal{D}}}{h}\right). \quad (4)$$

In Equation 4, $K(s_t) \geq 0$ is the kernel that estimates the density for the current state s_t over the states $s^{\mathcal{D}}$ found in the demonstrations. The parameter h is the bandwidth that trade-off the results between balance and variance. In this work, we adopt the *k-NN* based on *Manhattan distance* once it can provide suitable metric for real-values without parameter tuning and KDE with a *gaussian kernel* from Scikit-learn [1]. The Algorithm 1 summarizes the strategy proposed.

Algorithm 1: Safety through Intrinsically Motivated Imitation Learning (SIMIL)

Input: set of Q-Networks with weights θ^Q, set of Target Q'-Networks with weights $\theta^{Q'} \leftarrow \theta^Q$, replay memory $\mathcal{D}'$, demonstrations $\mathcal{D}$, frequency which update target net λ, importance factor ρ;
Output: Policy π

for $t \in \{1, 2, ...\}$ **do**
  Sample state s_t
  Select action a_t using *k-NN*(s_t) in $\mathcal{D}$
  Play (s_t, a_t), observe the reward r_t and the next state s'_t
  Calculate $\eta(s_t)$, sum it to a final reward $r'_t = r_t + \rho\eta(s_t)$
  Store transition (s_t, a_t, r'_t, s'_t) into $\mathcal{D}'$
  $s_t \leftarrow s'_t$
end
for $t \in \{1, 2, ...\}$ **do**
  Sample a mini-batch of n transitions from $\mathcal{D}'$
  Calculate loss $\delta(\theta^Q)$
  Perform a gradient descent step to update θ^Q
  if t ***mod*** $\lambda = 0$ **then**
    Update the set of weights $\theta^{Q'} \leftarrow \theta^Q$
  end
end

4 Results

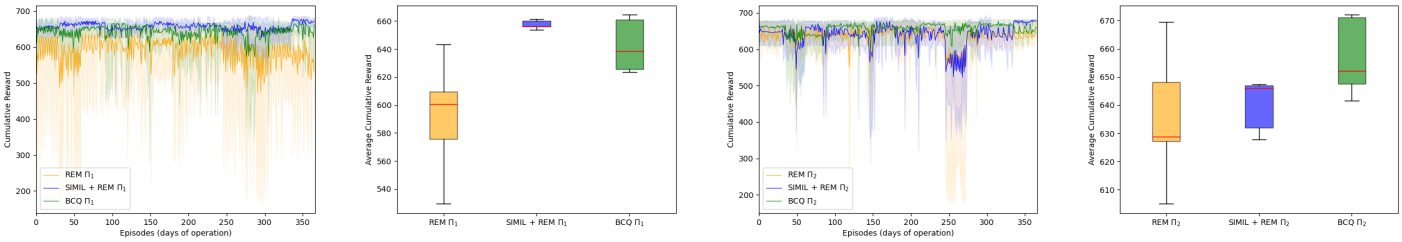
In this work, we aim to evaluate if (1) the proposed imitation learning strategy can generate policies that outperform offline methods baselines; (2) the proposed POMDP can obtain policies that offer a competitive performance relative to that observed in the real world. To this end, we conducted the experiments using the real-world dataset divided into one year for the learning process and one year for the evaluation. Both Offline RL methods and SIMIL use the same amount of data for learning. For accurate comparisons, all samples interact with the simulator for both training and evaluation. This means that the evaluation of the offline dataset is done through interactions with the simulator. We compare the policies BCQ [4, 5], REM [6], and SIMIL + REM using 5 models for each reward function due to the stochasticity in the learning process [13].

To analyze the performance, we call the set of policies obtained using the Equations 2 and 3 by Π_1 and Π_2 respectively. We show in Figure 2 the min, max, and average cumulative reward along with the episodes using the 5 policies obtained. The results show that SIMIL has lower variance and competitive performance relative to cumulative rewards compared to fully-offline policies. The lower peaks in performance are mainly due to not meeting the tank level safety constraints.

Policy	*Electricity Consumption (kW)*
REM Π_1	-1.11 $\pm$ 9.78
SIMIL + REM Π_1	-4.05 $\pm$ 1.97
BCQ Π_1	-3.54 $\pm$ 2.71
REM Π_2	4.08 $\pm$ 7.93
SIMIL + REM Π_2	-3.33 $\pm$ 5.77
BCQ Π_2	-1.40 $\pm$ 3.33

Table 1: Average electricity consumption (%) $\pm$ standard deviation compared to real-world operation.

The three sub-goals: electricity consumption, distribution of pump usage, and tank level are the counterparts of the policy. Thus, a suitable policy performs with lower electricity consumption/higher efficiency, reduces switches, and



(a) (b) (c) (d)

Figure 2: (a) and (c) are respectively the min, max, and average cumulative reward for the set of policies Π_1 and Π_2. (b) and (d) shows the average of the cumulative rewards along with the episodes for Π_1 and Π_2.

Policy	*NOP*	*NP1*	*NP2*	*NP3*	*NP4*
Real-world	30.47	8.30	43.42	8.31	9.50
REM π_1^*	11.38	4.93	0.87	82.82	0.0
SIMIL + REM π_1^*	17.05	0.17	28.54	5.29	48.95
BCQ π_1^*	22.87	17.79	8.13	51.09	0.12
REM π_2^*	32.64	25.85	0.04	41.47	0.0
SIMIL + REM π_2^*	28.08	3.12	36.04	4.89	27.87
BCQ π_2^*	37.11	37.48	0.06	25.35	0.0

Table 2: Action distribution (%)

distributes the pump operation while respecting the tank level constraints. Table 1 compares the electricity consumption for Π regarding real-world operation while Table 2 shows the action distribution for π^*. The results show that SIMIL policies achieve competitive results with real-world operations considering electricity consumption. Finally, generally, the policies presented an operation in the safety range of tank levels.

References

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[3] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[4] Fujimoto, Scott, David Meger, and Doina Precup. "Off-policy deep reinforcement learning without exploration." International Conference on Machine Learning. PMLR, 2019.

[5] Fujimoto, Scott, et al. "Benchmarking batch deep reinforcement learning algorithms." arXiv preprint arXiv:1910.01708 (2019).

[6] Agarwal, Rishabh, Dale Schuurmans, and Mohammad Norouzi. "An optimistic perspective on offline reinforcement learning." International Conference on Machine Learning. PMLR, 2020.

[7] Chentanez, Nuttapong, Andrew Barto, and Satinder Singh. "Intrinsically motivated reinforcement learning." Advances in neural information processing systems 17 (2004).

[8] Costa, Luis Henrique Magalhães, et al. "A branch-and-bound algorithm for optimal pump scheduling in water distribution networks." Water resources management 30.3 (2016): 1037-1052.

[9] Georgescu, Sanda-Carmen, and Andrei-Mugur Georgescu. "Pumping station scheduling for water distribution networks in EPANET." UPB Sci. Bull, Series D 77.2 (2015): 235-246

[10] Folorunso Taliha Abiodun, and Fatimah Sham Ismail. "Pump scheduling optimization model for water supply system using AWGA." 2013 IEEE Symposium on Computers Informatics (ISCI). IEEE, 2013.

[11] Hajgató, Gergely, György Paál, and Bálint Gyires-Tóth. "Deep reinforcement learning for real-time optimization of pumps in water distribution systems." Journal of Water Resources Planning and Management 146.11 (2020).

[12] Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." 2015 aaai fall symposium series. 2015.

[13] Henderson, Peter, et al. "Deep reinforcement learning that matters." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

Modularity benefits reinforcement learning agents with competing homeostatic drives

Zack Dulberg
Princeton University
Princeton, NJ 08544
zdulberg@princeton.edu

Rachit Dubey
Princeton University
Princeton, NJ 08544
rdubey@princeton.edu

Isabel M. Berwian
Princeton University
Princeton, NJ 08544
iberwian@princeton.edu

Jonathan D. Cohen
Princeton University
Princeton, NJ 08544
zdulberg@princeton.edu

Abstract

The problem of balancing conflicting needs is fundamental to intelligence. Standard reinforcement learning algorithms maximize a scalar reward, which requires combining different objective-specific rewards into a single number. Alternatively, different objectives could also be combined at the level of *action* value, such that specialist modules responsible for different objectives submit different action suggestions to a decision process, each based on rewards that are independent of one another. In this work, we explore the potential benefits of this alternative strategy. We investigate a biologically relevant multi-objective problem, the continual homeostasis of a set of variables, and compare a monolithic deep Q-network to a modular network with a dedicated Q-learner for each variable. We find that the modular agent: a) requires minimal exogenously determined exploration; b) has improved sample efficiency; and c) is more robust to out-of-domain perturbation.

Keywords: modular reinforcement learning, homeostasis, conflict, multi-objective decision-making, exploration

Acknowledgements

This project / publication was made possible through the support of a grant from the John Templeton Foundation.

1 Introduction

Humans (and other animals) must satisfy a large set of distinct and possibly conflicting objectives. For example, we must find food, water, shelter, socialize, maintain our temperature, reproduce, etc.. Artificial agents that need to function autonomously in natural environments may face similar problems (e.g., balancing the need to accomplish a specified goal with the need to recharge, etc.). Finding a way to balance disparate needs is thus an important challenge for intelligent agents, and can often be a source of psychological conflict in humans.

In standard reinforcement learning (RL), monolithic agents act in order to maximize a single future discounted reward [1]. The standard way to generalize this to problems with multiple objectives is scalarization. For example, in homeostatically-regulated reinforcement learning (HRRL), an agent is modelled as having separable homeostatic drives, and is rewarded based on its ability to maintain all its "homeostats" at their set points. This is done by *combining* deviations from *all* set-points into a single reward which, when maximized, minimizes homeostatic deviations overall [2].

This approach faces several challenges typical to RL. First, to avoid settling on a sub-optimal policy, an agent must trade-off exploitation of knowledge about its primary objective with some form of exogenous exploration (typically by acting randomly or according to an exploration-specific bonus). Second, sample inefficiency follows from the "curse of dimensionality": as environmental complexity increases, an agent must learn how exponentially more states relate to its objective. Third, RL agents tend to over-fit their environment, performing poorly out-of-domain (i.e. they are not robust to distribution shifts). In the broader context of balancing multiple objectives, reward scalarization might be undesirable if the relative importance of different objectives is unknown or variable [3]. Finally, is not clear that the brain itself uses a common currency to navigate such trade-offs [4].

What is the alternative? Given that objectives may conflict with each other due to environmental constraints, and that agents only have one body with which to act, conflict must be resolved at some point between affordance and action. The monolithic solution resolves conflict at the level of reward (i.e. close to affordance). We suggest resolution could occur later; a set of modules with separate reward functions could submit action values to a decision process that selects a final action [5, 6]. This approach has the potential to address the three aforementioned challenges. Exploration might emerge naturally as a property of the system rather than having to be imposed or regulated as a separate factor, as specialist modules are "dragged along" by other modules when those have the "upper hand" on action. Modules might also have smaller sub-sets of relevant features to learn about, improving sample efficiency, and be less sensitive to distribution shifts in irrelevant features, improving robustness.

Here, we report simulations that provide evidence for benefits of such a modular approach with respect to exploration, sample efficiency, and robustness using deep RL in the context of homeostatic objectives. We construct a simple but flexible environment of homeostatic tasks and construct a deep RL implementation of the HRRL reward function. We then use this framework to quantify differences between monolithic and modular deep Q-agents, finding that modular agents seem to explore well on their own, achieve homeostasis faster, and better maintain it after distribution shift. Together, these results highlight the potential learning benefits of modular RL, while at the same time offering a framework through which psychological conflict and resolution might be better understood.

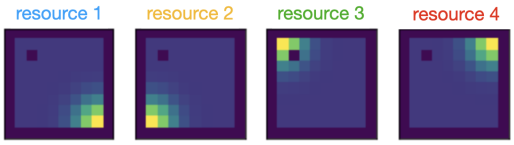
2 Methods

2.1 Environment

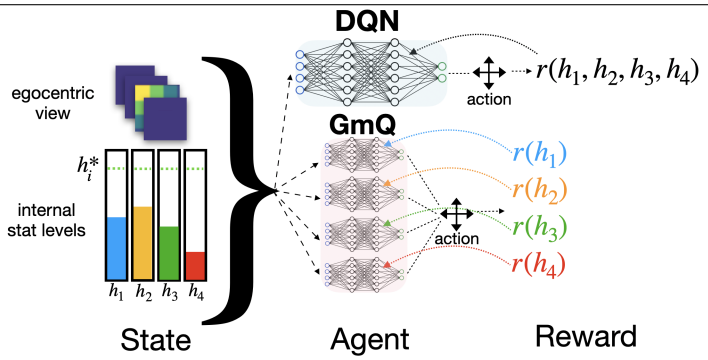
To study conflicting needs, we constructed a toy grid-world environment containing multiple different resources. Specifically, each location (x, y) in the environment contained a vector of resources of length N (i.e., there were N overlaid resource maps). The spatial distribution of each individual resource was specified by a normalized 2D Gaussian with mean μ_x, μ_y and co-variance matrix Σ (see also Figure 1a).

The agent received as perceptual input a 3x3 egocentric slice of the N resource maps (i.e. it could see all the resource levels at each position in its local vicinity). In addition to the resource landscape, the agent also perceived a vector $H_t = (h_{1,t}, h_{2,t}, ..., h_{N,t})$ consisting of N internal variables with each representing the agent's homeostatic need with respect to the resource. We refer to these variables as "internal stats" or just "stats" (such as osmostat, glucostat, etc.) which we assume are independent (h_i is only affected by acquisition of resource i) and have some desired set-point h_i^* (see Figure 1b). Set-points were fixed at $H^* = (h_1^*, h_2^*, ..., h_N^*)$ and did not change over the course of training.

The agent could move in each of four cardinal directions and, with each step, the individual stats, h_i, increased by the amount of resource i at the agent's next location. Additionally, each internal stat decayed at a constant rate to represent the natural depletion of internal resources over time (note: resources in the environment themselves did not deplete). Thus, if the agent discovered a location with a high level of resource for a single depleted stat, staying at that location would optimize that stat toward its set-point, however others would progressively deplete. Agents were initialized in the center of the grid, with internal stats below their set-points, and were trained for $30,000$ steps for a single episode (i.e. agents had to learn in real time as internal stats depleted). For all experiments, the internal stats started at the same level and shared the same set-points. Environmental parameters are summarized in Table 1.



(a) Gridworld environment



(b) Overview of monolithic DQN and modular GmQ agents

Figure 1: Environment and model schematics

2.2 Models

Monolithic agent We created a monolithic agent based on the deep Q network (DQN) [7]. The agent's perceptual input was a concatenation of all local resource levels along with all internal stat levels at each time step (we used neural networks as function approximators since stats were continuous variables). It's output was 4 action logits subsequently used for ϵ-greedy action selection. We used the HRRL reward function [2] which defined reward at each time-step r_t as drive reduction, where drive D was a convex function of set-point deviations; see equation (1).

$$r_t = D(H_t) - D(H_{t+1}) \quad \text{where} \quad D(H_t) = \sqrt[m]{\sum_{i=1}^{N} |h_i^* - h_{i,t}|^n} \tag{1}$$

Modular agent We created a modular agent based on greatest-mass Q-learning (GmQ) [5], which consisted of a separate DQN for each of the 4 resources/stats. Here, each module had the same input as the monolithic model (i.e. the full egocentric view and all 4 stat levels), but received a separate reward $r_{i,t}$ derived from only a single stat. The reward function for the ith module was therefore defined as in equation (2), where drive D depended on the ith resource only.

$$r_{i,t} = D(h_{i,t}) - D(h_{i,t+1}) \quad \text{where} \quad D(h_{i,t}) = \sqrt[m]{|h_i^* - h_{i,t}|^n} \tag{2}$$

To select a single action from the suggestions of the multiple modules, we used a simple additive heuristic. We first summed Q-values for each action across modules, and then performed standard ϵ-greedy action selection on the result. More specifically, if $Q_i(a)$ was the Q-value of action a suggested by module i, greedy actions were selected as $\arg\max_a \sum_i Q_i(a)$.

Common features Schematics for both models are shown in Figure 1b. All Q-networks were multi-layered perceptrons (MLP) with rectified linear nonlinearities trained using a standard temporal difference loss function with experience replay and target networks [7]. The Adam optimizer was used to perform one gradient update on each step in the environment. For both models, ϵ was annealed linearly from its initial to final value at the beginning of training at a rate that was experimentally manipulated as described below. Hyperparameters are summarized in Table 1.

Table 1: Parameter settings for environment and models

Model	DQN	GmQ	Environment	
Trainable parameters	1.09e6	1.09e6	# of resources N	4
MLP hidden layer units	1024	500	HRRL exponents (n, m)	$(4, 2)$
Learning rate	1e-3	1e-3	Stat set-points H^*	$(5, 5, 5, 5)$
Discount factor γ	0.5	0.5	Initial stat levels $H_{t=0}$	$(0.5, 0.5, 0.5, 0.5)$
Memory buffer capacity	30k	30k	Resource locations μ_x, μ_y	{0,10},{0,10}
Target network update frequency	200	200	Resource covariance Σ	$\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$
Batch size	512	512	Stat depletion per step	0.004
Initial ϵ	1	1		
Final ϵ	0.01	0.01		

3 Results

3.1 Optimizing monolithic DQN for homeostasis

We first characterized whether a standard DQN could reliably perform the task of homeostasis in our environment. Figure 2a summarizes the mean internal stat levels of 10 models averaged over all 4 stats and over the final 1k steps of training for different desired set-points. The model reliably achieved each set-point by the end of training, indicated by points tracking the identity line. The slight under-shooting (i.e. points slightly below the diagonal line) may reflect a bias from stats being initialized far below their set-points.

We then fixed the set-points $h_i^* = 5$ for all stats, and optimized the performance of the DQN baseline by performing a search over performance-relevant hyper-parameters, such as the discount factor γ. To quantify performance, we calculated the average homeostatic deviation per step Δ after the exploration annealing phase (using $t_1 = 15k$ and $t_2 = 30k$) as in equation (3). Lower Δ indicates better homeostatic performance.

$$\Delta = \frac{\sum_{t=t_1}^{t_2} \sum_i |h_i^* - h_{i,t}|}{t_2 - t_1} \tag{3}$$

Baseline DQN performance over a range of discount factors γ is shown in Figure 2b. We selected the best performing setting of $\gamma = 0.5$ and matched this and other parameters (see Table 1) between DQN and GmQ to compare them in the following two head-to-head experiments.

3.2 Modularity provides an exploration benefit

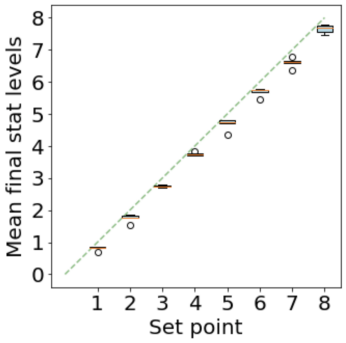
To investigate the impact of modularity on the need for exploration, we systematically varied the number of steps used to anneal ϵ in ϵ-greedy exploration from its initial to final value. We varied the ϵ annealing time for both models from 1 (i.e. minimal exploration of $\epsilon = 0.01$ only) to 10k (i.e. annealing from $\epsilon = 1$ to $\epsilon = 0.01$ over 10k steps).

Figure 3a shows the results of varying the amount of exploration annealing for both models. While DQN gains incremental performance benefits from increasing periods of initial exploration, GmQ displays a striking indifference to the exploration period; with only 1 step of annealing, it performs as well or better than the best DQN models. In other words, DQN requires careful tuning of an appropriate exploration annealing period, but GmQ achieves good performance with effectively no exogenously specified exploration.

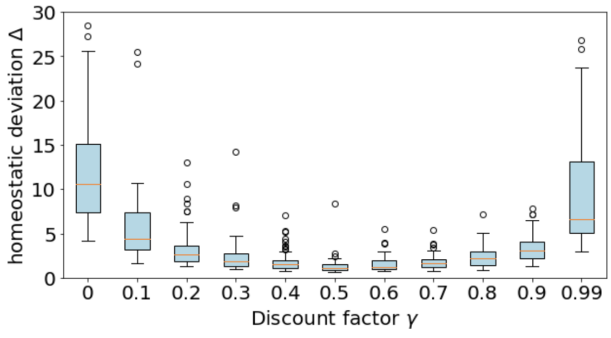
3.3 Modularity provides robustness in the face of perturbation

Finally, we tested how robust each model was to a perturbation out-of-domain that occurred halfway through training, i.e. at time-step 15k. At that point, a single internal stat variable (i.e. h_4) was clamped to a value of 20 (a value previously unseen by the network), and did not change (thus contributing no drive reduction and therefore 0 reward). We tested how well homeostasis was maintained for remaining stats after this perturbation.

Figure 3b shows the time-course of the four stats from the beginning of training, through a perturbation at time-step 15k, using 5000 ϵ-annealing steps. First, it can be seen that GmQ achieves stable homeostasis first, whereas DQN over-shoots set-points initially and takes longer to stabilize. Second, when stat 4 is clamped, DQN displays a significant disturbance to homeostasis of remaining stats, without clear recovery, whereas GmQ is robust in the face of the perturbation.

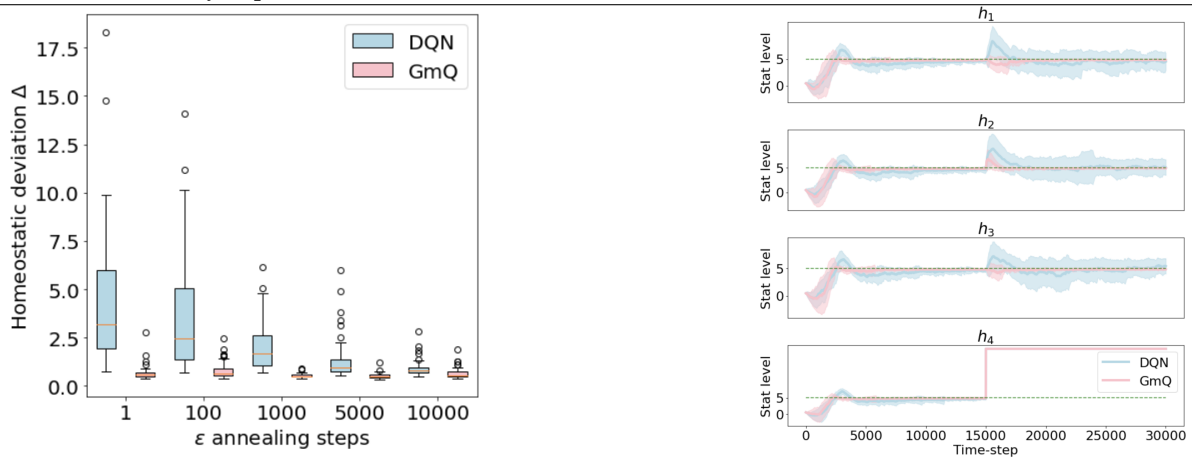


(a) Homeostasis is achieved for a range of set-points ($n = 10$)



(b) Performance over a range of discount factors ($n = 50$)

Figure 2: DQN baseline learnability and performance; Boxplots display inter-quartile range and outliers for n models



(a) Varying exploration period ($n = 50$)

(b) Stat time-courses ($n = 50$) perturbed at $t = 15k$ by clamping h_4. Green line shows set-point. Shading reflects s.d. accross models.

Figure 3: Experiments comparing DQN and GmQ with respect to (a) exploration and (b) perturbation

4 Discussion

We have shown that in a grid-world task with competing homeostatic drives, a simple modular agent based on greatest-mass Q-learning (GmQ) requires less hand-coded exploration, learns faster, and is more robust to environmental perturbations compared to a traditional monolithic deep Q-network (DQN). Our findings in the context of competing drives complement work showing mixture of expert systems display improved sample efficiency and generalization [8]. We also believe the exploration benefits we observed are novel. The problem of exploration in RL is fundamental, and existing solutions make use of noise, explicit exploratory drives/bonuses, and/or other forms of auto-annealing [9, 10]. We suggest an additional class of strategies, namely, exploration as an added benefit of having multiple independent drives, since exploitation from the perspective of one module is exploration from the perspective of another. We hypothesize that the ability of modules to suggest conflicting actions may provide modular agents with an implicit source of exploration.

Future Work Our toy environment has highlighted some initial benefits of modularity (exploration, sample efficiency and robustness), and we predict that these advantages will be amplified in more complex environments, or as the number of drives/objectives increases, due to the curse of dimensionality (more states to explore, learn about, or perturb). We aim to test our agents in rich 3D environments with homeostatic objectives. Next, modular drives immediately pose the problem of coordination. While we simply summed Q-values, more complex arbitrators (such as an additional RL agent that dynamically re-weights individual drives) might better exploit the benefits of modularity in the context of multiple objectives. Finally, humans experience psychological conflict, with various resolution mechanisms long described by psychodynamic theories [11]. Modular RL, with its implicit conflicts and resolutions, could, for the first time, offer a formal, computationally-explicit, and normative explanatory framework that could undergird and/or replace elements of psychodynamic theory for understanding the mechanisms responsible for conflict and resolution in the human brain.

References

[1] Richard S Sutton and Andrew G Barto. "Reinforcement learning: An introduction". In: (2018).

[2] Mehdi Keramati and Boris Gutkin. "Homeostatic reinforcement learning for integrating reward collection and physiological stability". In: *Elife* 3 (2014), e04811.

[3] Diederik M Roijers et al. "A survey of multi-objective sequential decision-making". In: *Journal of Artificial Intelligence Research* 48 (2013), pp. 67–113.

[4] Benjamin Y Hayden and Yael Niv. "The case against economic values in the orbitofrontal cortex (or anywhere else in the brain)." In: *Behavioral Neuroscience* 135.2 (2021), p. 192.

[5] Stuart J Russell and Andrew Zimdars. "Q-decomposition for reinforcement learning agents". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 656–663.

[6] Harm Van Seijen et al. "Hybrid reward architecture for reinforcement learning". In: *Advances in Neural Information Processing Systems* 30 (2017).

[7] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[8] Robert A Jacobs et al. "Adaptive mixtures of local experts". In: *Neural computation* 3.1 (1991), pp. 79–87.

[9] Tianpei Yang et al. "Exploration in deep reinforcement learning: a comprehensive survey". In: *arXiv:2109.06668* (2021).

[10] Samuel M McClure, Mark S Gilzenrat, and Jonathan D Cohen. "An exploration-exploitation model based on norepinepherine and dopamine activity". In: *Advances in neural information processing systems* 18 (2005).

[11] Anna Freud. *The ego and the mechanisms of defence*. Routledge, 2018.

SAAC: Safe Reinforcement Learning as an Adversarial Game of Actor-Critics

Yannis Flet-Berliac*
Stanford University
Stanford, CA, USA
yfletberliac@stanford.edu

Debabrota Basu
Équipe Scool, Inria Lille- Nord Europe
Université de Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL
F-59000 Lille, France
debabrota.basu@inria.fr

Abstract

Although Reinforcement Learning (RL) is effective for sequential decision-making problems under uncertainty, it still fails to thrive in real-world systems where *risk* or *safety* is a binding constraint. In this paper, we formulate the RL problem with safety constraints as a non-zero-sum game. While deployed with maximum entropy RL, this formulation leads to a safe adversarially guided soft actor-critic framework, called SAAC. In SAAC, the adversary aims to break the safety constraint while the RL agent aims to maximize the constrained value function given the adversary's policy. The safety constraint on the agent's value function manifests only as a repulsion term between the agent's and the adversary's policies. Unlike previous approaches, SAAC can address different safety criteria such as safe exploration, mean-variance risk sensitivity, and CVaR-like coherent risk sensitivity. We illustrate the design of the adversary for these constraints. Then, in each of these variations, we show the agent differentiates itself from the adversary's unsafe actions in addition to learning to solve the task. Finally, for challenging continuous control tasks, we demonstrate that SAAC achieves faster convergence, better efficiency, and fewer failures to satisfy the safety constraints than risk-averse distributional RL and risk-neutral soft actor-critic algorithms.

Keywords: Safe reinforcement learning, Maximum entropy RL, Constrained Markov decision processes, Soft actor-critic

Acknowledgements

...

*This work was done during Yannis's PhD at Scool, Inria Lille.

1 Introduction

Designing a Reinforcement Learning (RL) algorithm requires both efficient quantification of uncertainty regarding the incomplete information and the probabilistic decision making policy, and effective design of a policy that can leverage these quantifications to achieve optimal performance. Instead of recent success of RL in structured games and simulated environments, real-world deployment of RL in industrial processes, unmanned vehicles, robotics etc., does not only require efficiency in terms of performance but also being sensitive to risks involved in decisions [2]. This has propelled works quantifying risks in RL and designing safe (or robust, or risk-sensitive) RL algorithms [10, 4].

RL Formalization: Markov Decision Process (MDP). We consider the RL problems that can be modelled as a *Markov Decision Process (MDP)*. An MDP is defined as a tuple $\mathcal{M} \triangleq (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$. $\mathcal{S} \subseteq \mathbb{R}^d$ is the *state space*. $\mathcal{A}$ is the admissible *action space*. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function* that quantifies the goodness or badness of a state-action pair (s, a). $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the *transition kernel* that dictates the probability to go to a next state given the present state and action. Here, $\gamma \in (0, 1]$ is the *discount factor* that quantifies the effect of the reward at present step to the next one. The goal of the agent is to compute a *policy* $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ that maximizes the expected value of cumulative rewards obtained by a time horizon $T \in \mathbb{N}$. For a given policy π, the *value function* or the expected value of discounted cumulative rewards is $V_\pi(s) \triangleq \mathop{\mathbb{E}}_{s_t \sim \mathcal{T}(s_{t-1}, a_{t-1}), a_t \sim \pi(s_t)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) | s_0 = s \right] \triangleq \mathbb{E}_{\pi \mathcal{M}}[Z_\pi^T(s)]$. Here, $Z_\pi^T(s)$ is the *return* of policy π.

Safe RL: Safe Exploration and Risk Measures. In safe RL, risk-sensitivity or safety is embedded mainly using two approaches. The first approach is constraining the RL algorithm to converge in a restricted, 'safe' region of the state space [5, 10]. Here, the 'safe' region is the part of the state space that obeys some external risk-based constraints, such as the non-slippery part of the floor for a walker. RL algorithms developed using this approach either try to construct policies that generate trajectories which stay in this safe region with high probability [5], or to start with a conservative 'safe' policy and then to incrementally estimate the maximal safe region [1]. Due to existence of these 'error' or 'unsafe' states, even a policy with low variance can produce large risks (e.g. falls or accidents) [10].

The other approach is to define a risk-measure on the return $Z_\pi^T(s)$ of a policy π, and then to minimize the corresponding total risk [7, 9, 3]. A risk-measure is a statistics computed on the cumulative return and it quantifies either the spread of the return distribution around its mean value or the heaviness of this distribution's tails. Example of such risk measures are conditional value-at-risk (CVaR) [3], exponential utility [7], variance [9], etc. At tandem to investigating the risk quantifiers, researchers aimed to make the safe RL algorithms scalable [3] and to extend to the continuous MDPs [10]. Our approach is flexible to consider all these risk measures and both discrete and continuous MDP settings.

Our Contributions. In this paper, we unify both of these approaches as a constrained RL problem, and further derive an equivalent non-zero sum (NZS) stochastic game formulation [11] of it. In our NZS game formulation, *risk-sensitive RL reduces to a game between an agent and an adversary* (Sec. 2). The adversary tries to break the *safety constraints*, i.e. either to move out of the 'safe' region or to increase the risk measures corresponding to a given policy. In contrast, the agent tries to construct a policy that maximizes its expected long-term return given the adversarial feedback, which is a statistics computed on adversary's constraint breaking. Given this formulation, we propose a generic actor-critic framework where any two compatible actor-critic RL algorithms are employed to enact as the agent and the adversary to ensure risk-sensitive performance (Sec. 3). In order to instantiate our approach, we propose a specific algorithm, *Safe Adversarially guided Actor-Critic* (`SAAC`), that deploys two Soft Actor-Critics (SAC) [6] as the agent and the adversary. We further derive the policy gradients for the SACs corresponding to the agent and the adversary, which shows that the risk-sensitivity of the agent is ensured by a term repulsing it from the adversary in the policy space. Interestingly, this term can also be used to seek risk and explore more. In Sec. 4, we experimentally verify the risk-sensitive performance of `SAAC` under safe region, CVaR, and variance constraints for continuous control tasks from real-world RL suite [2]. We show that `SAAC` is not only risk-sensitive but it outperforms the state-of-the-art risk-sensitive RL and distributional RL algorithms.

2 Safe RL as a Non-Zero Sum Game

Safe RL as Constrained MDP (CMDP). Both the safe exploration and risk-measure based approaches can be expressed as a CMDP that aims to maximize the value function V_π of a policy π while constraining the total risk ρ_π below a threshold δ:

$$\arg\max_{\pi} V_\pi(s) \text{ s.t. } \rho_\pi(s) \leq \delta \text{ for } \delta > 0. \tag{1}$$

If Mean-Standard Deviation (MSD) [9] is the risk measure[1], $\rho_\pi(s) \triangleq \mathbb{E}\left[Z_\pi^T(s) | \pi, s_0 = s\right] + \lambda \sqrt{\mathbb{V}\left[Z_\pi^T(s) | \pi, s_0 = s\right]}$ ($\lambda < 0$). If CVaR is the risk measure, $\rho_\pi(s) \triangleq \text{CVaR}_\lambda\left[Z_\pi^T(s) | \pi, s_0 = s\right]$ for $\lambda \in [0, 1)$. For the constraint of staying in the 'safe' or 'non-error' states $\mathcal{S} \setminus \mathcal{E}$, $\rho_\pi(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{T} \mathbb{1}(s_{t+1} \in \mathcal{E}) | \pi, s_0 = s \in \mathcal{S} \setminus \mathcal{E}\right] = \sum_{t=0}^{T} \mathbb{P}_\pi[s_{t+1} \in \mathcal{E}]$ such that $s_0 = s$ is a non-error state and $\mathcal{E}$ is the set of 'unsafe' or 'error' states. We refer to this as *subspace risk* $\text{Risk}(\mathcal{E}, \mathcal{S})$ for $\mathcal{E} \subseteq \mathcal{S}$.

CMDP as a Non-Zero Sum (NZS) Game. We address the constraint optimization in Eq. (1) by formulating its Lagrangian.

$$\mathcal{L}(\pi, \beta) \triangleq V_\pi(s) - \beta_0 \rho_\pi(s), \text{ for } \beta_0 \geq 0. \tag{2}$$

[1] Variance is not a coherent risk but standard deviation is. Thus we choose to use Mean-Standard Deviation than Mean-Variance.

For $\beta_0 = 0$, this reduces to its risk-neutral counterpart. Instead, as $\beta_0 \to \infty$, this reduces to the unconstrained risk-sensitive approach. Thus, the choice of β_0 is important. We automatically tune this trade-off in our proposed method. Now, the important question is to estimate the risk function $\rho_\pi(s)$. Researchers have either solved an explicit optimization problem to estimate the parameter or subspace corresponding to the risk measure, or used a stochastic estimator of the risk gradients. These approaches are poorly scalable and lead to high variance estimates as there is no provably convergent CVaR estimator in RL settings. In order to circumvent these issues, we aim to sequentially learn and then adapt to these constraints. Specifically, we deploy *an adversary* that aims to maximize the cumulative risk $\rho_\pi(s)$ given the same initial state s and trajectory τ as *the agent* maximizing Eq. (2) and use it as a proxy for the risk constraint.

$$\theta^* \triangleq \arg\max_\theta \mathcal{L}(\theta, \beta) = V_{\pi_\theta}(s) - \beta_0 V_{\pi_\omega}(s), \qquad \omega^* \triangleq \arg\max_\omega V_{\pi_\omega}(s). \tag{3}$$

Here, we consider that the policies of the agent and the adversary are parameterized by θ and ω respectively. The value function of the adversary $V_{\pi_\omega}(s, \cdot)$ is designed to estimate the corresponding risk $\rho_\pi(s)$. This is a non-zero sum game (NZS) as the objectives of the adversary and the agent are not the same and does not sum up to 0. Following this formulation, any safe RL problem expressed as a CMDP (Eq. (1)), can be reduced to a corresponding agent-adversary non-zero sum game (Eq. (3)). The adversary tries to maximize the risk, and thus to shrink the feasibility region of the agent's value function. The agent tries to maximize the regularized Lagrangian objective in the shrinked feasibility region. We refer to this duelling game as *Risk-sensitive Non-zero Sum (RNS)* game.

3 SAAC: Safe Adversarial Soft Actor-Critics

Background: Maximum-Entropy RL. In this paper, we adopt the Maximum-Entropy RL (MaxEnt RL) framework [4], also known as entropy-regularized RL. In MaxEnt RL, we aim to maximize the sum of value function and the conditional action entropy, $\mathcal{H}_\pi(a|s)$, i.e. for a policy π: $\arg\max_\pi \quad V_\pi(s) + \mathcal{H}_\pi(a|s) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{T}(s_{t-1}, a_{t-1}), a_t \sim \pi(s_t)} \left[Z_\pi^T(s) - \log \pi(a_t|s_t) \mid s_0 = s \right].$

Unlike the classical value function maximizing RL that always has a deterministic policy as a solution, MaxEnt RL tries to learn stochastic policies such that states with multiple near-optimal actions has higher entropy and states with single optimal action has lower entropy. Solving MaxEnt RL is equivalent to computing a policy π that has minimum KL-divergence from a target trajectory distribution $\mathcal{T} \circ \mathcal{R}$:

$$\arg\max_\pi V_\pi(s) + \mathcal{H}_\pi(a|s) = \arg\min_\pi D_{\mathrm{KL}} \left(\pi(\tau) \parallel \mathcal{T} \circ \mathcal{R}(\tau) \right). \tag{4}$$

Here, τ is a trajectory $\{(s_0, a_0), \ldots, (s_T, a_T)\}$. Target distribution $\mathcal{T} \circ \mathcal{R}$ is a softmax or Boltzmann distribution on the cumulative rewards given the trajectory: $\mathcal{T} \circ \mathcal{R}(\tau) \propto p_0(s) \prod_{t=0}^T \mathcal{T}(s_{t+1}|s_t, a_t) \exp[Z_\pi^T(s)]$. Policy distribution is the distribution of generating trajectory τ given the policy π and MDP $\mathcal{M}$: $\pi(\tau) \propto p_0(s) \prod_{t=0}^T \mathcal{T}(s_{t+1}|s_t, a_t) \pi(a_t|s_t)$. Thus in MaxEnt RL, the optimal policy is a softmax or Boltzmann distribution over the expected future return of state-action pairs.

This perspective of MaxEnt RL allows us to design `SAAC` which transforms the robust RL into an adversarial game in the softmax policy space. MaxEnt RL is widely used in solving complex RL problems as: it enhances exploration [6], it transforms the optimal control problem in RL into a probabilistic inference problem [12], and it modifies the optimization problem by smoothing the value function landscape.

Risk-sensitive Non-zero Sum (RNS) Game with MaxEnt RL. In order to perform the RNS game with MaxEnt RL, we substitute the Q-values in Eq. (3) with corresponding soft Q-values. Thus, the adversary's objective is maximizing:

$$\omega^* = \arg\max_\omega \mathbb{E}_{\pi_\omega}[Q_\omega(s, \cdot)] + \alpha_0 \mathcal{H}_{\pi_\omega}(\pi_\omega(.|s)) = \arg\min_\omega D_{\mathrm{KL}} \left(\pi_\omega(.|s) \parallel \exp\left(\alpha_0^{-1} Q_\omega(s, \cdot)\right) / Z_\omega(s) \right). \tag{5}$$

for $\pi_\omega \in \Pi_\omega$, and the agent's objective is maximizing:

$$\theta^* = \arg\max_\theta \quad \mathbb{E}_{\pi_\theta}[Q_\theta(s, \cdot)] + \alpha_0 \mathcal{H}_{\pi_\theta}(\pi_\theta(.|s)) - \beta_0 (\mathbb{E}_{\pi_\theta}[Q_\omega(s, \cdot)] + \alpha_0 \mathcal{H}_{\pi_\omega}(\pi_\omega(.|s))) \tag{6}$$

$$= \arg\min_\theta D_{\mathrm{KL}} \left(\pi_\theta(.|s) \parallel \exp\left(\alpha^{-1} Q_\theta(s, \cdot)\right) / Z_\theta(s) \right) - \beta D_{\mathrm{KL}} \left(\pi_\theta(\cdot|s) \parallel \pi_{\omega^*}(\cdot|s) \right). \tag{7}$$

for $\pi_\theta \in \Pi_\theta$. Here, $\alpha = \alpha_0(1 + \beta_0)$ and $\beta = \alpha_0 \beta_0$. The last equality holds true as $\pi_{\omega^*}(.|s) = \exp\left(\alpha_0^{-1} Q_{\omega^*}(s, \cdot)\right) / Z_{\omega^*}(s)$ for the adversary's optimal policy π_{ω^*}, and since the optimization is over θ, adding $\ln Z_\omega(s)$ does not make a change.

Additionally, for $\omega \neq \omega^*$, the relaxed objective $-(D_{\mathrm{KL}} \left(\pi_\theta(.|s) \parallel \exp\left(\alpha^{-1} Q_\theta(s, \cdot)\right) / Z_\theta(s) \right) - \beta D_{\mathrm{KL}} \left(\pi_\theta(\cdot|s) \parallel \pi_\omega(\cdot|s) \right))$ is a strict lower bound of the goal of the agent in Eq. (6). Thus, maximizing it is similar to maximizing the lower bound on the actual objective. This is similar to the general EM algorithms for maximising likelihoods. Thus, not only in asymptotics, but at every step optimizing the reduced objective allows to maximize the agent's risk-sensitive soft Q-value.

Following this reduction, we observe that performing the RNS game with MaxEnt RL is equivalent to performing the traditional MaxEnt RL for adversary with a risk-seeking Q-function Q_ω, and a modified MaxEnt RL for the agent that includes the usual soft Q-function and a KL-divergence term repulsing the agent's policy π_θ from the adversary's policy π_ω. This behaviour of RNS game in policy space allows to propose a duelling soft actor-critic algorithm, namely `SAAC`.

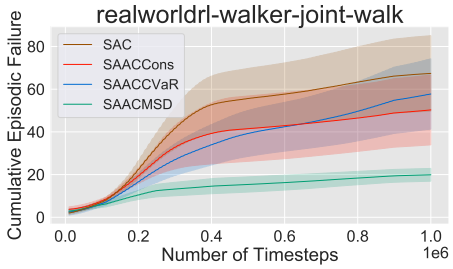


Figure 1: Constraints: SAAC variants.

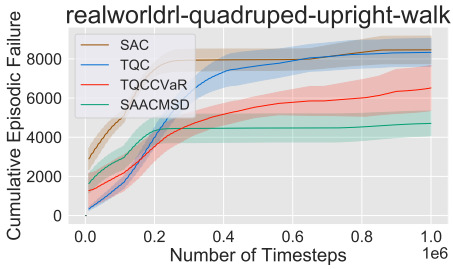


Figure 2: Constraints: SAAC vs. baselines.

Figure 3: In *walker-joint-walk*.

Method	Efficiency (xSAC)	# Failures $\pm\sigma$
SAC	$\times 1$	65.88 ± 17.25
SAAC-Cons	$\times 1.33$	48.66 ± 15.99
SAAC-CVaR	$\times 2.02$	54.39 ± 15.37
SAAC-MSD	$\times \mathbf{2.21}$	$\mathbf{19.31 \pm 3.02}$

Figure 4: In *quadruped-upright-walk*.

Method	Efficiency (xSAC)	# Failures $\pm\sigma$
SAC	$\times 1$	8443.93 ± 696.47
TQC	$\times 0.97$	8297.63 ± 697.88
TQC-CVaR	$\times 1.03$	6298.33 ± 1078.50
SAAC-MSD	$\times \mathbf{1.19}$	$\mathbf{4632.80 \pm 657.35}$

The SAAC Algorithm. We propose an algorithm SAAC to solve the objectives of the agent (Eq. (6)) and of the adversary (Eq. (5)). In SAAC, we deploy two soft actor-critics (SACs) to enact the agent and the adversary respectively.

As a building block for SAAC, we deploy the recent version of SAC [6] that uses two soft Q-functions to mitigate positive bias in the policy improvement step. In the design of SAAC, we introduce two new ideas: an off-policy deep actor-critic algorithm within the MaxEnt RL framework and a Risk-sensitive Non-zero Sum (RNS) game. SAAC engages the agent in safer strategies while finding the optimal actions to *maximize* the expected returns. The role of the adversary is to find a policy that maximizes the probability of breaking the constraints given by the environment. The adversary is trained online with off-policy data given by the agent. We denote the parameter of the adversary policy using ω[2]. For each sequence of transition from the replay buffer, the adversary should find actions that minimize the following loss:

$$J(\pi_\omega) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_\omega} \left[\alpha \log \left(\pi_\omega \left(a_t | s_t \right) \right) - Q_\psi \left(s_t, a_t \right) \right] \right].$$

Finally, leveraging the RNS based reduced objective, SAAC makes the agent's actor minimize $J(\pi_\theta)$:

$$J(\pi_\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} \Big[\mathbb{E}_{a_t \sim \pi_\theta} \Big[\alpha \log \left(\pi_\theta \left(a_t | s_t \right) \right) - Q_\phi \left(s_t, a_t \right) - \beta \Big(\log \pi_{\theta_{\text{old}}}(a_t|s_t) - \log \pi_{\omega_{\text{old}}}(a_t|s_t) \Big) \Big] \Big].$$

In blue is the repulsion term introduced by SAAC. The method alternates between collecting samples from the environment with the current agent's policy and updating the function approximators, namely the adversary's critic Q_ψ, the adversary's policy π_ω, the agent's critic Q_ϕ and the agent's policy π_θ. It performs stochastic gradient descent on corresponding loss functions with batches sampled from the replay buffer. Now, we provide a few examples of designing the adversary's critic Q_ψ for different safety constraints.

SAAC-Cons: *Subspace Risk.* At every step, the environment signals whether the constraints have been satisfied or not. We construct a reward signal based on this information. This constraint reward, denoted as r_c, is 1 if all the constraints have been broken, and 0 otherwise. $J(Q_\psi)$ is the soft Bellman residual for the critic responsible with constraint satisfaction:

$$J(Q_\psi) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \Big[\frac{1}{2} \Big(Q_\psi \left(s_t, a_t \right) - \big(r_c \left(s_t, a_t \right) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} \mathbb{E}_{a_t \sim \pi_\omega} \left[Q_{\bar{\psi}} \left(s_t, a_t \right) - \alpha \log \pi \left(a_t | s_t \right) \right] \big) \Big)^2 \Big]. \tag{8}$$

SAAC-MSD: *Mean-Standard Deviation (MSD).* In this case, we consider optimizing a Mean-Standard Deviation risk [9], which we estimate using: $Q_\psi(s,a) = Q_\phi(s,a) + \lambda \sqrt{\mathbb{V}[Q_\phi(s,a)]}$. $\lambda < 0$ is a hyperparameter that dictates the lower $\lambda -$ SD considered to represent the lower tail. In the experiments, we use $\lambda = -1$. In practice, we approximate the variance $\mathbb{V}[Q_\phi(s,a)]$ using the state-action pairs in the current batch of samples. We refer to the associated method as SAAC-MSD.

SAAC-CVaR: *CVaR.* Given a state-action pair (s,a), the Q-value distribution is approximated by a set of quantile values at quantile fractions [3]. Let $\{\tau_i\}_{i=0,\dots,N}$ denote a set of quantile fractions, which satisfy $\tau_0 = 0$, $\tau_N = 1$, $\tau_i < \tau_j \, \forall i < j$, $\tau_i \in [0,1] \, \forall i = 0, \dots, N$, and $\hat{\tau}_i = (\tau_i + \tau_{i+1})/2$. If $Z^\pi : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$ denotes the soft action-value of policy π, $Q_\psi(s,a) = -\sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) \, g'(\hat{\tau}_i) \, Z^{\pi_\theta}_{\hat{\tau}_i}(s,a;\phi)$ with $g(\tau) = \min\{\tau/\lambda, 1\}$, where $\lambda \in (0,1)$. In the experiments, we set $\lambda = 0.25$, i.e. we truncate the right tail of the return distribution by dropping 75% of the topmost atoms.

4 Experimental Analysis

Experimental Setup. To validate the proposed framework, we conduct a set of experiments in the real-world RL challenge [2], such as *realworldrl-walker-joint-walk*, and *realworldrl-quadruped-upright-walk*. Note that for all the experiments, the agents are trained for 1M timesteps and their performance is evaluated at every 1000-th step. Similar to [6], the adversary temperature β and the entropy temperature are automatically adjusted.

Comparison between Risk Quantifiers of SAAC. First, we compare the different variants of SAAC allowed by the method's framework in the *realworldrl-walker-joint-walk* task. From Table 3 and Fig. 1 (lines are average performances and shaded areas represent one standard deviation), we evaluate how our method affects the performance and risk aversion of agents.

[2] resp. ω_{old} the parameter at the previous iteration.

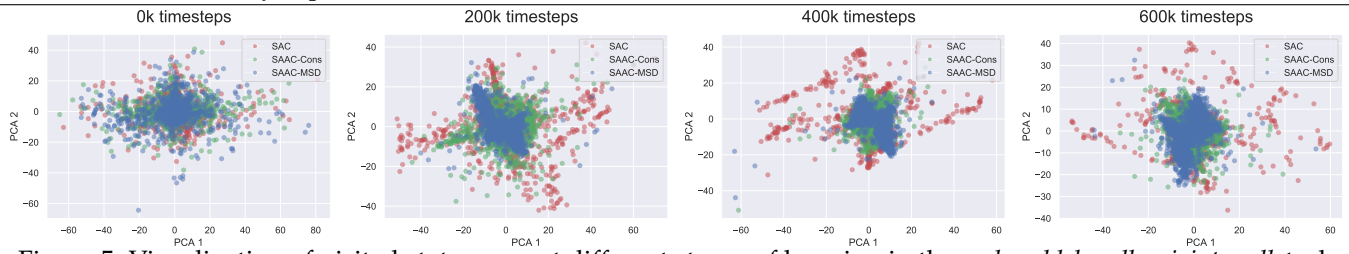


Figure 5: Visualization of visited state space at different stages of learning in the *realworldrl-walker-joint-walk* task.

In addition to the rate at which the maximum average return is reached by each of the methods compared to SAC, we compare the cumulative number of failures of the agents (the lower the better). Risk-sensitive agents such as `SAAC` decrease the probability of breaking safety constraints. Concurrently, they *achieve the maximum average return with higher sample efficiency and* `SAAC-MSD` *ahead*. Henceforth, we use the `SAAC-MSD` version of our method to compare with the baselines.

Comparison of `SAAC` to Baselines. Now, we compare the best performing `SAAC` variant `SAAC-MSD` with SAC [6], TQC [8] and TQC-CVaR, i.e. an extension of TQC with 16% of the topmost atoms dropped (cf. Table 6 in [8, Appendix B]) of all Q-function atoms. TQC-CVaR leverages distributional RL with risk measure to obtain safer policies. In Table 4 and Fig. 2, we evaluate `SAAC-MSD` in *realworldrl-quadruped-upright-walk*. Table 4 confirms the advantage of using `SAAC-MSD` as a risk-averse MaxEnt RL method over the risk-neutral MaxEntRL and risk-averse distributional RL baselines. `SAAC` *allows the agents to achieve faster convergence, using safer policies during training, better efficiency* ($\sim 1.19\times$ *more than SAC), and less number of failures to satisfy the safety constraints* (~ 26 *to* 45% *less)*.

Visualization of Safer State Space Visitation. In this experiment, we choose SAC, `SAAC-Cons` and `SAAC-MSD` to train a relatively wide spectrum of agents using the same experimental protocol as in Sec. 5.2., and on the *realworldrl-walker-joint-walk* task. We collect samples of states visited during the evaluation phase in a test environment at different stages of the training. The state vectors are projected from a 18D space to a 2D space using PCA. We present the results in Fig. 5. At the beginning of training, there is no clear distinction in terms of explored state regions, as the learning has not begun yet. On the contrary, during the 200k-600k timesteps, there is a significant difference in terms of state space visitation. In resonance with the cumulative number of failures shown in Fig. 1, the results suggest that SAC engages in actions leading to more unsafe states. Conversely, `SAAC` *demonstrates to successfully constraint the agents to the safe regions*.

References

[1] Felix Berkenkamp, Riccardo Moriconi, Angela P. Schoellig, and Andreas Krause. Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes. In *IEEE CDC*, pages 4661–4666, 2016.

[2] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

[3] Hannes Eriksson, Debabrota Basu, Mina Alibeigi, and Christos Dimitrakakis. SENTINEL: Taming uncertainty with ensemble-based distributional reinforcement learning. *arXiv preprint arXiv:2102.11075*, 2021.

[4] Benjamin Eysenbach and Sergey Levine. Maximum entropy RL (provably) solves some robust RL problems. *arXiv preprint arXiv:2103.06257*, 2021.

[5] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

[6] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[7] Ronald A Howard and James E Matheson. Risk-sensitive Markov decision processes. *Management science*, 18(7):356–369, 1972.

[8] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *ICML*, pages 5556–5566. PMLR, 2020.

[9] LA Prashanth and Mohammad Ghavamzadeh. Variance-constrained actor-critic algorithms for discounted and average reward MDPs. *Machine Learning*, 105(3):367–417, 2016.

[10] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 2019.

[11] S Sorin. Asymptotic properties of a non-zero sum stochastic game. *International Journal of Game Theory*, 15(2):101–107, 1986.

[12] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

Hierarchies of Reward Machines

Daniel Furelos-Blanco
Department of Computing
Imperial College London
d.furelos-blanco18@imperial.ac.uk

Mark Law
ILASP Limited
mark@ilasp.com

Anders Jonsson
Department of Information and Communication Technologies
Universitat Pompeu Fabra
anders.jonsson@upf.edu

Krysia Broda
Department of Computing
Imperial College London
k.broda@imperial.ac.uk

Alessandra Russo
Department of Computing
Imperial College London
a.russo@imperial.ac.uk

Abstract

Hierarchical reinforcement learning (HRL) algorithms decompose a task into simpler subtasks that can be independently solved. This enables tackling complex long-horizon and/or sparse reward tasks more efficiently. In recent years, several efforts have focused on proposing discrete structures, such as finite-state machines (FSMs), that can be exploited using HRL and learned from an agent's experience. In this paper, we introduce a formalism for hierarchically composing reward machines (RMs). RMs are FSMs where each edge is labeled by (1) a propositional logic formula over a set of high-level events that capture a task's landmark/subgoal, and (2) a reward for satisfying the formula. The structure of an RM is naturally exploited by HRL algorithms by treating each landmark as a subtask and deciding which subtask to pursue from each RM state. A hierarchy of reward machines (HRM) enables the constituent RMs to call each other, potentially defining an arbitrary number of increasingly abstract machines. Our formalism guarantees that an HRM can be converted into an equivalent flat one. We adapt HRL algorithms to HRMs by defining each RM in the hierarchy as a subtask itself. Given a set of tasks with hierarchical structure, we describe a curriculum-based method to induce an HRM for each task in the set. Each HRM is induced from a set of traces of high-level events and a set of callable RMs from lower level tasks. We evaluate our method in two domains with hierarchically composable tasks. We show that encapsulating each task's structure within an HRM makes the learning of a multi-level HRM more efficient than that of a flat HRM since the size of the root machine is potentially much smaller. We also study how efficient it is to use HRMs from lower levels to drive the search for example traces in higher level tasks.

Keywords: Automaton Learning
Hierarchical Reinforcement Learning
Reward Machines

Acknowledgements

The authors would like to thank the anonymous reviewers, Hadeel Al-Negheimish and Alex Spies for their helpful comments and suggestions. Anders Jonsson is partially funded by Spanish grant PID2019-108141GB-I00.

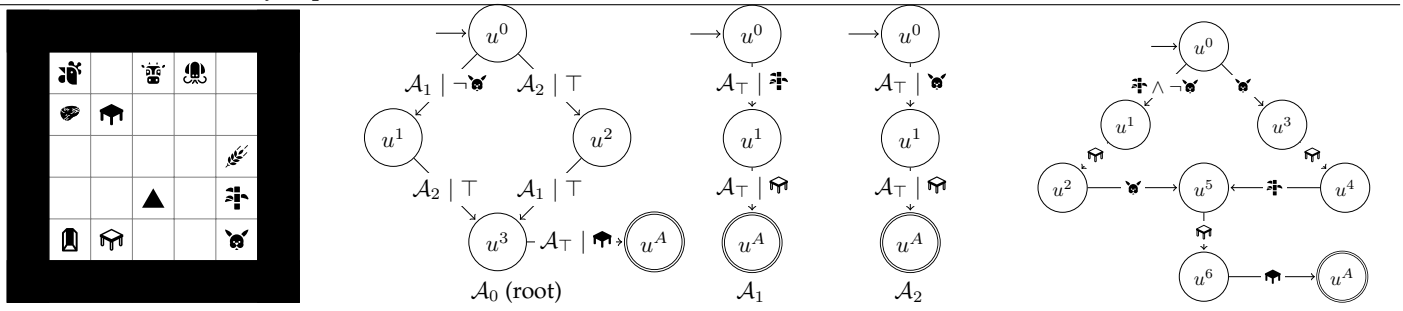


Figure 1: A CRAFTWORLD grid (a), a multi-level HRM for BOOK (b) and an equivalent flat HRM (c).

Task	Level	Description	Task	Level	Description	Task	Level	Description	Task	Level	Description
BATTER	1	(&) ;	PAPER	1	;	MAP	2	(PAPER & COMPASS) ;	CAKE	4	BATTER ; MILKB.SUGAR ;
BUCKET	1	;	QUILL	1	(&);	MILKBUCKET	2	BUCKET ;			
COMPASS	1	(&) ;	SUGAR	1	;	BOOKQUILL	3	BOOK & QUILL			
LEATHER	1	;	BOOK	2	(PAPER & LEATHER) ;	MILKB.SUGAR	3	MILKBUCKET & SUGAR			

Table 1: List of CRAFTWORLD tasks. Descriptions "x ; y" express sequential order (observe/do x then y), and descriptions "x & y" express that x and y can be done in any order.

1 Introduction

Reward machines (RMs) [8] are a recent formalism for tackling sparse reward tasks in partially observable environments by encoding the reward function of a given task. RMs are finite-state machines whose edges are labeled by propositional logic formulas over a set of high-level events and a reward scalar. RMs capture a task's subgoals through these formulas, and thus help to address partial observability by acting as an external memory. These structures are amenable to the use of hierarchical reinforcement learning (HRL) frameworks, such as options [7], by associating an option to each formula in the RM. Recent papers have proposed methods for learning RMs [9, 11, 4] and similar kinds of machines [3, 2]. The primary shortcoming of the RMs considered by previous work is that they cannot be reused within larger RMs, thus the same policies might be learned multiple times unnecessarily. Besides, methods for learning RMs do not usually scale well when the RMs consist of several states. In this work, we propose a formalism for hierarchically composing RMs by allowing calls between them. We introduce a curriculum-based method for inducing these hierarchies given a set of tasks classified into different levels according to their subtasks. Hierarchies of RMs (HRMs) enable reusability and ease the machine induction process since the constituent RMs are smaller. Our method successfully learns and exploits HRMs in environments with hierarchically composable tasks, outperforming the learning of equivalent flat RMs. We empirically show that using previously learned HRMs to explore allows for a more efficient collection of example traces in new tasks.

Preliminaries An episodic partially observable Markov decision process (POMDP) is a tuple $\mathcal{M} = \langle S, S_T, S_G, \Sigma, A, p, r, \gamma, \nu \rangle$, where S, A, p, r and γ are defined as for MDPs, $S_T \subseteq S$ is a set of terminal states, $S_G \subseteq S_T$ is a set of goal states, Σ is a set of observations, and $\nu : S \to \Delta(\Sigma)$ is a mapping from states to probability distributions over observations. The POMDP is enhanced with a set of *propositions* $\mathcal{P}$, and a *labeling function* $\mathcal{L} : \Sigma \to 2^{\mathcal{P}}$ mapping observations into subsets of propositions (or *labels*) $L \subseteq \mathcal{P}$. The aim is to find a policy $\pi : (\Sigma \times A)^* \times \Sigma \to A$, a mapping from histories of observation-action pairs to actions, which maximizes the expected sum of discounted rewards (or *return*), $R_t = \mathbb{E}_\pi[\sum_{k=t}^{n} \gamma^{k-t} r_t]$, where n is the last step of the episode. We assume that the combination of an observation and a history of labels seen during an episode is sufficient to obtain the Markov property, i.e. a policy can be defined as $\pi : (2^{\mathcal{P}})^* \times \Sigma \to A$.

The interaction between the agent and a POMDP environment is as follows. At time t, the state of the environment is $s_t \in S$, and the agent observes a tuple $\boldsymbol{\sigma}_t = \langle \sigma_t^\Sigma, \sigma_t^T, \sigma_t^G \rangle$, where $\sigma_t^\Sigma \sim \nu(\cdot|s_t)$ is an observation, and $\sigma_t^T = \mathbb{I}[s_t \in S_T]$ and $\sigma_t^G = \mathbb{I}[s_t \in S_G]$ indicate whether s_t is a terminal state and a goal state respectively. If the state is non-terminal, the agent executes action $a_t \in A$, the environment transitions to state $s_{t+1} \sim p(\cdot|s_t, a_t)$, and the agent observes a new tuple $\boldsymbol{\sigma}_{t+1}$ and receives reward $r(s_t, a_t, s_{t+1})$. At the end of the episode, we will have a trace $\lambda = \langle \boldsymbol{\sigma}_0, a_0, r_1, \boldsymbol{\sigma}_1, a_1, \ldots, a_{n-1}, r_n, \boldsymbol{\sigma}_n \rangle$, which can be of three types: a goal trace if $\sigma_n^G = \top$, a dead-end trace if $\sigma_n^T = \top \wedge \sigma_n^G = \bot$, and incomplete if $\sigma_n^T = \bot$. A label trace $\lambda_{\mathcal{L},\mathcal{P}}$ can be derived by applying the labeling function $\mathcal{L}$ to each observation $\sigma_{0 \le i \le n}^\Sigma$ in the previous trace.

The options framework [7] addresses temporal abstraction in reinforcement learning. An option is a tuple $\omega = \langle I_\omega, \pi_\omega, \beta_\omega \rangle$, where I_ω and β_ω respectively denote where the option initiates and terminates (e.g., a subset of states), and π_ω is the option's policy describing the behavior of the option between I_ω and β_ω.

2 Contributions

We propose the CraftWorld domain (cf. Figure 1a) to describe our method. The agent (▲) can move forward or rotate 90°, staying put if it moves towards a wall. Grid locations are labeled with propositions from $\mathcal{P} = \{$[icon], [icon], [icon], [icon], [icon], [icon], [icon], [icon], [icon], [icon]$\}$. The agent observes propositions that it steps on (e.g., {[icon]} in the top-left corner). Table 1 lists several tasks that consist of observing a sequence of propositions, where the reward is 1 if the goal is achieved and 0 otherwise. A label L is used as a truth assignment where propositions $p \in \mathcal{P}$ in L are true, else they are false (e.g., {[icon]} satisfies [icon] $\wedge \neg$[icon]).

Formalism A *hierarchy of reward machines (HRM)* is a tuple $\mathcal{H} = \langle \mathcal{A}, \mathcal{A}_r, \mathcal{P}, \delta_\mathcal{H} \rangle$, where $\mathcal{A} = \{\mathcal{A}_0, \ldots, \mathcal{A}_{m-1}\} \cup \{\mathcal{A}_\top\}$ is a set of m RMs and a leaf RM $\mathcal{A}_\top$, $\mathcal{A}_r \in \mathcal{A} \setminus \{\mathcal{A}_\top\}$ is the root RM, $\mathcal{P}$ is a finite set of propositions shared by all RMs, and $\delta_\mathcal{H} : U_\mathcal{H} \times 2^\mathcal{P} \to U_\mathcal{H}$ is a hierarchical transition function. The set $U_\mathcal{H}$ denotes the set of all possible *hierarchy states*, each a tuple $\langle \mathcal{A}_i, u, \Gamma \rangle$ where $\mathcal{A}_i \in \mathcal{A}$ is an RM, u is a state of $\mathcal{A}_i$, and Γ is a call stack. The call stack determines the RMs to which control must be returned once a call is completed. Each *reward machine (RM)* $\mathcal{A}_i \in \mathcal{A}$ is a tuple $\mathcal{A}_i = \langle U_i, \mathcal{P}, \varphi_i, r_i, u_i^0, U_i^A, U_i^R \rangle$, where U_i is a finite set of states, $\mathcal{P}$ is a finite set of propositions, $\varphi_i : U_i \times U_i \times \mathcal{A} \to \text{DNF}_\mathcal{P}$ is the state transition function, $r_i : U_i \times U_i \to \mathbb{R}$ is the reward transition function, $u_i^0 \in U_i$ is the initial state, $U_i^A \subseteq U_i$ is the set of accepting states, and $U_i^R \subseteq U_i$ is the set of rejecting states.[1,2] The *leaf machine* $\mathcal{A}_\top$ has a single state, which is accepting (i.e., $U_\top = U_\top^A = \{u_\top^0\}$). The expression $\varphi_i(u, u', \mathcal{A}_j) = \phi$ indicates that the transition from $u \in U_i$ to $u' \in U_i$ is associated with a call to RM $\mathcal{A}_j$ and DNF formula $\phi \in \text{DNF}_\mathcal{P}$, which must be satisfied to start the call (by default, $\phi = \bot$). Accepting and rejecting states do not have transitions to other states. Figure 1b shows Book's HRM, which consists of a root and two RMs for the subtasks Paper and Leather. An edge from state u to u' of an RM $\mathcal{A}_i$ is of the form $\mathcal{A}_j \mid \varphi_i(u, u', \mathcal{A}_j)$, double circled states are accepting states, and loop transitions are omitted.

The execution of an HRM starts in the root's initial state with an empty call stack. Given a hierarchy state and a label $L \subseteq \mathcal{P}$, the next hierarchy state is determined by the hierarchical transition function $\delta_\mathcal{H}$, which is recursively defined using the transition functions φ_i of the constituent RMs. Given a hierarchy state $\langle \mathcal{A}_i, u, \Gamma \rangle$, $\delta_\mathcal{H}$ covers three cases:

1. If $u \in U_i^A$ is accepting and Γ is non-empty, pop the top element of Γ and return control to the previous RM on the call stack, recursively applying $\delta_\mathcal{H}$ in case several accepting states are reached simultaneously.
2. If L satisfies the formula ϕ of a transition $\varphi_i(u, u', \mathcal{A}_j) = \phi$, as well as formulas from initial states of recursively called RMs, push $\mathcal{A}_j$ onto Γ and recursively apply $\delta_\mathcal{H}$ from its initial state. In Figure 1b, L must satisfy [icon] $\wedge \neg$[icon] to start $\mathcal{A}_1$ from the root's initial state, while it only needs to satisfy [icon] if the call is made from state u^2.
3. If none of the conditions in previous cases hold, the hierarchy state does not change.

The theorem below captures that the behavior of $\delta_\mathcal{H}$ is equivalent to the transition functions of flat RMs in previous works, specifically those using logic formulas to label the edges [2], so an HRM cannot have circular dependencies and must behave deterministically (two state transitions cannot be satisfied at once). We omit the proof due to space constraints.

Theorem 1. *Every HRM $\mathcal{H}$ and associated hierarchical transition function $\delta_\mathcal{H}$ corresponds to an equivalent flat RM.*

Reinforcement Learning An HRM can be exploited using options, similar to flat finite-state machines [8, 2]. Each transition $\varphi_i(u, u', \mathcal{A}_j) = \phi$ is associated with a *formula option* ϕ if $\mathcal{A}_j = \mathcal{A}_\top$, and a *call option* $\langle \mathcal{A}_j, \phi \rangle$ if $\mathcal{A}_j \neq \mathcal{A}_\top$. Both types of options are applicable in RM state u of $\mathcal{A}_i$. A formula option simply attempts to satisfy ϕ, while a call option additionally has to satisfy formulas in recursively called RMs. In Figure 1b, the set of formula options is {[icon], [icon] $\wedge \neg$[icon], [icon], [icon], [icon]}, and option [icon] $\wedge \neg$[icon] leads the agent to observe label {[icon]}. The call options are $\langle \mathcal{A}_1, \neg$[icon]$\rangle$, $\langle \mathcal{A}_1, \top \rangle$ and $\langle \mathcal{A}_2, \top \rangle$. Options $\langle \mathcal{A}_1, \neg$[icon]$\rangle$ and $\langle \mathcal{A}_2, \top \rangle$ are applicable in the root's initial state. In each RM state, a metacontroller chooses an option with the aim of reaching an RM's accepting state as soon as possible. The formula option policies and metacontrollers are trained with Q-learning using a pseudo-reward function tailored to the formula and the RM's reward transition function, respectively.

An *option hierarchy* $\boldsymbol{\omega}$ manages the options currently executing. Initially, $\boldsymbol{\omega}$ is empty. At each step, the agent uses metacontrollers to add options to $\boldsymbol{\omega}$ until a formula option is added. In Figure 1b, the metacontroller in the root's initial state may choose option $\langle \mathcal{A}_2, \top \rangle$ followed by option [icon], resulting in $\boldsymbol{\omega} = [\langle \mathcal{A}_2, \top \rangle,$ [icon]$]$, and actions are then selected according to option [icon]. After each action, formula option policies are updated and the new hierarchy state determines whether any option in $\boldsymbol{\omega}$ has terminated. A formula option terminates if the hierarchy state changes, while a call option terminates if it does not appear in the call stack of the new hierarchy state. Termination is applied bottom-up starting from the formula option, and stopped when an option does not terminate. Finally, the metacontrollers are updated for the terminated options, and call options that appear in the call stack of the new hierarchy state but not in $\boldsymbol{\omega}$ are added to $\boldsymbol{\omega}$ so that the corresponding metacontrollers can be updated later. We remark that the agent may not move through the hierarchy as intended: if the agent observes {[icon]} while pursuing [icon], it moves to RM $\mathcal{A}_1$ and not to $\mathcal{A}_2$ as originally intended.

[1] We assume reward functions are $r_i(u, u') = 1$ if $u \notin U_i^A$ and $u' \in U_i^A$, and 0 otherwise for all $0 \leq i < m$.

[2] These RMs are different from the original ones [8] in that (i) there are calls to other RMs in a hierarchy, (ii) there are explicit accepting and rejecting states, and (iii) transitions are given by propositional logic formulas over $\mathcal{P}$ instead of sets of propositions.

Learning the Hierarchies An HRM is automatically learned for each task in a set of composable tasks following a *curriculum learning* method [6]. Each task is assigned a level depending on its subtasks (e.g., Table 1 shows the levels for CRAFTWORLD tasks), and learning progresses from lower to higher levels. Initially, level 1 tasks are chosen with equal probability while higher level tasks cannot be chosen. When task i terminates, its average return R_i is updated using the last undiscounted return r as $R_i \leftarrow \beta R_i + (1-\beta)r$, where β is a hyperparameter. The probability of choosing task i in the next episode is given by $c_i / \sum_k c_k$, where $c_i = 1 - R_i$ (the maximum return is assumed to be 1). When the minimum average return of tasks in the current level or lower surpasses a threshold Δ, the current level increases by 1.

Learning an HRM is analogous to previous work for flat machines [2]. Given a set of label traces, a set of propositions, a set of callable RMs and a number of RM states, an inductive logic programming system, ILASP, learns a state transition function that correctly recognizes the traces (e.g., goal traces finish in a root's accepting state). The callable RMs include all RMs in lower levels, and each RM has one accepting state and one rejecting state. To ease the induction, label traces are compressed (i.e., consecutive equal labels are merged into a single one), RMs are forced to be acyclic, DNFs consist of a single disjunct, and a symmetry breaking method is applied. The induction of HRMs is *interleaved* with policy learning: a new HRM is learned when an episode's label trace is not correctly recognized by the current HRM (e.g., a goal trace does not finish in the root's accepting state).

The first HRM is learned from a set of κ goal traces collected by exploring, similar to other works [9, 11]. For level 1 tasks, the agent performs a random walk, while in higher levels the agent uses HRMs (i.e., call options) and formulas (i.e., formula options) from lower levels. Enhancing exploration with options allows collecting goal traces faster, especially when labels are sparse. Finally, the κ_s shortest traces are used to learn the HRMs in order to reduce the running time.

3 Evaluation and Discussion

We evaluate our method in two domains. CRAFTWORLD is a modification of MiniGrid [1] that adds new object types (one per proposition) and tasks. The grid is fully observable and can be of three types: an open plan 7×7 grid (OP) as in Figure 1a, an open plan 7×7 grid with a lava location (OPL), and a 13×13 four rooms (FR) [7]. OPL has an extra proposition for lava (🔥), which must always be avoided. WATERWORLD [8] is a 2D box containing 12 balls of 6 colors (2 per color), moving at constant speed in a fixed direction. The agent ball can change its velocity in any cardinal direction. Propositions $\mathcal{P} = \{r, g, b, c, y, m\}$ denote ball colors. The agent observes the color of the balls it overlaps with. The tasks consist in observing specific sequences of colors [8]. Unlike CRAFTWORLD, labels with multiple propositions are observable in WATERWORLD, motivating the use of propositional formulas as an extra level of abstraction. Both domains are partially observable since the agent does not know the accomplished subgoals, which are encoded by the HRM.

Each experiment consists of 10 runs on a set of 10 random instances (e.g., by placing objects randomly in CRAFTWORLD). In CRAFTWORLD, OP and OPL have one object for each proposition, while FR has one or two. All experiments run for 100,000 episodes, each lasting a maximum of 300 steps. The curriculum has parameters $\beta = \Delta = 0.95$ and uses returns from the greedy policies evaluated in each task-instance pair every 100 episodes. ILASP has 2 hours to learn the HRMs for all tasks. We use $\kappa = 25$ for level 1 tasks, $\kappa = 150$ for level 2 tasks onwards, and $\kappa_s = 10$ for all tasks.

We define multiple DQNs at different levels of abstraction [5]. Each formula option and each RM is associated with a DDQN [10]. Metacontrollers provide Q-values for each option in an RM given an observation and an RM state (the output is masked according to the options available in the input RM state). We adopt ϵ-greedy exploration: each formula option and RM state is associated with its own ϵ, which is linearly annealed. For formula options, ϵ decreases after each step performed with that formula, while for metacontrollers ϵ decreases after having finished an option that started in that state. Formula option policies and metacontrollers are trained using different discount factors γ. Each RM has its own experience replay buffer, whereas all formula options share a common buffer (a form of intra-option learning [7]).

Figure 2 shows learning curves for the tasks of each domain, each measuring the undiscounted return obtained by the greedy policy every 100 episodes. The dotted vertical lines correspond to episodes where an HRM is learned. To ease visibility, some curves are not displayed for all training episodes. All tasks generalize across instances, and the curriculum is visible in all domains: when the return for tasks at a level is close to 1, the HRMs and policies in the next level start to be learned. In CRAFTWORLD, learning in OP is easier than in OPL and FR. OPL is harder than OP because (1) dead-ends hinder observing goal examples for level 1 tasks using a random policy; (2) the root RMs must include rejecting states and use an extra proposition, complicating learning; (3) all non-lava policies must avoid the lava; (4) policies to reach the lava must be learned; and (5) metacontrollers must learn that edges labeled with 🔥 should not be chosen. Similar factors make FR harder than OP and OPL. The collection of goal examples in level 1 tasks is challenging with ϵ-greedy, especially those with several subgoals (e.g., BATTER) where convergence is delayed relative to those with fewer subgoals. It is also more difficult to generalize across instances since FR's grid is bigger. In all cases, we observe that once level 1 tasks are mastered, learning in higher level tasks is fast owing to the reuse of lower level tasks' RMs.

The average running times (in seconds) of the HRM learning system across runs are 1257.8 (163.2) for OP, 1706.0 (302.8) for OPL, and 669.9 (113.1) for FR (standard errors in brackets). Including the lava (OPL) increases the time needed to learn

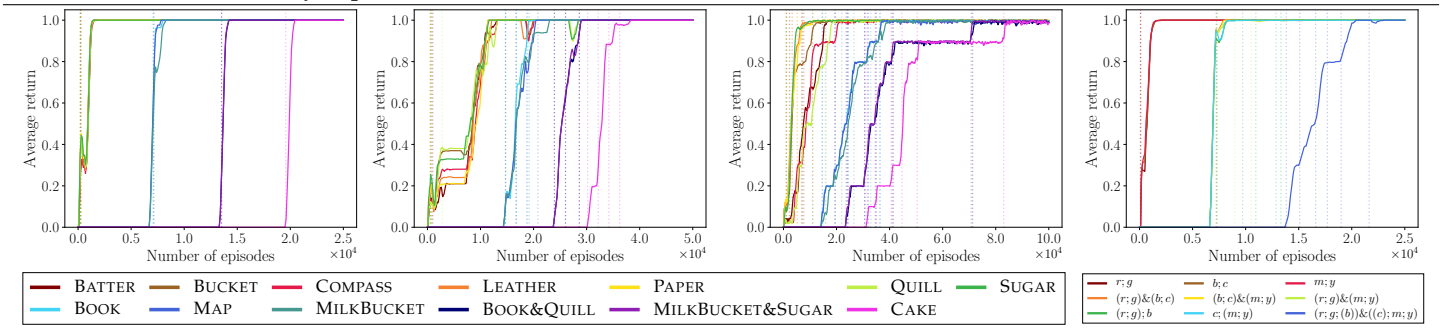


Figure 2: Learning curves for CRAFTWORLD (OP, OPL, and FR) and WATERWORLD.

the HRMs due to (1) a bigger hypothesis space caused by the increase in the number of propositions and the inclusion of rejecting states, and (2) the need to cover more example traces (OPL involves dead-end traces). In the case of FR, despite the learned HRMs are similar to those in OP, the running time is lower. This may be due to the example traces being shorter: propositions are sparsely distributed in FR and dense in OP. In all cases, around 90% of the running time is spent on the HRMs for BOOK, MAP and CAKE. Learning a multi-level HRM is less demanding than learning an equivalent flat one: the only task out of level 1 whose flat HRM can be learned within 2 hours is MILKBUCKET, which consists of 4 states. The flat HRM for BOOK, which is also a level 2 task, contains twice as many states (see Figure 1c). This shows that leveraging task compositionality helps learning RMs which could not have been learned in previous work.

The performance of exploration is evaluated by measuring for each task the number of episodes between the activation of its level and the learning of its first HRM. We compare the performance by using only primitive actions versus using call and formula options as well. Using only primitive actions leads to a higher number of episodes (i.e., it takes more time to collect the set of κ examples). While the BOOK task in the OP scenario needs 6957.2 (302.8) episodes using primitive actions, only 500.3 (9.6) episodes are required if options are also used. Delaying the learning of an HRM incurs a general delay since it takes longer to switch to higher levels. In the case of FR, observing goal examples is harder and surpassing level 2 never occurs if primitive actions are exclusively used. In addition, using sets of goal examples is shown to be convenient: using $\kappa = \kappa_s = 1$ in OP causes experiments to time out when on level 2 or higher in 9/10 runs. Finally, we evaluate policy learning alone by comparing the learning curves for handcrafted non-flat and equivalent flat HRMs. We observe both converge similarly in the simplest tasks, while non-flat HRMs speed up convergence in the hardest ones.

In future work, we plan to modify the curriculum learning component by removing the pre-established levels for each task, and allowing policies from lower levels to be used without being close to perfect.

References

[1] M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic Gridworld Environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[2] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *J. Artif. Intell. Res.*, 70:1031–1116, 2021.

[3] M. Gaon and R. I. Brafman. Reinforcement Learning with Non-Markovian Rewards. In *AAAI*, 2020.

[4] M. Hasanbeig, N. Y. Jeppu, A. Abate, T. Melham, and D. Kroening. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *AAAI*, 2021.

[5] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *NeurIPS*, 2016.

[6] T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning Compositional Neural Programs with Recursive Tree Search and Planning. In *NeurIPS*, 2019.

[7] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[8] R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *ICML*, 2018.

[9] R. Toro Icarte, E. Waldie, T. Q. Klassen, R. A. Valenzano, M. P. Castro, and S. A. McIlraith. Learning Reward Machines for Partially Observable Reinforcement Learning. In *NeurIPS*, 2019.

[10] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, 2016.

[11] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu. Joint Inference of Reward Machines and Policies for Reinforcement Learning. In *ICAPS*, 2020. 220

Making Policy Gradient Estimators for Softmax Policies More Robust to Non-stationarities

Shivam Garg
Department of Computing Science
University of Alberta
Edmonton, AB
sgarg2@ualberta.ca

Samuele Tosatto
Department of Computing Science
University of Alberta
Edmonton, AB
tosatto@ualberta.ca

Yangchen Pan*
Noah's Ark Lab
Huawei
Edmonton, AB
pan6@ualberta.ca

Martha White†
Department of Computing Science
University of Alberta
Edmonton, AB
whitem@ualberta.ca

A. Rupam Mahmood†
Department of Computing Science
University of Alberta
Edmonton, AB
armahmood@ualberta.ca

Abstract

Policy gradient (PG) estimators are ineffective in dealing with softmax policies that are sub-optimally saturated, which refers to the situation when the policy concentrates its probability mass on sub-optimal actions. Sub-optimal policy saturation may arise from a bad policy initialization or a sudden change, i.e. a non-stationarity, in the environment that occurs after the policy has already converged. Unfortunately, current softmax PG estimators require a large number of updates to overcome policy saturation, which causes low sample efficiency and poor adaptability to new situations. To mitigate this problem, we propose a novel policy gradient estimator, which we call as the *alternate estimator*, for softmax policies. This new estimator utilizes the bias in the critic estimate and the noise present in the reward signal to escape the saturated regions of the policy parameter space. We establish these properties by analyzing this estimator in the tabular bandit setting, and testing it on non-stationary reinforcement learning environments. Our results demonstrate that the alternate estimator is significantly more robust to policy saturation compared to the regular variant, and can be readily adapted to work with different PG algorithms and function approximation schemes.

(The full version of this paper is available at https://arxiv.org/abs/2112.11622.)

Keywords: Policy gradient, softmax policies, policy saturation, non-stationary environments.

Acknowledgements

The authors gratefully acknowledge funding from the Canada CIFAR AI Chairs program, the Reinforcement Learning and Artificial Intelligence (RLAI) laboratory, the Alberta Machine Intelligence Institute (Amii), and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

*Work done while at UofA.
†CIFAR AI Chair, Alberta Machine Intelligence Institute (Amii)

1 Introduction

Policy gradient (PG) algorithms aim to optimize a sequential decision making problem defined on a set of parameterized policies: the policy parameters are optimized using standard stochastic gradient-based update to maximize the net reward received by the agent. PG methods have been successfully deployed in a variety of real world tasks such as robotics (Mahmood, et al., 2018) and large scale simulated problems (Berner et al., 2019). For discrete action tasks, policies are typically represented using a categorical distribution parameterized by a softmax function. However, softmax policies have some inherent issues. Even with access to the true gradients, they can be slow in responding to non-stationarity and their performance heavily depends on the initialization of the policy parameters (Mei et al. 2020). Both these problems arise from an issue, which we call *sub-optimal policy saturation*.

Sub-optimal policy saturation refers to the situation when the policy places a high probability mass on sub-optimal actions. An agent with a saturated policy will not be able to explore other actions and may continue to remain in the sub-optimal region if an appropriate measure is not taken. Sub-optimal policy saturation arises in multiple scenarios: (1) Non-stationarity: in a constantly changing environment, what was once an optimal strategy may no longer work well. (2) Pre-training / transfer learning: deep reinforcement learning (RL) systems are often pre-trained on different tasks before being used on the main task, and the subtle differences in the structure of these tasks might lead to a bad policy initialization which is sub-optimally saturated. And (3) Stochastic updates: PG updates suffer from high variance and therefore, it is possible for a policy to become saturated on sub-optimal actions during the course of learning.

PG methods with softmax policies are particularly susceptible to policy saturation. Entropic regularization (Peters et al., 2010) is a popular approach to mitigate this issue: it works by making the policy more explorative. However, entropic regularization introduces additional terms in the objective, and therefore the resulting optimal policy can be different from the original one. We take a different approach to address this issue. Instead of augmenting the optimization objective with entropy, we introduce a PG estimator that inherently helps to escape the sub-optimally saturated regions.

Our proposed estimator is a simple yet effective approach for dealing with sub-optimally saturated policies thereby making PG algorithms more robust. The classic likelihood ratio estimator for softmax policies, which we call as the *regular estimator*, takes a frustratingly large amount of experience to escape sub-optimally saturated policy regions. The regular estimator produces near zero gradients at saturation as both its expectation and variance, even with reward noise, are vanishingly small at those regions. In contrast, our proposed estimator, which we call the *alternate estimator*, has a non-zero variance in the same scenario that can be utilized to escape the sub-optimal regions. Further, the alternate estimator naturally utilizes the bias in the critic estimate to increase the policy's entropy, thereby encouraging exploration. As the critic estimate improves, this effect reduces, allowing the policy to saturate towards the optimal actions.

2 Preliminaries

In RL, the decision making task is described using a Markov decision process (MD) $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mu, p, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R} \subset \mathbb{R}$ represent the sets of states, actions, and rewards; $\mu \in \Delta(\mathcal{S})$ is the start state distribution; $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$ is the transition dynamics; and $\gamma \in [0, 1]$ is the discount factor. (The object $\Delta(\mathcal{X})$ denotes the set of all possible probability distributions over the set $\mathcal{X}$.) The agent maintains a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that describes its interaction with the environment. This interaction results in the episode $\{S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T\}$, where $S_0 \sim \mu$, $A_t \sim \pi(\cdot|S_t)$, and $S_{t+1}, R_{t+1} \sim p(\cdot, \cdot|S_t, A_t)$ for $t \in \{0, \ldots, T-1\}$ with T being the (possibly random) episode termination length. The agent uses such interactions to learn a policy that maximizes the expected return $\mathcal{J} = \mathbb{E}_\pi[\sum_0^{T-1} \gamma^t R_{t+1}]$. PG methods provide one way to accomplish this task. We now describe these methods in two typical settings.

Gradient Bandits: In bandits the agent picks an action A_t from a discrete action set $\mathcal{A}$ at each timestep, and obtains a reward $R_t \sim p(\cdot|A_t)$. The goal is to maximize the expected immediate reward $\mathcal{J} := \mathbb{E}_\pi[R_t] =: r_\pi$. Gradient bandit algorithms maximize $\mathcal{J}$ using gradient ascent. Given a softmax policy $\pi_{\boldsymbol{\theta}}(a) = e^{\theta_a} / \sum_{b \in \mathcal{A}} e^{\theta_b}$, where θ_a is the action preference corresponding to action a, the gradient $\nabla \mathcal{J}$ (Sutton & Barto, 2018) is given by $[\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi]_i = r(a_i) \cdot \pi(a_i) \cdot (1 - \pi(a_i))$, where $r(a) := \mathbb{E}[R|a]$ is the reward function. However, the agent usually does not have access to r for all actions at the same time and must resort to using sample based gradient estimators $\hat{\mathbf{g}}$ which satisfy $\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi = \mathbb{E}_{A \sim \pi; R \sim p(\cdot|A)}[\hat{\mathbf{g}}(A, R)]$. And the typical choice is to use what we call the *regular estimator*

$$[\hat{\mathbf{g}}^{\text{REG}}(A, R)]_a := (R - b)(\mathbb{I}(A = a) - \pi(a)), \tag{1}$$

where $\mathbb{I}$ is the indicator function and b is the agent's estimate of the average reward r_π.

Policy Gradient: In an episodic MDP, the PG objective is $\mathcal{J} = \mathbb{E}_\pi[\sum_0^{T-1} \gamma^t R_{t+1}]$, and the policy gradient is given by

$$\nabla \mathcal{J} = \sum_{s \in \mathcal{S}} \nu_\pi(s) \sum_{a \in \mathcal{A}} \nabla \pi(a|s) q_\pi(s, a). \tag{2}$$

where $\nu_\pi(s) := \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(S_k = s)$ is the state occupancy measure under policy π.

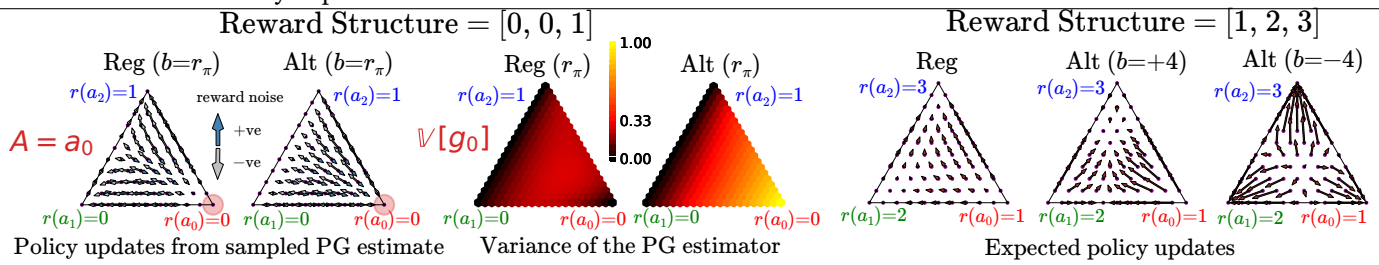


Figure 1: Policy updates and the variance for PG estimators on 3-armed bandits plotted on the probability simplex. The policy-update plots were drawn by updating the policy using the corresponding stochastic PG estimator, assuming that action a_0 was taken and a noise of constant magnitude was added to the reward. The blue and grey arrows correspond to the policy change direction for $+ve$ and $-ve$ reward noises. The variance plots show the heatmap for the zeroth element of the gradient estimator. Last three plots show policy updates using expectation of the estimators.

3 Warmup: Alternate Gradient Bandits

In this section, we derive the alternate PG estimator for the bandit setting using vector notation. For a k-armed bandit problem, let $\boldsymbol{\pi}, \mathbf{r} \in \mathbb{R}^k$ denote the policy and the reward vectors with $[\boldsymbol{\pi}]_k = \pi(a_k)$ and $[\mathbf{r}]_k = r(a_k)$. Using vectors, the expectations of scalar quantities become vector inner products: $\mathcal{J}_\pi \equiv r_\pi = \mathbb{E}_\pi[R] = \sum_a \pi(a) r(a) = \boldsymbol{\pi}^\top \mathbf{r}$. Further, the softmax policy can be written as $\boldsymbol{\pi} = e^{\boldsymbol{\theta}} / (\mathbf{1}^\top e^{\boldsymbol{\theta}})$, where the vector $e^{\boldsymbol{\theta}} \in \mathbb{R}^k$ is defined as $[e^{\boldsymbol{\theta}}]_a := e^{\theta_a}$. And $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi} = (\mathbf{I} - \boldsymbol{\pi}\, \mathbf{1}^\top) \operatorname{diag}(\boldsymbol{\pi})$. Hence, the regular policy gradient estimator can be derived as follows:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J} = \nabla_{\boldsymbol{\theta}} (\boldsymbol{\pi}^\top \mathbf{r}) = (\mathbf{I} - \boldsymbol{\pi}\, \mathbf{1}^\top) \operatorname{diag}(\boldsymbol{\pi})\, \mathbf{r} = \mathbb{E}_{A \sim \pi}[r(A)(\mathbf{e}_A - \boldsymbol{\pi})] = \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)}[(R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi})], \tag{3}$$

where $\mathbf{g}^{\text{REG}}(A, R) := (R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi})$ is the regular gradient bandit estimator and $\mathbf{e}_A \in \mathbb{R}^k$ represents the basis vector. The approximate version of it, as given before in Eq. 1, is $\hat{\mathbf{g}}^{\text{REG}}(A, R) = (R - b)(\mathbf{e}_A - \boldsymbol{\pi})$, where b is an estimate of r_π. Also note that this approximate estimator remains unbiased, i.e. $\nabla \mathcal{J} = \mathbb{E}[\hat{\mathbf{g}}^{\text{REG}}(A, R)]$.

The Alternate Gradient Bandit Estimator is derived by using an alternate form of the true gradient:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{J} &= (\mathbf{I} - \boldsymbol{\pi}\, \mathbf{1}^\top) \operatorname{diag}(\boldsymbol{\pi})\, \mathbf{r} = \left(\operatorname{diag}(\mathbf{r}) - \boldsymbol{\pi}\, \mathbf{r}^\top\right) \boldsymbol{\pi} = \operatorname{diag}(\mathbf{r})\, \boldsymbol{\pi} - \boldsymbol{\pi}^\top \mathbf{r}\, \boldsymbol{\pi} = \left(\operatorname{diag}(\mathbf{r}) - \boldsymbol{\pi}^\top \mathbf{r}\, \mathbf{I}\right) \boldsymbol{\pi} \\ &= \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)}[(R - r_\pi)\, \mathbf{e}_A], \end{aligned} \tag{4}$$

where $\mathbf{g}^{\text{ALT}}(A, R) := (R - r_\pi)\, \mathbf{e}_A$ is the *alternate* gradient bandit estimator. Using a baseline b to estimate the average reward r_π, gives the following estimator: $\hat{\mathbf{g}}^{\text{ALT}}(A, R) = (R - b)\, \mathbf{e}_A$.

4 Properties of the Alternate Gradient Bandits Estimator

We now discuss how the alternate estimator utilizes the reward noise and the bias in the critic estimate to escape saturated regions in the policy space.

Alternate Estimator Utilizes Reward Noise: Whenever the policy is saturated, i.e. it places a high probability mass on some actions, the expected gradient becomes close to zero, and consequently the policy weights are not updated. Further, for the regular estimator, the variance at these "corners" of the probability simplex is also zero. Therefore, there is neither any gradient signal nor any noise, and the agent is unable to escape the sub-optimal region. In contrast, the alternate estimator, despite being zero in expectation, has a non-zero variance and therefore the agent can utilize the reward noise to escape the sub-optimal region. We illustrate this point with Figure 1 (left and middle) and an accompanying example:

Example 1. *Consider a 3-armed bandit with $\mathcal{A} = \{a_0, a_1, a_2\}$, the expected reward $\mathbf{r} = [0\ 0\ 1]^\top$, and the policy $\boldsymbol{\pi} = [1\ 0\ 0]^\top$. The rewards are perturbed with a normally distributed noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$, so that upon picking action A, the agent receives the reward $r(A) + \epsilon$. Since $r_\pi = 0$, $R - r_\pi = 0 + \epsilon - 0 = \epsilon$. And because the agent samples $A = a_0$ at each timestep, we obtain*

$$\begin{aligned} \mathbf{g}^{REG}(A, R) &= (R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi}) = \epsilon \cdot \left([1\ 0\ 0]^\top - [1\ 0\ 0]^\top\right) = [0\ 0\ 0]^\top, \text{ and} \\ \mathbf{g}^{ALT}(A, R) &= (R - r_\pi)\, \mathbf{e}_A = \epsilon \cdot [1\ 0\ 0]^\top = [\epsilon\ 0\ 0]^\top. \end{aligned}$$

This example shows that at a sub-optimal corner, the agent with the alternate estimator effectively does a random walk until it escapes that region and can choose other actions to get a non-zero reward signal. Further, doing a random walk at the optimal corner is not problematic, since there is an attractive gradient signal as soon as the agent moves away from this corner. The following propositions formalize these points.

Proposition 1. *Define $\mathcal{I}_c := \{a \mid r(a) = c\}$ for some constant $c \in \mathbb{R}$. Assume that $\exists c$ such that $\mathcal{I}_c \neq \emptyset$ and that the policy is saturated on the actions in the set $\mathcal{I}_c$: $\sum_{a\in\mathcal{I}_c} \pi(a) = 1$. Then the expected policy gradient for softmax policies is zero: $\nabla_{\boldsymbol{\theta}}\mathcal{J} = \mathbb{E}[\mathbf{g}^{REG}(A,R)] = \mathbb{E}[\hat{\mathbf{g}}^{REG}(A,R)] = \mathbb{E}[\mathbf{g}^{ALT}(A,R)] = \mathbf{0}$.*

Proposition 2. *Let $\sigma(a)^2 := \mathbb{V}[R|A=a]$ be the variance of the reward corresponding to action a. Assume that the policy is saturated on the action c, i.e., $\pi(c) = 1$. Then, the variance of the regular PG estimator (with or without a baseline) is zero: $\mathbb{V}[\mathbf{g}^{REG}(A)] = \mathbb{V}[\hat{\mathbf{g}}^{REG}(A,R)] = \mathbf{0}$. Whereas, the variance of the alternate PG estimator is non-zero: $\mathbb{V}[\mathbf{g}^{ALT}(A,R)] = \sigma(c)^2\,\mathbf{e}_c$.*

Although, the above example and the propositions require the policy to lie at the boundary of the probability simplex which is unsatisfiable for softmax policies, using continuity arguments, we can still reason that for the regular estimator, both the expected gradient and its variance vanish in the proximity of the simplex boundary, whereas the alternate estimator will have non-zero variance. In addition to this, the stochastic update for the alternate estimator is much higher than that for the regular estimator. This can be seen from Figure 1 (left): look at the length of the update arrows near the bottom-right corner on the simplex.

Alternate Estimator Utilizes its Biasedness: The alternate estimator can also utilize the bias in the critic estimate b to escape saturation. To see this, consider a baseline $b \neq r_\pi$. Then the alternate estimator becomes biased: $\nabla_{\boldsymbol{\theta}}\mathcal{J} \neq \mathbb{E}[\hat{\mathbf{g}}^{\text{ALT}}(A,R)] = \mathbb{E}[(R-b)\,\mathbf{e}_A] = \boldsymbol{\pi}\odot(\mathbf{r} - b\mathbf{1})$. Interestingly, this biased update $\mathbb{E}[\hat{\mathbf{g}}^{\text{ALT}}(A,R)]$ is not the gradient of any function. Therefore, in order to understand its behavior, we look at its fixed point and under what conditions it acts as an attractor or a repellor. Figure 1 (right) illustrates this fixed point and its behavior based on how the baseline is initialized for one specific instance of a bandit problem. We now state a theorem that formalizes this property. Let $r_1 \leq r_2 \leq \cdots \leq r_k$ represent the true rewards for the bandit problem. Let $\pi_t = e^{\boldsymbol{\theta}^{(t)}}/\mathbf{1}^\top e^{\boldsymbol{\theta}^{(t)}}$ represent the softmax policy at timestep t. And let the action preference vector $\boldsymbol{\theta}^{(t)}$ be updated using $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha\,\mathbb{E}[\hat{\mathbf{g}}^{\text{ALT}}(A,R)]$ for $\alpha > 0$.

Lemma 2.1. ***Fixed points of the biased gradient bandit update.*** ***(1)*** *If there exists an $n \in \{1,2,\ldots,k-1\}$ such that $r_1 \leq \cdots \leq r_n < b < r_{n+1} \leq \cdots \leq r_k$, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{ALT}(A,R)]$ is never equal to zero.* ***(2)*** *If $b = r(a)$ for at least one action, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{ALT}(A,R)] = 0$ at any point on the face of the probability simplex given by $\sum_{a\in\mathcal{I}_b}\pi(a) = 1$ with $\mathcal{I}_b := \{a|r(a) = b\}$.* ***(3)*** *If $b < r_1$ or $b > r_k$, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{ALT}(A,R)] = 0$ at a point π^* within the simplex boundary given by $\pi^*(a) = \frac{1}{r(a)-b}\left(\sum_{c\in\mathcal{A}}\frac{1}{r(c)-b}\right)^{-1},\ \forall a \in \mathcal{A}$.*

Theorem 1. ***Nature of the fixed point $\boldsymbol{\pi}^*$.*** *Assume that $\pi_t \neq \pi^*$. If the baseline is pessimistic, i.e. $b < r_1$, then for any $\alpha > 0$, the fixed point π^* acts as a repellor: $D_{KL}(\pi^*\|\pi_{t+1}) > D_{KL}(\pi^*\|\pi_t)$, where D_{KL} is the KL-divergence. And if the baseline is optimistic, i.e. $b > r_k$, then given a sufficiently small positive stepsize α the fixed point π^* acts as an attractor: $D_{KL}(\pi^*\|\pi_{t+1}) < D_{KL}(\pi^*\|\pi_t)$.*

The above theorem illustrates that with an optimistic baseline, an agent using the alternate estimator is updated towards a more uniform distribution π^*. Therefore, if the agent were stuck in a sub-optimal corner of the probability simplex, an optimistic baseline would make its policy more uniform and encourage exploration. (On the flip side, with a pessimistically initialized baseline, the alternate estimator can pre-maturely saturate towards a sub-optimal corner; see Figure 1 (right).) And even though π^* is different from the optimal policy, the agent with an alternate estimator can still reach the optimal policy, because as the agent learns and improves its baseline estimate, the alternate PG estimator becomes asymptotically unbiased. And hopefully by this time, the agent has already escaped the saturated policy region.

5 Alternate PG Estimator for MDPs

The alternate gradient estimator can be readily extended to MDPs, where it enjoys properties analogous to the bandit case. For a given state s, let $\boldsymbol{\pi}(\cdot|s) \in \mathbb{R}^{|\mathcal{A}|}$ be the vector with $[\boldsymbol{\pi}(\cdot|s)]_a = \pi(a|s)$. Similarly, define the action value vector $[\mathbf{q}_\pi(s,\cdot)]_a = q_\pi(s,a)$. The softmax policy for MDPs then becomes $\pi(a|s) = e^{[\boldsymbol{\theta}_\mathbf{w}(s)]_a}/\sum_{b\in\mathcal{A}} e^{[\boldsymbol{\theta}_\mathbf{w}(s)]_b}$, where $\boldsymbol{\theta}_\mathbf{w} : \mathcal{S} \to \mathbb{R}^{\mathcal{A}}$ denotes the action preference vector, parameterized by $\mathbf{w}$, and $[\boldsymbol{\theta}_\mathbf{w}(s)]_a$ is its ath element. In practice, $\boldsymbol{\theta}_\mathbf{w}$ could be implemented using, say, a neural network. For brevity, we will drop $\mathbf{w}$, i.e. $\boldsymbol{\theta} \equiv \boldsymbol{\theta}_\mathbf{w}$. To obtain the results that follows, we analytically worked out the gradient of the action likelihood for the softmax policy (similar to what we did for the bandit case; see Eqs. 3 and 4), which gives us

$$\nabla_\mathbf{w}\mathcal{J}_\pi = \mathbb{E}\Big[\underbrace{\nabla_\mathbf{w}\log\pi(A|S)(q_\pi(S,A) - v_\pi(S))}_{=:\mathbf{g}^{\text{REG}}(S,A)}\Big] = \mathbb{E}\Big[\underbrace{\nabla_\mathbf{w}[\boldsymbol{\theta}(S)]_A(q_\pi(S,A) - v_\pi(S))}_{=:\mathbf{g}^{\text{ALT}}(S,A)}\Big]. \tag{5}$$

Note that even though the regular and the alternate PG estimators are equal in expectation, in general they are not equal for an arbitrary state-action pair: $\mathbf{g}^{\text{REG}}(S,A) \neq \mathbf{g}^{\text{ALT}}(S,A)$. Further, it is straightforward to adapt the alternate estimator given in Eq. 5 to work with different PG methods such as REINFORCE, Actor-Critic, TRPO, or PPO.

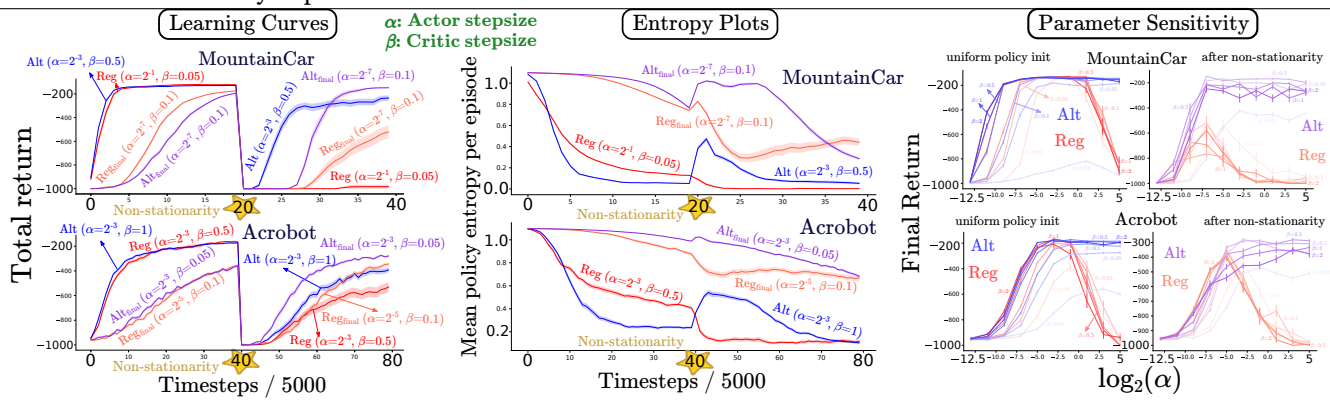


Figure 2: Performance of online Actor-Critic on MountainCar (200k timesteps) and Acrobot (400k timesteps). Learning curves are for the best performing stepsize configurations at the time non-stationarity was introduced and at the final timestep. Entropy plots show entropy of the policy on the exact states encountered during each episode. The sensitivity plots show the mean performance during the last 5000 timesteps. All the results were averaged over 50 runs.

6 Experiments on Non-stationary Environments

In this section, we demonstrate that the alternate estimator is able to effectively handle non-stationarity in an environment, whereas the regular estimator fails to do so. We trained the online Actor-Critic algorithm (Sutton and Barto, 2018) to solve the MountainCar and the Acrobot control tasks, with linear function approximation (using tile-coding). To induce non-stationarity in either task, we switched the `left` and `right` actions after half-time.

Figure 2 (left) shows the learning curves for the best parameter configuration. For each estimator, we selected two sets of parameter values: which performed best right before the non-stationarity hit and another which performed best at the end of the experiment (denoted by a `final` in the subscript). For MountainCar, the alternate estimator is superior to the regular estimator for both sets of stepsizes. Remarkably, the best performing parameter set for regular (red curve) at timestep 100k was unable to recover from the non-stationarity; whereas alternate (blue curve), despite having similar performance as regular at 100k timestep, was able to recover. On Acrobot, the difference in performance is still there but relatively smaller. We attribute the superior performance of the alternate estimator to the bias in the critic estimate. For instance, in MountainCar at 100k timesteps, the critic would have converged to predict a return of about -200. But when the non-stationarity hit, the agent would have started receiving returns much lower than -200. This means that at that time, the critic estimate became optimistic and encouraged exploration by pushing the policy towards a more uniform distribution. Figure 2 (middle) corroborates this point: the policy entropy for alternate (but not for regular) jumps right around the timestep when the non-stationarity hit. Figure 2 (right) shows the parameter sensitivity plots for these tasks.

7 Conclusions

We proposed an alternate policy gradient estimator for softmax policies that, as we demonstrated theoretically and empirically, effectively utilizes the reward noise and the bias in the critic to escape sub-optimally saturated regions in the policy space. Our analysis, conducted on multiple bandit and MDP tasks, suggests that this estimator works well with different PG algorithms and different function approximation schemes. The alternate estimator makes existing PG methods more viable for non-stationary problems, and by extension for many practical real-life control tasks.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680.*

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. *Conference on Robot Learning.*

Mei, J., Xiao, C., Dai, B., Li, L., Szepesvári, C., Schuurmans, D. (2020). Escaping the Gravitational Pull of Softmax. *Advances in Neural Information Processing Systems.*

Peters, J., Mulling, K., Altun, Y. (2010). Relative entropy policy search. *AAAI Conference on Artificial Intelligence.*

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, Second Edition. MIT Press.

From human behavioral experiments to improved autonomous agents through imitation and reinforcement learning

Vittorio Giammarino
Division of Systems Engineering
Boston University
Boston, MA 2446
vgiammar@bu.edu

Matthew F Dunne *
Cognitive Neuroimaging Center
Boston University
Boston, MA 2446
mdunne34@bu.edu

Kylie N Moore †
Cognitive Neuroimaging Center
Boston University
Boston, MA 2446
knmoore@bu.edu

Michael E Hasselmo
Center for Systems Neuroscience
Boston University
Boston, MA 2446
hasselmo@gmail.com

Chantal E Stern ‡
Cognitive Neuroimaging Center
Boston University
Boston, MA 2446
chantal@bu.edu

Ioannis Ch. Paschalidis §
Dept. of Electrical and Computer Engineering
Boston University
Boston, MA 2446
yannisp@bu.edu

Abstract

We develop a method to learn bio-inspired policies for autonomous agents from human behavioral data. The data were taken from a larger study investigating human foraging behavior and consist of 50 eight-minute trajectories collected from 5 different participants (10 eight-minute runs per participant). The human participants were virtually immersed in an open field foraging environment and trained to obtain the highest amount of rewards, in the given time, without having knowledge of the rewards distribution. We introduce a Markov Decision Process framework to model the human decision dynamics, incorporating both egocentric and allocentric information in the state. Autonomous agent policies are designed to map from human decisions to observed states. They are parameterized by a fully connected Neural Network with a single hidden layer and trained via Imitation Learning (IL) using maximum likelihood estimation. The results show that passive imitation substantially underperforms human performance. The human-inspired policies are then refined via on-policy Reinforcement Learning (RL), which shows to be more suitable than off-policy counterparts when combined with pre-trained networks. We show that the combination of IL and RL can match human performance and is robust to reward distribution shift. The developed methodology can be used to efficiently learn policies for unmanned vehicles which have to solve missions in an open field environment.

Keywords: Human Foraging, Imitation Learning, Reinforcement Learning

*also Graduate Program for Neuroscience and Center for Systems Neuroscience.
†also Graduate Program for Neuroscience and Center for Systems Neuroscience.
‡also Center for Systems Neuroscience.
§also Division of Systems Engineering and Dept. of Biomedical Engineering.

Distributional Reward Shaping: Point Estimates Are All You Need

Michael Gimelfarb **Scott Sanner** **Chi-Guhn Lee**
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Canada, M5S 3G8
{mike.gimelfarb,ssanner,cglee}@{mail,mie,mie}.utoronto.ca

Abstract

Potential-based reward shaping is a powerful approach for incorporating value-based advice in order to accelerate the convergence of reinforcement learning algorithms on problems with sparse reward. We propose the idea of distributional reward shaping, in which the shaping signal is a probability distribution over hypothetical returns in each state-action pair. A natural setting in which such advice could be useful for transferring knowledge is the distributional reinforcement learning (DRL), that has recently provided state-of-the-art results on a number of benchmark problems. However, it is largely unclear how to incorporate distributional advice while maintaining policy invariance guarantees as in standard RL. To this end, our first contribution is to show that distributional reward shaping maintains policy invariance if the policy is derived by maximization of the expected return. By drawing on several examples from the literature, our second contribution is to illustrate that such results do not hold generally in the risk-sensitive RL setting, in which the agent optimizes a non-linear utility function of the return. However, we show that the utility of the distributional reward shape could provide a powerful deterministic reward signal, that does not require making independence assumptions nor limiting the class of utility functions that can be used.

Keywords: reward shaping, transfer learning, distributional reinforcement learning, risk-sensitive, risk-aware, safety

1 Introduction

Potential-based reward shaping (PBRS) is a powerful technique for transforming a reinforcement learning problem with a sparse reward into one with a dense reward without changing the optimal policies [Ng et al., 1999]. Recent literature has demonstrated its ability and flexibility in transferring knowledge from a variety of sources [Brys et al., 2015a,b, Suay et al., 2016]. However, none of the existing work has studied the problem of transferring knowledge that occurs naturally in the form of a probability distribution over hypothetical return outcomes. Such "distributional" advice could be particularly useful as a means of communicating notions of danger or uncertainty between learning agents, or from humans to agents, and represents a richer vocabulary for transferring value-based advice than point estimates alone. One practical setting is the transfer of knowledge between RL agents following the *distributional RL* (DRL) framework [Bellemare et al., 2017], which estimates the full distribution of the return rather than its expectation alone. Here, the learnt value function distribution from one task could be used as input to another agent who later solves a different (but somewhat related) task, perhaps measuring risk in a different way. In many real-world domains that involve high risk, such as medical domains for example, a human expert could provide scenarios that might play out following a particular treatment (e.g. patient dies, patient survives), that could be best modeled as a distribution over returns due to the stochastic or uncertain nature of the problem.

The goal of this abstract is two-fold. First, we prove that policy invariance with distributional advice holds for DRL agents that optimize expected returns (Theorem 1). Second, we demonstrate that such results do not generalize directly to the risk-sensitive setting (Example 1, 2). Instead, the utility of the advice provides a PBRS signal that preserves invariance, and has favourable mathematical properties (Lemma 1).

2 Preliminaries

2.1 Distributional Reinforcement Learning

Given a *Markov decision process* (MDP) with state space $\mathcal{S}$, actions $\mathcal{A}$, reward function r, transition function P, and discount factor $\gamma \in (0,1)$, the goal of DRL is to learn the probability distribution of the discounted return $Z^*(s,a) = \sum_{t\geq 0}\gamma^t r(s_t,a_t,s_{t+1})$ following an optimal policy π^* (in a set of bounded random variables $\mathcal{Z}$) by solving the *distributional Bellman equation* (DBE):

$$Z^*(s,a) = r(s,a,S') + \gamma Z^*(S',\pi^*(S')), \quad \pi^*(s') \in \text{argmax}_{a'}\mathbb{E}_{Z^*}[Z^*(s',a')].$$

Bellemare et al. [2017] approximated $Z^*(s,a)$ by a histogram with equidistant points, and learnt a parametric representation $Z_\theta(s,a)$ by minimizing the KL-divergence between $Z_\theta(s,a)$ and the distribution of the single-step Bellman update. Furthermore, they showed that the DBE operator is a contraction mapping under the p-Wasserstein metric. More recently, Dabney et al. [2018b] proposed QR-DQN to estimate $Z(s,a)$ by discretizing the quantiles of the distribution rather than the outcomes, and used quantile regression to approximate the quantiles of $Z^*(s,a)$.

Dabney et al. [2018a] extended the framework of QR-DQN to learn risk-sensitive policies. Given a utility function $U : \mathbb{R} \to \mathbb{R}$, the goal of *risk-aware RL* is to learn a policy that optimizes the expected utility of the return associated with the Bellman equation:

$$Z^*(s,a) = r(s,a,S') + \gamma Z^*(S',\pi^*(S')), \quad \pi^*(s') \in \arg\max_{a'}\mathbb{E}_{Z^*}[U(Z^*(s',a'))].$$

Their approach, called IQN, restricted U to the class of *distortion risk measures*, that intuitively can be seen as computing the expected value of the return under a distorted distribution of $Z(s,a)$. More formally, U is associated with a distortion function $\beta : [0,1] \to [0,1]$, such that $\mathbb{E}_Z[U(Z(s,a))] = \mathbb{E}_{\tau\sim U([0,1])}[Z_{\beta(\tau)}(s,a)]$ where $Z_\tau(s,a) = F^{-1}_{Z(s,a)}(\tau)$ denotes the quantile function of $Z(s,a)$.

2.2 Utility Function

In our developments, we further abstract the idea of expected utility by considering the general class of *concave utility* functions $\mathcal{U} : \mathcal{Z} \to \mathbb{R}$ that satisfy [Föllmer and Schied, 2002]:

A1. $\mathcal{U}[0] = 0$

A2. if $Z_1, Z_2 \in \mathcal{Z}$ such that $\mathbb{P}(Z_1 \geq Z_2) = 1$, then $\mathcal{U}[Z_1] \geq \mathcal{U}[Z_2]$

A3. if $c \in \mathbb{R}$ and $Z \in \mathcal{Z}$, then $\mathcal{U}[Z + c] = \mathcal{U}[Z] + c$

A4. if $Z_1, Z_2 \in \mathcal{Z}$ then $\mathcal{U}[Z_1 + Z_2] \geq \mathcal{U}[Z_1] + \mathcal{U}[Z_2]$,

which can be seen as necessary criteria for rational and risk-averse decision-making. We also assume:

A5. if $Z_t \in \mathcal{Z}$ converges in distribution to $Z \in \mathcal{Z}$ with $\min Z_t \to \min Z$ and $\max Z_t \to \max Z$, then $\mathcal{U}[Z_t] \to \mathcal{U}[Z]$,

which ensures that the optimization criterion $\arg\max_a \mathcal{U}[Z_t(s,a)]$ remains meaningful as Z_t converges to the true return distribution.

2.3 Reward Shaping

The successful application of a reinforcement learning algorithm – and by extension DRL – depends largely on the quality of the chosen reward function. Given a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and a shaping function $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, reward shaping produces an augmented reward function $r'(s,a,s') = r(s,a,s') + F(s,a,s')$. The goal is to devise F such that the agent converges faster towards the optimal policy using r' than using r alone. However, care is necessary to ensure that the set of optimal policies remains *invariant* with respect to the above change in the reward function [Randløv and Alstrøm, 1998]. One way to guarantee policy invariance is *potential-based reward shaping* (PRBS). Specifically, letting $\Phi : \mathcal{S} \to \mathbb{R}$ be any function, PBRS defines the shaping function as $F(s,a,s') = \gamma\Phi(s') - \Phi(s)$. Intuitively, when the agent transitions from state s to s', PBRS "replaces" the incentive $\Phi(s)$ given in s with a new incentive $\Phi(s')$ in successor state s'. Ng et al. [1999] showed that PBRS is the only way to ensure policy invariance in general MDPs.

3 Distributional Reward Shaping

In this section, we discuss the distributional setting in which the advice and the agent are both return distributions in $\mathcal{Z}$.

3.1 Distributional Policy Invariance in DRL

To establish a characterization of policy invariance in DRL, we first study the risk-neutral setting $\mathcal{U} = \mathbb{E}$, where advice is given as a probability distribution over hypothetical return.

Theorem 1. *Let $\Phi : \mathcal{S} \to \mathcal{Z}$ be independent at each evaluation. If $\mathcal{U} = \mathbb{E}$, then PBRS leaves the optimal policies unchanged.*

Proof. We begin with the Bellman equation for an MDP $\mathcal{M}$ with dynamics P and reward function r,

$$Z_t(s,a) = r(s,a,S') + \gamma Z_{t+1}(S', \arg\max_{a'} \mathcal{U}[Z_{t+1}(S',a')]) := \mathcal{T} Z_{t+1}(s,a),$$

and subtract $\Phi(s)$ from both sides to obtain

$$Z_t(s,a) - \Phi(s) = r(s,a,S') + \gamma\Phi(S') - \Phi(s) + \gamma\left(Z_{t+1}(S', \arg\max_{a'} \mathcal{U}[Z_{t+1}(S',a')]) - \Phi(S')\right). \tag{1}$$

Here and throughout, the equality is to be understood as equivalence in distribution. Next, define the random variable $Z'_t(s,a)$ and $r'_t(s,a,s')$ such that:

$$Z'_t(s,a) := Z_t(s,a) - \Phi(s) \tag{2}$$

$$r'(s,a,S') := r(s,a,S') + \gamma\Phi(S') - \Phi(s). \tag{3}$$

Substituting the last two equations into (1), we obtain

$$Z'_t(s,a) = r'(s,a,S') + \gamma Z'_{t+1}(S', \arg\max_{a'} \mathcal{U}[Z_{t+1}(S',a')]).$$

Next, given **A3**, (2) and linearity of expectation:

$$\begin{aligned}\pi_{t+1}(s') \in \arg\max_{a'} \mathcal{U}[Z_{t+1}(s',a')] &= \arg\max_{a'} \left\{\mathcal{U}[Z'_{t+1}(s',a')] + \mathcal{U}[\Phi(s')]\right\} \\ &= \arg\max_{a'} \mathcal{U}[Z'_{t+1}(s',a')] := \pi'_{t+1}(s'),\end{aligned}$$

and hence:

$$\begin{aligned}Z'_t(s,a) &= r'(s,a,S') + \gamma Z'_{t+1}(S', \arg\max_{a'} \mathcal{U}[Z_{t+1}(S',a')]) \\ &= r'(s,a,S') + \gamma Z'_{t+1}(S', \arg\max_{a'} \mathcal{U}[Z'_{t+1}(S',a')]) := \mathcal{T}' Z'_{t+1}(s,a),\end{aligned}$$

is the distributional Bellman operator of the MDP with immediate reward function r'. Finally, we observe that the roles of the two MDPs with reward r and r' can be interchanged, and so the optimal policies are preserved, as claimed. □

Observe that (2) foreshadows a result in Ng et al. [1999], namely that PBRS shifts the return in each state-action pair down by the value of the potential $\Phi(s)$ in each state of the MDP. However, a critical distinction in the DRL setting is that PBRS leads to a *convolution* of $Z_t(s,a)$ and $\Phi(s)$. Also, (3) suggests that immediate rewards r' can be stochastic even after observing $s' \sim S'$. This does not complicate our analysis, however, since the stochasticity can be absorbed directly into the estimate $Z_t(s,a)$. Finally, we note that the consequences of this theorem also hold for arbitrary $\mathcal{U}$ if Φ_t is deterministic (e.g. risk-free).

3.2 The Failure of Policy Invariance in Risk-Sensitive DRL

A natural question to ask is whether policy invariance still holds for arbitrary concave utility functions. Unfortunately, this turns out to be false in general, since the analysis above shows that $\mathcal{U}$ must be strictly additive, e.g.

$$\mathcal{U}[Z_t(s,a) + \Phi(s)] = \mathcal{U}[Z_t(s,a)] + \mathcal{U}[\Phi(s)] \tag{4}$$

for all choices of Z_t and Φ. In more intuitive terms, *policy invariance holds if the potential function does not permit the learning agent to improve the overall risk through diversification among two lotteries: (1) the agent's current knowledge of the return $Z_t(s,a)$, and (2) the expert's current knowledge of the return $\Phi(s)$*. Without further knowledge about the dependence between $Z_t(s,a)$ and $\Phi(s)$, it is therefore not possible to establish policy invariance except when $\mathcal{U} = \mathbb{E}$. To make this point clearer, we illustrate how this would require making careful choices for $\mathcal{U}$ that depend on how $Z_t(s,a)$ and $\Phi(s)$ are jointly distributed.

Example 1. Suppose that $Z_t(s,a)$ and $\Phi(s)$ are independent for each (s,a), and consider the *entropic utility* function $\mathcal{U}_\beta$

$$\mathcal{U}_\beta[Z] = \frac{1}{\beta} \log \mathbb{E}[e^{\beta Z}],$$

where $\beta \in \mathbb{R}$ is an arbitrary control parameter, and which satisfies **A1-A4**. Furthermore, we also define the *weighted entropic utility* $\mathcal{U}_w$ as

$$\mathcal{U}_w[Z] = \int \mathcal{U}_\beta[Z]\, w(\beta)\, \mathrm{d}\beta$$

where $w : \mathbb{R} \to [0, \infty)$, and which satisfies **A1-A4** provided that $\int w(\beta)\, \mathrm{d}\beta = 1$. It is also easy to verify that:

$$\mathcal{U}_w[Z_t(s,a) + \Phi(s)] = \mathcal{U}_w[Z_t(s,a)] + \mathcal{U}_w[\Phi(s)].$$

Furthermore, it can be shown that $\mathcal{U}_w$ is the *only* class of utility functions that satisfies **A1-A4** and **A5** under the independence assumption [Goovaerts et al., 2004].

While the form of $\mathcal{U}$ in the aforementioned example appears restrictive, we note that entropic utility essentially guarantees policy invariance as long as $Z_t(s,a)$ and $\Phi(s)$ are simulated from uncoupled stochastic processes. This setting is quite natural, for instance, if $\Phi(s)$ and $Z_t(s,a)$ were learnt on different tasks using IQN in an end-to-end framework, since samples $\Phi(s) \sim \max_a Z_{\beta(\tau)}(s,a)$ and $G_t \sim Z_{\beta(\tau')}(s,a)$ are typically generated according to i.i.d. samples $\tau, \tau' \sim U([0,1])$. In many instances however, such as when the immediate rewards or the return realizations from different tasks are directly correlated [Zhang et al., 2021], the entropic utility is no longer a viable quantifier of risk.

Example 2. A converse example to independence is perfect dependence or *commonoticity*, in which $Z_t(s,a) = F^{-1}_{Z_t(s,a)}(\tau)$ and $\Phi(s) = F^{-1}_{\Phi(s)}(\tau)$ for the same realization of $\tau \sim U([0,1])$, where equality holds in distribution. We define $Q_X(p)$ to denote the p-quantile function of X, and the *tail value-at-risk*

$$\mathrm{TVaR}_p[X] = \frac{1}{1-p} \int_p^1 Q_X(q)\, \mathrm{d}q.$$

This utility function is additive for commonotone random variables [Dhaene et al., 2006], so (4) holds.

Taken together, the previous two examples have thus shown that *policy invariance holds only when the dependence structure between the model returns of the agent and the expert are precisely known, and the utility function $\mathcal{U}$ is chosen carefully*. In other words, ignoring the hidden interaction between the agents can lead to unintended behaviours in end-to-end or multi-task approaches. Another shortcoming of the distributional approach to shaping is that it requires computing the convolution of several probability distributions for each sample and can be computationally demanding. Finally, the choice of utility should be dictated by the external environment (e.g. regulators), rather than solely from the specificity of each task.

3.3 Point Estimates are All You Need

If arbitrary distributional advice cannot be accounted for directly without knowing the structure of Z_t and Φ, we ask whether it can be incorporated into PBRS in other ways. Given an arbitrary distribution-valued potential $\Phi_t \in \mathcal{Z}$, one such candidate that is consistent with the agent's decision-making criteria is $\phi(s) = \mathcal{U}[\Phi(s)]$. With this simplification, (4) holds immediately due to cash invariance, and thus $\phi(s)$ provides a policy-invariant reward signal.

The next natural question to ask is: what kind of advice allows DRL to learn *efficiently*? We can answer this question briefly by showing that the nature of the ideal *deterministic* potential Φ takes the form of the utility of the return distribution Z^*, with similar results having been shown only for standard RL [Zou et al., 2021].

Lemma 1. *For an MDP with return distribution Z^* under optimal policy π^*, the PBRS signal $\phi(s) = \max_a \mathcal{U}[Z^*(s,a)]$ defines an MDP $\mathcal{M}'$ with a dense reward, whose optimal utility is zero along the optimal trajectory $\pi^*(s)$ and negative everywhere else.*

Proof. The consequences of Theorem 1 hold for the deterministic signal $\phi(s)$, and so the optimal policy remains unchanged. Furthermore, according to (2) and **A3**:

$$\mathcal{U}[Z'(s,a)] = \mathcal{U}[Z^*(s,a) + (-\phi(s))] = \mathcal{U}[Z^*(s,a)] - \phi(s) = \mathcal{U}[Z^*(s,a)] - \max_a \mathcal{U}[Z^*(s,a)] \leq 0,$$

where equality holds only if $a \in \arg\max_a \mathcal{U}[Z^*(s,a)] = \pi^*(s)$, as claimed. □

3.4 Generalization to State-Action Potentials

As a final observation, we note that it is easy to extend our previous analysis to the setting in which $\Phi_t(s,a)$ is time and action-dependent, by considering the look-ahead reward shaping function

$$F_t(s,a,s',a') = \gamma\Phi_{t+1}(s',a') - \Phi_t(s,a),$$

where a is the action chosen in state s, a' is the corresponding action chosen in state s', and so forth. By extending the derivations of Wiewiora et al. [2003], Devlin and Kudenko [2012] to our setting, it is easy to check that

$$\mathcal{U}[Z'_t(s,a)] = \mathcal{U}[Z_t(s,a) - \Phi_t(s,a)].$$

Thus, under assumptions **A1-A5**, policy invariance holds once again as long as the actions in the new MDP are chosen according to

$$\pi'_t(s) \in \operatorname{argmax}_a \left\{\mathcal{U}[Z_t(s,a)] + \mathcal{U}[\Phi_t(s,a)]\right\},$$

where $\phi(s,a) = \mathcal{U}[Z^*(s,a)]$ satisfies the usual guarantees for PBRS.

4 Conclusion

We demonstrated that PBRS can be generalized to a distribution over returns, and that it preserves policy invariance in standard DRL. A negative result is illustrated, namely that policy invariance in the risk-sensitive setting cannot be guaranteed for arbitrary non-linear utility functions of the return. Instead, to accommodate many rational choices of utility functions such as CVaR, we propose to map the distributional advice to a point estimate by using the given utility function, in which case the resulting PBRS signal is risk-sensitive, policy-invariant, and is optimal in certain circumstances.

References

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *ICML*, pages 449–458. PMLR, 2017.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *AAAI*, 2015a.

Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. Policy transfer using reward shaping. In *AAMAS*, pages 181–188, 2015b.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *ICML*, pages 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *AAAI*, 2018b.

Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *AAMAS*, pages 433–440. IFAAMAS, 2012.

Jan Dhaene, Steven Vanduffel, Marc Goovaerts, Rob Kaas, Qihe Tang, and David Vyncke. Risk measures and comonotonicity: A review. *Stochastic Models*, 22:573–606, 12 2006. doi: 10.1080/15326340600878016.

Hans Föllmer and Alexander Schied. Convex measures of risk and trading constraints. *Finance and stochastics*, 6(4):429–447, 2002.

Marc J Goovaerts, Rob Kaas, Roger JA Laeven, and Qihe Tang. A comonotonic image of independence for additive risk measures. *Insurance: Mathematics and Economics*, 35(3):581–594, 2004.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, volume 98, pages 463–471. Citeseer, 1998.

Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pages 429–437, 2016.

Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *ICML*, pages 792–799, 2003.

Pushi Zhang, Xiaoyu Chen, Li Zhao, Wei Xiong, Tao Qin, and Tie-Yan Liu. Distributional reinforcement learning for multi-dimensional reward functions. *NeurIPS*, 34, 2021.

Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Learning task-distribution reward shaping with meta-learning. In *AAAI*, volume 35, pages 11210–11218, 2021.

Policy Optimization with Distributional Constraints: An Optimal Transport View

Arash Givchi
Rutgers University–Newark
arash.givchi@gmail.com

Pei Wang
Rutgers University–Newark
peiwang@rutgers.edu

Patrick Shafto
Rutgers University–Newark & Institute for Advanced Study (IAS)
patrick.shafto@gmail.com

Abstract

We consider constrained policy optimization in Reinforcement Learning (RL), where the constraints are in form of marginals on state visitations and global action executions. Given these distributions, we formulate policy optimization as unbalanced optimal transport over the set of occupancy measures. We propose a general purpose RL objective based on Bregman divergence and optimize using Dykstra's algorithm. The approach admits an actor-critic algorithm for when the state or action space is large and only samples from the marginals are available.

Keywords: Policy Optimization, Optimal Transport, Dykstra's algorithm

1 Introduction

Formulating RL as a linear programming problem, occupancy measures appear as an explicit constraint on the optimal policy [Put14]. Considering the reward function as negative cost of assigning an action to a state, we view RL as a stochastic assignment problem. We formulate policy optimization as an unbalanced optimal transport on the set of occupancy measures.

Optimal Transport (OT) [Vil08] is the problem of adapting two distributions on possibly different spaces via a cost function, unbalanced OT relaxes the marginal constraints on OT to arbitrary measures through penalty functions [LMS17, CPSV17]. We therefore define distributionally-constrained reinforcement learning as a problem of optimal transport. Given baseline marginals over states and actions, policy optimization is unbalanced optimal transport adapting the state marginal to action marginal via the reward function. Built upon mathematical tools of OT, we generalize the RL objective to the summation of a Bregman divergence D_Γ and any number of arbitrary lower-semicontinuous convex functions ϕ_i's and an initial distribution ξ as: $\min_{\mu\in\Delta} D_\Gamma(\mu|\xi) + \sum_i^N \phi_i(\mu)$, where Δ is the set of occupancy measures (joint distributions on states and actions generated from policies). We optimize this objective with *Dykstra's algorithm* [Dyk83] which is a method of iterative projections onto general closed convex constraint sets. Under Fenchel duality, this algorithm allows decomposition of the objective into Bregman projections on the subsets corresponding to each function.

As particular case, we can regularize over the state space distribution and/or the global action execution distribution of the desired occupancy measures. Given the reward function r and divergence functions D_{ψ_1} and D_{ψ_2}, our formulation allows constraints on the policy optimization problem in terms of distributions on state visitations ρ' and/or action executions η' as: $\max_{\mu\in\Delta} \mathbb{E}_\mu[r] - \epsilon_1 D_{\psi_1}\left(\mu\mathbf{1} \mid \rho'\right) - \epsilon_2 D_{\psi_2}\left(\mu^T\mathbf{1} \mid \eta'\right)$. We propose an actor-critic algorithm with function approximation for large scale RL, for when we have access to samples from a baseline policy (off-policy sampling or imitation learning) and samples from the constraint marginals.

2 Notation and Preliminaries

For any finite set $\mathcal{X}$, let $\mathcal{M}(\mathcal{X})$ be the set of probability distributions on $\mathcal{X}$. Further we define $\delta_{\mathcal{X}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{X} \\ \infty & \text{otherwise} \end{cases}$.

For $p, \in \mathcal{M}(\mathcal{X})$, $\mathcal{H}(p)$ denotes p's *entropy*. For a convex function $\psi : \mathcal{X} \to \mathbb{R}$ with $\psi(1) = 0$, we define ψ-divergence: $D_\psi(p|q) = \sum_x \psi\left(\frac{p(x)}{q(x)}\right) q(x)$. In particular, for $\psi(x) = x\log(x)$, $D_\psi(p|q) = \mathrm{KL}(p|q)$ is the *KL divergence* between p, q.

2.1 Optimal Transport

Given measures $a \in \mathcal{M}(\mathcal{X})$, $b \in \mathcal{M}(\mathcal{Y})$ on two sets $\mathcal{X}$ and $\mathcal{Y}$, with a cost function $C : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, Kantorovich *Optimal Transportation* is the problem of finding stochastic optimal plan $\mu \in \mathcal{M}(\mathcal{X}\times\mathcal{Y})$: $\min_\mu \mathbb{E}_\mu[C(x,y)] + \delta_{\{a\}}\left(\mu\mathbf{1}_\mathcal{Y}\right) + \delta_{\{b\}}\left(\mu^T\mathbf{1}_\mathcal{X}\right)$.

Unbalanced Optimal Transport replaces hard constraints $\delta_{\{a\}}$ and $\delta_{\{b\}}$, with penalty functions $\epsilon_1 D_{\psi_1}(\mu\mathbf{1}|a)$ and $\epsilon_2 D_{\psi_2}(\mu^T\mathbf{1}|b)$, where ϵ_1, ϵ_2 are positive scalars. This formulation also extends OT to measures of arbitrary mass. As $\epsilon_1, \epsilon_2 \to \infty$, the unbalanced OT approaches Kantorovich OT problem [LMS17].

To reduce the computational costs, an entropy term $\mathcal{H}(\mu)$ is usually added to the (U)OT objective to apply scaling algorithms [Cut13, CPSV17]. When $\mathcal{X}$ and $\mathcal{Y}$ are large or continuous spaces, often only samples from a, b are accessible. Stochastic approaches usually add a relative entropy $\mathrm{KL}(\mu \mid a \otimes b)$, instead of $\mathcal{H}(\mu)$ in order to take advantage of the Fenchel dual of the (U)OT optimization and estimate the objective from samples out of a, b [ACPB16, SDF+18].

2.2 Reinforcement Learning

Consider a discounted MDP $(\mathcal{S}, \mathcal{A}, P, r, \gamma, p_0)$, with finite state space $\mathcal{S}$ and action space $\mathcal{A}$, transition model $P : \mathcal{S}\times\mathcal{A} \to \mathcal{M}(\mathcal{S})$, initial distribution $p_0 \in \mathcal{M}(\mathcal{S})$, deterministic reward function $r : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ and discount factor $\gamma \in [0, 1)$. Letting Π be the set of stationary policies on the MDP, for any policy $\pi : \mathcal{S} \to \mathcal{M}(\mathcal{A})$, we denote $P^\pi : \mathcal{S} \to \mathcal{M}(\mathcal{S})$ to be the induced Markov chain on S. In policy optimization, the objective is

$$\max_{\pi\in\Pi} \sum_{s,a} \rho^\pi(s)\pi(a|s)r(s,a), \tag{1}$$

where $\rho^\pi(s) = (1-\gamma)\sum_{t=0}^\infty \gamma^t \mathrm{Pr}(s_t = s|\pi)$ is the discounted stationary distribution of P^π. For a policy π, we define its *occupancy measure* $\mu^\pi \in \mathcal{M}(\mathcal{S}\times\mathcal{A})$, as $\mu^\pi(s,a) = \rho^\pi(s)\pi(a|s)$. Let $\Delta := \{\mu^\pi : \pi \in \Pi\}$ be the set of occupancy measures of Π, the following lemma bridges the two sets Π and Δ: 233

Lemma 2.1. [SBS08][Theorem 2, Lemma2]

(i) $\Delta = \{\mu \in \mathcal{M}(\mathcal{S}\times\mathcal{A}) : \sum_a \mu(s,a) = (1-\gamma)p_0(s) + \gamma \sum_{s',a'} P(s|s',a')\mu(s',a') \ \forall s\}$

(ii) $\pi^\mu(a|s) := \mu(s,a)/\sum_a \mu(s,a)$ is a bijection from Δ to Π.

By multiplying $\pi = \pi^\mu$ to both sides of the equation in (i), one can obtain $\Delta = \{\mu : \mu \geq 0\,,\ A^\mu \mu = b^\mu\}$ where $A^\mu = \mathbb{I} - \gamma {P_\mu}^T$ and $b^\mu = (1-\gamma)\pi p_0$. In the rest of paper, we may drop the superscripts μ and π, when the context is clear. Rewriting the policy optimization objective (1) in terms of μ, we get $\max_{\mu\in\Delta} \mathbb{E}_\mu[r] = \max_\mu \mathbb{E}_\mu[r] - \delta_{\{b^\mu\}}(A^\mu\mu)$. Entropy-regularized version of this objective relative to a given baseline $\mu' \in \Delta$, is also studied [PMA10, NJG17]:

$$\max_{\mu\in\Delta} \mathbb{E}_\mu[r] - \epsilon\,\mathrm{KL}(\mu|\mu') \equiv \min_{\mu\in\Delta} \mathrm{KL}\left(\mu|\mu' e^{r/\epsilon}\right), \tag{2}$$

where ϵ is the regularization coefficient. By lemma 2.1, one can decompose the regularization term in (2) as $\mathrm{KL}(\mu|\mu') = \mathrm{KL}\left(\sum_a \mu|\sum_a \mu'\right) + \mathbb{E}_{\rho^\mu}\left[\mathrm{KL}\left(\pi|\pi'\right)\right]$, with the first term penalizing the shift on state distributions and the second penalty is over average shift of policies for every state. Since the goal is to optimize for the best policy, one might consider only regularizing relative to π' as in [SLA+15, NJG17]

$$\max_{\mu\in\Delta} \mathbb{E}_\mu[r] - \epsilon\,\mathbb{E}_{\rho^\mu}\left[\mathrm{KL}(\pi|\pi')\right]. \tag{3}$$

One can also regularize the objective by $\mathcal{H}(\mu)$ as

$$\max_{\mu\in\Delta} \mathbb{E}_\mu[r] - \epsilon\mathcal{H}(\mu) \equiv \min_{\mu\in\Delta} \mathrm{KL}(\mu|e^{r/\epsilon}), \tag{4}$$

which encourages exploration and avoids premature convergence [HTAL17].

3 A General RL Objective with Bregman Divergence

In this section, we propose a general RL objective based on Bregman divergence and optimize using Dykstra's algorithm.

Let Γ be a strictly convex, smooth function on relint(dom(Γ)), the relative interior of its domain, with convex conjugate Γ^*. For any $(\mu,\xi) \in \mathrm{dom}(\Gamma)\times\mathrm{int}(\mathrm{dom}(\Gamma))$, we define the Bregman divergence $D_\Gamma(\mu|\xi) = \Gamma(\mu) - \Gamma(\xi) - \langle\nabla\Gamma(\xi), \mu-\xi\rangle$. Given ξ, we consider the optimization

$$\min_\mu D_\Gamma(\mu|\xi) + \sum_i^N \phi_i(\mu), \tag{5}$$

where ϕ_i's are proper, lower-semicontinuous convex functions satisfying $\cap_i^N \mathrm{relint}(\mathrm{dom}(\phi_i)) \cap \mathrm{relint}(\mathrm{dom}(\Gamma)) \neq \emptyset$. Let dom($\Gamma$) be the simplex on $\mathcal{S}\times\mathcal{A}$, regularized RL algorithms in Section 2.2 can be observed as instances of optimization (5):

• Given a baseline μ', setting $\Gamma = \mathcal{H}$, $\phi_1(\mu) = \delta_{\{b^\mu\}}(A^\mu\mu)$, $\xi = \mu' e^{r/\epsilon}$, recovers objective (2).

• Similarly, as discussed in [NJG17], $\Gamma(\mu) = \sum_{s,a}\mu(s,a)\log\mu(s,a)/\sum_a\mu(s,a)$, $\phi_1 = \mathbb{E}_\mu[r/\epsilon]$, $\phi_2(\mu) = \delta_{\{b^\mu\}}(A^\mu\mu)$, and $\xi = \mu'$ recovers objective (3).

• Further, $\Gamma = \mathcal{H}$, $\phi_1(\mu) = \delta_{\{b^\mu\}}(A^\mu\mu)$, and $\xi = e^{r/\epsilon}$, entropy-regularizes the occupancy measure in objective (4).

The motivation behind using Bregman divergence is to generalize the KL divergence regularization usually used in RL algorithms. Moreover, one may replace the Bregman divergence term in (5) with a ψ-Divergence and attempt deriving similar arguments for the rest of the paper.

3.1 Dykstra's Algorithm

In this section, we use Dykstra's algorithm [Dyk83] optimize objective (5). Dykstra is a method of iterative projections onto general closed convex constraint sets, which is well suited because the occupancy measure constraint is on a compact convex polytope Δ. Defining the Proximal map of a convex function ϕ, with respect to D_Γ, as $\mathrm{Prox}_\phi^{D_\Gamma}(\mu) = \arg\min_{\tilde\mu} D_\Gamma(\tilde\mu|\mu) + \phi(\tilde\mu)$, for any $\mu \in \mathrm{dom}(\Gamma)$, we present the following proposition which is the generalization of Dykstra algorithm in [Pey15]:

Proposition 3.1 (Dykstra's algorithm)**.** For iteration $l > 0$,

$$\mu^{(l)} = \mathrm{Prox}_{\phi_{[l]_N}}^{D_\Gamma}\left(\nabla\Gamma^*\left(\nabla\Gamma(\mu^{(l-1)})\right) + \nu^{(l-N)}\right),\ \nu^{(l)} = \nu^{(l-N)} + \nabla\Gamma(\mu^{(l-1)}) - \nabla\Gamma(\mu^{(l)}), \tag{6}$$

with $[l]_N = \begin{cases} N & \text{if } l \bmod N = 0 \\ l \bmod N & \text{otherwise} \end{cases}$, converges to the solution of optimization (5), with $\mu^{(0)} = \xi$ and $\nu^{(i)} = \mathbf{0}$ for $-N+1 \leq i \leq 0$.

Intuitively, at step l, algorithm (6) projects $\mu^{(l-1)}$ into the convex constraint set corresponding to the function $\phi_{[l]_N}$.

As aforementioned RL objectives in Section 2.2 can be viewed as instances of optimization (5), Dykstra's algorithm can be used to optimize the objectives. In particular, as the constraint $\phi_N(\mu) = \delta_{\{b^\mu\}}(A^\mu \mu)$ occurs in each objective, each iteration of Dykstra requires

$$\text{Prox}_{\phi_N}^{D_\Gamma}(\mu) = \arg\min_{\tilde{\mu} \in \Delta} D_\Gamma(\tilde{\mu}|\mu), \tag{7}$$

which is the Bregman projection of μ onto the set of occupancy measures Δ. In the next section we apply Dykstra to solve unbalanced optimal transport on Δ.

4 Distributionally-Constrained Policy Optimization

A natural constraint in policy optimization is to enforce a global action execution allotment and/or state visitation frequency. In particular, given a positive baseline measure η', with $\eta'(a)$ being a rough execution allotment of action a over whole state space, for $a \in \mathcal{A}$, we can consider $D_{\psi_1}\left(\mathbb{E}_{\rho^\pi}[\pi] \mid \eta'\right)$ as a global penalty constraint of policy π under its natural state distribution ρ^π. Similarly, the penalty $D_{\psi_2}\left(\rho^\pi \mid \rho'\right)$ enforces a cautious or exploratory constraint on the policy behavior by avoiding or encouraging visitation of some states according to a given positive baseline measure ρ' on $\mathcal{S}$.

So, given baseline measures ρ' on $\mathcal{S}$ and η' on $\mathcal{A}$, we define distributionally-constrained policy optimization:

$$\max_{\mu \in \Delta} \mathbb{E}_\mu[r] - \epsilon_1 D_{\psi_1}\left(\mu \mathbf{1} \mid \rho'\right) - \epsilon_2 D_{\psi_2}\left(\mu^T \mathbf{1} \mid \eta'\right). \tag{8}$$

When $D_{\psi_1} = D_{\psi_2} = \text{KL}$, objective (8) looks similar to (2),but they are different. In (8), if $\rho' = \mu' \mathbf{1}$ and $\eta' = \mu'^T \mathbf{1}$, for some baseline $\mu' \in \Delta$, then the third term is $\text{KL}\left(\mathbb{E}_{\rho^\pi}[\pi] \mid \mathbb{E}_{\rho^{\pi'}}[\pi']\right)$ which is a global constraint on center of mass of π over the whole state space, whereas $\mathbb{E}_{\rho^\pi}[\text{KL}(\pi \mid \pi')]$ is a stronger constraint on closeness of policies on every single state. Thus (8) generally constrains the projected marginals of μ over $\mathcal{S}$ and $\mathcal{A}$, and (2) constrains μ element wise.

For regularization purposes in iterative policy optimization algorithm (e.g., using mirror descent), one natural choice of the state and action marginals is to take $\rho' = \sum_a \mu_{k-1}$, $\eta' = \sum_s \mu_{k-1}$ at the k'th iteration. Another source for the marginals ρ', η' is the empirical visitation of states and actions sampled out of an expert policy in imitation learning.

Formulation of Objective (8) is in form of UOT on the set of occupancy measures. So, for applying Dykstra algorithm, we can add an entropy term $\epsilon \mathcal{H}(\mu)$ to transform (8) into

$$\max_{\mu \in \Delta} -\text{KL}(\mu \mid \xi) - \epsilon_1 D_{\psi_1}\left(\mu \mathbf{1} \mid \rho'\right) - \epsilon_2 D_{\psi_2}\left(\mu^T \mathbf{1} \mid \eta'\right), \tag{9}$$

which means setting $N = 3$, $\phi_1 = \epsilon_1 D_{\psi_1}, \phi_2 = \epsilon_2 D_{\psi_2}$, $\phi_3(\mu) = \delta_{b^\mu}(A^\mu \mu)$, $\xi = e^{r/\epsilon}$ in Objective (5) [1]. Hence, the Dykstra's algorithm can be applied with following proximal functions:

$$\text{Prox}_{\phi_1}^{\text{KL}}(\mu) = \text{diag}\left(\frac{\text{Prox}_{\epsilon_1 D_{\psi_1}}^{\text{KL}}(\mu \mathbf{1})}{\mu \mathbf{1}}\right)\mu, \text{Prox}_{\phi_2}^{\text{KL}}(\mu) = \mu\, \text{diag}\left(\frac{\text{Prox}_{\epsilon_2 D_{\psi_2}}^{\text{KL}}(\mu^T \mathbf{1})}{\mu^T \mathbf{1}}\right), \text{Prox}_{\phi_3}^{\text{KL}}(\mu) = \arg\min_{\tilde{\mu} \in \Delta} \text{KL}(\tilde{\mu}|\mu). \tag{10}$$

In general, for appropriate choices of ϕ_1, ϕ_2 (e.g., $D_{\psi_1} = D_{\psi_2} = \text{KL}$) the proximal operators ave closed form solutions. However, as discussed, ϕ_3 in (10) has no closed form solution. We can also consider other functions for ϕ_1, ϕ_2 in this scenario, for example, setting $\phi_2(\mu) = \delta_{\{\eta'\}}(\mu^T \mathbf{1})$, changes the problem into finding a policy π which globally matches the distribution η' under its natural state distribution ρ^π, i.e., $\mathbb{E}_{s \sim \rho^\pi}[\pi(a|s)] = \eta(a)$ for any $a \in \mathcal{A}$[2].

In the next section we propose an actor-critic algorithm for large scale reinforcement learning.

4.1 Large Scale RL

When $\mathcal{S}, \mathcal{A}$ are large, policy optimization via Dykstra is challenging because tabular updating of $\mu(s,a)$ is time consuming or sometimes even impossible. In addition, it requires knowledge of reward function r for the initial distribution ξ and transition model P for projection onto Δ in (10). We derive an off-policy optimization scheme with function approximation to address these problems.

[1] Coefficients ϵ_1 and ϵ_2 in equation (9) are different from those in equation (8).

[2] As a constraint, $\pi \rho^\pi$ would be a feasible solution, when $\pi(a|s) \propto \eta'(a)$.

Replacing the last two terms of obj. (8) with their convex conjugates by dual variables u, v, Q, we get

$$\max_{\mu} \min_{u,v,Q} \mathbb{E}_{\mu}\left[r - A^{\mu *}Q - \epsilon_1 u \mathbf{1}_{\boldsymbol{A}}^T - \epsilon_2 \mathbf{1}_{\boldsymbol{S}} v^T\right] + \mathbb{E}_{b^\mu}[Q(s,a)] + \epsilon_1 \mathbb{E}_{\rho'}[\psi_1^*(u(s))] + \epsilon_2 \mathbb{E}_{\eta'}[\psi_2^*(v(a))], \tag{11}$$

where $A^{\mu *}$ is the convex conjugate (transpose) of A^μ.

Having access to samples from a baseline $\mu' \in \Delta$, both helps regularize the objective (11) into an easier problem to solve, and allows off-policy policy optimization or imitation learning [ND20]. Yet, by the discussion in Section 4, regularizing with the term $D_\psi(\mu|\mu')$ in (11) might make marginal penalties redundant, in particular when $\rho' = \sum_a \mu'$ and $\eta' = \sum_s \mu'$.

When $\rho' \neq \sum_a \mu'$ **or** $\eta' \neq \sum_s \mu'$, without the loss of generality, assume marginals $\rho' \neq \sum_a \mu'$ and $\eta' \neq \sum_s \mu'$. In this case, regularizing (11) with $D_\psi(\mu|\mu')$ and under Fenchel duality, we get the off-policy optimization objective

$$\max_{\pi} \min_{u,v,Q} \mathbb{E}_{\mu'}\left[\psi^*\left(r + \gamma P^\pi Q - Q - \epsilon_1 u - \epsilon_2 v\right)(s,a)\right] + (1-\gamma)\mathbb{E}_{\pi,p_0}[Q(s,a)] + \epsilon_1 \mathbb{E}_{\rho'}[\psi_1^*(u(s))] + \epsilon_2 \mathbb{E}_{\eta'}[\psi_2^*(v(a))], \tag{12}$$

where $u(s,a) := u(s)$ and $v(s,a) := v(a)$. Now, the first term is based on expectation of baseline μ' and can be estimated from offline samples. In a special case, when $D_{\psi_1} = D_{\psi_2} =$ KL in objective (8) and we take $D_\psi =$ KL as well, similar derivations yield

$$\max_{\pi} \min_{u,v,Q} \log\mathbb{E}_{\mu'}\left[\exp\left(r + \gamma P^\pi Q - Q - \epsilon_1 u - \epsilon_2 v\right)(s,a)\right] + (1-\gamma)\mathbb{E}_{p_0,\pi}[Q(s,a)] + \epsilon_1 \log\mathbb{E}_{\rho'}[\exp(u(s))] + \epsilon_2 \log \mathbb{E}_{\eta'}[\exp(v(a))].$$

Also, when $\rho' = \sum_a \mu'$ **and** $\eta' = \sum_s \mu'$, following the approach in [ND20], with the change of variable $\zeta(s,a) := \frac{\mu}{\mu'}(s,a)$, we can rewrite (11) with importance sampling weights as a policy optimization problem

$$\max_{\pi,\zeta} \min_{u,v,Q} \mathbb{E}_{\mu'}\left[\zeta(s,a)\left(r + \gamma P^\pi Q - Q - \epsilon_1 u - \epsilon_2 v\right)(s,a)\right] + (1-\gamma)\mathbb{E}_{p_0,\pi}[Q(s,a)] + \epsilon_1 \mathbb{E}_{\rho'}[\psi_1^*(u(s))] + \epsilon_2 \mathbb{E}_{\eta'}[\psi_2^*(v(a))].$$

We tested efficacy of our proposed algorithm on the grid-world problem. We also performed an ablation study in an imitation learning scenario on the cart-pole problem. Results show our algorithm is able to imitate a policy with an enforced global action distribution post-learning while maximizing the accumulated rewards. Results are omitted due to the paper length constraints.

References

[ACPB16] Genevay Aude, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport, 2016.

[CPSV17] Lenaic Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. Scaling algorithms for unbalanced transport problems, 2017.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances, 2013.

[Dyk83] Richard L Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983.

[HTAL17] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017.

[LMS17] Matthias Liero, Alexander Mielke, and Giuseppe Savaré. Optimal entropy-transport problems and a new hellinger–kantorovich distance between positive measures. *Inventiones mathematicae*, 211(3):969–1117, Dec 2017.

[ND20] Ofir Nachum and Bo Dai. Reinforcement learning via fenchel-rockafellar duality, 2020.

[NJG17] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

[Pey15] Gabriel Peyré. Entropic approximation of wasserstein gradient flows. *SIAM Journal on Imaging Sciences*, 8(4):2323–2351, 2015.

[PMA10] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[SBS08] Umar Syed, Michael Bowling, and Robert E Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039, 2008.

[SDF+18] Vivien Seguy, Bharath Bhushan Damodaran, Rémi Flamary, Nicolas Courty, Antoine Rolet, and Mathieu Blondel. Large-scale optimal transport and mapping estimation, 2018.

[SLA+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[Vil08] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Zero-Sum Stochastic Stackelberg Games

Denizalp Goktas
Department of Computer Science
Brown University
Providence, RI 02906
denizalp_goktas@brown.edu

Jiayi Zhao
Pomona College
Pomona, CA
jzae2019@mymail.pomona.edu

Amy Greenwald
Department of Computer Science
Brown University
Providence, RI 02906
amy_greenwald@brown.edu

Abstract

Min-max optimization problems (i.e., zero-sum games) have been used to model problems in a variety of fields in recent years, from machine learning to economics. The literature to date has mostly focused on static zero-sum games, assuming independent strategy sets. In this paper, we study a form of dynamic zero-sum games, called stochastic games, with dependent strategy sets. Just as zero-sum games with dependent strategy sets can be interpreted as zero-sum Stackelberg games, stochastic zero-sum games with dependent strategy sets can be interpreted as zero-sum stochastic Stackelberg games. We prove the existence of an optimal solution in zero-sum stochastic Stackelberg games (i.e., a recursive Stackelberg equilibrium), and show that a recursive Stackelberg equilibrium can be computed in polynomial time via value iteration. Finally, we show that stochastic Stackelberg games can model the problem of pricing and allocating goods across agents and time; more specifically, we propose a stochastic Stackelberg game whose solutions correspond to a recursive competitive equilibrium in a stochastic Fisher market. We close with a series of experiments which confirm our theoretical results and show how value iteration performs in practice.

Keywords: Equilibrium Computation; Stackelberg Games; Market Equilibrium; Stochastic games

Min-max optimization has paved the way for recent progress in a variety of fields, from machine learning to economics. These applications require computing solutions to **the min-max operator**, i.e., solving a **constrained min-max optimization problem**, also known as zero-sum games (with independent strategy sets): i.e., $\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{x},\boldsymbol{y})$, where the objective function $f : \mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ is continuous, and the strategy sets $\mathcal{X}\subset\mathbb{R}^n$ and $\mathcal{Y}\subset\mathbb{R}^m$ are nonempty and compact. When f is convex-concave, the seminal minimax theorem [14, 16] holds, and such a problem can be interpreted as a simultaneous-move zero-sum game between an outer player $\boldsymbol{x}$ and an inner player $\boldsymbol{y}$, with the solutions $(\boldsymbol{x}^*,\boldsymbol{y}^*)\in\mathcal{X}\times\mathcal{Y}$ of the min-max operator corresponding to a Nash equilibrium. More generally, one can consider **zero-sum stochastic games** (with independent strategy sets) $\mathcal{X}\subset\mathbb{R}^n$ and $\mathcal{Y}\subset\mathbb{R}^m$, nonempty and compact, and a state-dependent payoff function $r(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})$, for all $\boldsymbol{s}\in\mathcal{S}$, where the players seek to optimize their cumulative (discounted) payoffs, in expectation. When $r(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})$ is bounded, continuous, and concave-convex in $(\boldsymbol{x},\boldsymbol{y})$, for all $\boldsymbol{s}\in\mathcal{S}$, these games are guaranteed to have a unique solution [15],[1] while the **optimal policies**, i.e., the per-state collection of solutions to the min-max operator can be interpreted as a **recursive Nash equilibrium** of a zero-sum stochastic game, and can be computed in polynomial time by iterative application of the min-max operator [15]. Zero-sum *stochastic* games generalize zero-sum games from a single state to multiple states, and have found even more applications in a variety of fields [11].

Recently, Goktas and Greenwald [10] studied the computation of a **generalized min-max operator** i.e., solving a **constrained min-max game with dependent strategy sets**: i.e., $\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}:h(\boldsymbol{x},\boldsymbol{y})\geq\boldsymbol{0}} r(\boldsymbol{x},\boldsymbol{y})$ where, in addition to the aforementioned assumptions, $h : \mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ is continuous. Goktas and Greenwald have shown that even more problems of interest can be captured by solutions to the generalized min-max operator. However, a minimax theorem is not guaranteed to hold in this setting; as a result, these problems are best interpreted as zero-sum sequential, i.e., Stackelberg [17], games, and their solutions, as Stackelberg equilibria. One can likewise consider **zero-sum stochastic games with dependent feasible strategy sets** $\mathcal{X}\subset\mathbb{R}^n$ and $\mathcal{Y}\subset\mathbb{R}^m$, nonempty and compact and a state-dependent payoff function $r(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})$, as well as a state dependent constraint function $\boldsymbol{g}(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})$, for all $\boldsymbol{s}\in\mathcal{S}$, where the players seek to optimize their cumulative (discounted) payoffs, in expectation, while satisfying the constraint $\boldsymbol{g}(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})\geq\boldsymbol{0}$ at each state $\boldsymbol{s}\in\mathcal{S}$. Such a problem is best interpreted as a zero-sum stochastic Stackelberg game, for which the appropriate solution concept is the **recursive Stackelberg equilibrium (recSE)**. Although very little is known about the properties of such problems, they generalize both min-max games with dependent strategy sets and zero-sum stochastic games (with independent strategy sets), and, as we show, have novel applications.

In this paper, we prove the existence of recSE in zero-sum stochastic Stackelberg games, and show that a recSE can be computed in (weakly) polynomial time via value iteration. We further show that stochastic Stackelberg games can be used to solve problems of pricing and allocating goods across agents and time. In particular, we introduce **stochastic Fisher markets**, a stochastic generalization of the Fisher market [7], and a special case of Friesen's [9] financial market model, which itself is a stochastic generalization of the Arrow and Debreu model of a competitive economy [1]. We then prove the existence of recursive competitive equilibrium [13] in this model, under the assumption that consumers have continuous and homogeneous utility functions, by characterizing the competitive equilibria of any stochastic Fisher market as the Stackelberg equilibria of the corresponding stochastic Stackelberg game. Finally, we use value iteration to solve various stochastic Fisher markets, highlighting the issues that value iteration might face as a consequence of the smoothness properties of the utility functions.

1 Preliminaries

Notation We use caligraphic uppercase letters to denote sets (e.g., $\mathcal{X}$), bold uppercase letters to denote matrices (e.g., $\boldsymbol{X}$), bold lowercase letters to denote vectors (e.g., $\boldsymbol{p}$), bold uppercase letters to denote vector-valued random variables (e.g., $\boldsymbol{\Gamma}$), lowercase letters to denote scalar quantities, (e.g., x), and uppercase letters to denote scalar-valued random variables (e.g., X). We denote the ith row vector of a matrix (e.g., $\boldsymbol{X}$) by the corresponding bold lowercase letter with subscript i (e.g., $\boldsymbol{x}_i$). Similarly, we denote the jth entry of a vector (e.g., $\boldsymbol{p}$ or $\boldsymbol{x}_i$) by the corresponding Roman lowercase letter with subscript j (e.g., p_j or x_{ij}). We denote functions by a letter: e.g., f if the function is scalar valued, and $\boldsymbol{f}$ if the function is vector valued. We denote the vector of ones of size n by $\mathbf{1}_n$. We denote the set of integers $\{1,\dots,n\}$ by $[n]$, the set of natural numbers by $\mathbb{N}$, the set of real numbers by $\mathbb{R}$. We denote the postive and strictly positive elements of a set by a $+$ and $++$ subscript respectively, e.g., $\mathbb{R}_+$ and $\mathbb{R}_{++}$. We denote the orthogonal projection operator onto a set C by Π_C, i.e., $\Pi_C(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in C} \|\boldsymbol{x}-\boldsymbol{y}\|^2$. We denote by $\Delta_n = \{\boldsymbol{x}\in\mathbb{R}^n_+ \mid \sum_{i=1}^n x_i = 1\}$, and by $\Delta(A)$ the set of probability measures on the set A.

A **stochastic Stackelberg game** $(\mathcal{S},\mathcal{X},\mathcal{Y},\mu^{(0)},r_{\boldsymbol{x}},r_{\boldsymbol{y}},\boldsymbol{g},p,\gamma)$ is a dynamic two-player sequential, i.e., Stackelberg, game where one player we call the outer-player (resp. inner-player) moves through a set of states $\mathcal{S}$ picking an action to play from their continuous set of actions $\mathcal{X}\subset\mathbb{R}^n$ (resp. $\mathcal{Y}\subset\mathbb{R}^m$). Players start the game at an initial state determined by an initial state distribution $\mu^{(0)} : \mathcal{S}\to[0,1]$ s.t. for all states $\boldsymbol{s}\in\mathcal{S}$, $\mu^{(0)}(\boldsymbol{s})\geq 0$ denotes the probability of the game being initialized at state $\boldsymbol{s}$. At each state $\boldsymbol{s}\in\mathcal{S}$ the action $\boldsymbol{x}\in\mathcal{X}$ chosen by the outer player determines the set of **feasible** actions $\{\boldsymbol{y}\in\mathcal{Y} \mid \boldsymbol{g}(\boldsymbol{s},\boldsymbol{x},\boldsymbol{y})\geq\boldsymbol{0}\}$ that the inner player can in turn play. Once the outer and inner players make their moves, they receive payoffs $r_{\boldsymbol{x}} : \boldsymbol{s}\times\mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ and $r_{\boldsymbol{y}} : \mathcal{S}\times\mathcal{X}\times\mathcal{Y}\to\mathbb{R}$, respectively, and the game either ends with probability $1-\gamma$, where $\gamma\in(0,1)$ is called the **discount factor**, or transitions to a new state $\boldsymbol{s}'\in\mathcal{S}$, according to a **transition** probability function $p : \mathcal{S}\times\mathcal{S}\times\mathcal{X}\times\mathcal{Y}\to[0,1]$ s.t. $p(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{x},\boldsymbol{y})\in[0,1]$

[1] Although Shapley's original results concern payoffs which are bilinear in the outer and inner players' actions, they extend directly to payoffs which are convex-concave in the players' actions.

denotes the probability of transitioning to state $\boldsymbol{s}' \in \mathcal{S}$ from state $\boldsymbol{s} \in \mathcal{S}$ when a strategy profile $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ is chosen by the players.

In this paper, we focus on stochastic Stackelberg **zero-sum** games $\mathcal{G}^{(0)} = (\mathcal{S}, \mathcal{X}, \mathcal{Y}, \mu^{(0)}, r, \boldsymbol{g}, p, \gamma)$, in which the outer player's loss is the inner player's gain, i.e., $r_{\boldsymbol{x}} = -r_{\boldsymbol{y}}$. We say that a zero-sum stochastic Stackelberg game is **convex-concave** if, for all $\boldsymbol{s} \in \mathcal{S}$, $r(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}), g_1(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}), \ldots, g_d(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y})$ are convex-concave in $(\boldsymbol{x}, \boldsymbol{y})$. A zero-sum stochastic Stackelberg game reduces to zero-sum stochastic game [15] if for all state-action tuples $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S} \times \mathcal{X} \times \mathcal{Y}$, $\boldsymbol{g}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) \geq 0$. As we will show, it suffices to focus on deterministic policies in which a **policy** for the outer (resp. inner) player is a mapping from states to actions $\boldsymbol{\pi_x} : \mathcal{S} \to \mathcal{X}$ (resp. $\boldsymbol{\pi_y} : \mathcal{S} \to \mathcal{Y}$). The **outcome** of a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$ is a policy profile $(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \in \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}}$ consisting of policies for the outer and inner players, respectively. An outcome $(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \in \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}}$ is said to be **feasible** if, for all states $\boldsymbol{s} \in \mathcal{S}$, $\boldsymbol{g}(\boldsymbol{s}, \boldsymbol{\pi_x}(\boldsymbol{s}), \boldsymbol{\pi_y}(\boldsymbol{s})) \geq 0$. For simplicity, we introduce a function $\boldsymbol{G} : \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}} \to \mathbb{R}^{|\mathcal{S}| \times d}$ such that $\boldsymbol{G}(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) = (\boldsymbol{g}(\boldsymbol{s}, \boldsymbol{\pi_x}(\boldsymbol{s}), \boldsymbol{\pi_y}(\boldsymbol{s})))_{\boldsymbol{s} \in \mathcal{S}}$, and define feasible outcomes as those $(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \in \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}}$ s.t. $\boldsymbol{G}(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \geq \boldsymbol{0}$. For the rest of this paper, we assume:

Assumption 1.1. *1. For all states $\boldsymbol{s} \in \mathcal{S}$, the functions $r(\boldsymbol{s}, \cdot, \cdot), g_1(\boldsymbol{s}, \cdot, \cdot), \ldots, g_d(\boldsymbol{s}, \cdot, \cdot)$ are continuous in $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ with payoffs r bounded, i.e., $\|r\|_\infty \leq \alpha < \infty$, for some $\alpha \in \mathbb{R}_+$, and 2 $\mathcal{X}, \mathcal{Y}$ are non-empty and compact.*

Given a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$, the **state-value function**, $v : \mathcal{S} \times \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}} \to \mathbb{R}$, and the **action-value function**, $q : \mathcal{S} \times \mathcal{X} \times \mathcal{Y} \times \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}} \to \mathbb{R}$ are respectively defined as: $v(\boldsymbol{s}; \boldsymbol{\pi_x}, \boldsymbol{\pi_y}) = \mathbb{E}^{\boldsymbol{\pi_x}, \boldsymbol{\pi_y}}_{\boldsymbol{S}^{(t+1)} \sim p(\cdot | \boldsymbol{S}^{(t)}, \boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)})} \left[\sum_{t=0}^{\infty} (1-\gamma)\gamma^t r(\boldsymbol{S}^{(t)}, \boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)}) \mid \boldsymbol{S}^{(0)} = \boldsymbol{s}\right]$; $q(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\pi_x}, \boldsymbol{\pi_y}) = \mathbb{E}^{\boldsymbol{\pi_x}, \boldsymbol{\pi_y}}_{\boldsymbol{S}^{(t+1)} \sim p(\cdot | \boldsymbol{S}^{(t)}, \boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)})} \left[\sum_{t=0}^{\infty} (1-\gamma)\gamma^t r(\boldsymbol{S}^{(t)}, \boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)}) \mid \boldsymbol{S}^{(0)} = \boldsymbol{s} \;\; \boldsymbol{X}^{(0)} = \boldsymbol{x}, \boldsymbol{Y}^{(0)} = \boldsymbol{y}\right]$. For clarity, we write expectations conditional on $\boldsymbol{X}^{(t)} = \boldsymbol{\pi_x}(\boldsymbol{S}^{(t)})$ and $\boldsymbol{Y}^{(t)} = \boldsymbol{\pi_y}(\boldsymbol{S}^{(t)})$ as $\mathbb{E}^{\boldsymbol{\pi_x}, \boldsymbol{\pi_y}}$, and denote the state- and action-value functions by $v^{\boldsymbol{\pi_x}\boldsymbol{\pi_y}}(\boldsymbol{s})$, and $q^{\boldsymbol{\pi_x}\boldsymbol{\pi_y}}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y})$, respectively. Additionally, we let $\mathcal{V} = [-\alpha, \alpha]^{\mathcal{S}}$ be the space of all state-value functions of the form $v : \mathcal{S} \to [-\alpha, \alpha]$, and we let $\mathcal{Q} = [-\alpha, \alpha]^{\mathcal{S} \times \mathcal{X} \times \mathcal{Y}}$ be the space of all action-value functions of the form $q : \mathcal{S} \times \mathcal{X} \times \mathcal{Y} \to [-\alpha, \alpha]$. Note that by Assumption 1.1 the range of the state- and action-value functions is $[-\alpha, \alpha]$. The cumulative payoff function of the game $u : \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}} \to \mathbb{R}$ is the total expected loss (resp. gain) of the outer (resp. inner) player is then given by $u(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) = \mathbb{E}_{\boldsymbol{s} \sim \mu^{(0)}(\boldsymbol{s})} \left[v^{\boldsymbol{\pi_x}\boldsymbol{\pi_y}}(\boldsymbol{s})\right]$.

Definition 1.2. *A feasible policy profile $(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}}) \in \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}}$ is a **recursive Stackelberg equilibrium (recSE)** of a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$ iff $\max_{\boldsymbol{\pi_y} \in \mathcal{Y}^{\mathcal{S}} : \boldsymbol{G}(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi_y}) \geq \boldsymbol{0}} u\left(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi_y}\right) \leq u\left(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}}\right) \leq \min_{\boldsymbol{\pi_x} \in \mathcal{X}^{\mathcal{S}}} \max_{\boldsymbol{\pi_y} \in \mathcal{Y}^{\mathcal{S}} : \boldsymbol{G}(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \geq \boldsymbol{0}} u\left(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}\right)$.*

Mathematical Preliminaries A mapping $L : \mathcal{A} \to \mathcal{B}$ is said to be a **contraction mapping** (resp. **non-expansion**) w.r.t. norm $\|\cdot\|$ iff for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}$, and for $k \in [0, 1)$ (resp. $k = 1$) such that $\|L(\boldsymbol{x}) - L(\boldsymbol{y})\| \leq k \|\boldsymbol{x} - \boldsymbol{y}\|$.

2 Properties of Recursive Stackelberg equilibrium

We first state a necessary condition that the state-value function induced by a policy profile needs to satisfy in order for that policy profile to be a recSE. It is not directly clear if there exists a policy for any zero-sum stochastic Stackelberg game which satisfies this necessary condition since there is no way to guarantee that such an inequality will hold at all states.[2]

Lemma 2.1. *Consider a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$. A policy profile $(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}}) \in \mathcal{S}^{\mathcal{X}} \times \mathcal{S}^{\mathcal{Y}}$ is a recSE if, for all $\boldsymbol{s} \in \mathcal{S}$, $\max_{\boldsymbol{y} \in \mathcal{Y} : \boldsymbol{g}(\boldsymbol{s}, \boldsymbol{\pi}^*_{\boldsymbol{x}}(\boldsymbol{s}), \boldsymbol{y}) \geq 0} q^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}(\boldsymbol{s}, \boldsymbol{\pi}^*_{\boldsymbol{x}}(\boldsymbol{s}), \boldsymbol{y}) \leq q^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}(\boldsymbol{s}, \boldsymbol{\pi}^*_{\boldsymbol{x}}(\boldsymbol{x}), \boldsymbol{\pi}^*_{\boldsymbol{y}}(\boldsymbol{y})) \leq \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y} : \boldsymbol{g}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) \geq \boldsymbol{0}} q^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y})$. Equivalently, a policy profile $(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}})$ is a recSE if $(\boldsymbol{\pi}^*_{\boldsymbol{x}}(\boldsymbol{s}), \boldsymbol{\pi}^*_{\boldsymbol{y}}(\boldsymbol{s}))$ is a Stackelberg equilibrium: i.e., a solution to $\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y} : \boldsymbol{g}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) \geq \boldsymbol{0}} q^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y})$ at each $\boldsymbol{s} \in \mathcal{S}$.*

We define an operator $C : \mathcal{V} \to \mathcal{V}$ associated with a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$ whose fixed points satisfy the condition given in Lemma 2.1, and hence correspond to the value function associated with a recSE of $\mathcal{G}^{(0)}$. We then show that this operator is a contraction mapping, thereby establishing the existence of such a fixed point. This result generalizes a result first shown by [15] for zero-sum stochastic games, i.e., zero-sum stochastic Stackelberg games in which $\boldsymbol{G}(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \geq \boldsymbol{0}$, for all $(\boldsymbol{\pi_x}, \boldsymbol{\pi_y}) \in \mathcal{X}^{\mathcal{S}} \times \mathcal{Y}^{\mathcal{S}}$. Define $C : \mathcal{V} \to \mathcal{V}$ for a stochastic Stackelberg game $\mathcal{G}^{(0)}$ as the operator $(Cv)(\boldsymbol{s}) = \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y} : \boldsymbol{g}(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) \geq \boldsymbol{0}} \mathbb{E}_{\boldsymbol{S}' \sim p(\cdot | \boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y})} \left[r(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}) + \gamma v(\boldsymbol{S}')\right]$, we have the following two results.

Theorem 2.2. *$(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}})$ is a recSE of $\mathcal{G}^{(0)}$ of $v^{\boldsymbol{\pi_x}\boldsymbol{\pi_y}}$ iff it induces a value function which is a fixed point of C: i.e., $(\boldsymbol{\pi}^*_{\boldsymbol{x}}, \boldsymbol{\pi}^*_{\boldsymbol{y}})$ is a Stackelberg equilbrium iff, for all $\boldsymbol{s} \in \mathcal{S}$, $\left(Cv^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}\right)(\boldsymbol{s}) = v^{\boldsymbol{\pi}^*_{\boldsymbol{x}}\boldsymbol{\pi}^*_{\boldsymbol{y}}}(\boldsymbol{s})$.*

Theorem 2.3. *Consider the operator C associated with a stochastic Stackelberg game $\mathcal{G}^{(0)}$. If Assumption 1.1 holds, then C is a contraction mapping w.r.t. to the sup norm $\|.\|_\infty$ with constant γ.*

[2] All omitted results and proofs can be found in the full version. 239

Given some initial state-value function $v^{(0)} \in \mathcal{V}$, we define the **value iteration** process as $v^{(t+1)} = Cv^{(t)}$, for all $t \in \mathbb{N}_+$ (Algorithm 1). One way to interpret $v^{(t)}$ is as the function that returns the value $v^{(t)}(\boldsymbol{s})$ of each state $\boldsymbol{s} \in \mathcal{S}$ in the t stage zero-sum Stackelberg game starting at the last stage t and continuing until stage 0, with terminal payoffs given by $v^{(0)}$. The following theorem, which is a consequence of Theorems 2.2 and 2.3, not only states the existence of a recSE but also provides us with a means of computing a recSE via value iteration.

Theorem 2.4. *Consider a zero-sum stochastic Stackelberg game $\mathcal{G}^{(0)}$. If Assumption 1.1 holds, then $\mathcal{G}^{(0)}$ has at least one recSE $(\boldsymbol{\pi}_{\boldsymbol{x}}^*, \boldsymbol{\pi}_{\boldsymbol{y}}^*)$ with unique value function $v^{\boldsymbol{\pi}_{\boldsymbol{x}}^*\boldsymbol{\pi}_{\boldsymbol{y}}^*}$. Further, $v^{\boldsymbol{\pi}_{\boldsymbol{x}}^*\boldsymbol{\pi}_{\boldsymbol{y}}^*}$ can be computed by iteratively applying C to any initial state-value function $v^{(0)} \in \mathcal{V}$: $\lim_{t\to\infty} v^{(t)} = v^{\boldsymbol{\pi}_{\boldsymbol{x}}^*\boldsymbol{\pi}_{\boldsymbol{y}}^*}$.*

Remark 2.5. *Unlike Shapley's existence theorem for recursive Nash equilibria in zero-sum stochastic games, the above theorem does not require that the payoff function be convex-concave. The only conditions needed are continuity of the payoffs and constraints, and bounded payoffs. This makes the recSE a potentially useful solution concept, even for non-convex-non-concave stochastic games.*

Note that the proof of Theorem 2.4 shows that there exists a policy profile which satisfies the conditions of Lemma 2.1. Since this recSE definition is independent of the initial state distribution, we can infer that the recSE of any zero-sum Stackelberg game $\mathcal{G}^{(0)} = (\mathcal{S}, \mathcal{X}, \mathcal{Y}, \mu^{(0)}, r, \boldsymbol{g}, p, \gamma)$ is independent of the initial state distribution $\mu^{(0)}$. Hence, from now on, we denote a zero-sum Stackelberg game by $\mathcal{G}$. Theorem 2.4 tells us that value iteration converges to the value function associated with a recSE. Additionally, under Assumption 1.1, recSE is computable in (weakly) polynomial time.[3]

Theorem 2.6. *[Convergence of Value Iteration] Suppose value iteration is run on input $\mathcal{G}$. If Assumption 1.1 holds, and if we initialize $v^{(0)}(\boldsymbol{s}) = 0$, for all $\boldsymbol{s} \in \mathcal{S}$, then for $k \geq \frac{1}{1-\gamma} \log \frac{2\alpha}{\epsilon(1-\gamma)}$, we have $v^{(k)}(\boldsymbol{s}) - v^{\boldsymbol{\pi}_{\boldsymbol{x}}^*\boldsymbol{\pi}_{\boldsymbol{y}}^*}(\boldsymbol{s}) \leq \epsilon$.*

3 Recursive Market Equilibrium

We now introduce an application of zero-sum stochastic Stackelberg games, which generalizes a well known market model, the Fisher market [7], to a dynamic setting in which buyers not only participate in markets across time, but their wealth persists. A **(static) Fisher market** consists of n buyers and m divisible goods [7]. Each buyer $i \in [n]$ is endowed with a budget $b_i \in \mathcal{B}_i \subset \mathbb{R}_+$ and a utility function $u_i : \mathbb{R}_+^m \times \mathcal{T}_i \to \mathbb{R}$, which is parameterized by a type $t_i \in \mathcal{T}_i$ that defines a preference relation over the consumption space $\mathbb{R}_+^m$. Each good is characterized by a supply $q_j \in \mathcal{Q}_j \subset \mathbb{R}_+$. A **stochastic Fisher market** is a dynamic market in which each state corresponds to a static Fisher market: i.e., each state $\boldsymbol{s} \in \mathcal{S}$ is characterized by a tuple $(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q})$. In each state, the buyers choose their allocations $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T \in \mathbb{R}_+^{n\times m}$ and the market determines prices, after which the market comes to an end with probability $(1-\gamma)$, or it moves into a new state with probability $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{X})$.[4] A **stochastic Fisher market with savings** is a stochastic Fisher market in which, at each state, each buyer i, in addition to choosing their allocation, can set aside some **savings** $\beta_i \in \mathbb{R}_+$ to spend at some future state, and transitions depend on their savings choices as well: i.e., $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{X}, \boldsymbol{\beta})$. A **stochastic Fisher market with savings** is denoted by $(\mathcal{S}, \mathcal{U}, \boldsymbol{b}^{(0)}, p, \gamma)$. Given such a market, a **recursive competitive equilibrium (recCE)** [13] is a tuple $(\boldsymbol{X}^*, \boldsymbol{\beta}^*, \boldsymbol{p}^*) \in \mathbb{R}_+^{n\times m\times\mathcal{S}} \times \mathbb{R}_+^{n\times\mathcal{S}} \times \mathbb{R}_+^{m\times\mathcal{S}}$, which consists of an allocation, savings, and price system s.t. 1) the buyers are expected utility maximizing, constrained by their savings and spending constraints, i.e., for all buyers $i \in [n]$, $(\boldsymbol{x}_i^*, \beta_i^*)$ is the optimal policy that, for all states $(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) \in \mathcal{S}$, solves the Bellman equation $\nu_i(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) = \max_{(\boldsymbol{x}_i, \beta_i) \in \mathbb{R}_+^{m+1} : \boldsymbol{x}_i \cdot \boldsymbol{p}^*(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) + \beta_i \leq b_i} \left\{ u_i(\boldsymbol{x}_i, t_i) + \gamma\, \mathbb{E}_{(\boldsymbol{t}', \boldsymbol{b}', \boldsymbol{q}') \sim p(\cdot \mid \boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}, (\boldsymbol{x}_i, \boldsymbol{X}_{-i}^*(\boldsymbol{s})), (\beta_i, \boldsymbol{\beta}_{-i}^*(\boldsymbol{s})))} \left[\nu_i(\boldsymbol{t}', \boldsymbol{b}' + \beta_i, \boldsymbol{q}') \right] \right\}$, where $\boldsymbol{X}_{-i}^*, \boldsymbol{\beta}_{-i}^*$ denote the allocation and saving systems excluding buyer i, and 2) the market clears in each state so that unallocated goods in each state are priced at 0, i.e., for all $j \in [m]$ and $\boldsymbol{s} \in \mathcal{S}$, $p_j^*(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) > 0 \implies \sum_{i\in[n]} x_{ij}^*(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) = 1$ and $p_j^*(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) \geq 0 \implies \sum_{i\in[n]} x_{ij}^*(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) \leq 1$. The following theorem shows that the recSE of a stochastic Fisher market with savings are in fact recursive competitive equilibria. We note that if one ignores the savings terms, then the recSE are recursive competitive equilibria of the same market without savings.

Theorem 3.1. *A stochastic Fisher market with savings $(\mathcal{S}, \mathcal{U}, \boldsymbol{b}^{(0)}, p, \gamma)$ in which $\mathcal{U}$ is a vector of continuous and homogeneous utility functions has at least one recCE. Additionally, the recSE $(\boldsymbol{p}^*, \boldsymbol{X}^*, \boldsymbol{\beta}^*)$ that solves the following Bellman equation corresponds to the recCE of $(\mathcal{S}, \mathcal{U}, \boldsymbol{b}^{(0)}, p, \gamma)$:*

$$v(\boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}) = \min_{\boldsymbol{p}\in\mathbb{R}_+^m} \max_{(\boldsymbol{X}, \boldsymbol{\beta}) \in \mathbb{R}_+^{n\times(m+1)} : \boldsymbol{X}\boldsymbol{p} + \boldsymbol{\beta} \leq \boldsymbol{b}} \sum_{j\in[m]} q_j p_j + \sum_{i\in[n]} (b_i - \beta_i) \log(u_i(\boldsymbol{x}_i, t_i)) + \gamma \mathop{\mathbb{E}}_{(\boldsymbol{t}', \boldsymbol{b}', \boldsymbol{q}') \sim p(\cdot \mid \boldsymbol{t}, \boldsymbol{b}, \boldsymbol{q}, \boldsymbol{X}, \boldsymbol{\beta})} \left[v(\boldsymbol{t}', \boldsymbol{b}' + \boldsymbol{\beta}, \boldsymbol{q}') \right] \tag{1}$$

Remark 3.2. *This result could not be obtained by modifying the Lagrangian formulation, i.e., the simultaneous-move game form, of the Eisenberg-Gale program, because the saving problem is a convex-non-concave problem, and existence of recursive Nash equilibrium in deterministic policies requires convex-concavity of payoffs [11].*

[3] This convergence is only weakly polynomial time, since the computation of the generalized min-max operator applied to an arbitrary continuous function is an NP-hard problem; it is at least as hard as non-convex optimization. If, however, one restricts themselves to convex-concave stochastic Stackelberg games, Stackelberg equilibrium is computable in polynomial time, making the problem much more tractable.

[4] Note that, as is standard in the literature, we assume that prices do not determine the next state since market prices are set by a "fictional auctioneer." There is no market participant who determined prices! 240

4 Experiments

In order to better understand the iteration complexity of value iteration, and to understand how it performs in a continuous state space, we computed the recursive Stackelberg equilibria of various stochastic Fisher market with savings. We created markets with three different classes of utility functions, each of which endowed the state-value function with different smoothness properties.[5] Let $\boldsymbol{\theta}_i \in \mathbb{R}^m$ be a vector of parameters that describes the utility function of buyer $i \in [n]$. We considered the following (standard) utility function classes: 1. linear: $u_i(\boldsymbol{x}_i) = \sum_{j\in[m]} \theta_{ij} x_{ij}$; 2. Cobb-Douglas: $u_i(\boldsymbol{x}_i) = \prod_{j\in[m]} x_{ij}^{\theta_{ij}}$; and 3. Leontief: $u_i(\boldsymbol{x}_i) = \min_{j\in[m]} \left\{ \frac{x_{ij}}{\theta_{ij}} \right\}$. Since the state space is continuous, the value function is also continuous in stochastic Fisher markets. As a result, we used fitted value iteration, finding a fit via linear regression (e.g., [6]). To solve the generalized min-max operator at each step of value iteration, we used two methods: 1. **nested gradient descent ascent (GDA)**([10]; Algorithm 2), which is not guaranteed to converge to a global optimum since the zero-sum Stackelberg game for stochastic Fisher markets is convex-non-concave; and 2. **max-oracle gradient descent** ([10]; Algorithm 3), where we used simulated annealing [5], a metaheuristic which aims to find a global optimum, as the max-oracle. Although simulated annealing is not guaranteed to converge to a global optimum, we observed that it outperformed nested GDA, more often finding a global maximum for the inner player. To check whether the value function computed was optimal, we measured the exploitability of the market, meaning the distance between the recursive competitive equilibrium computed and the actual competitive equilibrium. To do so required that we check two conditions: 1) if each buyer's expected utility is maximized at the computed allocation and saving system at the price system output by the algorithm, and 2) if the market always clears. For both settings, given the value function computed by value iteration, we extracted the greedy policy and unrolled it across time to obtain the greedy actions $(\boldsymbol{X}^{(t)}, \boldsymbol{\beta}^{(t)}, \boldsymbol{p}^{(t)})$ at each state $\boldsymbol{s}^{(t)}$. We then computed the cumulative utility of the allocation and saving systems computed by the algorithms, i.e., for all $i \in [n]$, $\sum_{t=0}^{T} \gamma^t u_i(\boldsymbol{x}_i^{(t)})$, and compared this value to the expected maximum utility u_i^*, obtained by solving the consumption/saving problem for individual buyers given the price systems computed by our algorithms. We report the normalized distance between these two values: e.g., in the case of two buyers, we report $\frac{||(u_1,u_2)-(u_1^\star,u_2^\star)||}{||(u_1^*,u_2^*)||}$, Finally, we measured the excess demand, which we took as the distance to market clearance, i.e., $\frac{1}{T}\sum_{t=1}^{T} ||\sum_{i\in[n]} \boldsymbol{x}_i^{(t)} - \boldsymbol{q}^{(t)}||$. The interested reader can find in Appendix E of the full version of the paper, a graph of the exploitability of the recursive competitive equilibrium computed by both nested GDA and max-oracle gradient descent in all market types (Figure 2), and a a graph of the average value of the value function across all states as it varies with time (Figure 1).

Utility type	Nested GDA		Max-Oracle GD	
	Utility Distance	Market Clearance Distance	Utility Distance	Market Clearance Distance
Linear	6.312	1.513	0.388	2.165
Cobb-Douglas	142.717	2.189	142.03	2.169
Leontief	0.408	2.188	0.558	2.178

5 Conclusion

In this paper, we proved the existence of an optimal solution in zero-sum Stochastic Stackelberg games, and have shown that a Stackelberg equilibrium can be computed in polynomial time via value iteration. Many sequential game models have been proposed in recent years to formulate problems of mechanism design as learning problems, yet very little theoretical characterization of mechanisms has been achieved by such work. Our work provides a first step in this direction, and suggests that zero-sum Stackelberg games are tractable and useful models to learn mechanisms.

References

[1] Kenneth Arrow and Gerard Debreu. "Existence of an equilibrium for a competitive economy". In: *Econometrica: Journal of the Econometric Society* (1954), pp. 265–290.

[2] Alp E. Atakan. "Stochastic Convexity in Dynamic Programming". In: *Economic Theory* 22.2 (2003), pp. 447–455. ISSN: 09382259, 14320479. URL: http://www.jstor.org/stable/25055693.

[3] Stefan Banach. "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales". In: *Fund. math* 3.1 (1922), pp. 133–181.

[4] Richard Bellman. "On the theory of dynamic programming". In: *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952), p. 716.

[5] Dimitris Bertsimas and John Tsitsiklis. "Simulated annealing". In: *Statistical science* 8.1 (1993), pp. 10–15.

[6] Justin Boyan and Andrew Moore. "Generalization in reinforcement learning: Safely approximating the value function". In: *Advances in neural information processing systems* 7 (1994).

[7] William C Brainard, Herbert E Scarf, et al. *How to compute equilibrium prices in 1891*. Citeseer, 2000.

[8] Harley Flanders. "Differentiation Under the Integral Sign". In: *The American Mathematical Monthly* 80.6 (1973), pp. 615–627. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2319163.

[9] Peter H. Friesen. "The Arrow-Debreu Model Extended to Financial Markets". In: *Econometrica* 47.3 (1979), pp. 689–707. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1910415.

[10] Denizalp Goktas and Amy Greenwald. "Convex-Concave Min-Max Stackelberg Games". In: *Advances in Neural Information Processing Systems* 34 (2021).

[11] Anna Jaśkiewicz and Andrzej S Nowak. "Zero-sum stochastic games". In: *Handbook of dynamic game theory* (2018), pp. 1–64.

[12] HW Kuhn and AW Tucker. *Proceedings of 2nd berkeley symposium*. 1951.

[13] R Mehra and EC Prescott. "Recursive Competitive Equilibria and Capital Asset Pricing". In: *Essays in Financial Economics* (1977).

[14] J v Neumann. "Zur theorie der gesellschaftsspiele". In: *Mathematische annalen* 100.1 (1928), pp. 295–320.

[15] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[16] Maurice Sion et al. "On general minimax theorems." In: *Pacific Journal of mathematics* 8.1 (1958), pp. 171–176.

[17] Heinrich Von Stackelberg. *Marktform und gleichgewicht*. J. springer, 1934.

[5] Our code can be found here, and details of our experimental setup can be found in Appendix E of the full version of the paper.

On the Compatibility of Multistep Lookahead and Hessian Approximation for Neural Residual Gradient

Martin Gottwald
Chair for Data Processing,
Technical University of Munich,
Arcisstr. 21, 80333 Munich, Germany
martin.gottwald@tum.de

Hao Shen
fortiss,
Forschungsinstitut des Freistaats Bayern,
Guerickestr. 25, 80805 Munich, Germany
shen@fortiss.org

Abstract

In this work, we investigate, how multistep lookahead affects critical points of Residual Gradient algorithms. We set up a compound Bellman Operator for k consecutive transitions similar to TD(λ) methods and analyse the critical points of the associated Mean Squared Bellman Error (MSBE). By collecting per state multiple successors at once, one can create a more informative objective without increasing the requirements for function approximation architectures. In an empirical analysis, we observe that if one uses Hessian based optimisation to minimise the MSBE, it is not possible to benefit from larger lookahead. Already high convergence speeds and overall lower final error of a Gauss Newton algorithm seem to prevent further improvements by larger lookahead. Only first order gradient descent shows a significant boost in convergence for larger k, emphasizing the importance of multiple steps for existing and successful Deep Reinforcement Learning algorithms. Our results suggest that there are still open questions for Neural Network training in Reinforcement Learning applications.

Keywords: Critical Point Analysis, Gauss Newton Algorithm, Mean Squared Bellman Error, Multistep Lookahead, Residual Gradient

Acknowledgements

Supported by Deutsche Forschungsgemeinschaft (DFG) through TUM International Graduate School of Science and Engineering (IGSSE), GSC 81.

1 Introduction

Reinforcement Learning (RL) is a general approach to solve sequential decision making problems. When facing large or even continuous state spaces, Value Function Approximation (VFA) must be used as inevitable and effective tool [Bertsekas, 2012, Sutton and Barto, 2020]. Recent research efforts have focused more on Non-Linear Value Function Approximation (NL-VFA) methods, which use Neural Networks (NN), e.g. in the form of Multi-Layer Perceptrons (MLP), as approximation architecture. Impressive successes of NNs in solving challenging problems in pattern recognition, computer vision, speech recognition and game playing [LeCun et al., 2015, Yu and Deng, 2015, Mnih et al., 2015, Silver et al., 2017] have further triggered increasing efforts in applying NNs to VFA [van Hasselt et al., 2016].

A key ingredient in well performing RL algorithms, which make use of NL-VFA, are n-step returns [Mnih et al., 2016] or full TD(λ) like multistep lookahead mechanisms [Schulman et al., 2016, 2017]. This means, instead of using (s, a, r, s', a') tuples, one incorporates multiple consecutive transitions at once for learning. Due to the accumulated amount of information in the transition data, algorithms work more reliably with better convergence or higher quality approximations.

In this work, we investigate and analyse the impact of multistep lookahead on Residual Gradient (RG) algorithms [Baird III, 1995], especially when using a Gauss Newton (GN) algorithm for optimisation [Gottwald et al., 2021]. We extend the loss for training with multistep lookahead and derive its differential map. The influence of larger lookahead on critical points is outlined and empirical experiments regarding convergence and generalisation performance are described.

2 Method

We use a Markov Decision Process (MDP) with a continuous state space $\mathcal{S}$, discrete action space $\mathcal{A}$, one step reward r and discount factor γ to model the decision making. In this work, only deterministic transitions are considered to avoid the *Double Sampling* issue when working with RG algorithms [Baird III, 1995]. The goal is to learn an optimal policy π, which maps states to actions in such a way that an accumulated reward signal is maximised. To assess its quality, one employs the value function $V_\pi \colon \mathcal{S} \to \mathbb{R}$, which is defined as

$$V_\pi(s_0) = \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t), s_{t+1}). \tag{1}$$

Once V_π is known, it can be used to define an improved policy. The value function under policy π is the unique fixed point of the Bellman operator T_π

$$V_\pi(s) = (\mathrm{T}_\pi\, V_\pi)(s) = r(s, \pi(s), s') + \gamma V_\pi(s') \quad \forall s \in \mathcal{S}, \tag{2}$$

where s' is the successor of s when executing the action $a = \pi(s)$. For an arbitrary $V \colon \mathcal{S} \to \mathbb{R}$, the squared difference of the left and right hand side in Eq. (2) can be used as an objective function to convert the fixed point problem into an optimisation task. Multistep lookahead is now obtained by k repeated applications of the operator. This can either be done by summing all powers of T_π with exponential weighting to obtain the TD(λ) method, or by using only finite many powers and combining them as simple average. In the latter case, one defines T_π for several steps k by introducing the compound operator

$$\mathrm{T}_\pi^{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{T}_\pi^i \,. \tag{3}$$

As an example, three step lookahead results in $\mathrm{T}_\pi^{(3)} = 1/3 \left(\mathrm{T}_\pi^1 + \mathrm{T}_\pi^2 + \mathrm{T}_\pi^3\right)$ with $(\mathrm{T}_\pi^2\, V)(s) = r(s, \pi(s), s') + \gamma r(s', \pi(s'), s'') + \gamma^2 V(s'')$ and $(\mathrm{T}_\pi^3\, V)(s) = r(s, \pi(s), s') + \gamma r(s', \pi(s'), s'') + \gamma^2 r(s'', \pi(s''), s''') + \gamma^3 V(s''')$. Higher powers of T_π are defined similarly. The fixed point of $\mathrm{T}_\pi^{(k)}$ is still V_π as defined in Eq. (1) and yields the condition $V_\pi(s) = (\mathrm{T}_\pi^{(k)}\, V_\pi)(s)$. Hence, $\mathrm{T}_\pi^{(k)}$ also allows for a conversion of the fixed point iteration into a root finding problem by defining

$$\delta(s, s', s'', \ldots, s^{(k)}) \coloneqq V(s) - (\mathrm{T}_\pi^{(k)}\, V)(s). \tag{4}$$

The current state s and its k successors s', s'' until $s^{(k)}$ are collected from the dynamical system under control according to the current policy. After collecting a batch of N start states s_i, which are uniformly distributed in the whole state space, and their successors $s_i^{(j)}$, one can approximate a solution to the root finding problem with the MLP $f \colon \boldsymbol{\mathcal{W}} \times \mathcal{S} \to \mathbb{R}$ by minimising the multistep Neural Mean Squared Bellman Error (NMSBE)

$$\mathcal{J}(\mathbf{W}) \coloneqq \frac{1}{2N} \sum_{i=1}^{N} \left(f(\mathbf{W}, s_i) - \sum_{j=0}^{k-1} (k-j) \frac{\gamma^j}{k} r_{ij} - \sum_{j=1}^{k} \frac{\gamma^j}{k} f(\mathbf{W}, s_i^{(j)}) \right)^2 = \frac{1}{2N} \Delta_\pi^{(k)}(\mathbf{W})^\mathsf{T} \Delta_\pi^{(k)}(\mathbf{W}), \tag{5}$$

where the expression $\Delta_\pi^{(k)}(\mathbf{W}) \in \mathbb{R}^N$ takes the form

$$\Delta_\pi^{(k)}(\mathbf{W}) \coloneqq F(\mathbf{W}) - \frac{1}{k} R_\pi^{(k)} - \frac{1}{k} \sum_{j=1}^{k} \gamma^j F^{(j)}(\mathbf{W}) \tag{6}$$

by collecting for all i the evaluation of f for the states as vector $F(\mathbf{W}) \coloneqq [f(\mathbf{W}, s_1) \dots f(\mathbf{W}, s_N)]^\mathsf{T} \in \mathbb{R}^N$. Further, we use the abbreviation $r_{ij} = r(s_i^{(j)}, \pi(s_i^{(j)}), s_i^{(j+1)})$ and accumulate all reward terms in $R_\pi^{(k)}$. Critical points of $\mathcal{J}(\mathbf{W})$ are now characterised by the equation

$$\nabla_\mathbf{W} \mathcal{J}(\mathbf{W}) = \frac{1}{N} \underbrace{\Big(G(\mathbf{W}) - \frac{1}{k} \sum_{j=1}^{k} \gamma^j G^{(j)}(\mathbf{W}) \Big)^\mathsf{T}}_{=:\widetilde{G}(\mathbf{W}) \in \mathbb{R}^{N \times N_{net}}} \Delta_\pi^{(k)}(\mathbf{W}) = 0, \tag{7}$$

where $G(\mathbf{W})$ is the differential map of F with respect to the parameters $\mathbf{W}$ evaluated at all start states. With $G^{(j)}(\mathbf{W})$ we denote the same item but use the j-th successor states as input. We see that $\mathrm{T}_\pi^{(k)}$ provides a richer objective for the optimisation problem than $\mathrm{T}_\pi^{(1)}$ without increasing the required number of network parameters to allow for a full rank of $\widetilde{G}(\mathbf{W})$. A full rank of $\widetilde{G}(\mathbf{W})$ is important to ensure that any critical point achieves zero error, since in this case $\Delta_\pi^{(k)}(\mathbf{W}) = 0$ is the only way to satisfy the critical point condition of Eq. (7). This property is important, as it removes local minima and saddle points. In practical applications, the rank of $\widetilde{G}(\mathbf{W})$ is full, since sampling and numerical inaccuracies are present. If exact learning is out of reach, i.e., $\Delta_\pi^{(k)}(\mathbf{W}) \neq 0 \; \forall \; \mathbf{W} \in \mathcal{W}$, the objective is free of critical points. One can run a descent algorithm as long as possible.

There remains an open problem in the definition of $\Delta_\pi^{(k)}(\mathbf{W})$, which exists due to the sampling of states and their successors. We hypothesize that with only one step transitions, it is relatively easy to sample states with large pairwise distances. Especially in high dimensional spaces, this could be rather common. In such a case, an MLP could approximate a function, where $\Delta_\pi^{(1)}(\mathbf{W})$ becomes zero without being everywhere close to V_π, because for some states the transition information to remaining parts of the state space might not be present in the sampled data. Thus, the sampling based loss of Eq. (5) has uncontrollable degrees of freedom and for $k = 1$, one can even construct examples to demonstrate this issue. We expect that a multistep lookahead algorithm helps here. Due to the use of trajectories, one ensures that there are proper transitions between states and their various successors available. The chance that parts of the state space are not sufficiently connected is reduced. Hence, an MLP has less possibilities to approximate functions, which would shrink the length of $\Delta_\pi^{(k)}(\mathbf{W})$ to zero without becoming close to V_π.

From a theoretical perspective, one would expect a better performance when switching to multistep methods. This expectation is supported by the empirical behaviour of existing algorithms. When using k-step returns T_π^k for training, e.g. as done in [Mnih et al., 2016], or when employing full TD(λ)-like methods as in [Schulman et al., 2016, 2017], one obtains well performing algorithms. Multiple steps can compensate the convergence issues of Semi-Gradient algorithms, which exist due to the missing derivates of the MLP for successor states with respect to the parameters. Because expressions receive higher powers of the discount factor, they have less impact. If the lookahead becomes large, the terms vanish naturally such that omitted dependencies no longer cause any harm. Opposed to that, we observe that a Gauss Newton Residual Gradient algorithm converges already without problems with only single-step lookahead. Hence, we argue that a beneficial impact of multistep-lookahead depends directly on the algorithm and problem at hand. To clarify our considerations, we compare first order and Hessian based descent algorithms and test, whether different values for k create a meaningful difference.

3 Experiments

We use the Mountain Car environment from [Brockman et al., 2016] with additional transitions from the goal region to the valley to obtain an infinite horizon MDP. Training is performed in a batch setting with both a GN and first order only RG algorithm. For the first, we use a constant learning rate $\alpha = 0.1$, for the latter $\alpha = 0.01$. Start states are sampled uniformly from the whole space. We use $N = 300$ samples for training. Additional states, which are arranged on a grid with a high resolution (500×500), serve as test dataset. For representing value functions, we take an MLP consisting of two hidden layers with ten units each and employ Bent-Id activation functions. The input layer accepts two dimensional state vectors. The output is scalar with linear activation. Initial parameters are drawn elementwise uniformly from the interval $[-1, 1]$. We set the discount factor to $\gamma = 0.99$ and the policy for evaluation to accelerating in the direction of movement. For the lookahead we consider $k \in \{1, 2, 3, 4, 5\}$. We show results for 25 repetitions, where we randomise the training data and initial network parameters in each run.

When visualising the descent behaviour in Fig. 1, we observe a missing impact of $\mathrm{T}_\pi^{(k)}$ with increasing k for GN based optimisation. Using multiple transitions during training does not help with convergence. Opposed to that, when training with a first order only RG algorithm, the convergence speed can be enhanced significantly by varying k. The well-known slow convergence of RG algorithms for $k = 1$ vanishes with larger values. For both first order and Hessian based optimisation, a stepwise larger k results in worse final training errors. This could be explained by the fact that an MLP with identical capacity must fit a more complex objective. 244

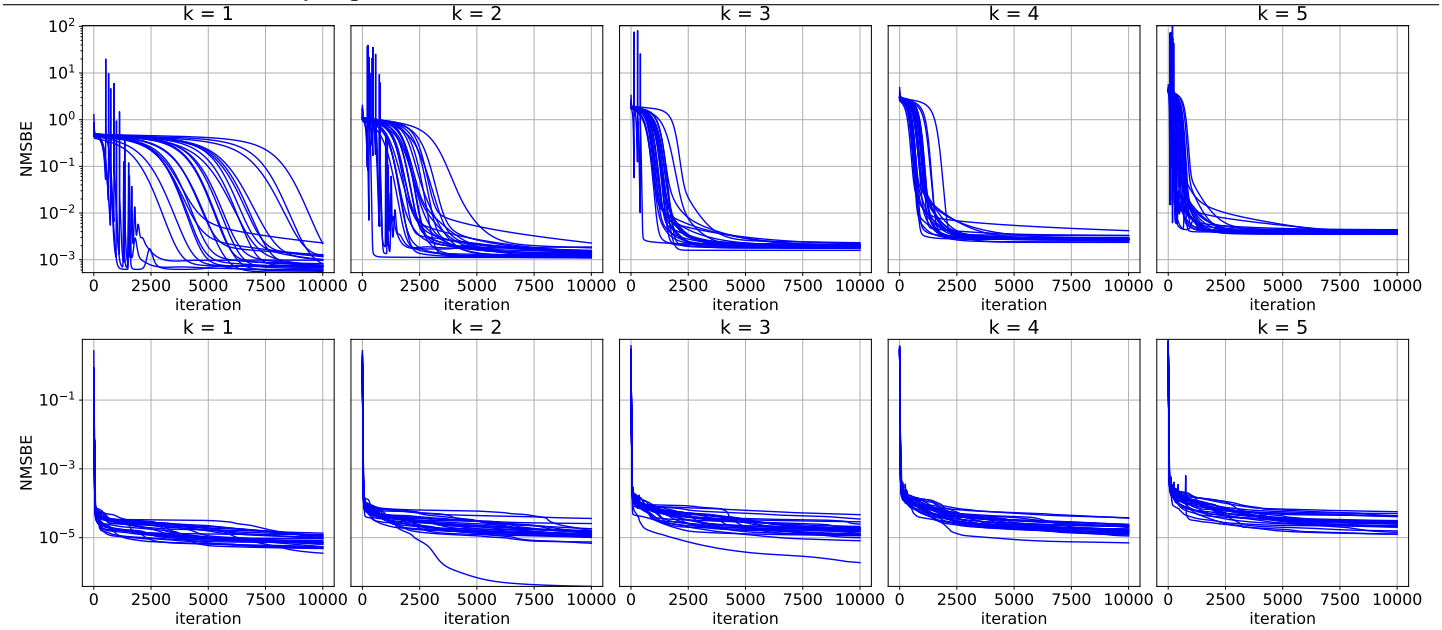


Figure 1: The impact of multistep lookahead onto first order and Hessian based RG algorithms. Only first order optimisation profits from larger k. For both methods, the final achieved training error increases slightly with bigger k. **Top:** First order optimisation. **Bottom:** Gauss Newton algorithm.

To compare training outcomes fairly, we evaluate the MLP at the end of training with the held out test set. Figs. 2a and 2b show the final test errors for all k for both optimisation methods. We see, that the average test error increases with k for first order gradient descent, implying that the solutions become worse. This matches the increase of training errors. For Hessian based optimisation, we observe a similar rise, but not as pronounced. Interestingly, the best approximated value function for $k = 5$, which is shown in Fig. 2f, possesses a qualitatively better shape despite a higher test error than the best value function for $k = 1$ in Fig. 2e. The value functions for first order optimisation are for all k almost identical to that of Fig. 2d and do not reflect the details of the actual value function V_π, which is obtained from Monte Carlo methods and shown in Fig. 2c. We conclude that using a test dataset is not a perfect reliable assessment. This could be due to the aforementioned open problems regarding the objective of Eq. (7) and its critical point condition.

4 Conclusion

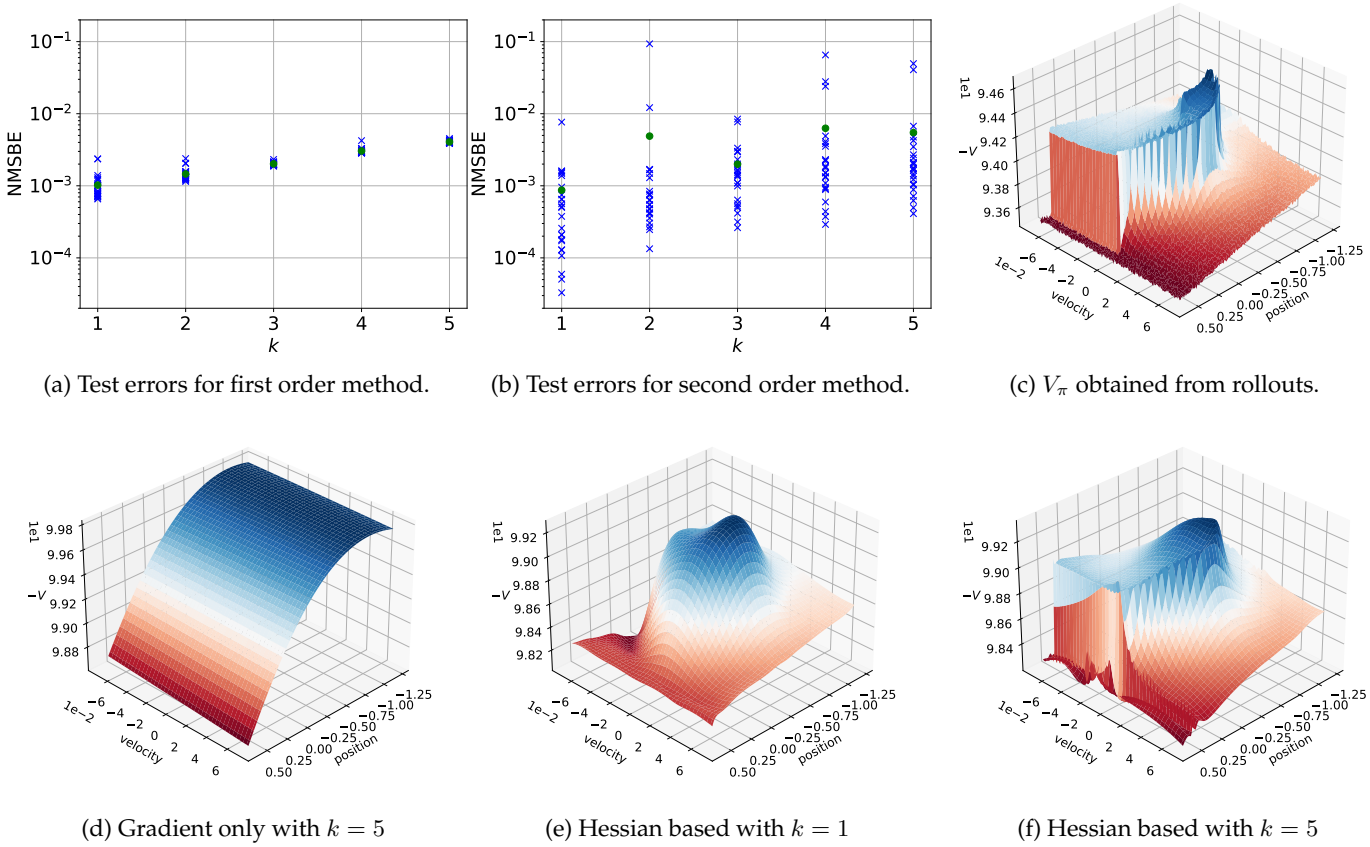
We have seen that the compatibility of multistep lookahead formulations is not given when employing Gauss Newton Residual Gradient algorithms. Although we can demonstrate the beneficial impact of multiple steps on a first order only Residual Gradient algorithm and we also see that a Hessian based optimisation performs well when using single transitions, their combination does not lead to an even more impressive algorithm. Since this happens in relatively easy control problems, our experiments suggest that a proper formulation of the optimisation problem is mandatory and, unfortunately, to some extent still an unsolved problem. There are open questions about the critical points of the objective, in particular when using sampling based approximations, and it is not known, whether this loss is suited for Deep Reinforcement Learning with Residual Gradient algorithms. Furthermore, a sound performance evaluation is also a challenge on its own.

Our results show that whenever one is restricted to first order only optimisation, it is important to use multistep methods to overcome the slow convergence of Residual Gradient algorithms. If the quality of the solution is more important, one should use Hessian information, at least in an approximated form. The final NMSBE is significantly smaller, convergence is fast and one only requires transition data consisting of a single step.

In this work, we have analysed a compound Bellman operator with uniform averaging. Hence, as a next step, it is important to know, whether also TD(λ) methods show this incompatibility.

References

L. C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *Proceeding of the* 12^{th} *International Conference on Machine Learning*, pages 30–37, 1995. 245



(a) Test errors for first order method.

(b) Test errors for second order method.

(c) V_π obtained from rollouts.

(d) Gradient only with $k = 5$

(e) Hessian based with $k = 1$

(f) Hessian based with $k = 5$

Figure 2: **a)** and **b)**: Final test errors for first and second order optimisation. **c)** to **f)**: Ground truth value function and the best approximations according to the smallest test errors.

D. P. Bertsekas. *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, volume 2. Athena Scientific, 4^{th} edition, 2012.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *Computing Research Repository*, arXiv:1606.01540, 2016.

M. Gottwald, S. Gronauer, H. Shen, and K. Diepold. Analysis and optimisation of bellman residual errors with neural function approximation. *Computing Research Repository*, arXiv:2106.08774, 2021.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Peterson, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The* 33^{rd} *International Conference on Machine Learning*, pages 1928–1937, 2016.

J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the* 4^{th} *International Conference on Learning Representations*, 2016.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *Computing Research Repository*, arXiv:1707.06347, 2017.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 550:354—359, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2^{nd} edition, 2020.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, pages 2094–2100, 2016.

D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer-Verlag, London, 2015.

Humans adapt their foraging strategies and computations to environment complexity

Nora C. Harhen
Department of Cognitive Sciences
University of California, Irvine
Irvine, CA 92697
nharhen@uci.edu

Aaron M. Bornstein
Department of Cognitive Sciences
University of California, Irvine
Irvine, CA 92697
aaron.bornstein@uci.edu

Abstract

Foraging has been suggested to provide a naturalistic context for studying decision-making. In the wild and in the laboratory, foragers come close to approximating the optimal decision strategy given by Marginal Value Theorem (MVT; Charnov, 1976). Recent work has used reinforcement learning to understand how the variables for decision-making under an MVT-policy are learned (Garrett & Daw, 2020; Simon & Daw, 2011). This work often implicitly assumes the forager begins with a specific, fixed representation of the environment. However, it is likely that this representation is also something that must be learned. Here we ask — can foragers learn a representation of the environment and, importantly, do they adapt their decision strategies and value computations to this representation as it evolves? We propose a model of how foragers could use principled statistical inference to organize their past experiences into a representation that guides decision-making. The model was tested in a variant of a serial stay/switch foraging task with multimodal reward distributions and non-uniform transition structure between patch types. In this task, participants adapted their foraging to both the richness of the local context and their internal uncertainty. These results are consistent with participants having learned and used a model of the environment to guide their decisions. Overall, these findings demonstrate the utility of combining representation learning and reinforcement learning to understand foraging behavior.

Keywords: foraging, structure learning, model-based reinforcement learning

Acknowledgements

This work was supported by NIMH P50MH096889 and a NARSAD Young Investigator Award by the Brain and Behavior Research Foundation to AMB. NCH was supported by a National Defense Science and Engineering Graduate fellowship.

1 Introduction

Decision-makers commonly choose between staying with a current option or foregoing it in hope of a better future alternative. Such decisions arise in ethology where they are known as patch leaving problems in which a patch refers to a concentration of resources in the environment. In solving these problems, foragers must weigh the the costs and benefits of harvesting an often depleting resource against those associated with searching for a new, unharvested one. An optimal solution is given by Marginal Value Theorem under a certain set of assumptions (MVT; Charnov, 1976) — a forager should leave the current depleting patch once its reward rate falls below the overall reward rate of the environment. Relative to MVT, it's been widely observed that foragers from rodents to humans stay longer than prescribed (Blanchard & Hayden, 2015; Constantino & Daw, 2015; Kane et al., 2019). This is known as overharvesting.

MVT's predictions assume the forager has complete knowledge of the environment and its dynamics. Consequently, MVT provides how the decision should be made, but it does not explicitly describe how its key decision variables, the local and global reward rates, should be learned. However, this assumption of complete knowledge is not often met in the real world. This suggests more naturalistic patching leaving is both a decision-making and a learning problem. A potential simple learning rule involves keeping a running average of rewards across all past patch experiences in the environment (Constantino & Daw, 2015).

These simple learning rules work well in simple, homogeneous environments in which patches are similar to one another. However, real world environments are often complex with regions varying in richness (McNamara & Houston, 1985; Sparrow, 1999). In more naturalistic environments, it may be beneficial to group patches of similar richness together to form a multi-state representation that affords contextually-appropriate and dynamic estimates of the local and global reward rates. In standard reinforcement settings, humans use a similar strategy — they track environmental statistics and leverage them to adaptively adjust reward-related computations (Behrens, Woolrich, Walton, & Rushworth, 2007; Simon & Daw, 2011). Thus, we asked — in a foraging context, do decision-makers learn an internal representation of the environment and adapt their strategies and computations with respect to it? We propose a model of how foragers may incrementally build such a representation from past experiences and use it during decision-making. We then test its predictions with a novel serial stay-switch task.

2 Latent Cause Model

2.1 Learning a state representation of the environment

Latent-cause inference provides a framework for building state space representations out of past experiences (Courville, Daw, & Touretzky, 2006; Gershman, Norman, & Niv, 2015). Under this framework, representation learning is treated as a clustering problem in which an experience is assigned to a pre-existing cluster based on its similarity to experiences previously assigned to the cluster. In a new environment, the learner begins with a single cluster, or state, to which experiences can belong. New experiences that differ significantly from past ones can initiate the creation of a new cluster. Thus, the complexity of the representation is allowed to grow incrementally as experience warrants it. Through principled statistical learning, the learner develops a representation with a useful number of clusters — enough to allow for contextually-specific predictions but few enough to enable generalization across experiences.

Within a foraging context, past experiences could correspond to reward decays experienced while harvesting patches and states to patch types that differ in their richness. To infer the current patch type or state, the observer must combine their past experiences with current experience. The prior probability of a patch belonging to a patch type, p, at time t is given by:

$$P(p) = \begin{cases} \frac{N_p}{t-1+\alpha} & \text{if p is an old patch type} \\ \frac{\alpha}{t-1+\alpha} & \text{if p is a new patch type} \end{cases} \tag{1}$$

Where N_p is the number of patches already assigned to that patch type and α is the prior over environment complexity. This formally instantiates the assumptions that 1) the current patch type is more likely to be a frequently visited one and 2) there always remains some probability that a new patch belongs to a previously unobserved patch type. In our model, we allow the structure learning parameter, α, to be a free parameter fit to individual participants' choice data.

A set of reward decays at time t, D_t, can be combined with the prior probability specified in Equation 1 to generate a posterior distribution over patch types.

$$P(p_t|D_t) = \frac{P(D_t|p_t)P(p_t)}{p(D_t)} \tag{2}$$

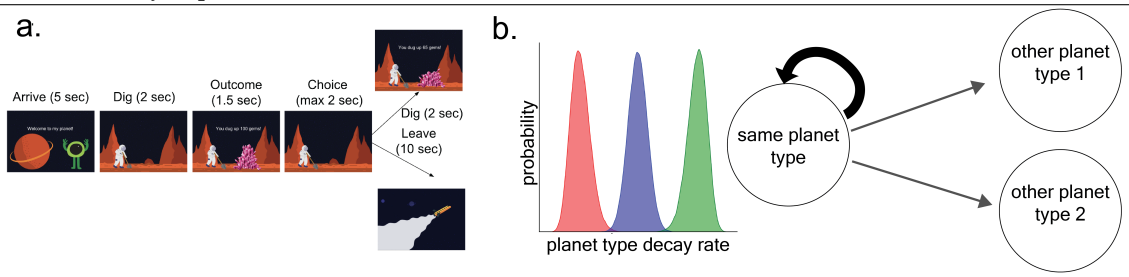


Figure 1: Task structure. **A.** Participants sequentially decided whether to stay dig from a depleting gem mine or incurring a time cost to leave for a new planet with a replenished mine. **B.** The decay rate distributions associated with rich, neutral, and poor planet types and and transition probabilities between planet types when leaving for a new planet.

where p_t is a potential patch type. Each patch type has a unique distribution over decay rates associated with it that determines $P(D_t|p_t)$.

Exact computation of this posterior is computationally demanding, so we use particle filtering as an approximate inference algorithm (Gershman et al., 2015; Sanborn, Griffiths, & Navarro, 2006). Harhen, Hartley, and Bornstein (2021) contains further implementation details.

2.2 Using structure to inform stay/leave decisions

The forager compares the value of staying and leaving and selects the higher-valued option. The value of staying is taken as the reward received on the last harvest, r_t, multiplied by the predicted decay rate, $\hat{d}$, if the forager were to stay again.

$$V_{stay} = r_t * \hat{d} \tag{3}$$

To generate $\hat{d}$ the agent samples from patch type-specific decay rate distributions. The probability of sampling from a planet type is proportional to its posterior probability of the current planet belonging to that cluster, $P(p_t|D_t)$.

The value of leaving is estimated by averaging over the reward rates from all previously encountered patches discounted by some factor, γ.

$$V_{leave} = \gamma \frac{r_{total}}{t_{total}} t_{harvest} \tag{4}$$

Theoretical work has suggested that discounting factors that adapt to an agent's internal uncertainty can be beneficial in complex environments (Jiang, Kulesza, Singh, & Lewis, 2015). Following this, we allow γ to be dynamic, flexibly adjusting to the individual's uncertainty over the accuracy of their internal representation. Here, we compute uncertainty as being proportional to the entropy of the samples drawn to generate, $\hat{d}$.

We compared this structure learning model to two models previously used to explain human foraging behavior in Constantino & Daw (2015) – a temporal difference learning model (TD) and a MVT learning model that learns the mean decay rate and global reward rate of the environment (MVT learn). Each model's fit to the data was evaluated using a 10-fold cross validation procedure. For each participant, we shuffled their PRTs on all visited planets and split them into 10 separate training/test datasets. The best fitting parameters were those that minimized the sum of squared error (SSE) between the participant's PRT and the model's predicted PRT on each planet in the training set. Then, with the held out test dataset, the model was simulated with the best fitting parameters and the SSE was calculated between the participant's true PRT and the model's PRT. To compute the model's final cross validation score, we summed over the test SSE from each fold.

3 Methods

We tested whether participants could learn an internal representation of a three patch type environment and use it to guide their foraging decisions. Past human foraging work has focused on either single patch type environments or multi-patch type environments in which patch types of differing richness are blocked off from one another. To more closely mimic real world conditions, we interleaved the three patch types and did not indicate that patches could differ from one another.

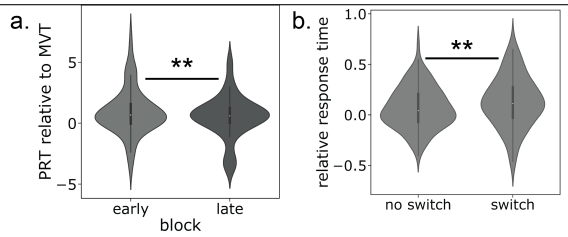


Figure 2: Behavioral results. **A.** With experience, participant's planet residence times (PRT) became closer to those prescribed by an MVT-optimal policy (at 0). **B.** Participants were slower in making their first decision on a new planet following a switch to a different planet type relative to if there was no switch.

Specifically, we investigated how humans learn in a serial stay/switch foraging task (Constantino & Daw, 2015). Participants visited planets where they would dig for space treasure (Figure 1A). On a planet, participants had to choose between staying and digging from a depleting mine or incurring a time cost to travel to a new planet with a replenished mine. Their goal was to collect as many gems as possible across the span of the game. Participants completed 5 blocks lasting 6 minutes each. The environment consisted of three planet types differing in richness – rich, neutral, and poor (Figure 1B). Rich planets had decay rates sampled from a distribution with a higher expected value and hence would deplete more slowly relative to the poor and neutral planets. Participants were not told that planets differed in quality requiring them to infer this from experienced rewards alone. Planets of a similar type temporally clustered together resembling natural environments' spatiotemporal correlations in richness. When the participant left a planet, there was a 80% probability they would travel to a planet of the same type. If traveling to a planet of differing quality, it was equally likely to be one of the two remaining planet types.

We recruited 198 participants from Amazon Mechanical Turk (ages 23-64, Mean=39.79, SD=10.56). Participation was restricted to workers who had completed at least 100 prior studies and had at least a 99% approval rate. Participants were paid $6 as a base payment and could earn a bonus contingent on performance ($0-4). We excluded 82 participants for having average planet residence times 2 standard deviations above or below the group mean, failing a quiz on the task instructions more than 2 times, and/or missing catch questions. The catch questions asked participant's to press the letter "Z" on their keyboard at random intervals throughout the task. This was meant to "catch" participants who were repeatedly making repetitive choices in a manner not guided by value.

4 Results

As predicted by MVT, participants stayed longer the richer the planet was (rich vs. neutral - t(115)= 19.77, $p < 0.0001$; neutral vs. poor - t(115) = 12.57, $p < 0.0001$). When directly comparing to MVT, participants on average overharvested across the entire experiment (t(115) = 3.88, $p = 0.00018$). MVT assumes perfect knowledge of the environment's structure. If participants learned the structure of the task environment, then the extent of over harvesting should diminish as they accumulate more experience. Participants did just that, overharvesting more in the initial two blocks relative to the final two (Figure 2A, t(115) = 3.27, $p = 0.0014$). Further suggesting a multi-state representation, participants demonstrated context sensitivity, overharvesting only on poor and neutral planets but not on rich (Figure 2d; poor - t(115) = 6.92, $p < 0.0001$; neutral - t(115) = 9.00, $p < 0.0001$; rich - t(115) = 1.38, $p = 0.17$). Participants' reaction times provided further evidence of structure learning. We reasoned that learners sensitive to structure should demonstrate switch costs when transitioning between planets of a different type. Consistent with this, participants were slower in making their initial choice on planets who differed in type from the most recent prior planet (Figure 2B, t(115) = 2.65,$p = 0.0093$).

Computational modeling results further supported participants' use of an internal model of the environment. Based on cross validation scores, the adaptive discounting model provided a better account of participants' choices relative to the two other models that assumed no structure learning (Figure 2AB). In the adaptive discounting model, the structure learning parameter α must be greater than 0 to allow for multi-state inference. In simulation, the lowest setting of α that lead to multiple states being inferred in at least 90% of simulation runs was 0.8. Thus, this value was used as our baseline for assessing structure learning. We found that 76% of participants had a fit α greater than than this threshold (Figure 3C). Validating α as a measure of individual structure learning ability, participant's with higher fit α demonstrated greater switch costs (Figure 3D, Kendall's τ = 0.24, $p = 0.00076$).

Prior theoretical work has demonstrated that decision-makers should monitor the uncertainty over the accuracy of their model of the environment and adapt their planning horizon to it (Jiang et al., 2015). Based on this normative work and empirical findings that humans similarly adapt their discounting of future value to internal uncertainty (Gershman & Bhui, 2020), we reasoned that participants' choices in this more complex, naturalistic environment environment would

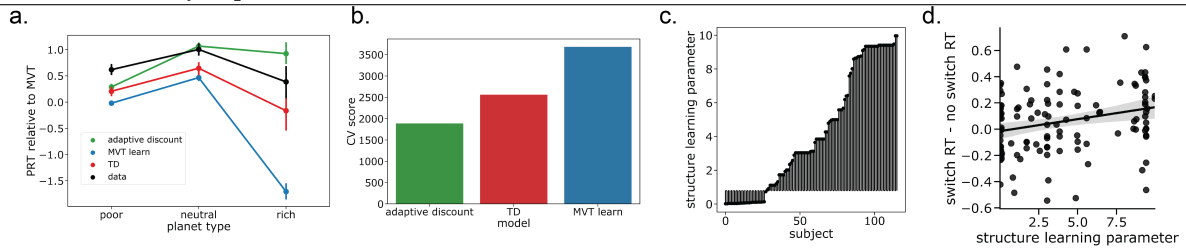


Figure 3: Model fitting results. **A.** The planet residence time (PRT) of the agent, whether real or simulated, is compared to the PRT of an MVT optimal agent who has complete knowledge of the environment. This done by taking the difference of the two PRTs. PRTs above 0 reflect overharvesting behavior while below underharvesting. Error bars are S.E.M. **B.** Model comparison using cross validation (CV) between the adaptive discounting, temporal difference learning, and MVT augmented with learning models. A lower CV score indicates the model provided a better account of participants behavior. **C.** Each subjects' best fitting structure learning parameter or α. The baseline was set to the lowest value that lead to multiple planet types being inferred in at least 90% of simulation runs (0.8). 76% of participants were above this threshold. **D.** There was a positive correlation between participants' structure learning parameter, α, and their reaction time switch cost.

be better fit by a model that dynamically adjusts its discounting rate as more experience is accumulated and uncertainty reduced. Consistent with this hypothesis, 93% of participants had an uncertainty adaptation parameter greater than 0.

5 Conclusion

We found evidence that participants learned the structure of their environment and adjusted both their strategies and computations to it. These findings highlight the importance of representation learning in shaping foraging decisions. Consistent with work examining the use of model-based strategies in standard reinforcement learning tasks, we observed considerable individual differences in the sensitivity to task structure and in the adaptation to it. Therefore, a potential future direction is to examine how individual differences in representation learning can explain variability in individuals' deviations from MVT optimality (Harhen & Bornstein, 2021).

References

Behrens, T. E. J., Woolrich, M. W., Walton, M. E., & Rushworth, M. F. S. (2007, September). Learning the value of information in an uncertain world. *Nat. Neurosci.*, *10*(9), 1214–1221.

Blanchard, T. C., & Hayden, B. Y. (2015, February). Monkeys are more patient in a foraging task than in a standard intertemporal choice task. *PLoS One*, *10*(2), e0117057.

Charnov, E. L. (1976, April). Optimal foraging, the marginal value theorem. *Theor. Popul. Biol.*, *9*(2), 129–136.

Constantino, S. M., & Daw, N. D. (2015, December). Learning the opportunity cost of time in a patch-foraging task. *Cogn. Affect. Behav. Neurosci.*, *15*(4), 837–853.

Courville, A. C., Daw, N. D., & Touretzky, D. S. (2006, July). Bayesian theories of conditioning in a changing world. *Trends Cogn. Sci.*, *10*(7), 294–300.

Garrett, N., & Daw, N. D. (2020, July). Biased belief updating and suboptimal choice in foraging decisions. *Nat. Commun.*, *11*(1), 3417.

Gershman, S. J., & Bhui, R. (2020, July). Rationally inattentive intertemporal choice. *Nat. Commun.*, *11*(1), 3365.

Gershman, S. J., Norman, K. A., & Niv, Y. (2015, October). Discovering latent causes in reinforcement learning. *Current Opinion in Behavioral Sciences*, *5*, 43–50.

Harhen, N. C., & Bornstein, A. M. (2021). Structure learning as a mechanism of overharvesting. In *Proceedings of the 19th international conference on cognitive modeling*.

Harhen, N. C., Hartley, C., & Bornstein, A. (2021). Model-based foraging using latent-cause inference. In *Proceedings of the annual meeting of the cognitive science society* (Vol. 43).

Jiang, N., Kulesza, A., Singh, S., & Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems* (pp. 1181–1189).

Kane, G. A., Bornstein, A. M., Shenhav, A., Wilson, R. C., Daw, N. D., & Cohen, J. D. (2019, September). Rats exhibit similar biases in foraging and intertemporal choice tasks. *Elife*, *8*.

McNamara, J. M., & Houston, A. I. (1985, November). Optimal foraging and learning. *J. Theor. Biol.*, *117*(2), 231–249.

Sanborn, A. N., Griffiths, T. L., & Navarro, D. J. (2006, January). A more rational model of categorization.

Simon, D., & Daw, N. (2011). Environmental statistics and the trade-off between model-based and TD learning in humans. *Adv. Neural Inf. Process. Syst.*, *24*.

Sparrow, A. D. (1999, November). A heterogeneity of heterogeneities. *Trends Ecol. Evol.*, *14*(11), 422–423.

Wittmann, M. K., Kolling, N., Akaishi, R., Chau, B. K. H., Brown, J. W., Nelissen, N., & Rushworth, M. F. S. (2016, August). Predictive decision making driven by multiple time-linked reward representations in the anterior cingulate cortex. *Nat. Commun.*, *7*, 12327.

Reinforcement Learning As End-User Trigger-Action Programming

Chace Hayhurst, Hyojae Park, Atrey Desai, Suheidy De Los Santos, Michael Littman

Brown University, Computer Science Department
{chace_hayhurst, hyojae_park, atrey_sanjeev_desai, suheidy_de_los_santos, michael_littman}@brown.edu

Abstract

We contend that the power of reinforcement learning comes from its fundamental declarative nature, allowing a system designer to consider *what* an agent's objective is instead of the details of *how* this objective can ultimately be achieved. This abstract provides some early design ideas for creating an end-user-oriented reinforcement-learning system based on the trigger-action programming model. We propose a study designed to highlight the similarities and differences of this end-user-based reinforcement-learning language to more established end-user trigger-action programming.

Introduction

Historically, much of research within the reinforcement-learning community has been directed at applying and designing algorithms to create intelligent agents that solve specific problems [Sutton and Barto 1998]. In recent years, this approach to reinforcement learning (RL) has produced exemplary results, with engineers being able to create agents that rival or even surpass the best human-level performances on some problems [Mnih et al. 2015, Silver et al. 2016]. In contrast, little effort has been put into studying how non-experts can interact with these systems.

An RL "programmer" needs to identify three things to the algorithm: (1) the *actions* an agent can take in the environment, (2) the *state variables* of the environment the agent should be concerned with, and (3) the reward function, or more simply, a *goal* that the agent should complete. That is, while RL *researchers* typically take actions, states, and goals as given and focus on how to design agents that can take actions to achieve goal states, RL *users* are the ones responsible for defining these parameters in the first place. Bad choices can lead to intractable learning problems because of either under-specification (critical aspects of the problem are not accessible by the learner) or over-specification (too many details are given to the learner, making learning and generalization difficult).

For some tasks and for some users, finding the right actions, states, and goals may be considerably easier than explicitly articulating the choices the agent should make. For others, perhaps not. Our research objective is understanding where that line might be.

Our baseline for comparisons is *trigger-action programming* [Ur et al. 2014, Huang and Cakmak 2015, Zhang et al. 2020], a methodology in broad use that allows end users to specify behaviors for data analysis, smart home devices, and other applications. A trigger-action program (TAP) consists of a set of rules, each of which has a trigger (something that has become true of the world) and action (an intervention that should be taken in response). Triggers can be primitive events or a primitive event combined with one or more conditions. An example TAP rule in the home-automation domain is: "**If** I arrive home (trigger event) **while** the indoor temperature is above 75 degrees (trigger condition) **then** turn on the AC (action)." Existing research shows that end users with no programming experience are able to construct and interpret TAPs, providing some support for the contention that this style of programming strikes a useful balance between ease of use and expressive power.

In this work, we observe that the actions in TAP are analogous to the actions in RL, while the triggers are akin to states. Thus, an RL task can be specified by a set of triggers (which constitute the learner's states), actions (which constitute the learner's actions), and a special trigger (which acts as the learner's goal).

Proposed Interface

Our interface is a modified version of the AutoTap project created by Zhang et al. (2019) to support the construction of TAPs. We have repurposed this framework to provide actions and triggers for a scenario in which a mobile robot is tasked with moving boxes throughout a house. We created two interfaces: TAP and RL. In both versions, users are greeted with a page allowing them to create new rules that guide agent behavior using the associated programming style.

In the TAP version, the user creates rules that pair a trigger and an action. (See Figure 1.)

In the RL version (see Figure 2), the user specifies a set of parts, each of which belongs to one of three selected categories: 'Consider doing:', 'Pay attention to:', and 'Get a 'yes' answer to:'. The part finishes the clause started by the corresponding category. Parts take the form of conditions that can be true or false given the current state of the envi-

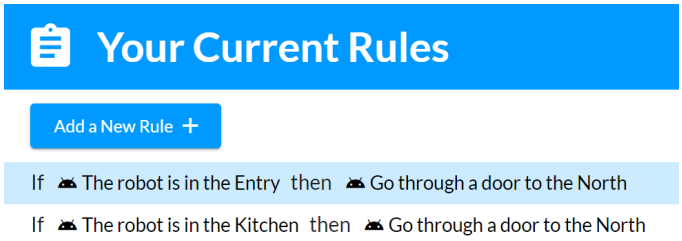


Figure 1: An example of a TAP ruleset in our system.

ronment or actions that can be performed by the agent. By combining a category with a part, a user specifies an action that the agent can take ('Consider doing:'), a state variable the agent should be concerned with ('Pay attention to:') or the goal of the agent ('Get a 'yes' answer to:').

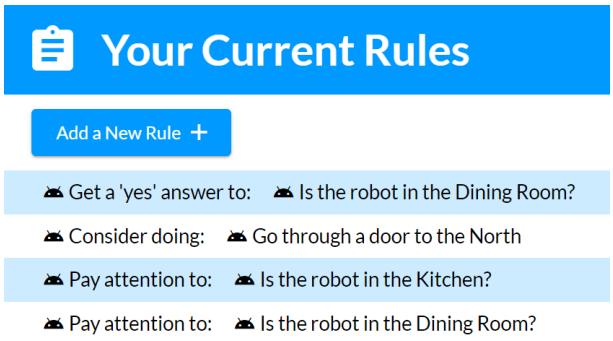


Figure 2: An example of an RL ruleset in our system.

Proposed Experiment Design

Our experiment will involve two groups of 20 participants with no prior experience in programming. One group will randomly be assigned to the TAP condition while the other will be assigned to the RL condition. Both groups will receive instructions on how to access the website with our program-creation framework.

Users would then be given four programming tasks of increasing difficulty and tasked with writing rules to control an agent to solve each task. The tasks involve a robot navigating through a house and moving items around in it (Figure 3).

Triggers in the system include 'Robot is in room X', 'There is a red/blue block in room X', and 'Robot is holding a red/blue block'. Actions that the robot can take include 'Go through a door to the North/South/East/West', 'Pick up a red/blue block in your current room', and 'Put down held block'.

Table 1 provides an example set of tasks and the number of TAP rules or RL parts needed to solve them. Although the number of required RL parts is consistently larger than the TAP rule sets, the assembly of RL parts is performed by the learner, which may make it easier to select them.

After users finish writing their programs, they submit

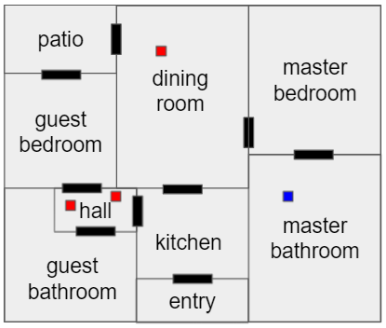


Figure 3: Floorplan of the house for our experiments. The blue dot is the agent controlled by the behavior specified by the user. Red dots are blocks in the enviornment that can be picked up and moved by the agent.

Table 1: Proposed tasks for our experiments.

Instruction	*TAP rules*	*RL parts*
Starting from the entry, go to the kitchen.	1	3
Wherever you start, go to the master bedroom.	8	12
Starting from the entry, bring back the blue box from the master bathroom.	7	13
Starting from the entry, clear all of the red boxes out of the hall.	6	12

them to our database for analysis. Solutions that miss components that are necessary to complete the task or put agents into situations where they have no available actions will be deemed incomplete. Solutions will be graded according to a rubric determined in advance for each task.

We are interested in statistics such as: How often does each programming style succeed on each task? Is there any pattern to how the success rates change as the task difficulty increases? Is a certain programming style better suited to certain tasks? How often do participants include unnecessary components? How often do they miss required components? Are there consistent errors that participants make that might be reduced through careful interface design? Ultimately, we are interested in assessing if RL can be a viable programming language for end users.

References

Huang, J.; and Cakmak, M. 2015. Supporting Mental Model Accuracy in Trigger-Action Programming. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg,

S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

Ur, B.; McManus, E.; Ho, M. P. Y.; and Littman, M. L. 2014. Practical Trigger-Action Programming in the Smart Home. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*.

Zhang, L.; He, W.; Martinez, J.; Brackenbury, N.; Lu, S.; and Ur, B. 2019. AutoTap: Synthesizing and repairing trigger-action programs using LTL properties. In *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 281–291.

Zhang, L.; He, W.; Morkved, O.; Zhao, V.; Littman, M. L.; Lu, S.; and Ur, B. 2020. Trace2TAP: Synthesizing Trigger-Action Programs from Traces of Behavior. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, volume 4, 104:1–104:26.

Does DQN *really* learn? Exploring adversarial training schemes in Pong

Bowen He
Department of Computer Science
Brown University
Providence, RI 02912
bowen_he@brown.edu

Sreehari Rammohan
Department of Computer Science
Brown University
Providence, RI 02912
sreehari@brown.edu

Jessica Forde
Department of Computer Science
Brown University
Providence, RI 02912
jessica_forde@brown.edu

Michael L. Littman
Department of Computer Science
Brown University
Providence, RI 02912
mlittman@cs.brown.edu

Abstract

In this work, we study two self-play training schemes, *Chainer* and *Pool*, and show they lead to improved agent performance in Atari Pong compared to a standard DQN agent—trained against the built-in Atari opponent. To measure agent performance, we define a robustness metric that captures how difficult it is to learn a strategy that beats the agent's learned policy.Through playing past versions of themselves, Chainer and Pool are able to target weaknesses in their policies and improve their resistance to attack. Agents trained using these methods score well on our robustness metric and can easily defeat the standard DQN agent. We conclude by using linear probing to illuminate what internal structures the different agents develop to play the game. We show that training agents with Chainer or Pool leads to richer network activations with greater predictive power to estimate critical game-state features compared to the standard DQN agent.

Keywords: Adversarial Training, DQN, Pong, Deep Reinforcement Learning, Robustness, Chainer, Pool

1 Introduction

Since the introduction in 2013, DQN [Mnih et al., 2013] has been accepted as outperforming human-level play in a range of Atari games. However, a fundamental question about what these agents truly learn remains: Are they picking up on gameplay techniques and fundamental scoring concepts, or are they merely exploiting idiosyncrasies in the game dynamics? As an example, a small change to the opponent policy can totally confuse a standard DQN-trained agent because it overfits to the opponent gameplay weaknesses instead of learning the essential skills of the game [Witty et al.]. Furthermore, are there ways of training Deep RL agents that can facilitate their learning of the essential rules of the game, leading to more robust policies? We aim to answer these questions in the game of Pong, showing that adversarial training schemes can extract more context-relevant information leading to stronger policies.

2 Background

The environment is modeled as a a time-homogeneous Markov Decision Process (X, A, R, P, γ). As usual, X and A represent state and action spaces, respectively, while the transition kernel $P(\cdot|x, a)$ defines the dynamics of the environment. Reward function $R(x, a)$ provides a signal to direct learning. Deep Q Networks, or DQN, is an approach to learning to optimize behavior in such environments. It applies several strategies to stabilize learning compared to naive learning approaches: replay buffer sampling, a target network, and frame pre-processing tricks.

Recent years have seen an uptick in the attention researchers have paid to questions of generalization of reinforcement-learning models. Packer et al. [2018] found that the vanilla deep RL algorithms have better generalization performance than specialized schemes that were proposed specifically to tackle generalization. In addition, Kansky et al. [2017] argued that generalizing from limited data and learning causal relationships are essential abilities on the path toward general intelligent systems.

Probe techniques used by Tenney et al. [2019] were useful to determine where different kinds of linguistic information is encoded in a large language model by correlating network activations with known properties of inputs. We apply these methods to our work, using linear probing to investigate the ability of the learned networks to support a heuristic task that well-trained agents should be able to perform well on.

Up until now, DQN in Pong has been trained non-adversarially, where a single agent competes against a static opponent modelled by the computer. The thesis of this paper is that adversarially trained agents are better than those trained non-adversarially in that they should develop policies prepared to handle a broader array of scenarios.

We observe in the training of a *standard DQN agent* (referring to a learner trained non-adversarial) in Pong that the agent learns to exploit weaknesses in the built-in Atari opponent—after a few million training steps, it perfects a "kill shot" aimed always at the upper or lower quadrants of the board that the built-in Atari agent cannot return. In our work, we change the dynamics of the opponent, and we demonstrate that doing so pushes the learner to focus on learning a more generalizable understanding of the game itself, robustly improving its skills in the process.

3 Methodology

Our adversarial training schemes pit the current agent against a previous version of itself. The key difference between the two algorithms we propose is *which* previous agent the current agent faces. We used the Gym Retro Atari implementation of Pong, which allows us to control both the left and right paddles independently. We chose Pong as a test bed because it is challenging, but small scale, allowing many training runs in a short amount of time.

In addition, Pong is well suited to adversarial training because the board is vertically symmetric, allowing us to easily create self-playing agents using a single policy. Specifically, if an agent is trained on one side, we can symmetrically reflect the game board as input to the agent to apply the same policy to the opposite side. To avoid overfitting the agent to a single side of the board, we randomly assign the agent a game side to play periodically. Under this uniform setting, we empirically tested two methods of self-play, *Pool* and *Chainer*, described in the following sections. We use the standard DQN agent as a baseline and our empirical results show that it is much weaker than the pool and chainer agents.

3.1 Pool

Pool, described in Algorithm 1, plays against a randomly selected previous agent from a queue of fixed size. Periodically, the current version of the agent is frozen and appended to the queue, creating a set of most up-to-date adversaries to play against. The opponent is updated periodically by sampling from this queue. In practice, we set the queue size to 5 and add new agents to the pool every $250,000$ steps. 256

3.2 Chainer

Chainer, described in Algorithm 2, plays the current version of the agent, A^n, solely against its immediate predecessor, A^{n-1}. When A_n achieves a fixed evaluation threshold, defined here as getting above an average score of 15 over 10 matches, the current agent A^n replaces the opponent's policy and a successor A^{n+1} will continue to be trained from the policy of A^n. We initialize the first member of the chain, A^0, with a standard DQN agent trained for 45 million steps. Agent A^1 is trained from scratch (*tabla rasa*) to play against agent A^0. This training approach allows the agent to continually improve its skill without overfitting to a specific opponent (which would stagnate the skill learning of the agent).

Algorithm 1 Pool Self-Play Algorithm

$P \leftarrow p$, pool update frequency
$S \leftarrow s$, step counter
Initialize *Agent*
Initialize *Env*
Initialize *Pool* of fixed size
Initialize *Opponent*
while True **do**
  interact(*Agent*, *Opponent*, *Env*)
  update(*Agent*)
  if $S\%P == 0$ **then**
    Pool $\leftarrow$ *freeze*(*Agent*)
    Opponent $\leftarrow$ *sample*(*Pool*)
  end if
  $S+ = 1$
end while

Algorithm 2 Chainer Self-Play Algorithm

$S \leftarrow 0$, the number of steps
$C \leftarrow c$, the evaluation frequency
$i \leftarrow 1$, current length of chain
$t \leftarrow 15$, the iteration threshold
Initialize A^i, the current agent.
Initialize Opponent A^0, the standard DQN agent
while True **do**
  interact(*Agent*: A^i, *Opponent*: A^{i-1}, *Env*)
  update(*Agent*: A^i)
  if $S\%C == 0$ **then**
    average_score = evaluate(A^i, A^{i-1})
    if average_score $>$ t **then**
      Opponent $\leftarrow A^i$
      $i+ = 1$
    end if
  end if
  $S+ = 1$
end while

3.3 Linear Probing

Linear Probing is a technique used to determine if the learned representation in layers of a neural network contains information relevant for solving a sample task. This entails using simple machine-learning models like linear regressors to map layer activations to target values [Tenney et al., 2019]. Designing the heuristic task is entirely problem-dependent; the task often includes targets that a human expects is important toward achieving the end goal.

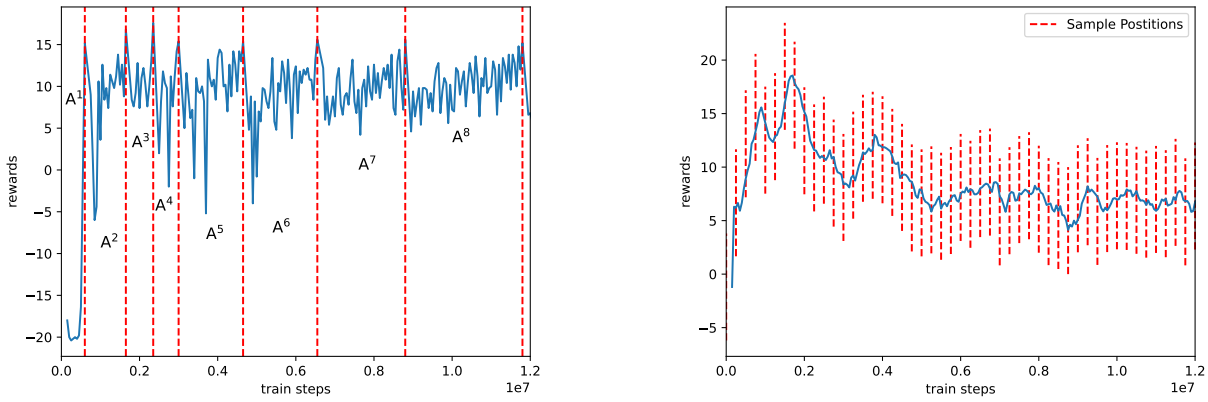
In the setting of Pong, we assume that a good agent should have the ability to predict, with high accuracy, where the ball is going to land on its side of the game board. Therefore, we attempted to linearly map the agent's final layer activations (a 512-length vector) to the landing y position of the ball. A ball landing at the top of the board on the agent's side has a landing value of 0, and a ball landing on the bottom has a landing value of 82, the height of board in pixels. We collect a training and test set of $10,000$ frames each; frames in this set are taken ≤ 30 steps before the ball collides with the agent's side of board. We report the MSE and R^2 statistic using A^7 from Chainer, Pool trained with 25 million steps, and the standard DQN agent trained for 45 million steps.

4 Experiments

In this section, we show empirical results for Chainer and Pool. We present learning curves, a new robustness metric, and the linear probing results.

4.1 Chainer and Pool Results

The Chainer learning curve shown in Figure 1 (a) shows the game reward achieved throughout training for different iterations of the opponent. Each vertical red dashed line represents a changing of the opponent (for example, the first dashed line represents the point in time when the opponent becomes A^2, the frozen agent at the second iteration of the chain). We move on to the next agent in the chain after the current agent has converged, which we define as reaching an average score of 15 over 10 matches against the opponent (21-point game). Notice how, as training progresses, the number of steps it takes for the agent to converge against its adversary increases—indicating the strength of opponents is increasing in the chain.



(a) Chainer

(b) Pooling

Figure 1: Learning Curves for Chainer and Pool

The Pooling learning curve in Figure 1 (b) shows the game reward achieved throughout training for different samplings of opponents from the queue. In this graph, the vertical dashed lines represent points in time where we sample a new opponent from the pool (every $250,000$ steps). Notice that, as time goes on, the overall reward curve dips downward. That is because the agents in the pool are getting better, making it more difficult for the current agent to perform well. Also note, in comparison to Figure 1 (a), this curve is not training agents to convergence. As a result, the agent finds it more difficult to adapt to the changing opponent. We show in Table 2 how, even though the agent is challenged by these new, more powerful opponents, it retains the ability to defeat the standard DQN agent.

4.2 Robustness

A truly robust agent should be hard to defeat by an opponent policy because its play is strong across the board. Thus, we define the *robustness* of an agent as the difficulty for an opponent to defeat it through learning. More robust agents will have flatter adversary score curves, indicating less reward achieved by the opponent over the course of learning. We picked the scores achieved by the opponent after $6e6$ steps, after which point the learning curve was stable with no large fluctuations.

In our robustness experiments, we tested 3 Pooling agents ($A_{0.5\text{M}}, A_{4\text{M}}, A_{25\text{M}}$) and 3 Chainer agents ($A^1, A^4, A^7$). Later agents are consistently more robust than earlier ones, as shown by their lower lines in the graphs. In comparison, the standard DQN agent is easily defeated by an opponent as shown in Figure 2 (opponent trained against standard DQN reaches 21 within about 1 million steps).

4.3 Linear Probing

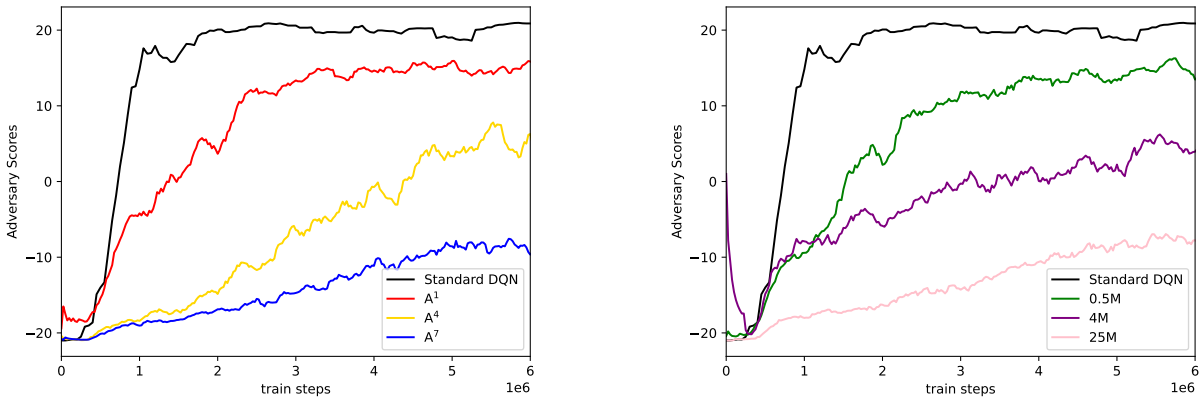
The results for Linear Probing are shown in Table 1. Probes for Chainer and Pool have significantly lower MSE and higher R^2 statistics than the probe for the standard DQN agent. A regression model that randomly guesses the landing y position of the ball would have MSE $\approx (82-41)^2 \approx 1600$. It is important to note that while the true model regressing network activations onto the landing y position of the ball is likely non-linear, even using the crudest form of estimation with a linear model shows that the activations from adversarially trained agents have greater predictive power on this task, suggesting these agents more thoroughly capture the dynamics of the game.

Training Scheme	MSE	R^2
Standard DQN	276.97	0.41
Pool	211.84	0.55
Chainer	202.41	0.57

Table 1: Linear Probing Results

4.4 Adversarial Agents vs. Standard DQN

An obvious question that the learning curves in Figure 1 raise is whether the adversarial agents can consistently defeat the standard DQN. For example, do Chainer agents in link A^{n+1} remember how to defeat the standard DQN agent (A^0, trained for 45 million steps) even after being trained to defeat more difficult opponents? To answer this type of question, we play Pooling agents ($A_{0.5\text{M}}, A_{4\text{M}}, A_{25\text{M}}$) and Chainer agents ($A^1, A^4, A^7$) against the standard DQN agent and report the 10-match average score achieved. The general trend is clear, as agents get further along in the adversarial training,



(a) Chainer Robustness Score

(b) Pooling Robustness

Figure 2: Robustness Tests (lower is more robust)

they are able to still easily defeat the standard DQN agent (indicating that these gameplay skills are not forgotten). In fact, their relative strength appears to be steadily increasing. We reiterate that it is not simply a matter of the adversarial agents being trained longer—the longest training agents have just over half the experience used to train compared to the standard DQN agent. The difference is that the adversarial agents are taught to handle a wider variety of situations, resulting in stronger overall play.

Chainer Agents	Scores Achieved Against Standard DQN	Pooling Agents	Scores Achieved Against Standard DQN
A^1	16.7	$0.5M$	18.2
A^4	19.5	$4M$	15.1
A^7	20	$10M$	20.2
		$25M$	20.4

Table 2: Adversarial Agents vs. Standard DQN Agent (trained for 45 million steps)

5 Conclusion

These results validate adversarial training schemes as a way to produce robust agents capable of opponent generalization in Pong. Agents trained using either Chainer or Pool are capable of handily defeating a standard DQN agent. Lastly, Chainer and Pool agents are more robust (more difficult to defeat by a learning opponent) and have better informed internal activations when compared to standard DQN agents.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1312.5602, December 2013.

Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael L. Littman, and David D. Jensen. Measuring and characterizing generalization in deep reinforcement learning. *Applied AI Letters*. doi: http://doi.org/10.1002/ail2.45.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing Generalization in Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1810.12282, October 2018.

Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. *arXiv e-prints*, art. arXiv:1706.04317, June 2017.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT Rediscovers the Classical NLP Pipeline. *arXiv e-prints*, art. arXiv:1905.05950, May 2019.

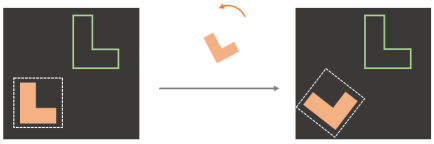
Equivariant Transporter Network

Haojie Huang **Dian Wang** **Robin Walters** **Robert Platt**
Department of Computer Science
Northeastern University
Boston, MA 02115
{huang.haoj; wang.dian; r.walters} @northeastern.edu; rplatt@ccs.neu.edu

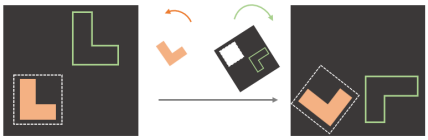
Abstract

Many challenging robotic manipulation problems can be viewed through the lens of a sequence of pick and pick-conditioned place actions. Recently, Transporter Net proposed a framework for pick and place that is able to learn good manipulation policies from a very few expert demonstrations [3]. A key reason why Transporter Net is so sample efficient is that the model incorporates rotational equivariance into the pick-conditioned place module, i.e., the model immediately generalizes learned pick-place knowledge to objects presented in different pick orientations. This work proposes a novel version of Transporter Net that is equivariant to both pick and place orientation. As a result, our model immediately generalizes pick-place knowledge to different place orientations in addition to generalizing pick orientation as before. Ultimately, our new model is more sample efficient and achieves better pick and place success rates than the baseline Transporter Net model. Our experiments show that only with 10 expert demonstrations, Equivariant Transporter Net can achieve greater than 95% success rate on 7/10 tasks of unseen configurations of Ravens-10 Benchmark. Finally, we augment our model with the ability to grasp using a parallel-jaw gripper rather than just a suction cup and demonstrate it on both simulation tasks and a real robot.

Keywords: Learning from Demonstrations, Robotics, Rotation Equivariance



(a) If Transporter Network [3] learns to pick and place an object when it is presented in one orientation, the model is immediately able to generalize to new object orientations.



(b) Our proposed Equivariant Transporter Network is able to generalize over both pick and place orientation. We view this as $C_n \times C_n$-equivariace of the model.

Figure 1. C_n-equivariace vs. $C_n \times C_n$-equivariace

1 Introduction

Pick and place is an important topic in manipulation due to its value in industry. Traditional assembly methods in factories require customized workstations so that fixed pick and place actions can be manually predefined. Recently, considerable research has focused on end-to-end visioned based models that directly map input observations to actions, which can learn quickly and generalize well. However, due to the large action space, these methods often require copious amounts of data.

A recent sample-efficient framework, Transporter Net, detects the pick-conditioned place pose by performing the cross convolution between an encoding of the scene and an encoding of a stack of differently rotated image patches around the pick. As a result of this design, Transporter Net is equivariant with respect to pick orientation. As shown in Figure 1a, if the model can correctly pick the pink object and place it inside the green outline when the object is presented in one orientation, it is automatically able to pick and place the same object when it is presented in a different orientation. This symmetry over object orientation enables Transporter Net to generalize well and it is fundamentally linked to the sample efficiency of the model. Assuming that pick orientation is discretized into n possible gripper rotations, we will refer to this as a C_n pick symmetry, where C_n is the finite cyclic subgroup of $\mathrm{SO}(2)$ that denotes a set of n rotations.

Although Transporter Net is C_n-equivariant with regard to pick, the model does not have a similar equivariance with regard to place. That is, if the model learns how to place an object in one orientation, that knowledge does not generalize immediately to different place orientations. This work seeks to add this type of equivariance to the Transporter Network model by incorporating C_n-equivarant convolutional layers into both the pick and place models. Our resulting model is equivariant both to changes in pick object orientation and changes in place orientation. This symmetry is illustrated in Figure 1b and can be viewed as a direct product of two cyclic groups, $C_n \times C_n$. Enforcing equivariance with respect to an even larger symmetry group than Transporter Net leads to even greater sample efficiency since equivariant neural networks learn effectively on a lower dimensional action space, the equivalence classes of samples under the group action.

2 Background on Symmetry Groups

2.1 Representation of a Group

We are primarily interested in rotations expressed by the group $\mathrm{SO}(2)$ and its cyclic subgroup $C_n \leq \mathrm{SO}(2)$. $\mathrm{SO}(2)$ contains the continuous planar rotations $\{\mathrm{Rot}_\theta : 0 \leq \theta < 2\pi\}$. The discrete subgroup $C_n = \{\mathrm{Rot}_\theta : \theta \in \{\frac{2\pi i}{n} | 0 \leq i < n\}\}$ contains only rotations by angles which are multiples of $2\pi/n$. The special Euclidean group $\mathrm{SE}(2) = \mathrm{SO}(2) \times \mathbb{R}^2$ describes all translations and rotations of $\mathbb{R}^2$.

A d-dimensional *representation* $\rho \colon G \to \mathrm{GL}_d$ of a group G assigns to each element $g \in G$ an invertible $d \times d$-matrix $\rho(g)$. Different representations of $\mathrm{SO}(2)$ or C_n help to describe how different signals are transformed under rotations. For example, the trivial representation $\rho_0 \colon \mathrm{SO}(2) \to \mathrm{GL}_1$ assigns $\rho_0(g) = 1$ for all $g \in G$, i.e. no transformation under rotation. The standard representation $\rho_1(\mathrm{Rot}_\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ represents each group element by its standard rotation matrix. The regular representation ρ_{reg} of C_n acts on a vector in $\mathbb{R}^n$ by cyclically permuting its coordinates $\rho_{\mathrm{reg}}(\mathrm{Rot}_{2\pi/n})(x_0, x_1, ..., x_{n-2}, x_{n-1}) = (x_{n-1}, x_0, x_1, ..., x_{n-2})$. For more details, we refer the reader to Weiler and Cesa [1].

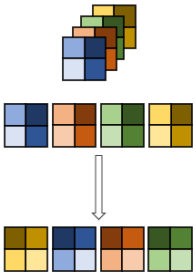


Figure 2. Illustration of the action of T_g^{reg} on a 2×2 image.

2.2 Equivariant Mappings

We formalize images and feature maps as feature vector fields, i.e. functions $f \colon \mathbb{R}^2 \to \mathbb{R}^c$, which assign a feature vector $f(\mathbf{x}) \in \mathbb{R}^c$ to each position $\mathbf{x} \in \mathbb{R}^2$. The action of a rotation $g \in \mathrm{SO}(2)$ on f is a combination of a rotation in the domain of f via ρ_1 (this rotates the pixel positions) and a rotation in the channel space $\mathbb{R}^c$ by $\rho \in \{\rho_0, \rho_{\mathrm{reg}}\}$. If $\rho = \rho_{\mathrm{reg}}$, then the channels cyclically permute according to the rotation. If $\rho = \rho_0$, the channels do not change. We denote this action (the action of g on f via ρ) by $T_g^\rho(f)$:

$$[T_g^\rho(f)](\mathbf{x}) = \rho(g) \cdot f(\rho_1(g)^{-1}\mathbf{x}). \tag{1}$$

For example, the action of $T_g^{\rho_{\text{reg}}}(f)$ is illustrated in Figure 2 for a rotation of $g = \pi/2$ on a 2×2 image f that uses ρ_{reg}. For brevity, we will denote $T_g^{\text{reg}} = T_g^{\rho_{\text{reg}}}$ and $T_g^0 = T_g^{\rho_0}$. A functional mapping F is *equivariant* to a group G if the output produced by F transforms consistently when the input transforms under the action of an element $g \in G$,

$$T_g^{\text{out}}[F(f)] = F(T_g^{\text{in}}[f]) \tag{2}$$

where T_g^{in} transforms the input of F by the group element g while T_g^{out} transforms the output of F by g.

3 Transporter Network

Before describing our variation on Transporter Net, we summarize the the pick and place problem and the original Transporter Net architecture described in [3].

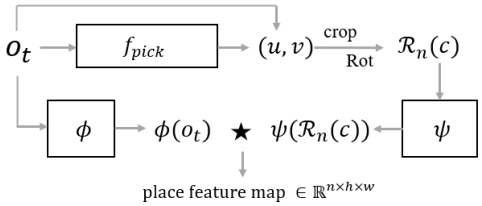


Figure 3. The Architecture of Transporter Net.

3.1 Problem Statement

We define the *Planar Pick and Place* problem as follows. Given a visual observation o_t, the problem is to learn a probability distribution $p(a_{\text{pick}}|o_t)$ over picking actions $a_{\text{pick}} \in \text{SE}(2)$ and a distribution $p(a_{\text{place}}|o_t, a_{\text{pick}})$ over placing actions $a_{\text{place}} \in \text{SE}(2)$ conditioned on a_{pick} that accomplishes some task of interest. The visual observation o_t is typically a projection of the scene (e.g., top-down RGB-D images) and the pose of the end effector is expressed as (u, v, θ) where u, v denote the pixel coordinates of the gripper position and θ denotes gripper orientation. (Since [3] uses suction cups to pick, that work ignores pick orientation.)

3.2 Description of Transporter Net

Transporter Network [3] solves the planar pick and place problem using the architecture shown in Figure 3. The pick network $f_{\text{pick}} : o_t \mapsto p(u, v)$ maps o_t onto a probability distribution $p(u, v)$ over pick position $(u, v) \in \mathbb{R}^2$. The output pick position a^*_{pick} is calculated by maximizing $f_{\text{pick}}(o_t)$ over (u, v). The place position and orientation is calculated as follows. First, an image patch c centered on a^*_{pick} is cropped from o_t to represent the pick action as well as the object. Then, the crop c is rotated n times to produce a stack of n rotated crops.

We will denote this stack of crops as: $\mathcal{R}_n(c) = (T^0_{2\pi i/n}(c))_{i=0}^{n-1}$, where we refer to $\mathcal{R}_n$ as the "lifting" operator. Then, $\mathcal{R}_n(c)$ is encoded using a neural network ψ. The original image, o_t, is encoded by a separate neural network ϕ. The distribution over place location is evaluated by taking the cross correlation between ψ and ϕ,

$$f_{\text{place}}(o_t, c) = \psi(\mathcal{R}_n(c)) \star \phi(o_t), \tag{3}$$

where ψ is applied independently to each of the rotated channels in $\mathcal{R}_n(c)$. Place position and orientation is calculated by maximizing f_{place} over the pixel position (for position) and the orientation channel (for orientation).

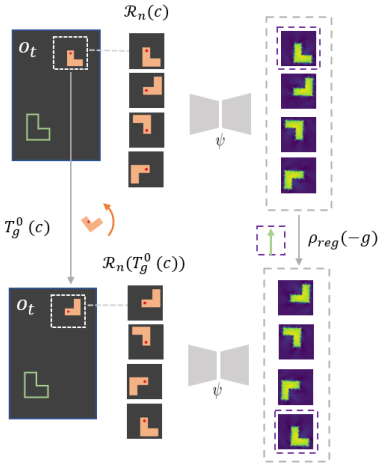


Figure 4. Illustration of the main part of the proof of Proposition 1. Rotating the crop c induces a cyclic shift in the channels of the output $\psi(\mathcal{R}_n(T_g^0 c)) = \rho_{\text{reg}}(-g)\psi(\mathcal{R}_n(c))$.

3.3 Equivariance of Transporter Net

Proposition 1 *The Transporter Net place network f_{place} is C_n-equivariant. That is, given $g \in C_n$, object image crop c and scene image o_t,*

$$f_{\text{place}}(o_t, T_g^0(c)) = \rho_{\text{reg}}(-g) f_{\text{place}}(o_t, c). \tag{4}$$

As shown in Figure 4, a rotation of g applied to the orientation of the object to be picked results in a $-g$ change in the placing angle, which is represented by a permutation along the channel axis of the placing feature maps. This is a symmetry over the cyclic group C_n which is encoded directly into the model. It enables it to immediately generalize over different orientations of the object to be picked and thereby improves sample efficiency. Note that here ψ is a simple CNN with no rotational equivariance. The equivariance results from the lifting $\mathcal{R}_n$. Instead, our proposed method incorporates the rotational equivariance in the pick network and $C_n \times C_n$-equivariance in the place network.

4 Equivariant Transporter

4.1 Equivariant Pick

We propose a C_n-equivariant model for detecting the planar pick action, as shown in Figure 5. First, we decompose the learning process of $a_{\text{pick}} \in \text{SE}(2)$ into two parts: $p(a_{\text{pick}}) = p(u, v)p(\theta|(u, v))$, where $p(u, v)$ denotes the probability that a pick exists at pixel coordinates u, v and $p(\theta|(u, v))$ is the conditioned probability of gripper orientation of θ. The distributions $p(u, v)$ and $p(\theta|(u, v))$ are modeled as two neural networks:

$$f_p(o_t) \mapsto p(u, v), \quad f_\theta(o_t, (u, v)) \mapsto p(\theta|(u, v)).$$

There are two equivariance relationships that we would expect to be satisfied for planar picking:

$$f_p(T_g^0(o_t)) = T_g^0(f_p(o_t)) \tag{5}$$

$$f_\theta(T_g^0(o_t), T_g^0(u, v)) = \rho_{reg}(g)(f_\theta(o_t, (u, v))). \tag{6}$$

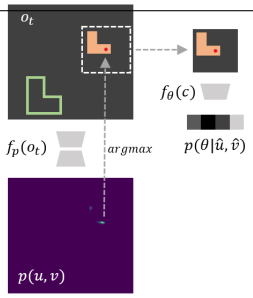


Figure 5. Equivariant Transporter Pick model. First, we find the pick position a^*_{pick} by evaluating the argmax over $f_p(o_t)$. Then, we evaluate f_θ for the image patch centered on a^*_{pick}.

Equation 5 states that the grasp points found in an image rotated by $g \in \text{SO}(2)$, (LHS of Equation 5), should correspond to the grasp points found in the original image subsequently rotated by g, (RHS of Equation 5). Equation 6 says that the grasp orientation at the rotated grasp point should be shifted by $g = 2\pi i$ relative to the grasp orientation at the original grasp points in the original image. We encode both f_p and f_θ using equivariant convolutional layers [1] which constrain the models to represent only those functions which satisfy Equations 5 and 6.

4.2 Equivariant Place

Given the picked object represented by the image patch c centered on a_{pick}, the place network models the distribution of $a_{\text{place}} = (u_{\text{place}}, v_{\text{place}}, \theta_{\text{place}})$ by: $f_{\text{place}}(o_t, c) \mapsto p(a_{\text{place}}|o_t, a_{\text{pick}})$. Our place model architecture closely follows that of Transporter Net [3] except that we explicitly encode equivariance constraints on both ϕ and ψ networks, i.e., $\psi(T_g^0(c)) = T_g^0(\psi(c))$ and $\phi(T_g^0(o_t)) = T_g^0(\phi(o_t))$. As a result of this change: 1) we are able to simplify the model by transposing the lifting operation $\mathcal{R}_n$ and the processing by ϕ; 2) our new model is equivariant with respect to a larger symmetry group $C_n \times C_n$, compared to Transporter Net which is only equivariant over C_n.

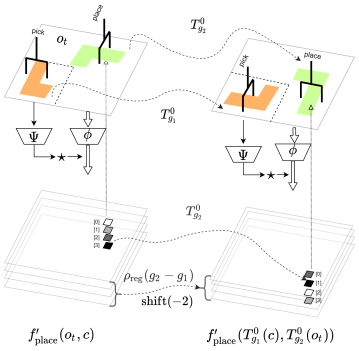


Figure 6. Equivariance of our placing network under the rotation of the object and the placement. A $\frac{\pi}{2}$ rotation on c and a $\frac{-\pi}{2}$ rotation on $o_t \backslash c$ are equivariant to: i), a $\frac{-\pi}{2}$ rotation on the placing location, and ii), the shift on the channel of placing rotation angle from $\frac{3\pi}{2}$ (the last channel) to $\frac{\pi}{2}$ (the second channel).

When ψ is C_n-equivariant, we can exchange $\mathcal{R}_n$ (the lifting operation) with ψ: $\psi(\mathcal{R}_n(c)) = \mathcal{R}_n(\psi(c))$. This equality is useful because it means that we only need to evaluate ψ for one image patch rather than the stack of image patches $\mathcal{R}_n(c)$ – something that is computationally cheaper. The resulting place model is then:

$$f'_{\text{place}}(o_t, c) = \mathcal{R}_n(\psi(c)) \star \phi(o_t) = \Psi(c) \star \phi(o_t), \tag{7}$$

where Equation 7 substitutes $\Psi(c) = \mathcal{R}_n[\psi(c)]$ to simplify the expression. Note that we use f'_{place} to denote Equivariant Transporter Net.

As Proposition 1 demonstrates, the baseline Transporter Net model [3] encodes the symmetry that rotations of the object to be picked (represented by c) should result in corresponding rotations of the place orientation for that object. However, pick and place problems have a second symmetry that is not encoded in Transporter Net: that rotations of the scene image (represented by o_t) should also result in corresponding rotations of the place orientation. We encode this second type of symmetry by enforcing the rotation equivariance of ϕ and ψ. Essentially, we go from a C_n-symmetric model to a $C_n \times C_n$-symmetric model.

Proposition 2 *Equivariant Transporter Net f'_{place} is $C_n \times C_n$-equivariant. That is, given rotations $g_1 \in C_n$ of the picked object and $g_2 \in C_n$ of the scene, we have that:*

$$f'_{\text{place}}(T^0_{g_1}(c), T^0_{g_2}(o_t)) = \rho_{\text{reg}}(g_2 - g_1) T^0_{g_2} f'_{\text{place}}(c, o_t). \tag{8}$$

The top of Figure 6 going left to right shows the rotation of both the object by g_1 (in orange) and the place pose by g_2 (in green). The LHS of Equation 8 evaluates f'_{place} for these two rotated images. The lower left of Figure 6 shows $f'_{\text{place}}(c, o_t)$. Going left to right at the bottom of Figure 6 shows the pixel-rotation by $T^0_{g_2}$ and the channel permutation by $g_2 - g_1$ (RHS of Equation 8).

Note that in additional to the two rotational symmetries enforced by our model, it also has translational symmetry. Since the rotational symmetry is realized by additional restrictions to the weights of kernels of convolutional networks, the rotational symmetry is in addition to the underlying shift equivariance of the convolutional network. Thus, the full symmetry group enforced is the group generated by $C_n \times C_n \times (\mathbb{R}^2, +)$.

5 Experiments

	block-insertion				place-red-in-green				towers-of-hanoi				align-box-corner				stack-block-pyramid			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Equivariant Transporter	**100**	**100**	**100**	**100**	**98.5**	**100**	**100**	**100**	**88.1**	**95.7**	**100**	**100**	41.0	**99.0**	**100**	**100**	**34.6**	**80.0**	**90.8**	**95.1**
Transporter Network	**100**	**100**	**100**	**100**	84.5	**100**	**100**	**100**	73.1	83.9	97.3	98.1	35.0	85.0	97.0	98.0	13.3	42.6	56.2	78.2

	palletizing-boxes				assembling-kits				packing-boxes				manipulating-rope				sweeping-piles			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Equivariant Transporter	**75.3**	**98.9**	**99.6**	**99.6**	**63.8**	**90.6**	**98.6**	**100**	**98.3**	**99.4**	**99.6**	**100**	**31.0**	**85.0**	**92.3**	**98.4**	**97.9**	**99.5**	**100**	**100**
Transporter Network	63.2	77.4	91.7	97.9	28.4	78.6	90.4	94.6	56.8	58.3	72.1	81.3	21.9	73.2	85.4	92.1	52.4	74.4	71.5	96.1

Table 1. **Performance comparisons on Ravens-10 benchmark (suction gripper).** Success rate (mean%) vs. the number of demonstration episodes (1, 10, 100, or 1000) used in training. Best performances are highlighted in bold.

	block-insertion			place-red-in-green			palletizing-boxes			align-box-corner			stack-block-pyramid		
Method	1	10	100	1	10	100	1	10	100	1	10	100	1	10	100
Equivariant Transporter	**100**	**100**	**100**	**95.6**	**100**	**100**	**96.1**	**100**	**100**	**64.0**	**99.0**	**100**	**62.1**	**85.6**	**98.3**
Transporter Network	98.0	**100**	**100**	82.3	94.8	**100**	84.2	99.6	**100**	45.0	85.0	99.0	16.6	63.3	75.0

Table 2. **Performance comparisons on tasks with a parallel-jaw end effector.** Success rate (mean%) vs. the number of demonstration episodes (1, 10, or 100) used in training.

5.1 Results for the Ravens-10 Benchmark Tasks

Ravens-10 [3] is a behaviour cloning simulation environment for manipulation built on Pybullet. For each task, it could produce a dataset of n expert demonstrations, where each demonstration contains a sequence of one or more observation-action pairs $(o_t, \bar{a}_t)$. We use $\bar{a}_{\text{pick}}$ and $\bar{a}_{\text{place}}$ to generate one-hot labels to train our networks with cross-entropy loss. Performance of our model is measured in the same way as [3] – using a metric in the range of 0 (failure) to 100 (success). Partial scores are assigned to multiple-action tasks. We report the highest test performance during training, averaged over 100 unseen tests for each task. As illustrated in Table 1, our proposed Equivariant Transporter Net outperform the baseline by orders of magnitude more sample efficiency. We believe it is due to the inductive bias of the $C_n \times C_n$-equivariance.

Task	# demos	# completions / # trials	success rate
stack-block-pyramid	10	17/20	95.8%
place-box-in-bowl	10	20/20	100%
block-insertion	10	20/20	100%

Table 3. Task success rates for physical robot evaluation tasks.

5.2 Results for Parallel Jaw Gripper Tasks

We selected 5 tasks from Ravens-10 and replaced the suction cup with the Franka Emika gripper, which require additional picking angle inference. To fit Transporter Network to parallel-jaw gripper tasks, we accomplish it by [2] lifting the input scene image over C_n, producing a stack of differently oriented input images as input to the pick network f_{pick}. The results are counter-rotated at the output of f_{pick}. Table 2 compares the performance of Equivariant Transporter with the baseline Transporter Net for the Parallel Jaw Gripper tasks.

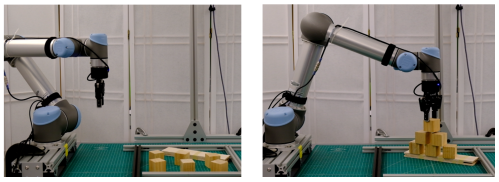


Figure 7. Stack-block-pyramid task on the real robot. The left figure shows the initial state; the right figure shows the completion state.

Finally, We evaluated Equivariant Transporter on a physical robot in our lab. As shown in Figure 7, we used a UR5 robot with a Robotiq-85 end effector. The workspace was a $40cm \times 40cm$ region on a table beneath the robot. The observations o_t were 200×200 depth images obtained using a Occipital Structure Sensor that was mounted pointing directly down at the table. We collected 10 human demonstrations for each of the three selected tasks. Table 3 shows results from 20 runs of each of the three tasks. Notice that the success rates here are higher than those in simulation (Table 2). This is likely caused by the fact that the criteria for task success in simulation (less than 1 cm translational error and $\frac{\pi}{12}$ rotation error) were more conservative than the real world experiment.

References

[1] Maurice Weiler and Gabriele Cesa. General $e(2)$-equivariant steerable cnns. *arXiv preprint arXiv:1911.08251*, 2019.

[2] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[3] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020. 264

Estimating the Capacity for Cognitive Control Based on Psychometric Choice Theory

Huang Ham
Department of Psychology
University of Pennsylvania
hamhuang@sas.upenn.edu

Ivan Grahek
Department of Cognitive, Linguistic
Psychological Sciences
Brown University
ivan_grahek@brown.edu

Laura A. Bustamante
Princeton Neuroscience Institute
Princeton University
lauraab@princeton.edu

Nathaniel Daw
Princeton Neuroscience Institute
Princeton University
ndaw@princeton.edu

Andrew Caplin
Department of Economics
New York University
ac1@nyu.edu

Sebastian Musslick
Department of Cognitive, Linguistic
Psychological Sciences,
Carney Institute for Brain Science
Brown University
sebastian_musslick@brown.edu

Abstract

Recent years have witnessed significant advances in our understanding of bounds on rationality in both cognitive psychology and economics. These two fields have been making separate progress, but time is ripe for unifying these efforts. In this article, we introduce recently developed economic tools, themselves rooted in the psychometric tradition, to quantify individual differences in the capacity for cognitive control. These tools suggest that a reliable assessment of the capacity for cognitive control may be accomplished by examining task performance as a function of reward. We demonstrate through simulation studies that an incentive-informed measure of task performance does a better job of recovering individual differences in one's capacity for cognitive control, compared to the commonly used congruency effect. Furthermore, we show that the economic approach can be used to predict control-dependent behavior across different task settings. We conclude by discussing future directions for the fruitful integration of behavioral economics and cognitive psychology with the aim of improved measurement of individual differences in the capacity for cognitive control.

Keywords: mental effort; executive function; individual differences; rational inattention; Bayesian revealed preference

Acknowledgements

This research is supported by Schmidt Science Fellows, in partnership with the Rhodes Trust.

1 Introduction

Research in value-based decision-making has typically focused on studying how people weigh the value of each option to guide their choice. However, different decision problems can require different levels of cognitive control—the ability to guide information processing in the service of a task—to adequately process all information relevant to the goal of the decision problem. Thus, quantifying a person's capacity for cognitive control may enable researchers to better predict their decisions and task performance in the real world. Researchers have developed a variety of paradigms to index a given person's capacity to exert cognitive control. Most commonly, these paradigms require the participant to categorize a target stimulus while ignoring one or multiple distractors [4]. For instance, in the Stroop task, participants are required to name the ink color of a color word (e.g., say "red" in response to "GREEN") while ignoring the word. Such interference tasks allow for the computation of a congruency cost: the difference in performance (e.g., accuracy) on trials in which target and distractor are associated with the same response (congruent trials, e.g. "RED" in the Stroop task) and trials in which the distractor interferes with the target to produce a competing response (incongruent trails, e.g., "GREEN" in the Stroop task). However, task performance measures that are supposed to target the same construct (e.g., the inhibition of distraction information) are poorly correlated with one another [12]. Mounting theoretical work suggests that the low predictive validity of performance- and choice-based measures derives from a confound between a person's *capacity* for cognitive control and their *willingness* to exert it [8, 7]. In this article, we leverage the psychometric theory of rational inattention [2], to derive an incentive-informed measure of cognitive control capacity. We then build on a well-established model of control allocation, to simulate individual differences in task performance. We examine the internal and predictive validity of the introduced measures, by extracting them from the simulated performance and by correlating them with internal variables that determine an agent's capacity for cognitive control. We demonstrate that the assessment of performance across incentives provides a better index of cognitive control capacity and that it may provide a reliable predictor of cognitive performance across individuals and tasks.

2 The Psychometrics of Rational Inattention

Our psychometric framework builds upon the intellectual tradition of bounded rationality, assuming that human cognition is rational within certain boundaries [3, 5]. While an unboundedly rational agent always chooses the 'best' action to take, the actions chosen by a boundedly rational agent may not always be the absolute 'best' but are on average as good as possible without requiring an unreasonable amount of effort. To formalize this framework in an experimental setting, we use A to denote the set of *actions* selected by a participant. For example, in a simple version of the Stroop task, a participant may choose from two relevant *actions*, to indicate the ink color of a color word (correct) or not (incorrect): $A = \{\text{correct}, \text{incorrect}\}$. The value of a chosen action typically depends on the experimental condition (e.g., whether the color of the Stroop stimulus is congruent or incongruent with the word). We call the set of experimental conditions states, denoted by S. Again in our Stroop task example, two possible experimental states (or conditions) may occur: the ink color may match the word (congruent condition) and the ink color mismatches the word (incongruent condition): $S = \{\text{congruent}, \text{incongruent}\}$. Finally, how good an action $a \in A$ is in a state $s \in S$ is quantified by an expected utility function $U : S \times A \to \mathbb{R}_{\geq 0}$. A participant that is boundedly rational must always choose according to an action strategy that maximizes the expected value of $U(s, a) - K(\text{experiment})$. Here $U(s, a)$ represents the *utility* the agent receives and K represents the (mental) cost the agent has to pay to resolve this choice problem. What we mean by an action strategy is simply the probability of choosing an action given a state, denoted as $p(a|s)$. Hence the expected value of $U(s, a) - K(\text{experiment})$ amounts to

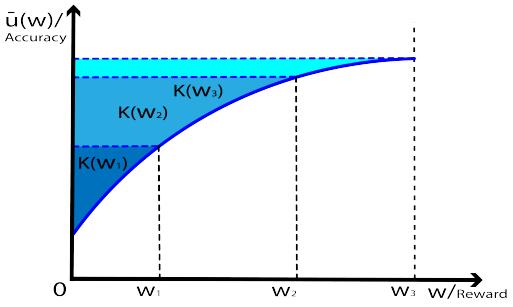


Figure 1: Caplin et al.'s K. $K(w)$ is the area of the region to the left of the revealed expected utility $\bar{u}$ curve between $\bar{u}(0)$ (the revealed expected utility for the base experiment) and $\bar{u}(w)$ (the revealed expected utility for the experiment that provides w times more of an incentive than the base experiment). w_1, w_2, and w_3 are three example values of w. In the context of Stroop tasks, $\bar{u}$ and w can be understood as response accuracy and incentive level awarded to the correct responses.

$$\sum_{s \in S} \sum_{a \in A} U(s, a) p(a|s) p(s) - K(\text{experiment})$$

where $p(s)$ denotes the probability of the state s occurring in the experiment[1]. The probability of each state $p(s)$ is set by the experimenter and the marginal probability of an action in a given state $p(a|s)$ can be estimated from the participant's

[1] Our formalism right now only considers the expected value of utility while ignoring other statistical properties such as variance. It would be an interesting extension of the current theory to incorporate a variance term to the above equation.

choice data using their choice frequency. Unfortunately, one cannot directly measure a participant's utility function U nor their cost of control K. However, U can be estimated based on the reward structure of the experiment. While there is plenty of work proposing ways to estimate U from rewards under different research contexts, [2] provide a first, general procedure for recovering the mental cost K—henceforth referred to as Caplin et al.'s K—from only the choice data. We next introduce the specific procedure of assessing K in 3 steps. We illustrate each step with both the general formulas and how they are specifically applied to our example Stroop task.

Step 1. We first obtain data from participants for a base experiment at which each participant has some utility function U. From the data, one can calculate $p(s, a) = p(a|s)p(s)$ for each pair of state and action. In this article, we obtain all the data from a simulated cognitive agent which we introduce in the next section. Because data are simulated, we can implement the assumption that reward points equal the participant's utility, for example:

$$U(s, a) = R(s, a) = \begin{cases} 1 & \text{if } a = \text{correct} \\ 0 & \text{if } a = \text{incorrect} \end{cases}$$

Step 2. Repeat step 1 for a set of n experiments that are identical to the base experiment except for the reward structure. Importantly, the reward structures differ in such a way that the participant's utility function in each experiment is some positive multiple w of U. Next for each of the experiments, we can compute the marginal probability $p_w(s, a) = p_w(a|s)p_w(s)$ of the state s and the action a from the frequencies of each unique pair of state and action in the choice data. Caplin et al. denote the expected utility $\hat{u}(w)$ of the experiment as a function of w. The main quantity of interest in our framework is the expected utility normalized by the weight w ($\bar{u}(w)$):

$$\hat{u}(w) = \sum_{s \in S} \sum_{a \in A} wU(s, a)p_w(s, a) \qquad \bar{u}(w) = \frac{\hat{u}(w)}{w} = \sum_{s \in S} \sum_{a \in A} U(s, a)p_w(s, a) \tag{1}$$

where $\bar{u}(w)$ is the revealed expected utility calculated using the choice data of the w weighted experiment and the utility function of the base experiment. In our Stroop task example, we need to simulate data from versions of the Stroop experiment with the same S and A, and $\exists w \in \mathbb{R}_{\geq 0}$ such that its reward/utility function

$$U_w(s, a) = \begin{cases} w & \text{if } a = \text{correct} \\ 0 & \text{if } a = \text{incorrect} \end{cases}$$

For each experiment, if the action amounts to the correct response (i.e., 'accurate'), the participant receives the reward of $w * 1 = w$ point, otherwise she gets $w * 0 = 0$ points. The expected utility of each of these data divided by w, i.e. $\bar{u}(w)$, can be understood as the participant's response *accuracy* where the reward is w points. Similarly, a participant's accuracy in only the congruent condition is equivalent to $\bar{u}(w|s = \text{congruent})$ and participant's accuracy in only the incongruent condition is equivalent to $\bar{u}(w|s = \text{incongruent})$. To emphasize these operational interpretations, from now on we denote

- w as R (reward for accurate responses),
- $\bar{u}(w)$ as $A(R)$ (accuracy as a function of reward),
- $\bar{u}(w|s = \text{congruent})$ as $A_C(R)$ (congruent trial accuracy), and
- $\bar{u}(w|s = \text{congruent})$ as $A_I(R)$ (incongruent trial accuracy).

Step 3 Caplin et al. prove that if the participant is indeed a boundedly rational agent, $\bar{u}$ should be a monotonically non-decreasing function [2]. If the data supports such monotonicity, one can recover K:

$$K(w) = w\bar{u}(w) - \int_0^w \bar{u}(t)dt. \tag{2}$$

The geometric intuition is illustrated in Figure 1. Using our new notations for the example Stroop task, we can compute the Caplin et al.'s K as $K(R) = RA(R) - \int_0^R A(t)dt$. Next, we compare and contrast Caplin et al.'s K with the congruency effect, which has is one of the most commonly used indicators of the capacity for cognitive control [4]. The congruency effect in the Stroop experiment with reward R, denoted by $\Delta A(R)$, is defined as the difference in accuracies on congruent and incongruent trials in the experiment with reward R, i.e., $\Delta A(R) = A_C(R) - A_I(R)$. These two measures have two crucial distinctions. First, whereas the congruency effect is restricted to interference tasks like the Stroop paradigm, Caplin et al.'s K can be applied in any task involving two or more responses. Second, while the congruency effect only depends on one experiment, Caplin et al.'s K takes advantage of the accuracy function at all reward levels. Finally, whereas the congruency effect is a theory-free metric, Caplin et al.'s K traces its roots to a rigorous and general theory rooted in the bounded rationality framework. We proceed by examining the internal and predictive validity of these metrics based on the behavior of a simulated agent, leveraging a different but related theory of control allocation.

3 Assessing the Internal and Predictive Validity of Caplin et al.'s K

267

The validity of any performance-based metric of control-demanding behavior can be assessed based on its ability to (a) track internal variables of the control system, and (b) predict behavior across tasks. Here, we considered three different metrics: the traditional congruency effect (assessed at reward $= 0$), the mean congruency effect (averaged across reward conditions), and Caplin et al.'s K. In this section, we extract these metrics from the behavior of a simulated agent in an interference task. The agent implements a model of control allocation based on the Expected Value of Control (EVC) [11]. We examine how well the introduced metrics relate to latent variables that determine the agent's capacity of control, namely (a) the efficacy with which control translates into performance and (b) the cost of control, and how well they predict performance in a second, unrelated task.

Table 1: Uniform intervals of sampled EVC parameters.

Parameter	Description	Min	Max
ϵ	control efficacy	4	5
c	control cost	0.8	0.9
ν	reward sensitivity	0.95	1.05
α_A	automaticity of Task A	-4	-2
α_B	automaticity of Task B	-4	-2
i_A	distractor interference of Task A	-2	-1
i_B	distractor interference of Task B	-2	-1
b	accuracy bias	5	5

3.1 Parameterization of Simulated EVC Agents

For the simulations reported in this article, we assume that the probability of an agent responding correctly in an interference task increases monotonically with the amount of control intensity allocated (u):

$$P(\text{correct}|u, S) = \frac{1}{1 + e^{\epsilon \cdot u - \alpha}}$$

where ϵ characterizes the control efficacy, i.e. how an increase in control signal intensity translates into changes in task accuracy. The parameter α determines the degree of task automaticity: The higher α, the easier the task, that is, the less cognitive control is needed to reach the correct outcome. Note that $\alpha = \alpha_0 - i$ depends on the amount of interference i in the current trial. For congruent trials, we assume that the agent receives no interference $i = 0$ whereas in incongruent trials, the agent receives some amount of interference $i > 0$ (effectively lowering the probability of responding correctly). The agent is assumed to choose the control signal that maximizes $EVC(u, S) = P(\text{correct}|u, S)V(R) - \text{Cost}(u)$.

The subjective value can be described as a function of the reward R provided for a correct response. Here we assume that the subjective value of the correct outcome amounts to

$$V(R) = \nu \cdot R + b$$

where R is a monetary reward that is provided in the event of a correct response, ν is the reward sensitivity of the agent and b is the baseline value that the agent assigns to being accurate (accuracy bias). Finally, the EVC agent is assumed to allocate control by taking into account an intrinsic cost that scales with control signal intensity. For the simulations reported below, we adopt an exponential cost function used in prior work [7, 8]:

$$\text{Cost}(u) = e^{c \cdot u} - 1$$

where c scales the increase in the cost of control with one unit of control signal intensity u. To simulate individual differences in EVC agents, we sampled parameters of the model from the uniform intervals listed in Table 1. The intervals were chosen such that the agents would allocate at least some amount control ($u^* > 0$) for each task condition. Note that the control efficacy ϵ and the control cost c both characterize an agent's capacity to exert control across different tasks.

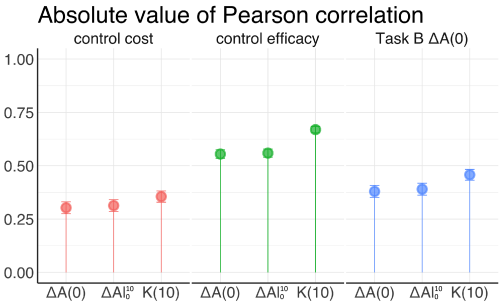


Figure 2: Correlation strength. The x-axis represents different measurements calculated from the simulated Stroop task. The y-axis represents the absolute value of Pearson correlation between the measurement and the quantity shown in the panel label. $\Delta A(0)$ is the congruency effect at reward 0, $\Delta A|_0^{10}$ is the mean congruency effect from reward 0 to 10, K(10) is Caplin et al.'s K at reward 10. Error bars are bootstrapped confidence intervals using 6000 samples.

3.2 Results

Internal Validity. To compare the internal validity in a first task (Task A), we correlated our measures each with the simulating parameter values. We found that all of our three measures correlate significantly with the control cost and control efficacy parameters ($|t| > 10$, $df = 998$, $p < 10^{-15}$). However, the correlation between the control cost and the congruency effect for $R = 0$ ($r = 0.302$) is significantly weaker than between the mean congruency effect from $R = 0$ to $R = 10$ ($r = 0.313$, $p = 0.025$) and between Caplin et al.'s K at $R = 10$ ($r = 0.355$, $p = 0.002$). The correlation between the control cost and the mean congruency effect from $R = 0$ to $R = 10$ is also significantly weaker than that between Caplin et al.'s K at $R = 10$ ($p = 0.008$). However, the correlation between the control efficacy and the congruency effect for $R = 0$

($r = -0.555$) is not significantly weaker than between the mean congruency effect from $R = 0$ to $R = 10$ ($r = -0.559$, $p = 0.155$). Note that both measures yield a significantly weaker correlation than Caplin et al.'s K at $R = 10$ ($r = -0.669$, $p < 10^{-3}$). This suggests that Caplin et al.'s K can better predict the control cost and efficacy than the congruency effect.

Predictive Validity. We found that all three metrics derived from the agents' performance in Task A correlate significantly with the congruency effect at $R = 0$ in a different task with an independently sampled automaticity and distractor interference (Task B) ($t > 12$, $df = 998$, $p < 10^{-15}$). Similarly, the correlation between the the congruency effect at reward 0 in task B and the congruency effect for $R = 0$ in task A ($r = 0.379$) is significantly weaker than between the mean congruency effect from $R = 0$ to $R = 10$ in task A ($r = 0.390$, $p = 0.017$). Caplin et al.'s K at $R = 10$ in task A correlates more strongly with the congruency effect in Task B than the other two measures ($r = 0.458$, $p < 10^{-3}$). This indicates that Caplin et al.'s K can predict better the congruency effect of another cognitive control task.

4 General Discussion

People rely on cognitive control to adjust how they process information in order to achieve their decision-making goals. Although reliable measurement of the capacity to exert cognitive control is crucial for predicting human decision-making, current measures of control (e.g., the congruency effect) show poor internal and external validity [6, 10]. Here we propose that one of the causes of this issue is that the current measurement approaches are focused on measuring control capacity at a fixed point. In this work, we introduced Caplin et al.'s K as a novel measurement technique that relies on the integration of task performance across reward conditions. This measurement device is implied by economic theories of rational inattention, resulting in a formal relationship with established metrics [2, 1], such as the congruency effect.

Building on the bounded rationality framework, we introduced Caplin et al.'s K as a choice-based measure, computed across linearly increasing incentives. We examined the validity of these metrics based on the behavior of simulated agents implementing a theory of control allocation [11, 9]. We find that Caplin et al.'s K does a better job of predicting latent variables that determine an agent's capacity for cognitive control (control efficacy and control cost) compared to the traditional congruency effect. Moreover, Caplin et al.'s K can be leveraged to predict an agent's behavior across unrelated interference tasks. These results provide initial support for the idea that considering how performance varies as a function of incentive levels is crucial for assessing the capacity for cognitive control in choice-based tasks.

References

[1] A. Caplin. Economic data engineering. Working Paper 29378, National Bureau of Economic Research, October 2021.

[2] A. Caplin, D. Csaba, J. Leahy, and O. Nov. Rational inattention, competitive supply, and psychometrics. *The Quarterly Journal of Economics*, 135(3):1681–1724, 2020.

[3] N. Chater and M. Oaksford. Ten years of the rational analysis of cognition. *Trends in Cognitive Sciences*, 3(2):57–65, Feb. 1999.

[4] G. Dreisbach and K. Fröber. On how to be flexible (or not): Modulation of the stability-flexibility balance. *Current Directions in Psychological Science*, 28(1):3–9, 2019.

[5] T. L. Griffiths, F. Lieder, and N. D. Goodman. Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic. *Topics in cognitive science*, 7(2):217–229, 2015.

[6] C. Hedge, G. Powell, and P. Sumner. The reliability paradox: Why robust cognitive tasks do not produce reliable individual differences. *Behavior research methods*, 50(3):1166–1186, 2018.

[7] S. Musslick, J. D. Cohen, and A. Shenhav. Estimating the costs of cognitive control from task performance: theoretical validation and potential pitfalls. In *Proceedings of the 40th Annual Conference of the Cognitive Science Society*, pages 800–805, Madison, WI, 2018.

[8] S. Musslick, J. D. Cohen, and A. Shenhav. Decomposing individual differences in cognitive control: a model-based approach. *In Proceedings of the 41st Annual Meeting of the Cognitive Science Society. Cognitive Science Society, Montreal, CA*, pages 2427–2433, 2019.

[9] S. Musslick, A. Shenhav, M. M. Botvinick, and J. D. Cohen. A computational model of control allocation based on the expected value of control. In *Reinforcement Learning and Decision Making Conference*. 2015.

[10] B. Saunders, M. Milyavskaya, A. Etz, D. Randles, M. Inzlicht, and S. Vazire. Reported self-control is not meaningfully associated with inhibition-related executive function: A bayesian analysis. *Collabra: Psychology*, 4(1), 2018.

[11] A. Shenhav, M. M. Botvinick, and J. D. Cohen. The expected value of control: an integrative theory of anterior cingulate cortex function. *Neuron*, 79(2):217–240, 2013.

[12] P. S. Whitehead, G. A. Brewer, and C. Blais. Are cognitive control processes reliable? *Journal of experimental psychology: learning, memory, and cognition*, 45(5):765, 2019.

Discovering Options that Minimize Average Planning Time

Alexander Ivanov
Department of Computer Science
Brown University
Providence, RI, USA
alexander_ivanov@brown.edu

Akhil Bagaria
Department of Computer Science
Brown University
Providence, RI, USA
akhil_bagaria@brown.edu

George Konidaris
Department of Computer Science
Brown University
Providence, RI, USA
gdk@cs.brown.edu

Abstract

Temporally extended actions, options, are often used to accelerate exploration and planning in sparse reward reinforcement learning tasks. We present an option discovery algorithm to accelerate planning by minimizing the average shortest distance in the task state-action graph. The proposed algorithm is proven to minimize planning time in a multitask setting and is shown to be a worst case $(4\alpha, 2)$-approximation of the optimal solution. The algorithm is generalized to stochastic communicating Markov decision processes and the bounds on optimality and runtime are shown to be preserved. Furthermore, we present a variant, Fast Average Options, with improved run-time and describe a general avenue for variants based on selection of a k-medians subroutine. We empirically evaluate our method and show that it outperforms comparable option discovery algorithms on a number of discrete and continuous domains.

Keywords: options, planning, graph, k-medians

1 Introduction

Most option discovery methods in the literature are based on heuristics [8]. Eigen options uses the eigen vectors of the graph Laplacian to create options that traverse the principle directions of the state space [6]. While this method is well defined and is shown to improve learning empirically, there is no formal guarantee or bound on improvement. Covering options is the first paper to establish a formal objective for option discovery and derive options that provably improve that objective [3]. Covering options minimizes the graph cover time and so bounds the expected number of steps needed to reach a rewarding state with a random walk. Assuming the agent acts randomly gives a bound on the worst case time to travel between two states. This assumption tends to be overly pessimistic in practice and minimizing cover time biases options for single task settings. We address both these points by consider the problem of selecting options that improve performance for planning in the multitask setting. We propose a new option discovery method, Average Options, which focuses on improving planning time in the multitask setting. We derive an option discovery method that provably minimizes planning time and present a $(2\alpha, 2)$-approximation algorithm. We evaluate our method empirically on several domains and compare against other approaches such as covering options and eigen-options.

2 Background

2.1 Value Iteration

In general we define value iteration for a general MDP (S, A, P, R, γ) with the following initialization and update of $V_t : S \to \mathbb{R}$

$$\begin{aligned} V_0(s) &= 0 \qquad \forall s \in S \\ V_{t+1}(s) &= \max_{a \in A} \left(\sum_{s' \in S} P(s, a, s')(R(s, a, s') + \gamma V_t(s')) \right) \end{aligned} \tag{1}$$

The update step is repeated until $V_{t+1}(s) - V_t(s) < \epsilon$ for all $s \in S$.

2.2 Point Options

For MDPs with many far apart states, pairs of states for which the shortest sequence of actions to reach one state from the other is relatively large compared to the number of states in the MDP, value iteration can be prohibitively slow. This is because it takes many update steps for the values of these states to propagate to the rest of the states in the MDP. We can reduce the needed number of iterations by reducing the number of actions needed to travel between such states. This can be done with options. Options are temporally extended actions that can facilitate exploration, learning, and long-horizon planning [9]. In this paper we will consider a special case of options called point options [3]. A point option is defined using an initiation set of a single state $I = \{s\}$, a termination set of a single state $B = \{s\}$, and a policy $\pi* : S \to A$ which induces the shortest sequence of actions $a_0, a_1, \dots$ to reach the termination state starting from the initiation state. We also say an option is "bidirectional" if that option o has a policy to go from the initiation state to termination state and a policy to go from termination state to initiation state. Let O be a set of such point options. For a graph G we denote adding options to the graph as $G + O = G'$. This new G' has the same vertices as G and for each $o \in O$ the edge going from the initiation state of o to the termination state of o is added. We take the added edges in $G + O$ to have length 1 and so the distance on $G + O$ is defined as normal. For brevity, we will refer to bidirectional point options as options in this paper [3].

2.3 k-Medians with Penalties Subroutine

The k-medians with penalties problem, also known as uncapacitated facility location with penalties, is used as a subroutine for the average options algorithm. Specifically, it is used to find which states in the state space should be connected with options. The k-medians with penalties problem aims to find the best set of k facilities F to open out of all facilities $\mathcal{F}$ such that the cost to serve, or deny service, to the cities C is minimized. The cost to serve a city $c \in C$ with facility f is $d(c, f)$ and the cost of denying service, a penalty, is p_c. The function $d : C \times F \to \mathbb{R}$ is an arbitrary distance metric that is symmetric and satisfies the triangle inequality. In this paper we will use k-medians with penalties as a subroutine and for a given k-medians with penalties algorithm we will say it is an α-approximation if it finds a solution F with cost no less than α times the cost of the optimal solution, F^*. There are many choices of such subroutines and they can be chosen based on their run-time and approximation guarantees [2][10][1]. For our bound on suboptimality we will consider an algorithm with $\alpha < 2$ from Jain et al. (2002) [2] and for our experiments we use an 5-approximation algorithm with improved run-time from Arya et al. (2001) [1].

3 Average Options

We propose a new method of selecting options that provably minimizes the average planning time for an MDP. Specifically, we consider the problem of finding a set of k bidirectional point options that minimize the number of iterations of VI necessary for convergence. We will first consider the case of deterministic MDPs with invertible actions and then generalize the results to stochastic communicating MDPs. We will show that our algorithm is able to achieve a 4α- approximation of the optimal set of options in polynomial time, $\mathcal{O}(n^2 \log(n) + n^3)$. Then we will show how the algorithm generalizes to stochastic MDP's with non-invertible actions while also preserving the approximation ration and run-time.

3.1 Deterministic MDPs with Invertible Actions

We first consider the case where the given MDP is deterministic, written as $M = (S, A, T, R, \gamma)$ where $T : S \times A \to S$ is the transition function. We can represent such an MDP as an undirected graph G where the vertices are states S and states s,s' are connected by an edge if there exists an action $a \in A$ with $T(s, a) = s'$. We also require that if $T(s, a) = s'$ there must exist an action a' such that $T(s', a') = s$.

We propose selecting options that minimize the average distance for the graph $G + O$. Specifically, we want to find the set of k point options O such that AvgDist$(G + O)$ is minimized.

$$d(G + O) = \sum_{s \in S} \sum_{s' \in S} d_{G+O}(s, s') \tag{2}$$

$$\text{AvgDist}(G + O) = d(G + O)/|S|^2 \tag{3}$$

Note that minimizing $d(G)$ is equivalent to minimizing AvgDist(G). To find such a set of k point options we propose the following algorithm:

1. Given a graph G and distance function d define a set the set of possible facilities $\mathcal{F} = S$, the vertices of G. Construct the set of cities C by duplicating each node u $2|S| - 2$ times to form u_{uv} and u_{vu} for each $v \in S$. Define the distance $d(i, u_{vu})$ to be $d(i, u)$ for all $i \in S$ and $u_{vu} \in C$ and assign each u_{vu} a penalty of $d(u, v)$.
2. Solve the $k + 1$-medians problem given the set of facilities $\mathcal{F}$, cities C, cost function d, and penalties as defined in the previous step. Let F be the chosen set of $k + 1$ facilities to open.
3. Select a state s in F and construct k options connecting the states in $F \setminus \{s\}$ to s.

Approximation Ratio and Run-Time Analysis

We will frame our problem of finding a set of options that minimizes the average graph distance in terms of adding shortcut edges to a graph to match the formulation of theorem 4 in section 6 of Meyerson et al. [7]. Because we consider the special case of point options which connect only pairs of states, we can equivalently think of them as shortcut edges that we add to the underlying graph. These edges will have length $\delta = 1$ as they still represent 1 step of planning. The weights w_{uv} for each pair of states $u, v \in S$ are also set to be 1 because we care about the average distance between all states equally. Adjusting the values of w_{uv} can let us emphasize the connections between specific regions of the MDP although we do not consider this case in this work. Theorem 4 of Meyerson et al. now gives us that there exists a $(4\alpha, 2)$-approximation algorithm that specifically follows the construction we give above.

Computing the distances between all pairs of states in the graph G with n vertices and m edges can be done in $\mathcal{O}(n^2 \log(n))$ time [5]. The k-median subroutine with $\alpha < 2$ can be done in $\mathcal{O}(n^3)$ time [2] although the α and run-time vary with choice of k-median algorithm. The total run-time of our algorithm is $\mathcal{O}(n^2 \log(n) + n^3)$

Bound on Planning Time

We now demonstrate that $d(G)$ bounds the number of iterations necessary for convergence of VI.

For a deterministic MDP we can rewrite the VI update as

$$V_{t+1}(s) = \max_{a \in A} \left(R(s') + \gamma V_t(s') \right) \qquad \text{with } s' = T(s, a) \tag{4}$$

In the context of planning we can assume that the reward function R is 1 for a single state $g \in S$ and 0 everywhere else and that the state g is absorbing. With these assumptions we can consider running VI until convergence in a finite number of iterations and take $V(s)$ for some $s \in S$. By expanding the VI update equation above we have that $V(s)$ obtains its value from the state g along the shortest path from g to s. The value of $V(s)$ is consequently $\gamma^{d(g,s)}$. This also

tells us that the $V(s)$ obtains this value after $d(g, s)$ iterations and by assumption never changes again. This means that the number of iterations of VI necessary to plan to reach a certain goal $g \in S$ can be written as $\max_{s \in S} d(s, g)$ and the average over all goal states is $\sum_{g \in S} \max_{s \in S} d(s, g)$. Notice that the average steps to convergence of VI over goals is upper bounded by the average graph distance.

$$\sum_{g \in S} \max_{s \in S} d(s, g) < d(G) = \sum_{s \in S} \sum_{s' \in S} d(s, s') \tag{5}$$

Therefore, by minimizing the average distance for a given graph we minimize the upper bound for number iterations of VI before convergence.

4 Empirical Results

We compare average options with two other notable methods, eigen options and covering options. Eigen options selects options that traverse the principle directions of the state space based on the eigen vectors of the graph Laplacian. Covering options selects options which bound the expected cover time of the state space.

We first qualitatively compare the options generated by our method to those obtained with covering options and eigen-options. For the 9x9 grid, covering options connect the furthest apart corners; average options connect states that are more central. In Four Rooms, average options connects the centers of the rooms whereas covering options and eigen-options again join the corner states.

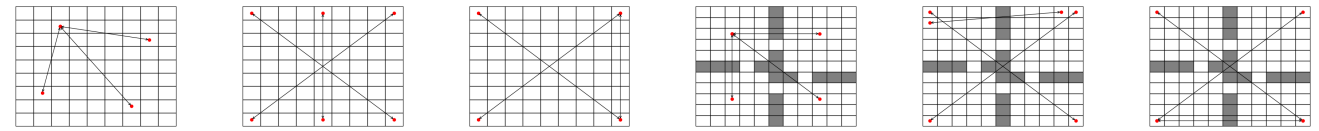
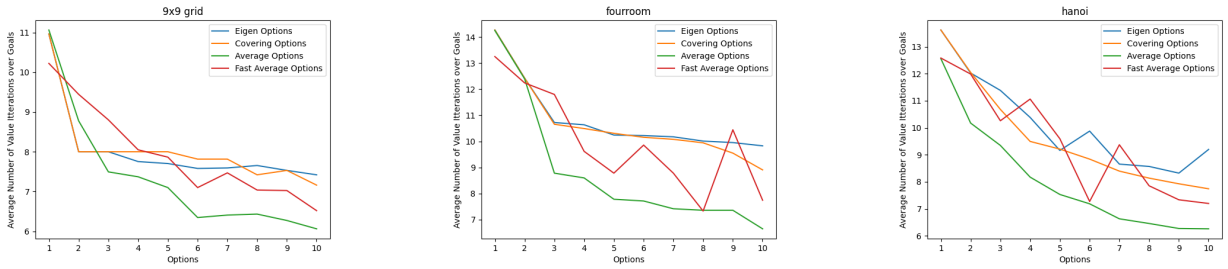


Figure 1: Visualization of options generated with average options, eigen options and covering options for small discrete domains.

We also quantitatively evaluate the reduction in planning time obtained by using average options as compared to eigen options and covering options. For average options, we use a k-median with penalties subroutine with approximation ratio of $\alpha = 5$ for improved run-time [1]. We additionally consider a variant of average options, fast average options, where the k-medians with penalties subroutine is replaced by just k-medians, greatly improving the run time but without a guarantee on the approximation ratio outside of empirical results [4].



(a) 9x9 grid (b) four-room (c) Towers of Hanoi

Figure 2: Comparison of the average over goals of time to convergence of VI for average options, eigen options and covering options.

We evaluate the option generation methods on three discrete domains: 9x9 grid, Four Rooms, and Towers of Hanoi. For each domain we generate a set of n options O using each method and then run value iteration for every state $g \in S$ in the graph $G + O$. For each run of VI we make g an absorbing state with reward 1 and give all other states a reward of

0. Because these are small discrete domains we can run VI until exact convergence and record the number of iterations taken. We then average the number of iterations over all the goal states.

Average options consistently performs better across the tested domains. Fast average options has variable performance but tends to outperform covering options and eigen-options as domains get larger and more options are selected.

5 Discussion

We note that in the qualitative evaluation that covering options and eigen-options join similar pairs of states with options for the 9x9 grid and Four Rooms domains. This can be explained by both methods having an objective function based on the eigen-vectors of the graph Laplacian. This similarity in objective function leads both methods to produce qualitatively similar options and so have similar effect on planning time.

We seek to provide some intuition about why average options outperforms covering options and eigen-ptions. Covering options tends to connect far away states, potentially aiding exploration. However, the majority of states are not at the fringes of the domain, so when planning to reach an arbitrary state, covering options are often unused. Similarly with eigen-options, when states at opposite ends of a principle direction are linked with an option, planning between the more common 'average' states is unaffected.

6 Conclusion

In this paper we presented a formal objective for selecting options to minimize planning time in a multitask setting. We derived an algorithm that provably minimizes the objective and bounded the algorithms suboptimality. We evaluated our method to empirically demonstrate the reduction in planning time and compared our method against covering options, another principled option discovery approach, and eignen options, a similar heuristic based approach. We also present a variant of average options, fast average options, with improved run-time and comparable empirical results to average options.

7 References

[1] Vijay Arya et al. "Local search heuristic for k-median and facility location problems". In: Jan. 2001, pp. 21–29. DOI: 10.1145/380752.380755.

[2] Kamal Jain et al. *Greedy Facility Location Algorithms Analyzed using Dual Fitting with Factor-Revealing LP*. 2002. arXiv: cs/0207028 [cs.DS].

[3] Yuu Jinnai et al. "Discovering Options for Exploration by Minimizing Cover Time". In: *Proceedings of the 36th International Conference on Machine learning*. 2019.

[4] Atsuyoshi Nakamura Keisuke Todo and Mineichi Kudo. "A Fast Approximate Algorithm for k-Median Problem on a Graph". In: *Proceedings of the 15th International Workshop on Mining and Learning with Graphs (MLG)*. 2019.

[5] Udaya Kumar, Reddy R, and K. Iyer. "All-pairs shortest-paths problem for unweighted graphs in O(n2 log n) time". In: *World Academy of Science, Engineering and Technology* 38 (Feb. 2009).

[6] Marlos C. Machado et al. *Eigenoption Discovery through the Deep Successor Representation*. 2018. arXiv: 1710.11089 [cs.LG].

[7] Adam Meyerson and Brian Tagiku. "Minimizing average shortest path distances via shortcut edge addition". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 272–285.

[8] Özgür Şimşek and Andrew Barto. "Skill Characterization Based on Betweenness". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller et al. Vol. 21. Curran Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper/2008/file/934815ad542a4a7c5e8a2dfa04fea9f5-Paper.pdf.

[9] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112.1 (1999), pp. 181–211. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(99)00052-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370299000521.

[10] Guang Xu and Jinhui Xu. "An LP rounding algorithm for approximating uncapacitated facility location problem with penalties". In: *Inf. Process. Lett.* 94 (May 2005), pp. 119–123. DOI: 10.1016/j.ipl.2005.01.005.

Probing Compositional Inference in Natural and Artificial Agents

Akshay K. Jagadish
Max Planck Institute for Biological Cybernetics
Tübingen, Germany
akshay.jagadish@tue.mpg.de

Tankred Saanum
Max Planck Institute for Biological Cybernetics
Tübingen, Germany

Jane X. Wang
DeepMind
London, United Kingdom

Marcel Binz
Max Planck Institute for Biological Cybernetics
Tübingen, Germany

Eric Schulz
Max Planck Institute for Biological Cybernetics
Tübingen, Germany

Abstract

People can easily evoke previously encountered concepts, compose them, and apply the result to novel contexts in a zero-shot manner. What computational mechanisms underpin this ability? To study this question, we propose an extension to the structured multi-armed bandit paradigm, which has been used to probe human function learning in previous works. This new paradigm involves a learning curriculum where agents first perform two sub-tasks in which rewards were sampled from differently structured reward functions, followed by a third sub-task in which rewards were set to a composition of the previously encountered reward functions. This setup allows us to investigate how people reason compositionally over learned functions, while still being simple enough to be tractable. Human behavior in such tasks has been predominantly modeled by computational models with hard-coded structures such as Bayesian grammars. We indeed find that such a model performs well on our task. However, they do not explain how people learn to compose reward functions via trial and error but have, instead, been hand-designed to generalize compositionally by expert researchers. How could the ability to compose ever emerge through trial and error? We propose a model based on the principle of meta-learning to tackle this challenge and find that – upon training on the previously described curriculum – meta-learned agents exhibit characteristics comparable to those of a Bayesian agent with compositional priors. Model simulations suggest that both models can compose earlier learned functions to generalize in a zero-shot manner. We complemented these model simulations results with a behavioral study, in which we investigated how human participants approach our task. We find that they are indeed able to perform zero-shot compositional reasoning as predicted by our models. Taken together, our study paves a way for studying compositional reinforcement learning in humans, symbolic, and sub-symbolic agents.

Keywords: Compositional Inference; Meta Reinforcement Learning; Zero-shot learning; Meta-learning; Reinforcement Learning; Human Decision Making. Bayesian Grammar.

Acknowledgements

This work was supported by the Max Planck Society and the Volkswagen Foundation.

1 Introduction

Every day we bring together concepts from seemingly different settings to bear on problems where they have not been used before. To illustrate, let us take the example of reading trends in COVID vaccination numbers. Figure 1 shows the vaccination numbers in Germany from December 2020 to March 2021. You can clearly identify a pattern in the trajectory of these numbers. They increase roughly exponentially (indicated by the black dots) but with a periodic rise and fall every week. If you were shown the trend until the first week of March and asked to extrapolate to the second week (in the red box), you would likely predict something close to the true trend. How do we accomplish this?

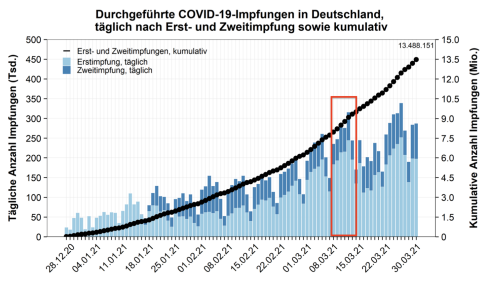


Figure 1: COVID vaccination numbers

The aforementioned example involves two aspects of learning that have been studied extensively in cognitive science but mostly in isolation [1, 2, 3]. First, function learning (i.e., learning the exponential and periodic function), and second, compositional reasoning over learned functions (i.e., combining the exponential and periodic function to extrapolate the trend for next week). Recent work of [2], for instance, has investigated the first aspect. They introduced a structured multi-armed bandit (sMAB) task, where rewards for different options followed a latent correlation structure, to study how people explore in structured environments. The latent reward functions were either linear or periodic (structured condition) or set to random uncorrelated values (unstructured condition). Participants were told to earn as many reward points as possible but were not provided any information regarding the underlying reward functions. They found that participants could pick up on the latent functions in the structured condition after a few rounds and use it to guide their exploration in later rounds.

While the study of [2] provides insights into how people learn latent reward functions, it does not tell us how they re-use earlier learned functions to compose new functions that were never encountered before. We attempt to close this gap in the present article. To this end, we extended the paradigm of [2] by introducing a learning curriculum. Participants in our task first performed two sub-tasks in which rewards were sampled from differently structured reward functions, followed by a third sub-task in which the rewards were set to a composition of the previously sampled functions. We were primarily interested in the following questions:

1. Do people compose previously learned functions in the final sub-task in a zero-shot manner?
2. Which computational models capture people's compositional reasoning abilities?

We demonstrate that people can indeed learn to compose new functions in a zero-shot manner. Historically, human behavior in such tasks has been modeled using algorithms with hard-coded structures, such as Bayesian grammars [4]. We find that these models also perform well in our task. However, they cannot answer the questions of where these structures come from and how they are acquired in the first place. To address these questions, we propose an alternative model based on the principle of meta-learning [5]. When trained on an appropriate curriculum, this model exhibits characteristics comparable to those of a Bayesian agent with compositional priors. We briefly discuss the advantages and disadvantages of the two modeling approaches and suggest that they can be used to understand distinct aspects of human cognition.

2 Methods & Results

2.1 Compositional Multi-Armed Bandit Task

To probe compositional reasoning in humans and artificial agents, we developed a task based on earlier work of [2]. Each task consists of three sMAB sub-tasks, where rewards follow a function that is dependent on the spatial position of arms. In each sMAB, rewards were determined as follows:

$$r_t = f(a_t) + \epsilon_t \tag{1}$$

where t denotes the time-step, $a_t \in \{0, \ldots, 5\}$ the action taken in time-step t, and $\epsilon_t \sim \mathcal{N}(0, 0.3)$ is an additive noise term.

The first two sub-tasks involved latent functions from either the linear or the periodic family:

$$f_{\text{linear}}(a_t) = \left(\frac{2a_t}{5} - 1\right) w + b \qquad w \sim \mathbf{U}(-2.5, 2.5), b \sim \mathbf{U}(2.5, 7.5) \tag{2}$$

$$f_{\text{periodic}}(a_t) = A \left|\sin\left(0.5\pi\left(a_t - \phi\right)\right)\right| + b \qquad A \sim \mathbf{U}(0, 7.5), \phi \in \{0, 1\}, b \sim \mathbf{U}\left(0, \frac{A}{1.4}\right) \tag{3}$$

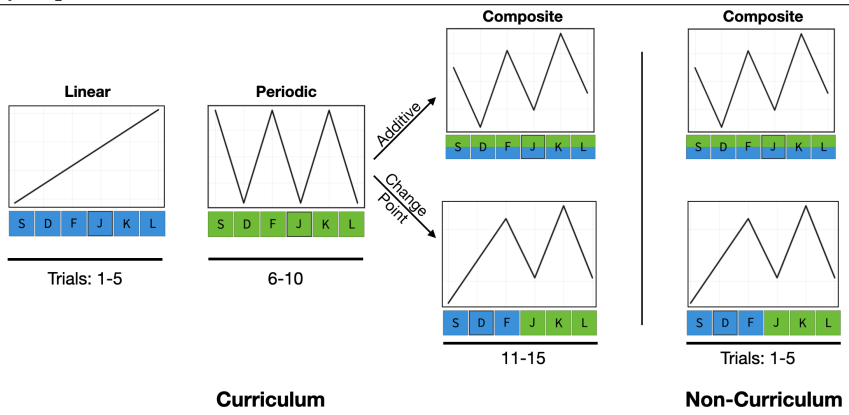


Figure 2: **Task Overview**. Example tasks for both compositional rules and conditions.

Latent functions in the final sub-task were composed from the functions encountered in the two earlier sub-tasks. We considered both additive and change-point compositions:

$$f_{\text{additive}}(a_t) = f_{\text{linear}}(a_t) + f_{\text{periodic}}(a_t) \tag{4}$$

$$f_{\text{change-point}}(a_t) = \begin{cases} f_{\text{linear}}(a_t) & \text{if } a_t \in \{0, 1, 2\} \\ f_{\text{periodic}}(a_t) & \text{otherwise} \end{cases} \tag{5}$$

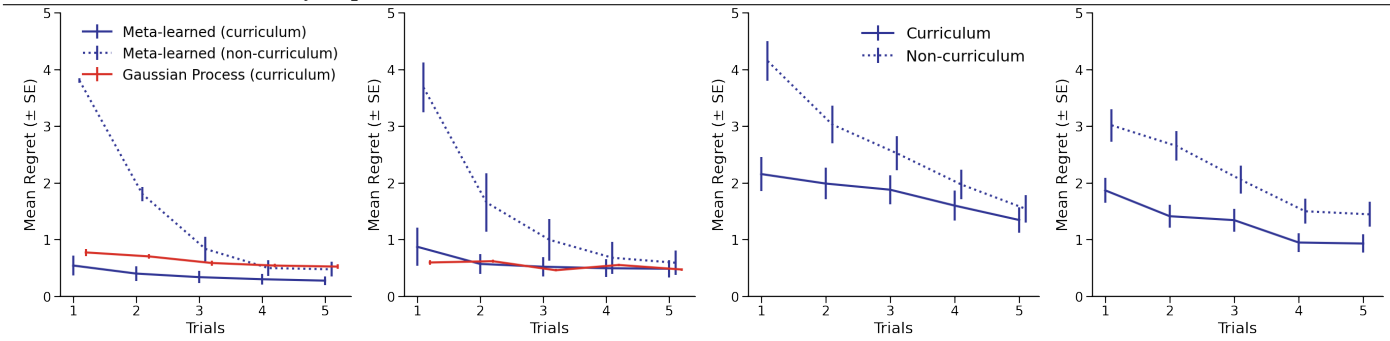
The order of composition was randomized in the change-point function. The length of each sub-task was set to 5 trials, leading to 15 overall trials per task. Note that the number of trials per sub-task is less than the number of available options. This prevents an agent from exhaustively trying out all options and forces it to generalize based on the underlying function. Figure 2 shows an example for both the additive and change-point condition. To have a comparison, we also consider a condition without a curriculum. In this non-curriculum condition, the agent did not interact with the first two sub-tasks and instead directly observed the composite function from the final sub-task (as illustrated on the right in Figure 2).

2.2 Computational Models

The goal of an agent used to model our task is to maximize the total amount of obtained rewards. We outline two models for achieving this goal. The first is a Bayesian model that incorporates knowledge of priors via the Gaussian Process (GP) regression model; the second is a recurrent neural network model that is meta-learned based on repeated interactions with randomly sampled tasks.

Gaussian Process Model: We implemented a GP regression model as the Bayesian model for our task as they have been successful at capturing human function learning in previous works [4, 2]. The latent reward functions were learned using a linear kernel for the first sub-task, a periodic kernel for the second sub-task, and a kernel using a composition of the linear and periodic kernel for the last sub-task. For additive compositions, the compositional kernel was the sum of a linear and periodic kernel, and the prior mean was set to the mean of the previously learned functions from the linear and periodic sub-tasks. For change-point compositions, the kernel entries were set to that of the linear kernel if both arms belonged to the linear function, the periodic kernel if both belonged to the periodic function, and zero otherwise. The prior mean in the change-point composition was set to the means learned in linear and periodic sub-tasks respectively. The model acted using an ϵ-greedy policy with an ϵ-value of 0.9 in the first two sub-tasks and acted greedily on the last sub-task. These ϵ-values were chosen such that the model explored and learned the latent functions in the first two sub-tasks and exploited the learned latent functions in the last sub-task.

Meta-Reinforcement Learning: The prior over possible reward functions and how these are composed are hard-coded in the Gaussian Process model. To test whether such priors and compositions can also be learned from experience, we implemented a meta-reinforcement learning agent similar to that of [5]. The agent architecture consisted of a long short-term memory (LSTM) network followed by an actor-critic module. The LSTM network had 48 hidden units and takes the trial index, the action from the previous step, the reward from the previous step, and – depending on the condition – the sub-task index as well as a one-hot encoding of the compositional rule as inputs. The actor-critic module comprised of a two-layer neural network with 48 hidden units in each layer with the first layer shared between actor and critic. It outputted an estimate of the value function and the policy. We trained the model using the standard A2C loss at the end of each episode with an additional entropy regularization term to prevent premature convergence [6]. The parameter that controlled the strength of the entropy regularization term was annealed to zero over the course of training. We used the ADAM optimizer with a learning rate of 0.001 and trained all agents for a total of $2 \cdot 10^5$ episodes.



(a) Models: Additive (b) Models: Change-Point (c) Human: Additive (d) Human: Change-Point

Figure 3: **Model and Human Performance**: (a-b) Mean regrets for both models averaged over 100 runs in the final sub-task for the additive and change-point rule. (c-d) Mean regrets for participants in the final sub-task for the additive and change-point rule. Error-bars for models show standard errors over five random seeds; for human data, standard errors were computed over participants.

2.3 Model Simulations

We simulated both of these models on the task described in Section 2.1. To investigate whether the models can reason compositionally, we measured their mean regrets in the final sub-task. Regret is defined as the difference between the optimal reward for a given latent function and the reward of the action selected by the agent. The result of this analysis is shown in Figure 3 (a) and (b). For both models, we find that the initial mean regret in the curriculum condition is close to zero. This indicates that the agents re-used earlier learned functions to compose new functions in a zero-shot manner. In contrast to this, the mean regret in the non-curriculum condition starts at chance-level and takes three to four trials to reach similar performance.

2.4 Human Experiment

How does human behavior compare to these two algorithms? To answer this question, we conducted a behavioral study in which the experimental design followed the previously outlined task structure. Participants were told that they are gamblers visiting the fictional town of Bandit City. This entailed visiting multiple casinos in order to play different sets of slot machines. Each casino had two slot machines by different companies, with their color indicating the manufacturer. Coins were earned after choosing an option on the slot machine. Participants were told that all slot machines from a company behaved similarly. That is, the expected payoffs across options for machines from the same company followed a similar function (either linear or periodic) but were provided no information about which function belonged to which company.

In each casino, participants had five trials per slot machine, and their goal was to win as many coins as possible. Afterward, they were then tested on a new slot machine, which was a composition of the two previously played machines. The rewards of this new slot machine were given by either an additive or a change-point composition. Machines with an additive composition had options that were half-green and half-blue in color, while options on machines with a change-point composition were either green or blue depending on the machine from which its reward was drawn (as shown in Figure 2).

Participants: We recruited 200 participants (103 female, M_{age} = 28.90) for the additive and 211 participants (96 female, M_{age} = 27.58) for the change-point composition through the Prolific platform. Participants were randomly assigned to the curriculum or non-curriculum condition. Each condition involved 20 tasks in total. All participants had an approval rate of 95% or more, were fluent English speakers from the United States and were 18 years of age or older. Participants were rewarded a base payment of $\pounds 2$ and a performance-dependent bonus payment up to $\pounds 2.5$. The study was approved by the local ethics committee. We removed 25 participants from our analysis as their performance on the last sub-task was not above chance level.

Results: If people can learn to compose in our task, we would expect them to pick the most rewarding option in the curriculum condition on the first trial of the final sub-task with a higher probability than people in the non-curriculum condition. We consider the last task as an evaluation task as we are interested in the converged behavior. The main quantity of interest is again the regret in the compositional sub-task. Figure 3 (c) and (d) shows the mean regret (averaged over participants) for both compositional rules. 278

A mixed linear regression analysis revealed that participants in the curriculum condition had a significantly lower regret on the first trial of the final sub-task compared to participants in the non-curriculum condition ($\hat{\beta} = -1.49 \pm 0.06, p < .001$). This result indicates that people were able to perform some form of zero-shot compositional reasoning in our task, similar to the investigated models.

However, there were also significant differences to the behavior of both models. The regret in the compositional sub-task for the participants was larger than that of the Bayesian and the meta-learned model. This is to be expected, as both models provide us with idealized predictions. Participants furthermore kept on learning during the final sub-task in the curriculum condition, while both computational models did not. To quantify this effect we fitted a mixed-effects linear regression using per-trial regret as the dependent variable, and the corresponding trial number as both fixed effects and random effects over participants. The results of this model showed a significant fixed effect of trial number ($\hat{\beta} = -0.31 \pm 0.02, p < .001$) onto regret. This confirms that the performance of participants improved with additional interactions. The improvement in the curriculum condition ($\hat{\beta} = -0.21 \pm 0.02$) however was weaker than that in the non-curriculum condition ($\hat{\beta} = -0.42 \pm 0.02$), indicating that participants could exploit the given task structure.

3 Discussion

People effortlessly compose previously learned functions into new ones when navigating their everyday lives. Previous research has examined this ability in both humans and artificial agents [7, 8] but in a different setting. In the present article, we proposed a novel experimental task to probe how different agents generalize compositionally. We found that two very different computational models – a Gaussian Process model and a meta-learned recurrent neural network – can compose previously learned functions into a new one in a zero-shot manner. We furthermore compared the behavior of these models to that of human subjects. While we demonstrated that people can learn to compose reward functions, our analysis also revealed a gap between the behavior of both models and people. We hope to address this gap in future work by building agents that take the complexity of their solutions into account [9].

It is generally thought that Bayesian models and neural networks offer different theories for understanding human cognition. In neural network models, cognition emerges from the interaction between a large set of simple processing units. Bayesian models, on the other hand, explain cognition by appealing to the idea of ideal statistical inference. The concept of meta-learning offers a bridge between the two frameworks. In the current study, we have seen that it is possible to meta-learn a recurrent neural network that can reason compositionally on a challenging task – an ability that has been historically investigated using Bayesian models. In general, we believe that future work should view both frameworks not as competing hypotheses but rather as symbiotic tools. Each of them comes with its own advantages and disadvantages: Bayesian models are often interpretable but hard to scale, while meta-learning offers an answer to the question of how to learn useful priors from experience but comes at the cost of losing theoretical guarantees.

References

[1] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, 2013.

[2] Eric Schulz, Nicholas T Franklin, and Samuel J Gershman. Finding structure in multi-armed bandits. *Cogn. Psychol.*, 119:101261, 2020.

[3] Daniel E Acuña and Paul Schrater. Structure learning in human sequential decision-making. *PLoS Comput. Biol.*, 6(12):e1001003, 2010.

[4] Tankred Saanum, Eric Schulz, and Maarten Speekenbrink. Compositional generalization in multi-armed bandits. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 43, 2021.

[5] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. In *CogSci*, 2017.

[6] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[7] Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. *Advances in neural information processing systems*, 32, 2019.

[8] Sreejan Kumar, Ishita Dasgupta, Jonathan Cohen, Nathaniel Daw, and Thomas Griffiths. Meta-learning of structured task distributions in humans and machines. In *International Conference on Learning Representations*, 2020.

[9] Marcel Binz, Samuel J Gershman, Eric Schulz, and Dominik Endres. Heuristics from bounded meta-learned inference. *Psychological review*, 2022.

Towards Painless Policy Optimization for Constrained MDPs

Arushi Jain[*]
McGill University & Mila
arushi.jain@mail.mcgill.ca

Sharan Vaswani[*]
Simon Fraser University

Reza Babanezhad
SAIT AI lab, Montreal

Doina Precup
McGill University & Mila

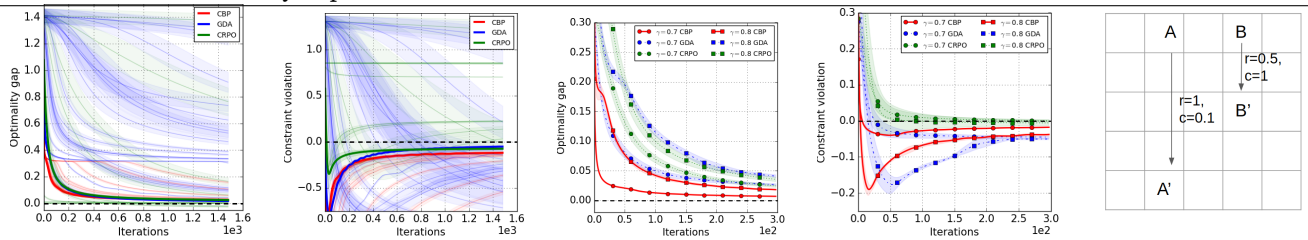
Csaba Szepesvari
Amii, University of Alberta

Abstract

In many safety-critical applications, e.g., robotics, finance, autonomous driving, agents must be subjected to satisfy certain constraints on a cost function. The Constrained Markov decision process (CMDPs) (Altman, 1999) is a natural framework for modelling such constraints. The typical objective for CMDP is to maximize a cumulative function of the reward (like in unconstrained MDPs), while (approximately) satisfying the constraints. In this paper, we study policy optimization with both tabular and linear function approximation in infinite-horizon, discounted constrained Markov decision process (CMDP). We propose a generic primal-dual optimization framework, which allows us to bound the sub-optimality gap and constraint violation for arbitrary algorithms in terms of the primal and dual regret on online linear optimization problems. We instantiate this framework in a way which allows us to use coin-betting algorithms from online linear optimization to control both the primal and dual regret. We call the resulting algorithm **Coin Betting Politex (CBP)**. Assuming that the action-value functions are ε-close to the span of d-dimensional state-action features and γ is the discount factor, T iterations of CBP result in an $O\left(\frac{1}{(1-\gamma)^3\sqrt{T}} + \frac{\varepsilon\sqrt{d}}{(1-\gamma)^2}\right)$ bound on the reward sub-optimality and a constraint violation of $O\left(\frac{1}{(1-\gamma)^2\sqrt{T}} + \frac{\varepsilon\sqrt{d}}{1-\gamma}\right)$. Unlike gradient descent-ascent and primal-only methods, our proposed *CBP is robust to the choice of hyper-parameters*. We empirically demonstrate the superior performance and robustness of CBP in both tabular and linear function approximation setting, on both gridworld environments and OpenAI gym tasks.

Keywords: reinforcement learning, constrained MDPs, primal-dual framework, coin-betting algorithm

[*]Equal authorship



(a) Hyperparameter sensitivity OG (b) Hyperparameter sensitivity CV (c) Environment misspecification OG (d) Environment misspecification CV (e) Gridworld environment

Figure 1: **Performance in model-based setting:** Performance metric – *Optimality gap (OG)* and *constraint violation (CV)* – of our proposed algorithm, CBP, and baselines GDA, CRPO in a gridworld environment (Fig. 1e) with access to the true CMDP model. Ideally OG and CV should converge to 0. Figs. 1a and 1b show the performance *sensitivity to hyperparameters*. The dark lines corresponds to the performance with the best hyperparameters. The lighter color lines show performance with other hyperparameter values. Figs. 1c and 1d shows the effect of *environment mis-specification* by varying the discount factor $\gamma = \{0.7, 0.8\}$ and keeping the best hyperparameters obtained from the left two figures (obtained in the original CMDP with $\gamma = 0.9$). GDA and CRPO, exhibit huge variance in performance with different hyperparameters, while CBP is quite *robust to variations in hyperparameters and environment mis-specification*.

1 Introduction

In many safety-critical applications, e.g., robotics, finance, autonomous driving, agents must satisfy certain constraints in addition to optimizing long-term returns. This objective can be formalized in a natural way through the framework of constrained Markov decision process (CMDP) (Altman, 1999).

In this paper, we focus on the problem of finding an approximately feasible policy (i.e. a policy which is allowed to violate constraints by a small amount), while (approximately) maximizing the cumulative reward in CMDPs. The literature on this topic can be divided into two types of methods. Methods of the first type, referred to as *primal-only* methods, only update the policy parameters while enforcing the constraints (Achiam et al., 2017). The recent work of Xu et al. (2021) guarantees global convergence for such methods to the optimal feasible policy in both the tabular and function approximation settings . The second approach for planning in CMDPs is to form the Lagrangian, and solve the resulting saddle-point problem using *primal-dual algorithms* (Altman, 1999). Such approaches update both the policy parameters (primal variables), and the Lagrange multipliers (dual variables). The recent work of Ding et al. (2020) proves that this approach converges to the optimal policy in both tabular and function approximation settings as well.

Although there has been substantial progress in designing planning algorithms for CMDPs, *all of the proposed algorithms are highly sensitive to hyperparameter tuning*. For example, Figs. 1a and 1b illustrate the effect of varying the hyperparameters of two provably efficient algorithms – the primal-dual Gradient Descent Ascent (GDA) (Ding et al., 2020) and the primal-only CRPO (Xu et al., 2021)– on a gridworld task. We can see that the magnitude of both optimality gap and constraint violations vary greatly for different hyperparameters. Hence, in order to obtain reasonably good performance on a new task, the hyperparameters of these algorithms need to be tuned from scratch, incurring significant computational overhead. Designing robust planning algorithms that require minimal hyperparameter tuning is the main motivation for this work. We propose Coin-Betting Politex (CBP) as an algorithm that can control both the primal and the dual regret, ensuring better robustness to hyperparameters (as evidenced by the red lines in the figures).

2 Problem Formulation

An infinite-horizon discounted CMDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, c, b, \rho, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition probability function, $\Delta_{\mathcal{S}}$ is the $\mathcal{S}$-dimensional probability simplex, $\rho \in \Delta_{\mathcal{S}}$ is the initial distribution of states and $\gamma \in [0, 1)$ is the discount factor. The primary reward to be maximized is denoted by $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The expected discounted return or *reward value* of a policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ is defined as $V_r^\pi(\rho) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$, where $s_0 \sim \rho, a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$. Similarly, the constrained reward is denoted by $c : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and the *constrained reward* of policy π is denoted by $V_c^\pi(\rho)$. For each state-action pair (s, a) and policy π, the reward action-value function is defined as $Q_r^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and satisfies the relation: $V_r^\pi(\rho)(s) = \langle \pi(\cdot|s), Q_r^\pi(s, \cdot) \rangle$. We define Q_c^π analogously. Given a class of policies Π, the agent's objective is to return a policy $\pi \in \Pi$ that maximizes $V_r^\pi(\rho)$, while ensuring that $V_c^\pi(\rho) \geq b$. Formally,

$$\max_{\pi \in \Pi} V_r^\pi(\rho) \quad \text{s.t.} \quad V_c^\pi(\rho) \geq b. \tag{1}$$

281

Throughout, we will assume the existence of a feasible policy, and denote the optimal feasible policy by π^*. In this work, we will consider an easier problem with a relaxed feasibility requirement. In particular, given a target error ε, our aim is to return a policy $\hat{\pi}$ such that, $V_r^{\hat{\pi}}(\rho) \geq V_r^{\pi^*}(\rho) - \varepsilon$, s.t. $V_c^{\hat{\pi}}(\rho) \geq b - \varepsilon$. In the next section, we specify a generic primal-dual framework to solve the problem in Eq. (1) and introduce our algorithm **CBP**.

3 Methodology

3.1 Primal-Dual framework

In order to specify the primal-dual framework, we use the Lagrangian formulation, and express the constrained optimization problem in Eq. (1) as the following saddle-point problem:

$$\max_{\pi \in \Pi} \min_{\lambda \geq 0} V_r^{\pi}(\rho) + \lambda [V_c^{\pi}(\rho) - b]. \tag{2}$$

Here, $\lambda \in \mathbb{R}$ is the Lagrange multiplier for the constraint and λ^* refers to its optimal value, meaning that (π^*, λ^*) is the solution to the above saddle-point problem.

We will solve the above primal-dual saddle-point problem iteratively, by alternatively updating the policy (primal variable) and the Lagrange multiplier (dual variable). If T is the total number of iterations of such an algorithm, we define π_t and λ_t to be the primal and dual iterates for $t \in [T]$. We define $\hat{Q}_r^t := \hat{Q}_r^{\pi_t}$ and $\hat{Q}_c^t := \hat{Q}_c^{\pi_t}$ as the *estimated* action-value functions corresponding to the policy π_t. In this section, we assume that we have uniform control over the approximation errors in the action-value functions for every state-action pair implying that $\|Q_r^t - \hat{Q}_r^t\|_\infty \leq \tilde{\varepsilon}$ and $\|Q_c^t - \hat{Q}_c^t\|_\infty \leq \tilde{\varepsilon}$.

Given a generic primal-dual algorithm, our task is to characterize its performance in terms of its cumulative reward and constraint violation. For a sequence of policies π_t and Lagrange multipliers λ_t generated by an algorithm, we define the **average optimality gap (OG)** and **average constraint violation (CV)** as follows:

$$\mathit{OG} := \frac{1}{T} \sum_{t=0}^{T-1} [V_r^{\pi^*}(\rho) - V_r^t(\rho)] \quad ; \quad \mathit{CV} := \frac{1}{T} \sum_{t=0}^{T-1} [b - V_c^t(\rho)]_+. \tag{3}$$

Here, $[x]_+ = \max\{x, 0\}$. We define the *primal regret* and *dual regret* with respect to the optimal policy π^* as follows. If $\hat{V}_c^t(\rho)(s) = \langle \pi_t(\cdot|s), \hat{Q}_c^t(s, \cdot) \rangle$ is the cost value function at state s, then:

$$\mathcal{R}^p(\pi^*, T) := \sum_{t=0}^{T-1} \sum_{s=0}^{\mathcal{S}-1} \langle \pi^*(\cdot|s) - \pi_t(\cdot|s), \hat{Q}_r^t + \lambda_t \hat{Q}_c^t] \rangle \quad ; \quad \mathcal{R}^d(\lambda, T) := \sum_{t=0}^{T-1} \langle \lambda_t - \lambda, \hat{V}_c^t(\rho) - b \rangle. \tag{4}$$

The above quantities correspond to the regret for online linear optimization algorithms that can independently update the primal and dual variables. In the unconstrained setting, reducing the policy optimization problem to that of online linear optimization has been previously explored through the *Politex* algorithm (Abbasi-Yadkori et al., 2019). We build upon Politex to reduce the problem from Eq. (2) to that of online linear optimization in Eq. (4). Theorem 3.1 characterizes the performance of a generic algorithm in terms of its primal and dual regret (proof omitted due to lack of space).

Theorem 3.1. *Assume that $\|Q_r^t - \hat{Q}_r^t\|_\infty \leq \tilde{\varepsilon}$ and $\|Q_c^t - \hat{Q}_c^t\|_\infty \leq \tilde{\varepsilon}$, for a generic algorithm producing a sequence of polices $\{\pi_0, \pi_1, \ldots, \pi_{T-1}\}$ and dual variables $\{\lambda_0, \lambda_1, \ldots, \lambda_{T-1}\}$ such that for all t. λ_t is constrained to lie in the $[0, U]$, where $U > \lambda^*$. Here $g(U) = \frac{\tilde{\varepsilon}}{1-\gamma}(1 + U) + U\tilde{\varepsilon}$. Then, the average optimality gap (OG) and constraint violation (CV) are bounded by:*

$$\mathit{OG} \leq \frac{\mathcal{R}^p(\pi^*, T) + (1-\gamma)\mathcal{R}^d(0, T)}{(1-\gamma)T} + g(U) \quad ; \quad \mathit{CV} \leq \frac{\mathcal{R}^p(\pi^*, T) + (1-\gamma)\mathcal{R}^d(U, T)}{(U - \lambda^*)(1-\gamma)T} + g(U).$$

Importantly, the above result does not depend on the class of policies Π, nor does it require any assumption about the underlying CMDP. In order to bound the average reward optimality gap and the average constraint violation, we need to (i) project the dual variables onto the $[0, U]$ interval and ensure that $U > \lambda^*$, (ii) update the primal and dual variables to control the respective regret in Eq. (4), and (iii) control the approximation error $\tilde{\varepsilon}$. In the next section, we use this recipe to design algorithms with provable guarantees.

3.2 Coin-Betting Politex Algorithm

Orabona and Pal (2016) and Orabona and Tommasi (2017) propose *coin-betting* algorithms that reduce the online linear optimization problems in Eq. (4) to an online betting, and leverage this idea to design parameter-free algorithms. First, we instantiate the Algorithm 2 in Orabona and Pal (2016) for updating the policy, which is completely parameter-free, to

our problem. The policy update requires the computation of additional variables w_t for each (s, a) pair and iteration t, as follows:

$$w_{t+1}(s,a) = \frac{\sum_{i=0}^{t} \tilde{A}_l^i(s,a)}{(t+1)+T/2}\left(1+\sum_{i=0}^{t}\tilde{A}_l^i(s,a)\, w_i(s,a)\right)$$
$$\pi_{t+1}(a|s) = \begin{cases} \pi_0(a|s), & \text{if } \sum_a \pi_0(a|s)\,[w_{t+1}(s,a)]_+ = 0 \\ \frac{\pi_0(a|s)\,[w_{t+1}(s,a)]_+}{\sum_a \pi_0(a|s)\,[w_{t+1}(s,a)]_+}, & \text{otherwise,} \end{cases} \tag{5}$$

where, $\tilde{A}_l^t(s,a) = \hat{A}_l^t(s,a)\,\mathcal{I}\{w_t(s,a) > 0\} + [\hat{A}_l^t(s,a)]_+\,\mathcal{I}\{w_t(s,a) \le 0\}$. Here, $\hat{A}_l^t(s,a)$ can be interpreted as the normalized advantage function, $\hat{A}_l^t(s,a) = \frac{1-\gamma}{1+U}\left[\hat{Q}_l^t(s,a) - \left\langle \hat{Q}_l^t(s,\cdot), \pi_t(\cdot|s)\right\rangle\right]$. $\mathcal{I}\{\omega\}$ is the indicator function with value 1 when condition ω satisfy. We normalize the action-value function to ensure boundedness within $[0,1]$ range. Note that the dual variables are projected in $[0, U]$ range, where $U \ge \lambda^*$.

Similarly, we instantiate Algorithm 2 in Orabona and Tommasi (2017) for updating the dual variable λ:

$$\lambda_{t+1} = \lambda_0 + \frac{\sum_{i=0}^{t} g_i}{L_t \max(\sum_{i=0}^{t}|g_i| + L_t, \alpha_\lambda L_t)}\left(L_t + \sum_{i=0}^{t}[(\lambda_i - \lambda_0)g_i]_+\right), \tag{6}$$

where $g_t := b - \hat{V}_c^t(\rho)$, α_λ is a tunable hyper-parameter, $L_t = \max(L_{t-1}, |g_t|)$ and L_0 is initialized to 0. Algorithm 1 summarizes CBP in the linear function approximation setting. In particular, for a given *coreset* $\mathcal{C}$, we estimate the action-value functions for a subset of $(s,a) \in \mathcal{C}$.

Algorithm 1: Coin-Betting Politex (CBP)

Input: $\alpha_\lambda > 0$ (parameter), π_0 (random policy initialization), $\lambda_0 = 1$ (dual variable initialization), m (Number of trajectories), T (Number of iterations), Feature map Φ, Coreset $\mathcal{C}$.
for $t = 0, \ldots, T-1$ **do**
  Using m trajectories for every $(s,a) \in \mathcal{C}$, compute weight vectors $\theta_r^{\pi_t}$ and $\theta_c^{\pi_t}$ of Q_r and Q_c respectively using LSTDQ (Lagoudakis and Parr, 2003).
  Compute $\hat{Q}_r^t(s,a) = \langle \theta_r^{\pi_t}, \phi(s,a)\rangle$, $\hat{Q}_c^t(s,a) = \langle \theta_c^{\pi_t}, \phi(s,a)\rangle$ and $\hat{Q}_l^t(s,a) = \hat{Q}_r^t(s,a) + \lambda_t\,\hat{Q}_c^t(s,a)$.
  Use stored vectors $\{\theta_r^{\pi_i}, \theta_c^{\pi_i}\}_{i=0}^t$ and compute $\pi_{t+1}(a|s)$ using Eq. (5).
  Compute $\hat{V}_c^{\pi_t}(\rho)$ and update λ_{t+1} using Eq. (6).
end
return mixture policy $\bar{\pi}_T := \frac{\sum_{t=0}^{T-1}\pi_t}{T}$.

Setting $U = \frac{2}{([\max_\pi V_c^\pi(\rho)]-b)(1-\gamma)}$, and using the primal and dual regret expressions for coin-betting (Orabona and Tommasi, 2017), we can show that a variant of CBP results in an $O\left(\frac{1}{(1-\gamma)^3\sqrt{T}} + \frac{\varepsilon\sqrt{d}}{(1-\gamma)^2}\right)$ bound on the reward sub-optimality and a constraint violation of $O\left(\frac{1}{(1-\gamma)^2\sqrt{T}} + \frac{\varepsilon\sqrt{d}}{1-\gamma}\right)$. The proof is omitted due to lack of space.

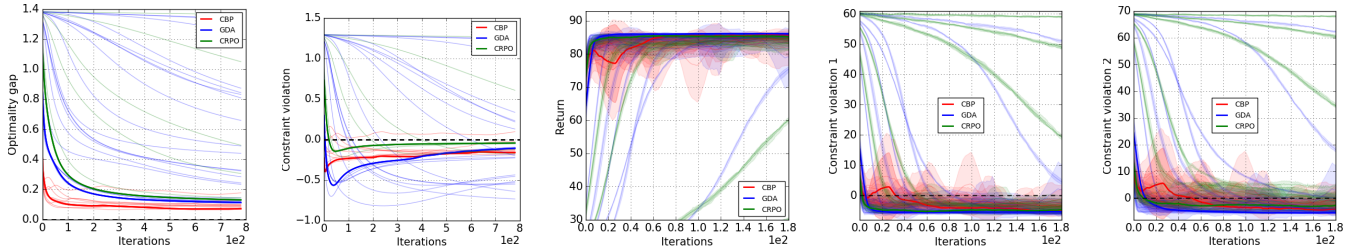
4 Experiments

4.1 Model-based setting

We show experiments on a simple tabular environment to compare our proposed algorithm, CBP, with the primal-dual approach, exemplified by GDA (Ding et al., 2020), and the primal-only approach exemplified by CRPO (Xu et al., 2021). We consider the $5X5$ gridworld introduced in (Sutton and Barto, 2018) and depicted in Fig. 1e. All four cardinal actions in the special states A and B receive a non-zero reward and cost values and land in states A' and B' respectively. The remaining states receive 0 reward and cost for all actions. Here $\gamma = 0.9$. In this section, we assume access to the true CMDP model. Figs. 1a and 1b show the performance metrics, OG and CV, for the three algorithms. CBP is *robust* to hyperparameter settings. The *best hyperparameters* corresponds to least OG and satisfy $CV \in [-0.25, 0]$. Next, we verify the robustness of the algorithms with respect to environment misspecification. We use the best value of the hyperparameters obtained for discount factor $\gamma = 0.9$ and use them for different CMDP corresponding to two other values of $\gamma = \{0.7, 0.8\}$ in Figs. 1c and 1d. This experiment suggests that CBP is robust to environment misspecification and consistently converges faster than the baselines. The **Github code** is available for all the experiments at https://github.com/arushijain94/CoinBettingPolite283

4.2 Model-free setting

In this section, we assume the model-free setting with no access to the true CMDP model. We use linear function approximation (LFA) to learn estimates of action-values using the LSTDQ algorithm (Lagoudakis and Parr, 2003). Tile coding (Sutton and Barto, 2018) is used to obtain features. We conduct experiments on two environments: Gridworld (Fig. 1e) and Cartpole (OpenAI gym). Along with the usual reward, we add two penalty constraints in the Cartpole environment, when (a) entering specific areas on x-axis and (b) having the angle of the pole larger than a threshold. Fig. 2a shows the OG and CV in the Gridworld environment with LFA. Similarly, Figs. 2c to 2e show the return and two constraint violations for constrained Cartpole environment. For the Cartpole experiment, we kept the $CV \in [-6, 0]$. In both the environments, CBP is robust to variations in the values of the hyperparameters.



(a) Gridworld OG (b) Gridworld CV (c) Cartpole return (d) Cartpole CV 1 (e) Cartpole CV2

Figure 2: **Model-free with linear function approximation:** We compare performance of CBP with baselines GDA and CRPO. Figs. 2a and 2b shows the performance in the gridworld environment with function approximation. Figs. 2c to 2e shows the return, CV1 and CV2 in constrained Cartpole environment. Here, dark lines correspond to the best hyper-parameters for each of the three algorithms. Lighter lines show the variation in performance with different hyper-parameters.CBP is robust to hyperparameter settings in both domains.

5 Conclusion

We proposed a general primal-dual framework to solve CMDPs with function approximation. We instantiated this framework using coin-betting algorithms from online linear optimization, and proposed the CBP algorithm. We proved that CBP is theoretically sound and has a good regret bounds. In addition, we showed empirically that CBP is robust to hyper-parameter tuning and environment mis-specification. We believe that developing robust, parameter-free algorithms is important for reproducibility in RL, and hope that our work will encourage future research in this area.

References

Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, 2019.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

Eitan Altman. *Constrained Markov decision processes*. CRC Press, 1999.

Dongsheng Ding, Kaiqing Zhang, Tamer Basar, and Mihailo Jovanovic. Natural policy gradient primal-dual method for constrained Markov decision processes. *Advances in Neural Information Processing Systems*, 33, 2020.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

Francesco Orabona and David Pal. Coin betting and parameter-free online learning. *Advances in Neural Information Processing Systems*, 29:577–585, 2016.

Francesco Orabona and Tatiana Tommasi. Training deep networks without learning rates through coin betting. *Advances in Neural Information Processing Systems*, 30, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction. Second Edition*. MIT Press, 2018.

Tengyu Xu, Yingbin Liang, and Guanghui Lan. CRPO: A new approach for safe reinforcement learning with convergence guarantee. In *International Conference on Machine Learning*, pages 11480–11491. PMLR, 2021.

Learning Abstract and Transferable Representations for Planning

Steven James, Benjamin Rosman
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
{steven.james,benjamin.rosman1}@wits.ac.za

George Konidaris
Department of Computer Science
Brown University
Providence RI, 02912
gdk@cs.brown.edu

Abstract

We are concerned with the question of how an agent can acquire its own representations from sensory data. We restrict our focus to learning representations for long-term planning, a class of problems that state-of-the-art learning methods are unable to solve. We propose a framework for autonomously learning state abstractions of an agent's environment, given a set of skills. Importantly, these abstractions are task-independent, and so can be reused to solve new tasks. We demonstrate how an agent can use an existing set of options to acquire representations from ego- and object-centric observations. These abstractions can immediately be reused by the same agent in new environments. We show how to combine these portable representations with problem-specific ones to generate a sound description of a specific task that can be used for abstract planning. Finally, we show how to autonomously construct a multi-level hierarchy consisting of increasingly abstract representations. Since these hierarchies are transferable, higher-order concepts can be reused in new tasks, relieving the agent from relearning them and improving sample efficiency. Our results demonstrate that our approach allows an agent to transfer previous knowledge to new tasks, improving sample efficiency as the number of tasks increases.

Keywords: representation learning, planning, abstractions, hierarchy

1 Introduction

Recently, state-of-the-art reinforcement learning (RL) approaches have made several significant advances in challenging domains, such as controlling nuclear reactors [1]. Despite these successes, it is clear that these approaches do not capture a remarkable aspect of human intelligence—namely, that humans can solve not just a single problem, but a massively diverse array of tasks. Consider the ALPHAZERO agent which attained superhuman performance in the grand challenge of Go [2]. While this is an immensely difficult task, the input to this agent is a set of binary vectors specifying stone locations, while the output is a location at which to place a stone. This input-output format is provided to the agent by a human designer because it captures exactly the task that must be accomplished. However, it means that the agent cannot solve any other tasks by definition; it cannot drive a car or cook a meal. While the former approach is useful in designing narrow, application-specific solutions, it falls short of the ultimate aim of generally intelligent agents.

In general, tasks in RL are formulated by human designers and provided to agents in a standardised, compact form. Though this practice is widespread, it sidesteps an important question: *where do these representations come from in the first place?* It is obvious that this approach is infeasible in the long run: we cannot preprogram an agent with every task it may encounter before deploying it in the real world. Nor can we require that a human designer accompany the agent throughout its lifetime, providing task representations as and when required. Clearly then, the only option is for the agent to learn its own representation for any newly encountered task directly from its observations of the world.

If we are to design a *single* agent capable of solving multiple tasks in the real world, it must necessarily have a complex sensorimotor space. However, solving long-horizon tasks at this low level is typically infeasible. A common approach to tackling this problem is hierarchical RL, which makes use of *abstractions* to simplify the problem. Action abstractions (also known as *skills*) alleviates the need to reason using low-level actions, while the use of state abstraction (where states are aggregated into high-level states) reduces the size of the problem. However, if an agent's abstractions are too high-level, it risks omitting important and necessary details. Conversely, if it seeks to preserve every last detail of the environment, then its representations will be too low-level and planning will once again be infeasible. The key question is how best to construct an abstract model of an environment while retaining only the information required for planning.

In this work, we outline a framework for learning transferable abstract representations from low-level data that can be used for long-term planning. More concretely, we extend the framework of Konidaris et al. [3] so that the learned representations are portable—given a new task, an agent can reuse the representations it has learned previously to speed up learning. We apply our framework to learn both agent- and object-centric representations in several high-dimensional domains, and demonstrate that our approach results in agents that are i) more sample efficient; ii) able to learn their own representations; and iii) able to use their learned representations to solve a variety of tasks.

2 Preliminaries

We begin by assuming that an agent is equipped with a set of skills and model tasks as semi-Markov decision processes $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$ where (i) $\mathcal{S}$ is the state space; (ii) $\mathcal{O}(s)$ is the set of temporally-extended actions known as *options* available at state s; (iii) $\mathcal{T}$ describes the transition dynamics, specifying the probability of arriving in state s' after option o is executed from s; and (iv) $\mathcal{R}$ specifies the reward for reaching state s' after executing option o in state s. An option o is defined by the tuple $\langle I_o, \pi_o; \beta_o \rangle$, where I_o is the *initiation set* specifying the states where the option can be executed, π_o is the *option policy* which specifies the actions to execute, and β_o the probability of the option terminating in each state [4].

We intend to learn an abstract representation suitable for planning. Prior work has shown that a sound and complete abstract representation must necessarily be able to estimate the set of initiating and terminating states for each option [3]. In classical planning, this corresponds to the *precondition* and *effect* of each high-level action operator. The precondition is defined as $\text{Pre}(o) = \Pr(s \in I_o)$, which is a probabilistic classifier that expresses the probability that option o can be executed at state s. Similarly, the effect represents the distribution of states an agent may find itself in after executing an option from states drawn from some starting distribution [3]. Since the precondition is a probabilistic classifier and the effect is a density estimator, they can be learned directly from option execution data. We can use preconditions and effects to evaluate the probability of an arbitrary sequence of options—a plan—executing successfully.

Partitioned Options For large or continuous state spaces, estimating $\Pr(s' \mid s, o)$ is difficult; however, if we assume that terminating states are independent of starting states, we can make the simplification $\Pr(s' \mid s, o) = \Pr(s' \mid o)$. These *subgoal* options are not overly restrictive, since they refer to options that drive an agent to some set of states with high reliability. While many options are not subgoal, it is often possible to *partition* an option's initiation set into a finite number of subsets. That is, we partition an option o's start states into finite regions $\mathcal{C}$ such that $\Pr(s' \mid s, o, c) \approx \Pr(s' \mid o, c), c \in \mathcal{C}$. Given (partitioned) subgoal options, we can estimate their preconditions and effects using the approach outlined by Konidaris et al. [3].

3 Agent-Centric Abstractions

Central to the field of artificial intelligence is the notion of the *agent*. Real-world agents are robots, which perceive their environments through sensors and act upon them with effectors. In practice, a human designer will usually build upon the observations produced by the agent's sensors to construct the Markov state space for the problem at hand, while discarding unnecessary perceptual information. Instead we will seek to effect transfer by using both the agent's sensor information—which is typically egocentric—in addition to the Markov state space. We assume that tasks are related because they are faced by the same agent. For example, consider a robot (equipped with various sensors) that is required to perform a number of as yet unspecified tasks. The only aspect that remains constant across all these tasks is the presence of the robot and, more importantly, its sensors, which map the state space $\mathcal{S}$ to a portable, lossy and egocentric observation space $\mathcal{D}$ known as *agent space*. We can use $\mathcal{D}$ to define portable options, whose option policies, initiation sets and termination conditions are all defined egocentrically. Because $\mathcal{D}$ remains constant regardless of the underlying SMDP, these options can be transferred across tasks [5].

Having made this distinction, we can write the state space of any given task $\mathcal{M}_i$ as the tuple $\langle \mathcal{X}_i, \mathcal{D} \rangle$, where $\mathcal{D}$ is shared across tasks and $\mathcal{X}_i$ represents task-specific state variables. Given this representation, we can follow a two-step process. The first phase uses the procedure outlined in Section 2 to learn portable abstract rules using agent-space transition data only. The second phase uses problem-space transitions to partition options in $\mathcal{X}_i$. Each partition is assigned a unique label, and these labels are used as parameters to ground the previously learned portable representations in the current task. For a new task, the agent need only estimate how the partition labels change under each option execution. Figure 1 illustrates this entire process, but see James et al. [6] for more details.

We test our approach in the *Treasure Game* [3], where an agent navigates a maze in search of treasure. This domain contains ladders and doors which impede the agent. Some doors can be opened and closed with levers, while others require a key to unlock. We first learn an abstract representation using agent-space transitions only, following the same procedure above. Once we have learned sufficiently accurate portable abstractions, they need only be instantiated for the given task by learning the linking between partitions. This requires far fewer samples than learning a task-specific representation from scratch. To illustrate, we construct ten levels and gather transition samples from each task. We use these samples to build both task-specific and egocentric (portable) models. For each level, we collect data until a model is sufficiently accurate at which point we continue to the next task. Results are given by Figure 2.

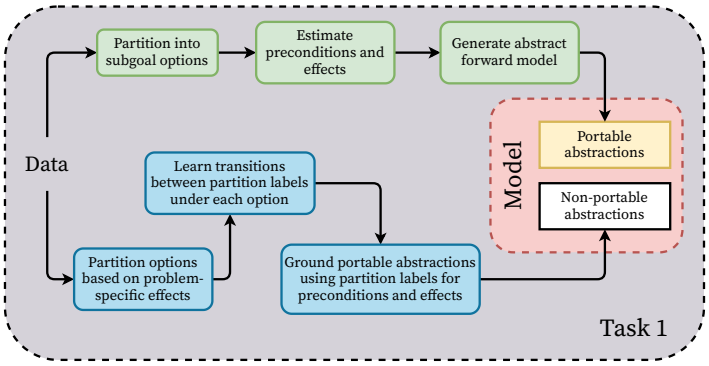


Figure 1: The agent learns transferable representations, which are then combined with problem-specific abstractions to form a model suitable for planning.

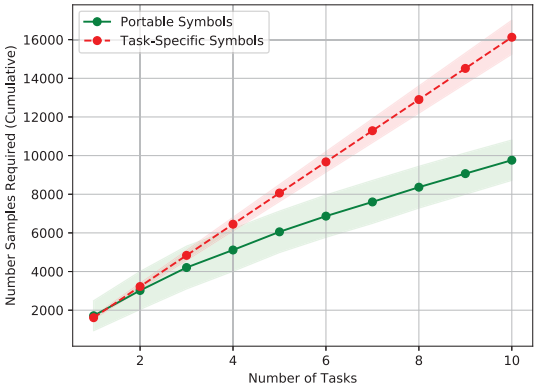


Figure 2: Owing to transfer, the number of samples required by the agent to learn a sufficiently accurate model decreases with the number of tasks faced.

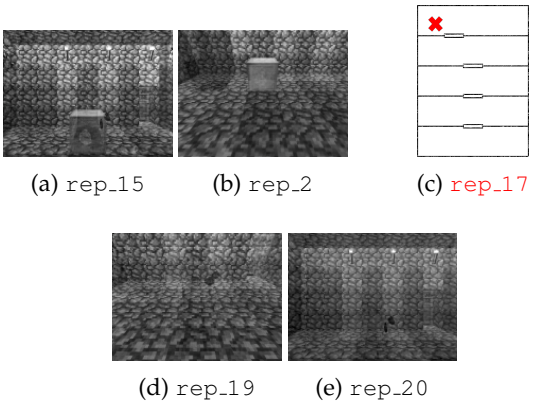
4 Object-Centric Abstractions

Having assumed the existence of an agent, it is natural to make another assumption—that the world consist of objects, and that similar objects are common amongst tasks. Previously, we assumed the existence of an agent equipped with sensors, which led to the idea of agent space. Since we are now assuming the existence of objects, a natural extension is to introduce the notion of *object space*. We adopt an object-centric formulation: in a task with n objects, the state is represented by the set $\{\mathbf{f}_a, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n\}$, where $\mathbf{f}_a$ is a vector of the agent's features and $\mathbf{f}_i$ are the features of object i [7].

The process to learn a grounded representation is now three-fold and is summarised by Figure 4. We first follow the same procedure outlined in Section 2 to construct a non-portable representation of a single task. Since object space is already factored into the constituent objects, each abstraction will refer to a distribution over a particular object's state. Next, we merge these representations where objects fall into the same "type" using the notion of *effect equivalence* [8]—two objects

are grouped into the same type when they undergo similar effects under the same set of options. Finally, we once more use the problem-specific state data to construct partition labels, which are used to ground previously learned portable representations in the current task. See James et al. [9] for more details.

We demonstrate our approach in a series of Minecraft levels, where each consists of five rooms with various items positioned throughout. Rooms are connected with either regular doors, which can be opened by direct interaction, or puzzle doors requiring the agent to pull a lever to open. The world is described by the state of each of the objects (given directly by each object's appearance as a 600×800 RGB image), the agent's view, and current inventory. The agent is given high-level skills, such as `ToggleDoor` and `WalkToItem`. To simplify learning, we downscale images and applying PCA to a greyscaled version, preserving the top 40 principal components. We follow the process in Figure 4 to learn portable object-centric representations, and ground them with task-specific partition labels derived from the agent's xyz-location. As mentioned, objects are grouped into types based on their effects, which is made easier because certain objects do not undergo effects under certain options. For example, the chest cannot be toggled, while a door can, and thus it is immediately clear that they are not of the same type. We investigate transferring abstractions between five procedurally-generated tasks, where each task differs in the location of the objects and doors. For a given task, the agent transfers all operators learned from previous tasks, and continues to collect samples using uniform random exploration until it produces a model that predicts the optimal plan can be executed. Figure 3 illustrates a learned abstraction, while Figure 5 shows the number of abstract option representations (operators) transferred between tasks.



(a) `rep_15` (b) `rep_2` (c) `rep_17`

(d) `rep_19` (e) `rep_20`

Figure 3: Abstract precondition and effect for breaking a gold block. The agent must be standing in front of a gold block (`rep_15`) at a particular location (`rep_17`), and the gold block must be whole (`rep_2`). As a result, the agent finds itself in front of a disintegrated block (`rep_20`), and the gold block is disintegrated (`rep_19`). Only the red abstraction must be relearned for each new task.

5 Hierarchies of Abstractions

The previous approaches, whether agent- or object-centric, resulted in an abstract decision problem. If we apply our framework repeatedly, it will discover increasingly higher order representations, which are themselves distributions over the representations at the level below; in the agent-centric setting, an abstract state space at level $i > 0$ is the tuple $\langle \mathcal{X}^{(i)}, \mathcal{D}^{(i)} \rangle$, where each $x \in \mathcal{X}^{(i)}$ and $d \in \mathcal{D}^{(i)}$ is a distribution over states in $\mathcal{X}^{(i-1)}$ and $\mathcal{D}^{(i-1)}$ respectively. The above formulation means that, as we construct more levels in the hierarchy, the resulting representations become increasingly compact and faster to plan with, but so does the degree of uncertainty.

Our first step is to construct an abstract representation using the approach in Section 3. Next we must decide how best to discover higher order skills in this new representation. We achieve this by converting our representation to a transition graph, identify "important" nodes (using the VOTERANK metric), and then construct options to reach these *subgoal* nodes using Djikstra's algorithm. Edges along these paths constitute our higher-order options. Note that since all the options contain only a single node in their termination set, they are subgoal by construction. We can then simply iterate this approach to construct an entire abstraction hierarchy.

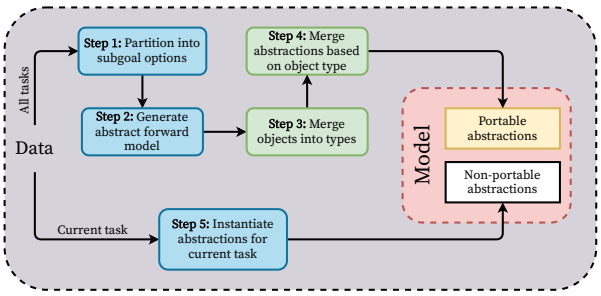


Figure 4: Learning object-relative representations from data. Blue nodes represent problem-specific representations, while green nodes are abstractions that can be transferred between tasks.

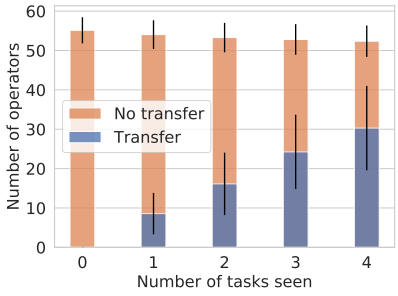


Figure 5: Orange: number of operators that must be learned to produce a sufficiently accurate model of a task. Blue: number of operators transferred between tasks. Mean and standard deviation over 80 runs.

We again apply our approach to the *Treasure Game* to construct portable hierarchies. To illustrate the effect of the hierarchy, we compute the distribution of the length of all pairs of shortest paths for each of the tasks when using abstractions from varying levels of the hierarchy. Results for the first task are given by Figure 6 and indicate that incorporating information at increasingly abstract levels of the hierarchy reduces the size of the graph (this trend holds across all other levels too). Consequently, the maximum planning horizon is shortened, which greatly simplifies the planning problem. We also investigate transfer by presenting the agent with each of the ten tasks in sequence. Unlike previously, portable representations here consist of representations at various levels in the hierarchy. We measure the number of samples required to learn a model of a new task, with the results illustrated by Figure 7. Although the results exhibit high variance (due to the exploration strategy, the differences in tasks, and the randomised task order), sample efficiency is clearly improved when an agent is able to reuse past knowledge.

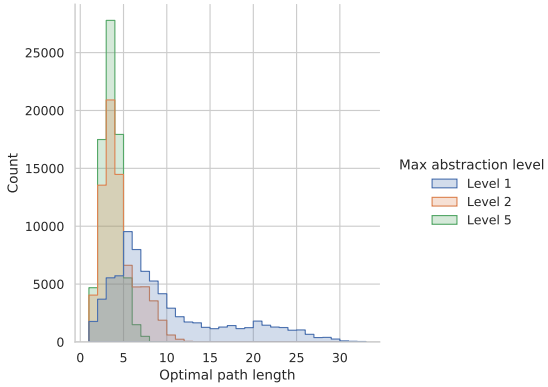


Figure 6: Distribution of optimal plan lengths in the first task when using hierarchies of varying heights.

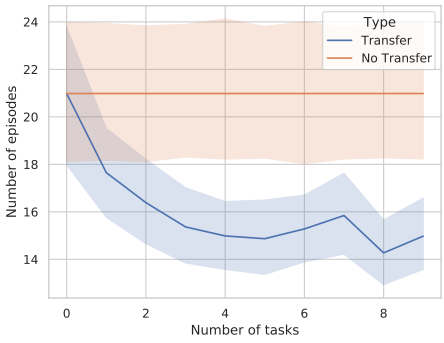


Figure 7: Number of episodes required to learn a model of a given task, decreasing as the agent observes more tasks. Mean and variance reported over 100 runs.

6 Conclusion

We proposed a framework for autonomously learning reusable representations. We showed how to learn an agent- and object-centric representation that can be used for planning. These representations can be transferred to new tasks, reducing the number of times an agent is required to interact with the world. We also showed how to construct a portable hierarchy of abstractions that can be used to plan at different levels. Altogether, our results indicate that the learned abstractions can be reused in new tasks, reducing the number of times an agent is required to interact with its environment. We believe this will be critical to scaling abstraction learning approaches to real world tasks in the future.

References

[1] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, pp. 414–419, 2022.

[2] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, and J. o. Schrittwieser, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] G. Konidaris, L. Kaelbling, and T. Lozano-Pérez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, no. January, pp. 215–289, 2018.

[4] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[5] G. Konidaris and A. Barto, "Building portable options: skill transfer in reinforcement learning." in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, vol. 7, 2007, pp. 895–900.

[6] S. James, B. Rosman, and G. Konidaris, "Learning to plan with portable symbols," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2020, pp. 4682–4691.

[7] E. Ugur and J. Piater, "Bottom-up learning of object categories, action effects and logical rules: from continuous manipulative exploration to symbolic planning," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation*, 2015, pp. 2627–2633.

[8] E. Şahin, M. Cakmak, M. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[9] S. James, B. Rosman, and G. Konidaris, "Autonomous learning of object-centric abstractions for high-level planning," in *International Conference on Learning Representations*, 2022.

Neuro-Nav: A Library for Neurally-Plausible Reinforcement Learning

Arthur Juliani
Araya Inc.
Minato-ku, Tokyo Japan
arthur_juliani@araya.org

Samuel A. Barnett
Princeton University
Princeton, NJ USA
samuelab@princeton.edu

Brandon Davis
Massachusetts Institute of Technology
Cambridge, MA USA
davisb@mit.edu

Margaret Sereno
University of Oregon
Eugene, OR USA
msereno@uoregon.edu

Ida Momennejad
Microsoft Research
New York City, NY USA
idamo@microsoft.com

Abstract

In this work we propose Neuro-Nav, an open-source library for neurally plausible reinforcement learning (RL). RL is among the most common modeling frameworks for studying decision making, learning, and navigation in biological organisms. In utilizing RL, cognitive scientists often handcraft environments and agents to meet the needs of their particular studies. On the other hand, artificial intelligence researchers often struggle to find benchmarks for neurally and biologically plausible representation and behavior (e.g., in decision making or navigation). In order to streamline this process across both fields with transparency and reproducibility, Neuro-Nav offers a set of standardized environments and RL algorithms drawn from canonical behavioral and neural studies in rodents and humans. We demonstrate that the toolkit replicates relevant findings from a number of studies across both cognitive science and RL literatures. We furthermore describe ways in which the library can be extended with novel algorithms (including deep RL) and environments to address future research needs of the field.

Keywords: reinforcement learning, open source, navigation, neuroscience

1 Introduction

Understanding how humans and other animals make non-reactive decisions in their environments remains a longstanding interest of psychologists and neuroscientists alike, especially following the proposal of cognitive maps a century ago [1]. Applying the formalism of reinforcement learning (RL) to decision making has accelerated the progress in this field within recent decades [2]. Indeed, there is neuroscientific evidence that various forms of learning in humans and other animals can be modeled using the formalism of RL, and thus be described by classical algorithms from the RL literature [3, 4].

Applying the RL paradigm has led to a deeper understanding of learning from describing simple forms of conditioning in animals [5], to providing a basis for complex navigational strategies in humans [6]. This work is often made possible by abstracting the underlying complexity of the real environment-agent system into an idealized model. This process involves the real environment of the animal being abstracted into a computationally tractable Markov Decision Process (MDP) [7], consisting of an underlying graph of state nodes, which are connected to one another via edges corresponding to actions taken by the agent. Likewise, the biological organism making the decisions is abstracted into an artificial agent modeled by a policy and value function, which can interact with and potentially learn the relevant dynamics of the MDP.

This process of abstraction is largely non-standardized, and has seen various ad-hoc implementations over the decades, often driven by the particular experimental needs of the researchers at the time. Furthermore, many of these concrete implementations are not available in open-source forms, change when translated from one coding language to another, or the code has been lost altogether. As a result, the ability to replicate or build on previous work has been limited, confusing trainees, slowing progress in the field, and preventing the straightforward comparison of various models or algorithms to animal data. On the other hand, most AI benchmarks remain limited to maximizing scores and superhuman performance, which misses the opportunity for neurally and biologically plausible representation and behavior.

In this work, we present Neuro-Nav[1], an open-source library providing a standardized set of benchmark environments and RL algorithms which can be used to both replicate previous findings, test the biological plausibility of new models, and provide scaffolding for future experimental work in the field. The benchmarks focus on domains that are the basis of many experimental studies: spatial navigation in mazes (Figure 1 Top), associative learning, and graph navigation tasks (Figure 1 Bottom) [4]. In addition, we provide standardized implementations of classical RL algorithms such as Q-Learning, Successor Representation, and Model-based RL with Value Iteration, which can be evaluated using the benchmark environments. We provide these all within the context of a documented and tested open-source repository, which can be freely used and developed further by the broader research community.

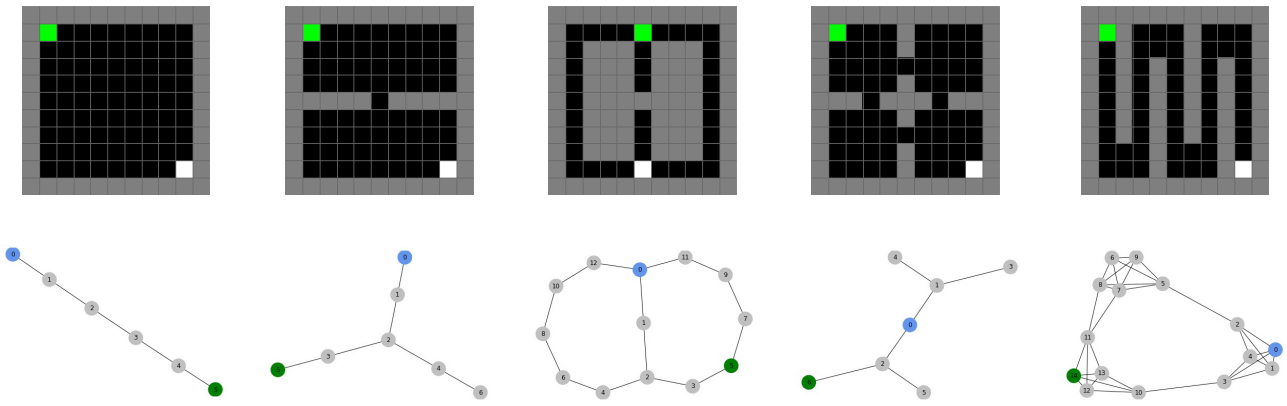


Figure 1: **Top**: Examples of maze environments. White square corresponds to agent location. Green square corresponds to goal location. Grey squares correspond to walls. **Bottom**: Examples of graph environments. Blue circle corresponds to agent location. Green circle corresponds to goal location. Environments drawn from studies in neuroscience [8, 9], classical RL [2], and cognitive science [10, 11].

2 Neuro-Nav Library

Neuro-Nav is an open source library consisting of three components: a set of benchmark environments, a toolkit containing artificial reinforcement learning agents and algorithms, and a set of interactive notebooks for replicating experimental

[1] https://github.com/awjuliani/neuro-nav

findings in the literature. All of the code in Neuro-Nav is written in the python programming language, and utilizes the numpy computational programming library to accelerate the relevant tensor mathematics.

2.1 Benchmark Environments

The environments included in Neuro-Nav consist of a set of graph and maze navigation tasks. In both cases, the underlying environment dynamics consists of a graph of 'state' nodes connected by 'action' edges. The agent occupies one of the nodes in the graph. The graph may also contain nodes that provide reward upon the agent occupying them, as well as a goal node, which both provides a reward as well as terminates the episode. Maze environments may be seen as a specific class of graph environments where the structure is mapped to a 2D grid, and the action space always consists of four actions: movement in each of the cardinal directions. Both types of environments are implemented using the OpenAI Gym interface, thus allowing for integration with a wide number of pre-existing open source RL codebases [12].

For each class of environments, we provide both an underlying library for defining and interacting with the environment as well as a set of pre-defined environments, which can be utilized for evaluation. These environments are often taken from existing literature, and serve to aid in the partial or complete replication of various findings from those original papers. In particular, we draw from environments defined in the context of neuroscience [8, 9], classical RL [2], and cognitive science [10, 11]. See Figure 1 for examples of a representative subset of the included maze and graph environments.

We also provide an abstraction for defining the observation space of the environment, which is completely orthogonal to that of the structure or topography of the environment. By default, we utilize a one-hot encoding for states, but also provide a set of alternative observational spaces such as Euclidean distance from walls in the maze tasks, or CIFAR images in the graph tasks, as well as many others.

2.2 Algorithms Toolkit

Algorithm	Function(s)	TD	Replay	Value Iteration
TD-Q	$Q(s,a)$	Yes	No	No
TD-AC	$V(s)$, $\pi(a\|s)$	Yes	No	No
TD-SR	$\psi(s,a)$, $\omega(s)$	Yes	No	No
Dyna-Q	$Q(s,a)$	Yes	Yes	No
Dyna-SR	$\psi(s,a)$, $\omega(s)$	Yes	Yes	No
Dyna-AC	$V(s)$, $\pi(a\|s)$	Yes	Yes	No
MBV	$Q(s,a)$, $T(s'\|s,a)$	No	No	Yes
MBSR	$\psi(s,a)$, $\omega(s)$, $Q(s,a)$, $T(s'\|s,a)$	Yes	No	Yes

Table 1: Reinforcement learning algorithms included in the Neuro-Nav toolkit. 'TD' column: the algorithm utilizes an online temporal difference learning rule. 'Replay' column: the algorithm utilizes offline replay algorithm. 'Value Iteration' column: the algorithm utilizes an offline model-based learning procedure.

In addition to graph and maze environments, we provide a set of artificial agents, which implement canonical algorithms from the RL literature. We focus on tabular rather than deep learning algorithms, as this has likewise been the focus of much of the literature at the intersection of neuroscience and RL [11]. We include algorithms implemented either as model-free or model-based learning algorithms, as well as hybrid algorithms such as Dyna. See Table 1 for a description of the provided agent types. Each agent implements one of multiple possible algorithms for learning a policy and value function. Decisions are taken by the agent with either a softmax policy or epsilon greedy strategy, both of which are available to use in any agent type.

3 Experimental Results

We validate Neuro-Nav by replicating known results from human and rodent navigation and decision-making experiments. In particular, we replicate two classes of results: behavioral, which compares the observable behavior of biological and artificial agents, and representational, comparing learned representations of biological agents (using neuroscience) and artificial agents. Here we present Neuro-Nav results using benchmark from both categories.

In order to validate Neuro-Nav for replication of behavioral results, we focused on two recent studies in the literature, [10] and [11]. In the former, agent performance is compared to that of humans in a set of navigation tasks, where aspects of the environments are changed partway during learning. In the latter, agent performance is compared to that of rodents in a set of maze navigation tasks. We present replications of the findings from Experiment 1 of [10] in Figure 2. We present results consistent with the results of [11] in Figure 3. 292

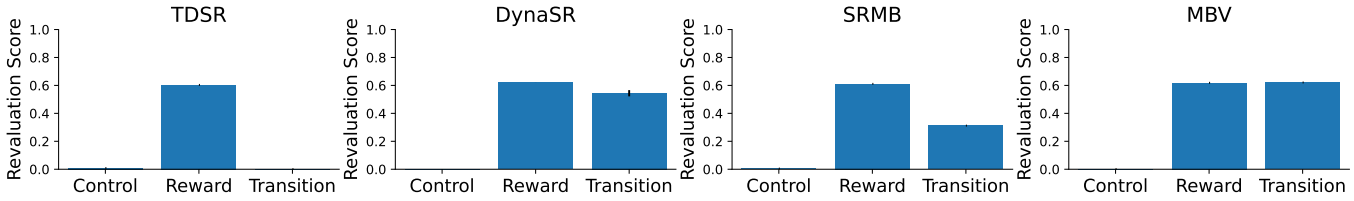


Figure 2: Replication of results from revaluation task described in Experiment 1 from [10]. Higher revaluation score corresponds to greater change in behavior after the re-learning phase. Both DynaSR and SRMB algorithms show most human-like revaluation behavior. Results averaged over ten experiments.

Beyond simply analyzing behavior, the internal content and structure of information utilized for decision making is of interest to many researchers. Here we focus on three relevant internal representations: value estimations, synthetic place fields, and grid fields [9]. We present Figure 4, which contains examples of each of these within an open field maze environment taken from an agent utilizing a successor representation.

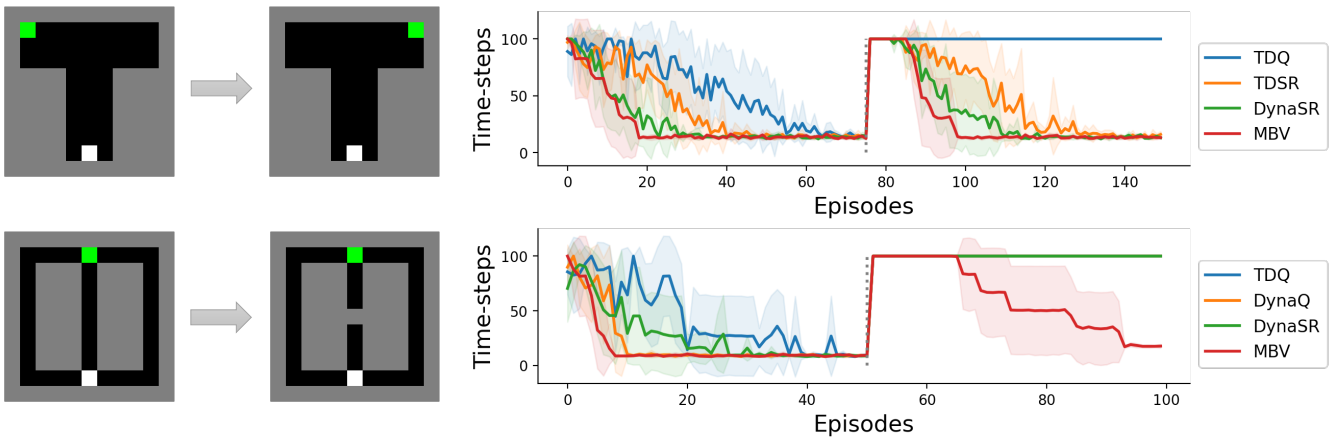


Figure 3: **Top**: Performance comparison between agent types on a reward transfer problem. Goal location changes at episode 75. SR and model-based algorithms successfully adapt to change. **Bottom**: Performance comparison between agents on a structure transfer problem. Environment structure changes at episode 50. Only model-based algorithm adapts to structural change. Graphs display average performance over five separate experiments. Tasks adapted from [11].

4 Discussion

Here we presented an open-source library of benchmark environments and algorithms for performing reinforcement learning experiments on decision making and navigation tasks. Neuro-Nav aims to empower reproducibility and standardization of evaluation within research in neurally plausible RL models of navigation and decision making. This project was developed with future extensions in mind, and as such, Neuro-Nav empowers users to easily develop new environments and algorithms beyond what is demonstrated in this work.

Neuro-Nav users can extend the benchmark environments in two ways. The first is by creating novel environments, i.e., MDPs, graphs, and maze topographies, to evaluate specific navigation, associative learning, or decision making experiments. The second is by defining novel observation spaces for the agents, such as using fractal or face/scene images as nodes or states, or perceiving Euclidean distance from maze walls. These examples, among others, capture relevant paradigms in the neuroscience literature [8, 13]. Future work using Neuro-Nav can add novel environments and novel algorithms to the open-source library, compare the performance of RL algorithms on all environments, and thus replicate a broader class of studies within the field.

Currently, the Neuro-Nav library only supports tabular learning agents. As such, the agents provided here are not capable of learning from the more varied class of possible observation spaces. In contrast, humans and other animals learn from complex multi-dimensional sensory signals. While much unrelated work in the field of deep RL has focused

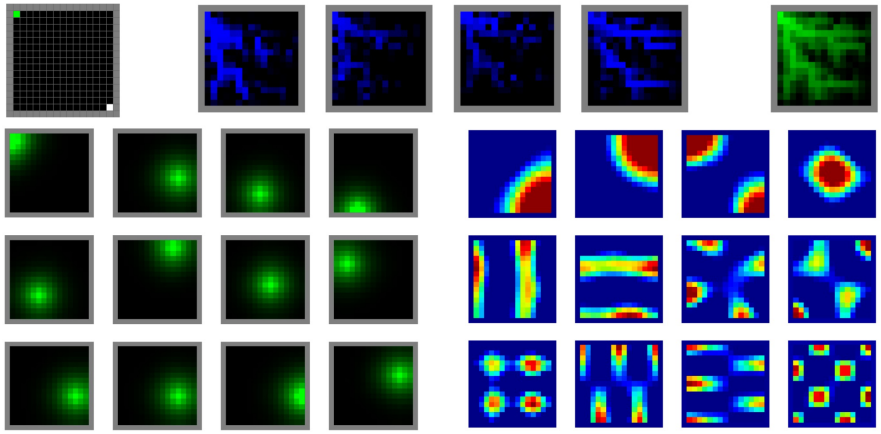


Figure 4: **Top Left**: Open field maze environment. **Top Middle**: Values of $Q(a|s)$ function in a maze environment after learning for each of the four actions. **Top Right**: Averaged value function over all actions, i.e. $V(s)$. **Bottom Left**: Values of a selection of units in the $\psi(s)$ function in a maze environment after learning. Units show place-like spatially selective fields. **Bottom Right**: PCA of $\phi(s)$ function in a maze environment after learning. Units show grid-like spatial selectivity. Procedure for generating place and grid fields adapted from [9].

on learning from high-dimensional observation spaces [14], we believe there is an opportunity for a middle path of lower-dimensional, but biologically grounded observation spaces, which enables linear models or simple neural networks to learn more expressive and generalizable behavioral policies. While we plan to include deep RL algorithms in future versions, we believe the current version of Neuro-Nav offers a promising step toward biologically plausible benchmarks, and a toolkit with potentially significant contributions to the field.

References

[1] Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Yael Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, 2009.

[4] Ida Momennejad. Learning structures: predictive representations, replay, and generalization. *Current Opinion in Behavioral Sciences*, 32:155–166, 2020.

[5] Richard S Sutton and Andrew G Barto. Time-derivative models of pavlovian reinforcement. 1990.

[6] Dylan Alexander Simon and Nathaniel D Daw. Neural correlates of forward planning in a spatial decision task in humans. *Journal of Neuroscience*, 31(14):5526–5539, 2011.

[7] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[8] Anna C Schapiro, Timothy T Rogers, Natalia I Cordova, Nicholas B Turk-Browne, and Matthew M Botvinick. Neural representations of events arise from temporal community structure. *Nature neuroscience*, 16(4):486–492, 2013.

[9] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653, 2017.

[10] Ida Momennejad, Evan M Russek, Jin H Cheong, Matthew M Botvinick, Nathaniel Douglass Daw, and Samuel J Gershman. The successor representation in human reinforcement learning. *Nature human behaviour*, 1(9):680–692, 2017.

[11] Evan M Russek, Ida Momennejad, Matthew M Botvinick, Samuel J Gershman, and Nathaniel D Daw. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLoS computational biology*, 13(9):e1005768, 2017.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[13] William de Cothi and Caswell Barry. Neurobiological successor features for spatial navigation. *Hippocampus*, 30(12):1347–1355, 2020.

[14] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Communication Emergence in a Goal-Oriented Environment: Towards Situated Communication in Multi-Step Interactions

Aleksandra Kalinowska
Northwestern University, Evanston, IL
& DeepMind, Edmonton, AB
ola@u.northwestern.edu

Elnaz Davoodi
DeepMind
Montreal, QC
elnazd@deepmind.com

Kory W. Mathewson
DeepMind
Montreal, QC
korymath@deepmind.com

Todd D. Murphey
Northwestern University
Evanston, IL
t-murphey@northwestern.edu

Patrick M. Pilarski
DeepMind & University of Alberta
Edmonton, AB
ppilarski@deepmind.com

Abstract

Effective communication enables agents to collaborate to achieve a goal. Understanding the process of communication emergence allows us to create optimal learning environments for multi-agent settings. Thus far, most of the research in the field explores *unsituated* communication in one-step referential tasks. These tasks are not temporally interactive and lack time pressures typically present in natural communication and language learning. In these settings, reinforcement learning (RL) agents can successfully learn *what* to communicate but not *when* or *whether* to communicate. Convergence is slow and agents tend to develop non-efficient codes, contrary to patterns observed in natural languages. Here, we extend the literature by assessing emergence of communication between RL agents in a temporally interactive, cooperative task of navigating a gridworld environment. Moreover, we *situate* the communication in the task—we allow the acting agent to actively choose between (i) taking an environmental action and (ii) soliciting information from the speaker, imposing an opportunity cost on communication. We find that, with situated communication, agents converge on a shared communication protocol more quickly. The acting agent learns to solicit information sparingly, in line with the Gricean maxim of quantity. In the same multi-step navigation task, we compare real-time to upfront messaging. We find that real-time messaging significantly improves communication emergence , suggesting that it is easier for agents to learn to communicate if they can exchange information when it is immediately actionable. Our findings point towards the importance of studying language emergence through situated communication in multi-step interactions.

Keywords: emergent communication; multi-agent reinforcement learning; cooperative AI

Acknowledgements

Work conducted while AK was an intern with DeepMind. We greatly appreciate conversations with Florian Strub, Ivana Kajic, and Michael Bowling at DeepMind that helped shape the research.

1 Introduction

Communication is a key skill for collaboration and hence largely beneficial in multi-agent settings. As humans, we share well-established communication protocols that have evolved over thousands of generations—shaped by functional pressures, such as time and articulation effort—to suit the needs of our daily tasks and to take advantage of our cognitive and physical capabilities. As an example, natural languages are known to be compositional, making them easier to learn and use [Kirby and Hurford, 2002]. Similarly, when we communicate, we are known to follow Grice's maxim of quantity—we try to be as informative as possible, giving only as much information as is needed [Grice, 1975]. If future artificial systems are to cooperate with humans, it will be beneficial for their communication protocols to follow these patterns. Understanding communication emergence among artificial agents will allow us to create optimal learning environments for multi-agent settings and supports the design of machines that will work well with each other and with people [Crandall et al., 2018, Steels, 2003].

With a recent increase in available computational power, the field has seen a lot of progress [Wagner et al., 2003, Lazaridou and Baroni, 2020]. Thus far, emergent communication has largely been studied in one-step referential games, such as the Lewis signalling task [Chaabouni et al., 2019, Li and Bowling, 2019, Lazaridou et al., 2018]. This type of learning environment is known to successfully enable language development [Kirby and Hurford, 2002] but does not allow agents to accelerate the learning process through back-and-forth interaction. In line with prior work [Evtimova et al., 2018] , we show that multi-step interactions can be beneficial for communication emergence, both in terms of agents' ability to converge to a collaborative solution and the time needed for convergence.

In most studies, the emerged language structures are analyzed for shared commonalities with natural languages, such as compositionality or encoding efficiency. Although desired, it is nontrivial for such properties to emerge spontaneously between artificial agents [Kottur et al., 2017]. For instance, artificial agents tend towards an anti-efficient encoding [Chaabouni et al., 2019]. This likely happens because in the Lewis signalling task, as well as in other simulated environments [Cao et al., 2018], agents have no incentive to be concise. In our approach, we show it is possible to obtain sparse communication by providing the agent with an action-communication trade-off, in line with the idea that *reward is enough to shape language* [Silver et al., 2021].

In our work, we explore the emergence of communication in a cooperative multi-step navigation task. Importantly, we *situate* the communication in the environment—we allow the acting agent to actively choose between (i) taking an action to move through the maze and (ii) soliciting information from the speaker. Our contributions are two-fold: (1) we study the emergence of *situated* communication and how it affects the communication protocol, and (2) we explore the effect of multi-step interactions on communication emergence.

2 Experimental Setup

The environment. We define a cooperative navigation task as a Markov Decision Process (MDP) with two reinforcement learning (RL) agents. The environment is set up as a pixel-based gridworld (7 by 7 cells). As illustrated in Figure 1, the maze includes 3 T-junctions, each allowing a right and left turn. Features of the world are represented with colors: walls are black, the maze is white, the agent is green, and the target is blue. The features are encoded with binary vectors.

The agents. There are two agents, a speaker and a listener (i.e. acting agent). The listener is embedded inside the gridworld and can take actions to move between cells. The action space of the listener spans 5 actions [move up, move down, move right, move left, stay in place]. The listener's observation consists of the environmental view (if any) concatenated with the message from the speaker. We test the listener under two conditions: (1) with no visibility, where the listener's observation consists solely of the speaker's message, and (2) with partial visibility, where the listener can see the 3 pixels directly in front of them. The second variant gives the listener environmental context to take actions without needing to rely solely on communication. The speaker does not reside within the gridworld and cannot take environmental actions (i.e. navigate the maze) but instead can communicate information to the listener. The message space of the speaker spans 5 symbols $[0, 1, ..., 4]$. At each timestep, the speaker can see the entire gridworld, including the location of the agent and the location of the goal. The speaker's view of the world map is rotated to align with the direction that the listener is facing. In our experiments, we test agents with and without memory. Agents without memory have to rely only on their current observations to generate messages or pick actions. Agents with memory have an internal representation of the history of an episode—they can use accumulated knowledge from prior timesteps to make decisions in the current timestep.

Agent architectures. The speaker and the listener share the same architecture without sharing weights or gradient values. They both have a 2-layer Convolutional Neural Network (CNN) that generates an 8 to 32 bit representation of the environment. In the case of the listener, this representation of the environment gets concatenated with the message received from the speaker. In both cases, the vector gets passed into a fully connected layer that generates the agent's action (a move or a message). Agents with memory have an additional single-layer LSTM [Hochreiter and Schmidhuber, 1997] after their fully connected layer. We train the agents using neural fitted Q learning [Riedmiller, 2005], with an Adam optimizer [Kingma and Ba, 2015] and $Q_t(\lambda)$ where $\lambda = 0.9$ and $\gamma = 0.99$. During training, agents use an ϵ-greedy policy with the exploration rate set as $\epsilon = 0.01$.

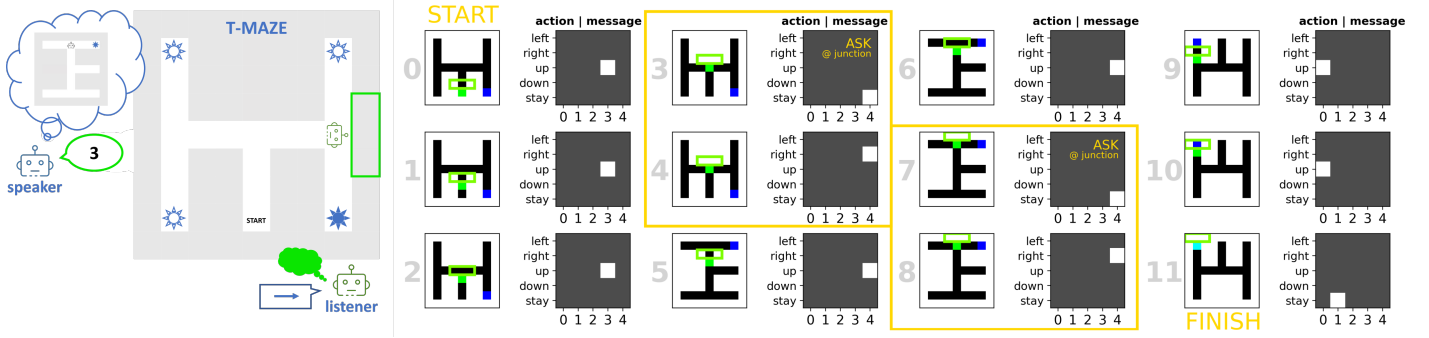


Figure 1: **Experimental setup and a walk-through of an example episode with *situated* communication.** In the maze on the left, stars indicate possible goal locations. To the right, we visualize an example episode of an active listener with partial visibility. The listener learns to solve the task optimally, deciding to stay and ask for information when at a junction (twice during the episode).

The task. The goal of the agents is to cooperate so that the listener reaches the target. In each experimental episode, both agents receive a reward $R = 1$ if the listener reaches the target before the episode terminates. Episode timeout is set to 100 steps. The goal locations are randomly assigned to one of 4 corners in the T-maze, as indicated with stars in Figure 1. In each episode, the listener agent starts from the bottom middle cell. We evaluate agent performance using 3 metrics: (1) task success (via a mean return per episode), (2) optimality of task solution (via a normalized reward per step), and (3) communication sparsity (via the number of asks per episode).

Communication modes. We compare three modes of communication: (1) real-time messaging with a passive listener, (2) real-time messaging with an active listener, and (3) upfront messaging with a passive listener. In mode 1, the speaker generates a 1-token message at every timestep and the message gets broadcasted to the listener before they choose an action. The speaker has to reason about both the content and timing of their message, deciding both *what* and *when* to communicate. In mode 2, we implement real-time messaging with an active listener. Here, the message is only broadcasted to the listener after they ask for information. The active listener can solicit to receive information in the next timestep by choosing to stay in place at the current timestep. The active listener has to learn *whether* to communicate at all. In mode 3, the speaker generates a 1-, 2-, or 3-token message at the beginning of each episode and that message gets broadcasted to the listener at each timestep throughout the episode.

We define the communication in mode 1 and 3 as *unsituated*—it is free and guaranteed to the agent at every timestep. There is no opportunity cost to communication. The communication in mode 2 is *situated*—we allow the acting agent to actively choose between (i) taking an environmental action and (ii) soliciting information from the speaker. As a result, the active listener experiences an opportunity cost to communication. They have to forego a move in the environment in order to obtain information from the speaker and make an informed decision.

Experimental parameters. For each experiment, we run a hyperparameter sweep over learning rates of the speaker and listener $\alpha = [10^{-5}, 10^{-6}, 10^{-7}]$ and over the size of the environmental representation $s = [4, 8, 16, 32]$. We run the simulation with each hyperparameter setting with 10 different random seeds. In the figures, we present the best performing agent pair from our hyperparameter sweep and/or the mean over the 10 replicas with the same hyperparameters as the best performing pair. When we plot metric means, we include the standard error of the mean.

3 Results

We start by generating a baseline for the task. Experiments confirm that without communication agents are unable to reliably solve the task. Under partial visibility, agents without communication can succeed in the task with a mean return of ≈ 0.25 per episode. With memory, baseline performance improves. However, due to the random location of the target, the listener cannot consistently solve the task in an optimal number of steps, converging to a normalized reward per step of ≈ 0.45. When allowed to communicate, all agents in the T-maze environment learn to solve the task and best agent pairs find an optimal solution, as visualized with the grey line in Figure 2.

The pressure of time in a multi-step interaction can incentivise sparse communication. In the first set of experiments, we evaluate the impact of situated communication on language emergence. Figure 1 shows a step-by-step example episode for an active listener with partial visibility. Under the partial visibility condition, information solicitation takes place mostly at the junctions, where the acting agent has a choice between two viable environmental actions. The active listener can learn to near optimally solicit information, asking ≈ 9.76 and ≈ 2.06 times per episode under the two visibility conditions, respectively.

In Figure 2 on the left, we illustrate the learning curves of the best performing agent pairs. Note that the active listeners ask for information frequently at the beginning of the interaction and gradually less over time. This suggests that agents initially have opportunities to align on a protocol. Over time, listeners learn *when* and *whether* to solicit information as

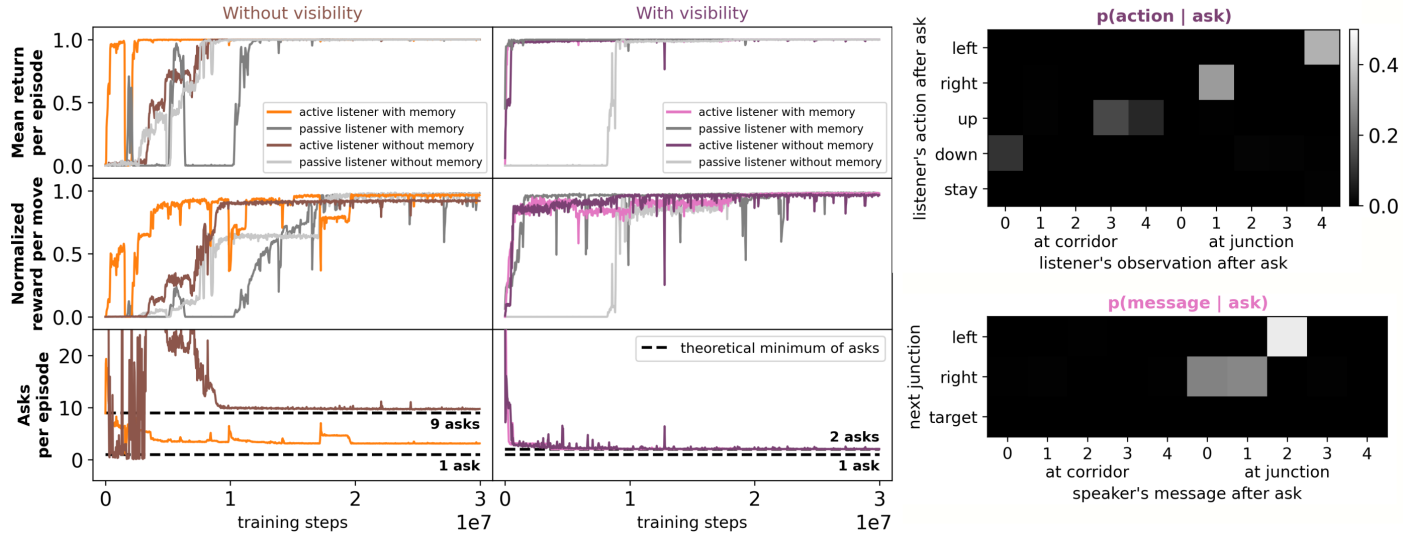


Figure 2: **Best performing pairs of agents with an active listener.** Under all conditions (with/without memory and with/without visibility) agents learn to solve the task via the shortest path. Listeners without memory learn to query the speakers in the optimal number of asks (once per step when the listener has no visibility and once per junction when the listener sees environmental context). Listeners with memory persist to ask for information when it is immediately actionable (instead of once at the beginning of an episode).

communication comes with a cost. We also observe that the best performing agent pairs with an active listener converge to an optimal solution faster than the best performing agent pairs with a passive listener. The results suggest that situated communication not only allows agents to learn a sparse communication protocol, in line with the Gricean maxim of quantity, but also has a positive impact on convergence speed.

The active listener exhibits a preference for just-in-time communication. Interestingly, when we test situated communication between agents with memory, agents continue to ask for information at the junctions (note the bottom heatmap in Figure 2). This is non-obvious—given memory, the active listener could ask for information at any point in the maze. In fact, if the agent were to be optimally sparse, they could (1) ask for information only once at the beginning of an episode, (2) receive a message encoding the address of the target, and (3) follow the relevant policy from memory. Instead, the active listener with memory learns sparse communication relative to a passive listener but they do not achieve the theoretically maximal sparsity, continuing to ask for information at the junctions when it is immediately actionable. This result suggests that it may be easier for agents to succeed at the task when they can control the timing of communication.

Real-time communication improves language emergence compared to upfront messaging. In our final experiment, we compare the real-time communication protocol (mode 1) with upfront messaging (mode 3). In both scenarios, the theoretical capacity of the communication channel allows the agents to communicate the necessary information, whether the agents choose to communicate directions, e.g., 'turn right', or a goal address, e.g., 'top left corner'. With upfront messages of length 1, 2, and 3, the speaker has 5, 25, or 125 unique messages available for communication, respectively.

With both real-time and upfront messaging, agents succeed in establishing a successful communication protocol when the listener has partial visibility—they converge to a mean return of 1 per episode. With no visibility for the acting agent, agent pairs with upfront messaging do not succeed at solving the task. Moreover, the real-time agents are more likely to converge to an optimal solution, being able to solve the T-maze task in 9 moves. With 1 upfront token, even the best agents learn to at-best solve the task in 12 steps. These agents seem to reliably learn unique messages to encode the action required at the first turn or the right/left part of the address, but they do not establish a unique encoding for the top/bottom portion of the address, as visible in the top heatmap in Figure 3. With 3 upfront tokens, the best agent pair agrees on 4 distinct symbols to encode the 4 possible goal locations. However, convergence is slow and on average agent pairs perform less optimally than under the real-time communication paradigm. We hypothesize that there are benefits to allowing communication to emerge from multi-step interactions. Our findings suggest that it is easier for agents to learn to communicate if they can exchange information when it is immediately actionable.

4 Conclusion & Discussion

Our results point towards the importance of studying emergent communication in multi-step interactions. The interactive aspect of communicating over time enables agents to learn both *what* and *when* to communicate. It improves overall task performance and speeds up convergence to an optimal solution. Secondly, we find that there is value in situating the communication in the task and giving the listener agency to choose *whether* to communicate at all. In this way, we allow the reward to shape the emergent communication protocol to exhibit properties of natural languages, such as sparsity.

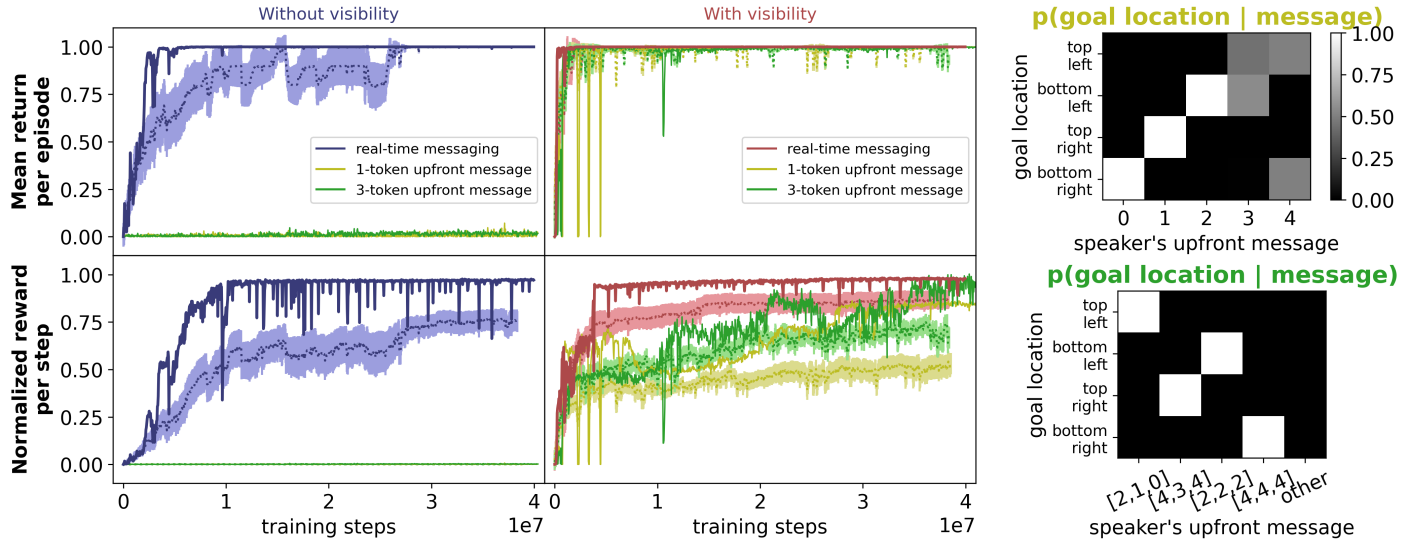


Figure 3: **Comparison of upfront and real-time messaging; agents have memory.** Real-time messaging improves convergence on a successful communication protocol. With upfront messaging, agents learn to solve the task before episode timeout when the listener has partial visibility. However, convergence is slow and agents are unlikely to solve the task in the optimal number of steps.

Our ongoing work will expand this idea and situate both the speaker and listener in the environment, allowing both agents to communicate and take actions in the gridworld environment.

References

[Cao et al., 2018] Cao, K., Lazaridou, A., Lanctot, M., Leibo, J. Z., Tuyls, K., and Clark, S. (2018). Emergent communication through negotiation. *Int. Conf. on Learning Representations*.

[Chaabouni et al., 2019] Chaabouni, R., Kharitonov, E., Dupoux, E., and Baroni, M. (2019). Anti-efficient encoding in emergent communication. *Advances in Neural Information Processing Systems*.

[Crandall et al., 2018] Crandall, J. W., Oudah, M., Ishowo-Oloko, F., Abdallah, S., Bonnefon, J.-F., Cebrian, M., Shariff, A., Goodrich, M. A., Rahwan, I., et al. (2018). Cooperating with machines. *Nature Communications*.

[Evtimova et al., 2018] Evtimova, K., Drozdov, A., Kiela, D., and Cho, K. (2018). Emergent communication in a multi-modal, multi-step referential game. *Int. Conf. on Learning Representations*.

[Grice, 1975] Grice, H. P. (1975). Logic and conversation. *Speech Acts*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *Int. Conf. on Learning Representations*.

[Kirby and Hurford, 2002] Kirby, S. and Hurford, J. R. (2002). The emergence of linguistic structure: An overview of the iterated learning model. *Simulating the Evolution of Language*.

[Kottur et al., 2017] Kottur, S., Moura, J. M., Lee, S., and Batra, D. (2017). Natural language does not emerge 'naturally' in multi-agent dialog. *Conf. on Empirical Methods in NLP*.

[Lazaridou and Baroni, 2020] Lazaridou, A. and Baroni, M. (2020). Emergent multi-agent communication in the deep learning era. *arXiv preprint arXiv:2006.02419*.

[Lazaridou et al., 2018] Lazaridou, A., Hermann, K. M., Tuyls, K., and Clark, S. (2018). Emergence of linguistic communication from referential games with symbolic and pixel input. *Int. Conf. on Learning Representations*.

[Li and Bowling, 2019] Li, F. and Bowling, M. (2019). Ease-of-teaching and language structure from emergent communication. *Advances in Neural Information Processing Systems*.

[Riedmiller, 2005] Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. *European Conf. on Machine Learning*.

[Silver et al., 2021] Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*.

[Steels, 2003] Steels, L. (2003). Evolving grounded communication for robots. *Trends in Cognitive Sciences*.

[Wagner et al., 2003] Wagner, K., Reggia, J. A., Uriagereka, J., and Wilkinson, G. S. (2003). Progress in the simulation of emergent communication and language. *Adaptive Behavior*.

Constructing and using cognitive maps for model-based control

Ata B. Karagoz*
Department of Psychological and Brain Sciences
Washington University in St. Louis
St. Louis, MO 63130
a.b.karagoz@wustl.edu

Zachariah M. Reagh
Department of Psychological and Brain Sciences
Washington University in St. Louis
St. Louis, MO 63130
zreagh@wustl.edu

Wouter Kool
Department of Psychological and Brain Sciences
Washington University in St. Louis
St. Louis, MO 63130
wkool@wustl.edu

Abstract

When making decisions, we sometimes rely on habit and at other times plan towards goals. Planning requires the construction and use of an internal representation of the environment, a cognitive map. How are these maps constructed, and how do they guide goal-directed decisions? Here we present work from an experiment where we coupled a sequential decision-making task with a behavioral representational similarity analysis approach to examine how relationships between choice options change when people build a cognitive map of the task structure.

In this pre-registered replication (n=161), we found that abstract representations reflecting higher-order relationships among items encountered in the task were associated with increased planning and better performance. In contrast, lower-order relationships such as simple visual co-occurrence of objects did not predict goal-directed planning. We also found that higher-order relationships were more strongly encoded among items associated with high-reward contexts, indicating a role for motivation during cognitive map construction. These results show that humans actively construct and use cognitive maps of task structure to make goal-directed decisions.

Keywords: cognitive maps, representational similarity analysis, behavioral, model-based learning

Acknowledgements

We'd like to thank Michael Freund, Gabriel James, and the members of the Complex Memory Lab, as well as the Control and Decision Making lab at Washington University. We'd also like to thank Dr. Alison Preston for the novel objects used here.

*personal github: github.com/atakaragoz

1 Introduction

Decisions are not made in a vacuum, but rather capitalize on the structure of the world around us. Imagine moving to a new city and learning about its structure as you navigate it. Ideally, your internal model will be set up to guide effective planning, strongly encoding relationships among major streets that connect different neighborhoods. However, this model may also incorporate features that are less relevant for planning, such as which street names rhyme with one another.

These internal models are commonly referred to as 'cognitive maps', a term coined by Tolman to explain how rodents were able to use spatial features of a maze to navigate towards goals (see [1] for review). The discovery of place cells and grid cells, which indicate an animal's current location in space and which are used in simulating possible future trajectories, provide the neural underpinnings for cognitive maps[2]. Recent work suggests that cognitive maps are not restricted to spatial domain but also encode more abstract features such as social dimensions[3]. However, much remains unclear about how humans construct abstract cognitive maps, specifically components that are goal-relevant, and how they use them to plan towards goals.

Recent studies of decision making formalize goal-directed planning as "model-based" reinforcement learning [4, 5, 6]. A model-based learner computes expected values for available actions by using a representation of the task structure. This flexible but computationally costly strategy enables it to apply the values learned at a given goal to every path that leads to that goal. Meanwhile, a model-free learner uses a more efficient but inflexible strategy, only updating the value of actions that led to reward, without considering the structure of the task.

Here, we assess the construction of cognitive maps to drive goal-directed decisions. To do so, participants (n=161) performed a variant of the 'two-step' task. This task dissociates model-based and model-free control by exploiting the ability of the model-based system to plan using an internal representation of the task structure, which contrasts with the model-free reliance on direct action-reward associations. Many prior studies using this task assume or ensure that an effective representation of the task is present. However, individual differences in cognitive maps may critically influence differences in model-based control. How does the nature and quality of one's cognitive map influence behavior? To index cognitive map structure, we developed a behavioral representational similarity analysis (behRSA) inspired by neuroimaging analyses. Participants were asked to rate "how related" pairs of objects were prior to and after learning the two-step task with minimal other instruction, allowing us to measure the amount of planning-relevant and planning-irrelevant information incorporated into their maps, and the influences of motivation on map formation.

2 Reinforcement learning model

We adapted an established hybrid reinforcement-learning model to assess participants' behavior in the decision-making task, specifically dissociating model-free and model-based decision making. Every trial t started out in one of four first-stage states $(s_{1,t})$ where one of two possible actions a_A and a_B could be selected $(a_{1,t})$. Depending on their selection, the participant deterministically transitioned to one of two second-stage states $(s_{2,t})$ where they could perform only one action $(a_{2,t})$ and then obtain a reward (r_t).

The model described here contains both a model-free learner and a model-based learner that learn expectations of long-term future reward $Q(s, a)$ for each combination of state and action. The model-free system learns reward expectations for each of the eight first-stage objects and two second-state objects, by updating their values based on reward prediction errors. The model-based system, on the other hand, learns a transition structure that represents which second-stage state each first-stage object leads to. It then combines this with the model-free reward expectations of the terminal, second-stage, states to select between first-stage options.

Model-free system: All model-free reward expectations were instantiated with a reward expectation of 4.5 (arithmetic mean of minimum and maximum possible reward) to all actions and states. The model-free learner would then use the $SARSA()$ temporal difference learning algorithm [7] to update its cached reward expectations based on the difference between predicted and received rewards. In the decision-making task this resulted in a reward prediction error being calculated at each stage according to:

$$\delta_{1,t} = Q_{MF}(s_{2,t}, a_{2,t}) - Q_{MF}(s_{1,t}, a_{1,t})$$
$$\delta_{2,t} = r_t - Q_{MF}(s_{2,t}, a_{2,t})$$

Notice that the second-stage prediction error incorporates the immediate reward outcome for that trial, but that the first-stage prediction error only incorporates expectations of future reward. The values of each prediction error were then used to update the reward expectations of the model-free learner at both the first and second stage:

$$Q_{MF}(s_{1,t}, a_{1,t}) \leftarrow Q_{MF}(s_{1,t}, a_{1,t}) + \alpha\delta_{1,t} + \alpha\lambda\delta_{2,t}$$

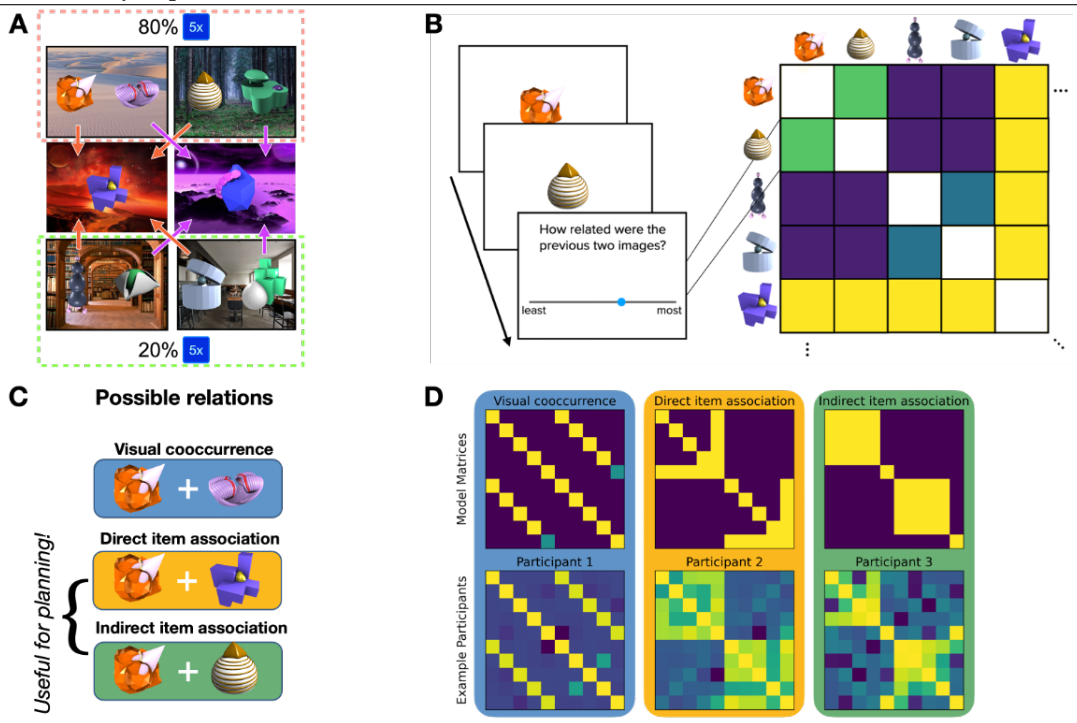


Figure 1: **A.** Task transition structure. There are four distinct first-stage states that each contain two unique objects. Each of these objects deterministically leads to a one of two second-stage states, as depicted by the colored arrows. These first-stage states differed in the amount of reward that could be received. For the two 'high-reward' states (top row), 80% of the trials were high stake, with a multiplier cue indicating that any points would be multiplied by five. For the other two 'low-reward' first-stage states, high-stake trials occurred on only 20% of trials. **B.** Representative trials of the pre- and post-behRSA task. Participants are shown each novel object pairing twice. A portion of similarity matrix generated from the behRSA task. The two similarity ratings for each pairing are averaged together and used to populate a symmetric representational similarity matrix where each cell is the pairwise similarity of the row and column object. In the example here, objects from the desert and forest become more similar to each other as well as their associated reward object. Items from the library and restaurant also increase similarity between themselves but less so while still being strongly associated with the reward object. **C.** Hypothesized relations that participants will use for the task. The first of these, visual cooccurrence, is not useful for planning. The later two, which are useful for planning, assess similarity between a first-stage object and the reward object it leads to as well as first-stage objects that lead to the same reward object. **D.** Example participants behRSA matrices for each of the 3 possible relations we hypothesized.

$$Q_{MF}(s_{2,t}, a_{2,t}) \leftarrow Q_{MF}(s_{2,t}, a_{2,t}) + \alpha\delta_{2,t}$$

Here, α is the reward learning rate (between 0 to 1) that determines how quickly new information about rewards is incorporated into the model-free learner expectations. The eligibility trace decay parameter λ (between 0 to 1) determines how much a reward prediction error experienced after the second stage choice changes first-stage reward expectations.

Model-based system: The model-based system combines learns the transitions structure of the task, and uses this to flexibly compute reward expectations for each available first-stage object. Specifically, it learns a transition matrix $T(s_1, a_1)$ that encodes the probability of moving to the second-stage state s_2 after choosing the action a_1 in the first-stage state s_1. In order to compute the model-based reward expectations, these probabilities were then combined with the reward expectations at the second stage:

$$Q_{MB}(s_{1,t}, a_{1,t}) = \sum_{s2} T(s_{1,t}, a_{1,t}) Q_{MB}(s_2, a_2)$$

$$Q_{MB}(s_{2,t}, a_{2,t}) = Q_{MF}(s_{2,t}, a_{2,t})$$

Choice rule: The model-free and model-based learners reward expectations in the first stage are integrated using a model-based weighting parameter w (ranging from 0 to 1) using the following rule:

$$Q_{net}(s_1, a_1) = 1 - wQ_{MF}(s_1, a_1) + wQ_{MB}(s_1, a_1)$$

We then used a softmax function to map the reward expectation to choice probabilities:

$$P(a_{1,t} = a_1|s_{1,t}) = \frac{exp(\beta[Q_{net}(s_{1,t},a_1)+\pi*rep(a_1)+\rho*resp(a_1)]}{\sum_{a_1'} exp(\beta[Q_{net}(s_{1,t},a_1')+\pi*rep(a_1')+\rho*resp(a_1')]}$$

Here, β is the inverse softmax temperature (left-bounded to 0) that determines how much influence reward expectations have on choice probabilities and can be thought of as a measure of exploration and exploitation. High softmax temperatures mean that the model is more likely to explore and low softmax temperatures mean the model more commonly exploits its knowledge. In order to capture choice perseveration, we included two parameters to capture both response key and stimulus 'stickiness'. The variable $rep(a_1)$ is defined as 1 if a_1 was the action that was chosen on the previous trial and 0 otherwise. The choice stickiness parameter π (left unbounded) related to choice perseveration when positive and choice switching when negative. The variable $resp(a_1)$ was defined as 1 if the action a_1 could be selected with the same response key that was used in the previous trial and 0 otherwise. The response stickiness parameter ρ captured perseveration of the response key press when positive and switching of response key press when negative. Together this results in a model with 8 free parameters.

3 Results

The decision-making task, modeled after an established paradigm, required participants to learn its structure in order to plan towards reward (Figure 1A). Each trial started pseudorandomly in one of four first-stage states, where participants chose between two objects. This choice determined which second-stage state – a red or a purple 'planet' – would then be encountered. For each first-stage state, one object always led to the red planet, and the other always to the purple planet. On each planet, participants then interacted with an object that provided a scalar reward which slowly changed over time.

We used *maximum a-posteriori* estimation to fit a dual-system reinforcement-learning model to behavior on this task[8]. This model describes behavior as a mixture of model-free and model-based control weighted according to a mixture parameter w. This parameter is fit closer to 1 for pure model-based control and closer to 0 for pure model-free control. Mirroring prior work, we found that participants' behavior reflected a mixture of model-free and model-based control (mean w = 0.57). This suggests that participants learned an internal representation of the task, a cognitive map, and that they used this for goal-directed decision making.

3.1 Behavioral indices of representational change track task structure and are effected by motivation

We used participants' relatedness ratings of objects to measure the structure of their cognitive maps (Figure 1B). Specifically, we formulated three ways in which objects could become more 'related' through task experience. First, we hypothesized that objects could become related if they co-occurred in a first-stage state (Figure 1C). Such a representational shift, although capturing some task structure, is not useful for planning because it does not reflect the consequences of actions. We also hypothesized two forms of representational shifts that related to the task's transitions (Figure 1C). In one case, we hypothesized that first-stage state objects and the second-stage objects they lead would become related. This representation, which we call a "direct item association", allows for planning from a first-stage action to the related second-stage state. We also hypothesized that all first-stage objects leading to the same second-stage state would become related. We call this representation an "indirect item association", because it encodes relations between objects that never occurred on the same trial.

To measure how strongly participants encoded these aspects of the task structure, we formalized these three representational shifts from pre- to post-task behRSA data with pre-registered hypothetical model matrices. Then, we used multiple linear regressions models to fit the changes in behRSA data to these matrices, for each participant separately. This produces three regression coefficients (co-occurrence, direct-item, indirect-item), each reflecting the strengths of one of the hypothesized representational shifts.

At the group level, participants judged item similarity in a manner consistent with each model matrix (example subjects with strong fits to each coefficient can be seen in Fig 1D). Participants judged objects as more related when they had occurred in the same first-stage state ($t(160) = 4.64,\ p < 0.001,\ d = 0.37$), when they constituted a pair where one object transitioned to the other ($t(160) = 5.98,\ p < 0.001,\ d = 0.47$), and when they both led to the same second-stage state ($t(160) = 5.29,\ p < 0.001,\ d = 0.42$). In other words, all three hypothesized components of the task were represented at the group-level.

We further sought to understand whether the difference in incentives between contexts affect the representation of task structure. We predicted that aspects of the task related to higher-order structure - both direct item associations and indirect item associations - would be more strongly encoded for the high-stake compared to the low-stake context items (Figure 1A).

To test this, we ran new multiple linear regressions, estimating the coefficients separately for each stake context. We found no difference between the high- and low-stake context representations of visual cooccurrence ($t(160) = 0.45,\ p = 0.65,\ d = 0.04$). We found a non-significant trend toward a context-driven difference in the direct item outcome representations ($t(160) = 1.74,\ p = 0.08,\ d = 0.14$). However, the indirect item associations were encoded more strongly for the high-stakes context compared to the low-stake context ($t(160) = 3.22,\ p = 0.0015,\ d = 0.25$). These results indicate that incentives lead to stronger representations for goal-directed information in task structure.

3.2 Correlation with w parameter and points earned

If the behRSA data measures the cognitive map, and cognitive maps enable planning, then individual differences in these representations of similarity structure should correlate with task performance (average reward rate) and reliance on model-based control. We predicted positive correlations for structured aspects of the task structure that are important for goal-directed planning but not for lower-order relationships.

Consistent with our hypotheses, we did not observe a relationship between performance and the strength of the visual cooccurrence component ($r(159) = 0.004, 95\%\ CI = [-0.15,\ 0.16],\ p = 0.96$). However, we found that performance was positively correlated with the encoding of direct item associations ($r(159) = 0.32, 95\%\ CI = [0.18,\ 0.46],\ p < 0.001$) and indirect item associations ($r(159) = 0.45, 95\%\ CI = [0.31,\ 0.56],\ p < 0.001$).

Next, we found a trending but non-significant negative correlation between model-based control and the strength of the visual cooccurrence component ($r(159) = -0.14, 95\%\ CI = [-0.29,\ 0.01],\ p = 0.0753$). Critically, however, we found that use of model-based control was positively correlated with the encoding of direct item associations ($r(159) = 0.33, 95\%\ CI = [0.18,\ 0.46],\ p < 0.001$), and indirect item associations ($r(159) = 0.48, 95\%\ CI = [0.35,\ 0.59],\ p < 0.001$).

4 Discussion

The way we plan toward goals to make optimal decisions is a key issue in the study of human behavior. In prior studies of goal-directed decision making, it has largely been assumed or ensured that the structure of a task was fully learned. As a consequence, it has been generally taken as a premise that model-based control simply operates over these representations, rather than considering the representations themselves as important sources of variance. In this preregistered experiment, we show the nature of these task representations, or cognitive maps, critically relate to one's ability to plan and maximize reward. Using a behavioral variant of RSA that measures similarity at a higher-level than in fMRI, we assessed the representation of differing components of a validated two-step task, testing *a-priori* hypotheses about which components of this task might be represented. Strikingly, the principal ways in which participants differed were largely accounted for by our *a-priori* models. We found that participants' representation of both directly and indirectly associated components of the task, which were relevant to higher-order task structure, were correlated with greater model-based control and performance in the task. This goes beyond simply representing the transition structure, as seen in the indirect association case. Moreover, these indirectly associated representations were more strongly present in contexts with increased incentives, further indicating goal-relevance of these cognitive maps. In contrast, goal-irrelevant information such as mere item co-occurrence did not relate to cognitive control or task performance.

References

[1] T. E. J. Behrens, T. H. Muller, J. C. R. Whittington, S. Mark, A. B. Baram, K. L. Stachenfeld, and Z. Kurth-Nelson, "What is a cognitive map? organizing knowledge for flexible behavior," *Neuron*, vol. 100, no. 2, pp. 490–509, 2018.

[2] J. O'Keefe and L. Nadel, *The Hippocampus as a Cognitive Map*. Clarendon Press, 1978.

[3] S. A. Park, D. S. Miller, and E. D. Boorman, "Inferences on a multidimensional social hierarchy use a grid-like code," *Nature Neuroscience*, vol. 24, pp. 1292–1301, Sept. 2021. Number: 9 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Decision;Social neuroscience Subject_term_id: decision;social-neuroscience.

[4] N. Daw, S. Gershman, B. Seymour, P. Dayan, and R. Dolan, "Model-based influences on humans' choices and striatal prediction errors," *Neuron*, vol. 69, no. 6, pp. 1204–1215, 2011.

[5] W. Kool, F. A. Cushman, and S. J. Gershman, "When does model-based control pay off?," *PLOS Computational Biology*, vol. 12, no. 8, p. e1005090, 2016. Publisher: Public Library of Science.

[6] W. Kool, S. J. Gershman, and F. A. Cushman, "Cost-benefit arbitration between multiple reinforcement-learning systems," *Psychological Science*, vol. 28, no. 9, pp. 1321–1333, 2017. Publisher: SAGE Publications Inc.

[7] G. A. Rummery and M. Niranjan, "On-Line Q-Learning Using Connectionist Systems," tech. rep., 1994.

[8] S. J. Gershman, "Empirical priors for reinforcement learning models," *Journal of Mathematical Psychology*, vol. 71, pp. 1–6, 2016.

Explaining Reinforcement Learning Agents By Policy Comparison

Jun Ki Lee
Department of Computer Science
Brown University
Providence, RI 02906
jun_ki_lee@brown.edu

Michael L. Littman
Department of Computer Science
Brown University
Providence, RI 02906
mlittman@cs.brown.edu

Abstract

To explain a reinforcement-learning agent, we propose comparing its policy to a baseline policy at a set of automatically identified decision points. Our novel method for selecting important decision points considers each possible state and decomposes the agent's value into the reward obtained before vs. after visiting that state. A state is considered important if the reward obtained by the agent's policy and the baseline policy are very different after the policy has changed. We demonstrate the utility of this approach on a grid world domain.

Keywords: Interpretability and explainability, occupancy frequency, Value decomposition

Acknowledgements

DARPA XAI.

1 Introduction

Reinforcement learning (RL) is a method that lets an agent interact with its environment making decisions sequentially and learns along the process of achieving specified goals [6]. As reinforcement learning is applied to larger domains such videos games and robotics [3, 1], explaining a learned agent is becoming more important [5].

In this paper, we propose a novel method for providing a tool to decompose the agent's value into the accumulated reward obtained before vs. after visiting a state and use this metric to select an important decision state where switching to a different policy would result in the most impact in the overall value of starting states. Based on this analysis, we generate an explanation for the most influential state.

2 Backgrounds

2.1 Markov Decision Process

Most RL agents are modelled by a Markov Decision Process (MDP). An MDP is defined by a tuple $\langle S, A, T, R, \gamma \rangle$. S denotes all possible states in an agent's environment and S_0 is a set of possible start states. The transition function $T : S, A \rightarrow S$ maps a state and action pair to its next state. If $s \in S$ and $T(s, a, s') = 0$ for $\forall s', a$, s is a terminal state and a set of terminal states are denoted as S_T. In this paper, the reward function $R : S \rightarrow \mathbb{R}$ is defined only for arrival states regardless of its initial state and an action, $R(\cdot, \cdot, s)$. γ is a discount rate. The objective of an agent is to maximize the accumulate sum of rewards discounted by γ.

A policy, $\pi : S \rightarrow A$, maps a state to an action and characterizes the behavior of an agent. The Q-value, $Q_\pi(s, a)$ is the expected return of a state s by following a policy π after taking ac action a and can be defined as $\mathbb{E}_\pi \left[\sum_{k=1}^{\infty} \gamma^k R(s_{t+k} \,|\, s_t = s, a_t = a\right]$. An optimal policy π^* is the policy that maximizes $V_\pi(s), \forall s \in S$. A value of a state is given as $V_\pi(s) = Q(s, \pi(a))$.

2.2 Explaining a policy for a MDP

Khan, Poupart, and Black first developed a method to explain an RL agent represented in a MDP using an occupancy frequency [2]. Using an occupancy frequency they developed a method to distinguish the most influential sets of states that each set has the same reward value. Templates for explanations can then be generated including an occupancy frequency for that set of states and factors of states that are generalized from the set of states.

Our approach is built upon this framework and add an algorithm to compare two different policies and find a state that switching to a different policy from this state has the most impact in the overall utility of an agent, $\mathbb{E}_{s_0 \in S_0} [V(s_0)]$.

3 Methods

This section describes are algorithmic approach.

3.1 Occupancy Frequency and Value Decomposition

A (discounted) *occupancy frequency* K of a state s' starting from s_0 following a policy, π, is defined by

$$K^{s_0,\pi}(s') = \sum_{t=0}^{\infty} \gamma^t P(s_{t+1} = s' \,|\, s_0). \tag{1}$$

It can be computed by a recurrence relation derived from the above definition.

$$K^{s_0,\pi}(s') = \sum_{s} T(s'|s, \pi(s)) \left(P(s_0 = s) + \gamma K^{s_0,\pi}(s)\right). \tag{2}$$

Given a policy π, the occupancy frequency can be used to calculate a value of a starting state s_0 together with a reward function, R.

$$V_\pi(s_0) = \sum_{s \in S} \left[K^{s_0,\pi}(s) \cdot R(\cdot, \cdot, s)\right]. \tag{3}$$

In this paper, we propose a new decomposition of this occupancy frequency K into two parts. Given a state $s^* \in S$, $s^* \notin S_T$, occupancy frequencies of a state $s' \in S$ before and after visiting s^* can be separated into anterior and posterior occupancy frequencies, L and M respectively:

$$K_{s^*}^{s_0,\pi}(s') = L_{s^*}^{s_0,\pi}(s') + M_{s^*}^{s_0,\pi}(s'). \tag{4}$$

The recurrence relationships for calculating L and M are given as

$$L_{s^*}^{s_0,\pi}(s') = \sum_{s \in S \setminus \{s^*\}} T(s' \mid s, \pi(s)) \left(P(s_0 = s) + \gamma L_{s^*}^{s_0,\pi}(s)\right), \tag{5}$$

and

$$M_{s^*}^{s_0,\pi}(s') = \gamma T(s' \mid s^*, \pi(s^*)) L_{s^*}^{s_0,\pi}(s^*) + \gamma \sum_s T(s' \mid s, \pi(s)) M_{s^*}^{s_0,\pi}(s). \tag{6}$$

The different M value can be calculated for a counterfactual policy change from π_0 to π_1 when the state s^* is reached if we use the L from the policy π_0:

$$M_{s^*}^{s_0,\pi_0,\pi_1}(s') = \gamma T(s' \mid s^*, \pi_1(s^*)) L_{s^*}^{s_0,\pi_0}(s^*) + \gamma \sum_s T(s' \mid s, \pi_1(s)) M_{s^*}^{s_0,\pi_0,\pi_1}(s). \tag{7}$$

Using the linearity of the occupancy frequencies, we can derive the value decomposition:

$$V_{s^*,\pi}(s_0) = V_{s^*,\pi}^{L}(s_0) + V_{s^*,\pi}^{M}(s_0). \tag{8}$$

3.2 Contrasting two different policies

Our approach to explaining RL agents contrasts a policy π_0 with the other policy π_1. Specifically, we want to answer the question of *why* an agent should choose its preferred policy π_0 over a contrasting policy π_1. The comparison of two polices is achieved by situating an agent in a counterfactual situation. We run an agent from a start state or a set of start states and let it follow its preferred policy π_0 until a significant state s^* is reached. We compare the behavior of an agent that continues with its π_0 policy to one that switches at that point to the contrasting policy π_1.

The value decomposition approach from the previous section can efficiently compare the value of the original and counterfactual policy. Both policies' expected values can be calculated via:

$$V_{s,\pi_0,\pi_0}(s_0) = V_{s,\pi_0}^{L}(s_0) + V_{s,\pi_0,\pi_0}^{M}(s_0), \tag{9}$$

$$V_{s,\pi_0,\pi_1}(s_0) = V_{s,\pi_0}^{L}(s_0) + V_{s,\pi_0,\pi_1}^{M}(s_0). \tag{10}$$

By looking into the theses value differences, we are comparing the *impact* of switching policies, $I_{s,\pi_0,\pi_1}(s_0)$, after an agent first visits the intermediate state s. This impact is measured by

$$I_{s,\pi_0,\pi_1}(s_0) = V_{s,\pi_0,\pi_0}(s_0) - V_{s,\pi_0,\pi_1}(s_0) = V_{s,\pi_0,\pi_0}^{M}(s_0) - V_{s,\pi_0,\pi_1}^{M}(s_0). \tag{11}$$

$$\mathop{\mathbb{E}}_{s_0 \in S_0}\left[I_{s,\pi_0,\pi_1}(s_0)\right] = \mathop{\mathbb{E}}_{s_0 \in S_0}\left[V_{s,\pi_0,\pi_0}^{M}(s_0) - V_{s,\pi_0,\pi_1}^{M}(s_0)\right]. \tag{12}$$

The expectation of the impact over all the start states measures the overall impacts of switching policies with respect to a intermediate state s. To generate a useful explanation, we want to find the state s^* that maximizes this impact, $s^* = \mathrm{argmax}_s \, \mathbb{E}_{s_0 \in S_0}\left[I_{s,\pi_0,\pi_1}(s_0)\right]$. The state is the one that, if an agent changes from π_0 to an alternative policy π_1 upon reaching that state, it will have the most impact on the overall performance of an agent.

4 Experiment and Evaluation

We demonstrate our method with an experiment on the 4×3 GridWorld domain introduced in the Russell and Norvig's book [4]. In this domain, we look at three different methods for choosing an influential state to use in our explanation.

4.1 Metrics

(A) Maximum Q-value difference: $s_q = \mathrm{argmax}_s \left[Q_{\pi_0}(s, \pi_0(s)) - Q_{\pi_0}(s, \pi_1(s))\right]$. This method answers the question: Which state would have its value change the most if it switched from following π_0 to following the action proposed by π_1 for one step? As Q-values encode a simple kind of counter-factual, and Q-values are produced in the context of many RL algorithms, this method is a very natural one to consider. A drawback of this method, however, is it does not take into account where the agent actually starts: s_0 (or a distribution over such states). A state could have very different Q-values, but be so unlikely to be reached that is hypothetical difference is moot.

(B) Maximum value difference: $s_v = \mathrm{argmax}_s \left[V_{\pi_0}(s) - V_{\pi_1}(s)\right]$. This method chooses the state where the two policies differ the most in terms of their state values V. The value of a state $V_\pi(s)$ is the expected discounted return starting from s and following π. The maximum value difference is a direct and simple way to select a state that is very different for the two different policies—it's the place in the state space where following the policies leads to the most extreme difference in value. It is like the maximum Q-value difference except it considers switching behavior indefinitely, instead of for

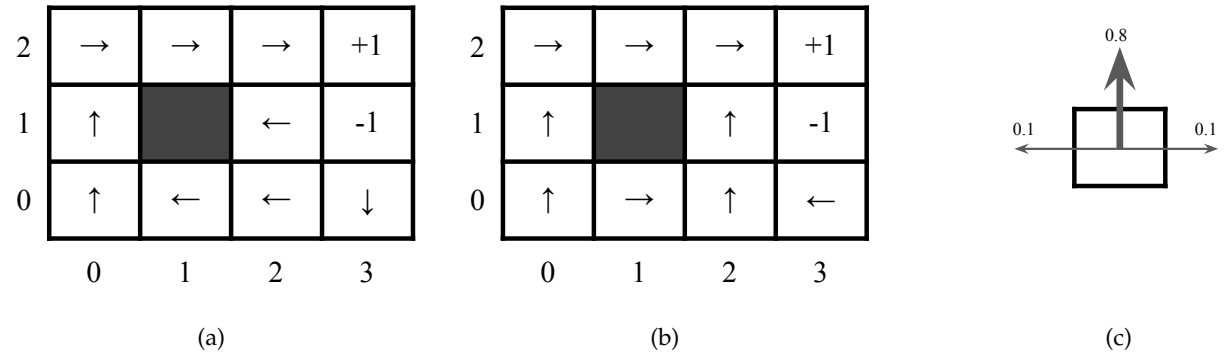


Figure 1: GridWorld policies. (a) An optimal policy π_0 with $R(\cdot, \cdot, s) = -0.04, \forall s \in S, s \notin S_T$. (b) Shortest path policy π_1 with the same R. (c) A diagram showing the slip probability of an action.

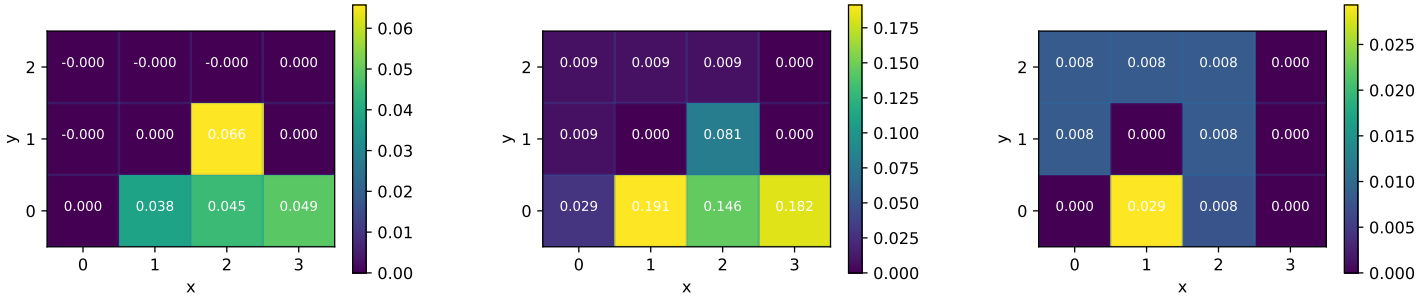


Figure 2: The comparison of three evaluation metrics. (a) Q-value difference. (b) Value difference. (c) Impact using anterior and posterior occupancy frequencies.

the single step used in Q values. As such, it provides a stronger contrast and a more meaningful counter-factual. Like the maximum Q-value difference, however, it does not consider the likelihood that the state is reached, resulting in a potentially misleading choice of state.

(C) Impact using anterior and posterior occupancy frequencies: $s^* = \text{argmax}_s \mathbb{E}_{s_0 \in S_0} [I_{s,\pi_0,\pi_1}(s_0)]$. Our proposed approach evaluates the difference resulting from following π_0 until s is reached, and then following pi_1 after that point. Although marginally more computationally complex than the prior two methods, the main advantage of this impact measure is that it accounts the overall value difference on the start state, and not just what happens when starting at the state. In this sense, it is a much better choice as the answer to the *why* question of how changing policies impacts the results.

4.2 GridWorld

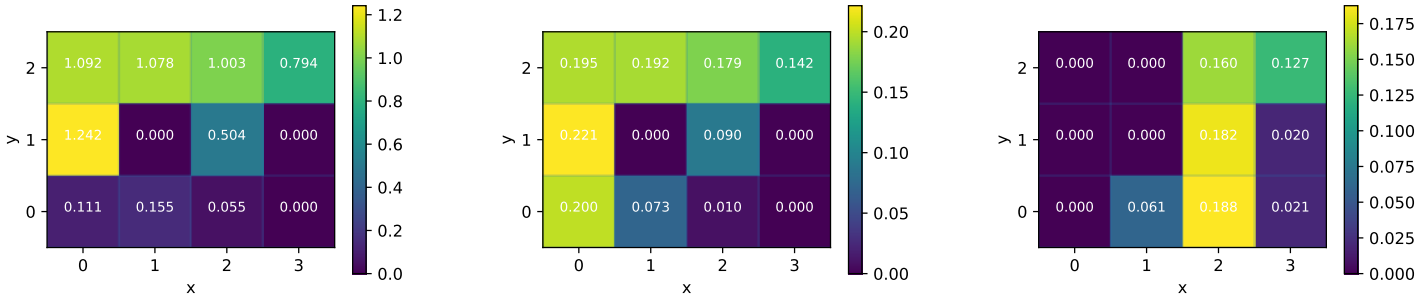
The GridWorld is fully observable and has four actions, *Up*, *Down*, *Left*, and *Right*. For each action, an agent will go to an intended state with the probability 0.8 and will move at the right angles to the original direction for the rest. Taking each step at non-terminal states, the agent receives the reward -0.04 and at two terminal states, the goal and lava, the ,mkj either $+1$ or -1 respectively as in Fig. 1 (c).

The polices for the comparison are depicted in Fig. 1 (a) and (b). The optimal policy is generated from the value iteration method and the shorted path policy is generated to minimize the total number of steps before reaching the goal state with the reward $+1$. The occupancy frequencies L and M can also be generated using the similar value iteration method and the calculated L and M for the state $(1, 0)$ are depicted in Fig. 2.

The results on the three metrics is in Fig. 3. The state $(2, 1)$ is chosen for the metric (A) and (B). For (A) we can explain the result as

> If an agent starts at $(2, 1)$ with two different policies, the difference of two expected returns will be 0.066.

For (B), the generated explanations are



(a) (b) (c)

Figure 3: The anterior and posterior occupancy frequencies of the state $s^* = (1, 0)$ (a) $L_{s^*}^{s_0,\pi_0}(s)$. (b) $M_{s^*}^{s_0,\pi_0,\pi_1}(s)$. (c) $M_{s^*}^{s_0,\pi_0,\pi_1}(s)$.

> If an agent starts at $(2, 1)$ with two different policies, if an agent takes the optimal action *Left* it will receive more reward by 0.081 compared to taking an action *Up* which is from the alternative shortest path policy.

The proposed *Impact* metric has chosen a different state $(1, 0)$. We can generate the below explanations with our occupancy frequencies and factoring states by [2].

> At the state $(1, 0)$, if an agent switches from the optimal policy π_0 to the shortest path policy π_1, the overall value of a start state $(0, 0)$ will differ by 0.029 which is most compared to any other states. Before reaching a state $(1, 0)$, the goal state $(3, 2)$ will be reached by 0.794 times. If an agent continues applying the optimal policy, it will reach the goal state $(3, 2)$ 0.142 times which is 0.015 times more than switching to the shortest path policy. If an agent switches to shortest path policy it will reach the lava state $(3, 1)$ 0.20 times more compare to 0.0 times when the policy is unchanged.

5 Conclusion

We presented the framework for choosing a state that has the most impact in overall value and provide detailed explanations based the calculated anterior and posterior occupancies. This also gives the power to reason counterfactually since we can assume the same precondition and reason about the outcome based on the decision a designer can make by continuing or switching to a different policy. In our future work, we would like to expand this framework to more domains and possibly to deep RL domains.

References

[1] Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, 3389–3396. IEEE.

[2] Khan, O.; Poupart, P.; and Black, J. 2009. Minimal sufficient explanations for factored markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 19, 194–200.

[3] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529.

[4] Russell, S.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

[5] Sequeira, P.; Yeh, E.; and Gervasio, M. T. 2019. Interestingness Elements for Explainable Reinforcement Learning through Introspection. In *IUI workshops*, volume 1.

[6] Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Confidently conflicted: The impact of value confidence on choice varies with choice context

Joonhwa Kim
Department of Cognitive, Linguistic, and Psychological Sciences
Brown University
Providence, RI 02912
joonhwa_kim@brown.edu

Romy Frömer
Department of Cognitive, Linguistic, and Psychological Sciences
Brown University
Providence, RI 02912
romy_fromer@brown.edu

Xiamin Leng
Department of Cognitive, Linguistic, and Psychological Sciences
Brown University
Providence, RI 02912
xiamin_leng@brown.edu

Amitai Shenhav
Department of Cognitive, Linguistic, and Psychological Sciences
Brown University
Providence, RI 02912
amitai_shenhav@brown.edu

Abstract

How people choose among a set of options is affected both by how they evaluate each option, and how they perceive the competition among those options. For instance, separate lines of work have shown that people weigh their options differently depending on (a) how confident they are in their valuation of each, and (b) whether or not selecting one option from a set excludes the possibility of selecting others. It remains unclear whether and how these two factors interact in shaping not only choices but also how difficult it feels to make a choice. To examine this interaction, we compared typical exclusive choices to non-exclusive choices, in which participants can choose additional items from the set after their initial choice. We tested how the value a person assigned to each option interacted with their confidence in those values to shape initial choices, subsequent choices, and experiences of choice conflict. When participants were required to choose one option from a set, we found that they were more likely to choose a low-value option that they had low confidence in than one they had high confidence in, and vice versa for high-value options. However, when participants had the flexibility to continue choosing additional items or not, we found that this effect was either absent or even reversed. We also replicated previous findings that participants experience the most conflict when choosing among the most and the least valuable options, but showed that this U-shaped effect was attenuated with lower levels of confidence in one's value estimates. Our work sheds new light on mechanisms of decision-making by highlighting that the impact of value confidence on choices critically depends on whether an option needs to be chosen at all. By adding nuance to previous findings our results provide a starting point for better understanding the mechanisms underlying value-based decisions, and what makes some choices harder than others.

Keywords: value-based decision-making, value confidence, choice conflict, choice exclusivity

1 Introduction

As early as breakfast time, we need to weigh our options and choose among them. These choices are guided by how much we value each of our options (e.g., scone vs. croissant), information that accumulates for the respective options and competes to determine our ultimate response. However, this evidence accumulation process can be modulated both by properties of value estimation and of the choice context itself.

For instance, recent work has shown that a person's confidence in the values of each of their options (e.g., based on more consistent or mixed experiences with a given breakfast item) influences how they choose between those options. When people are overall more confident in their estimates of option values, their choices are faster and more accurate/consistent [1, 2, 3]. These findings have been accounted for by Bayesian value estimation, where lower precision samples lead to less updating and the momentary value estimate is consequently shrunk towards one's prior (e.g., the average value of options in one's environment) [4].

A separate line of work has shown that how we accumulate value-related evidence is influenced by our perception of the level of competition between our options. When selecting our favorite option precludes the choice of other options (as if selecting from a breakfast menu; *exclusive choices*), we are slower and experience greater choice conflict than when we are allowed to subsequently choose additional options (as if selecting from a breakfast buffet; *non-exclusive choices*)[5]. These results have been accounted for with choice models that vary in the level of competition (e.g., mutual inhibition) across options during non-exclusive compared to exclusive choice [5, 6].

Important questions remain unanswered at the intersection of these two lines of work. For instance, how does the influence of value confidence on choice differ when an initial choice excludes or doesn't exclude additional choices? How does it further influence choices of whether to select additional options, and how many? How do all of these factors collectively influence the subjective experience of conflict during a decision? To address these questions, we had participants make decisions that were either exclusive (requiring only one choice) or non-exclusive (requiring at least one choice, but allowing for additional selections from the same set) (Fig. 1). We measured how confident participants were in each of their option values, and tested how this value confidence impacted choice behavior, and perceived choice conflict during initial and subsequent choices. We hypothesized that low confidence during subsequent choices (where multiple options *can* be chosen, but no option *needs* to be chosen), would produce less down-weighting of high-value items and less up-weighting of low-value items. We verify this and other hypotheses across two studies, showing that the interaction of confidence and value depends on one's choice context.

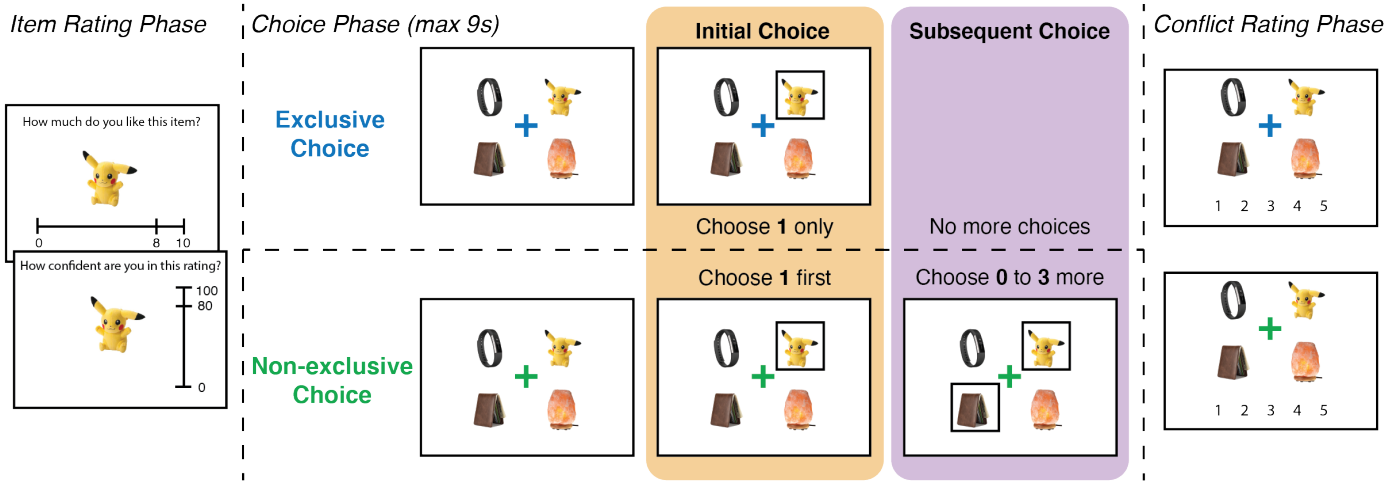


Figure 1. Task paradigm. Participants rate how much they like items and their confidence in these value ratings. Participants then choose among sets of 4 options, each. On exclusive choice trials, the trial ends post-initial choice. On non-exclusive choice trials, participants are allowed to select as many additional products as they like. Finally, participants rate the level of conflict they experienced during each choice.

2 Method

Participants across two studies (Study 1; N = 56, 25 males, 31 females, age = 36.5 ± 18.5 ys, Study 2; N = 77, 30 males, 47 females, age = 34.4 ± 20.6 ys) performed an experiment consisting of three phases: 1) item rating, 2) choice, and 3) conflict rating (Fig. 1). During item ratings (200 items total), participants were asked to rate how much they liked a consumer item on a scale from 0 (not at all) to 10 (a great deal), followed by a prompt to rate how confident they were

in this rating on a scale from 0 (not confident at all) to 100 (absolutely confident). Based on the individual ratings, we constructed personalized choice sets that varied in the relative and overall values of options (120 sets total). In the choice phase, participants made either *exclusive* (menu-type), or *non-exclusive* (buffet-type) choices among sets of 4 options. On *exclusive* choice trials, participants were allowed to choose only one product from the choice set. Once they clicked on this product, a box appeared around it and they proceeded to the next trial. On *non-exclusive* choice trials, participants were able to continue selecting as many options as they preferred after they chose the most preferred item first. The choice types were intermixed, occurred with equal likelihood, and were color-coded by a fixation cross in the center. Participants had 9s to complete their choice(s) for a given option set. Following the choice phase, participants were then asked to rate their subjective experience of conflict during each choice on a 5 point Likert-scale.

3 Results

3.1 The influence of value confidence and choice exclusivity on initial choices.

To test how confidence and choice exclusivity jointly impact choice behavior, we analyzed initial choices and response times as a function of choice condition, overall value, value difference, set confidence (mean confidence in all the items' values for a given set), and confidence in the chosen option, as within subject regressors, while controlling for confidence bias (a given participant's average level of confidence across all rated items).

Consistent with previous findings from separate lines of work [3, 5, 7], participants were faster (S1: $b = -0.01, p < .001$, S2: $b = -0.01, p < .001$) and more consistent (S1: $b = .04, p = .001$, S2: $b = 0.09, p < .001$) the higher the set confidence [Fig. 2; [3, 7]], and slower making exclusive compared to nonexclusive choices (S1: $b = -0.02, p < .001$; S2: $b = -0.02, p < .001$), in line with greater mutual inhibition in exclusive choice [5]. We found no reliable interactions between confidence and context ($ps > .05$). This suggests that confidence plays similar roles during initial choice (when one item has to be chosen), irrespective of choice exclusivity.

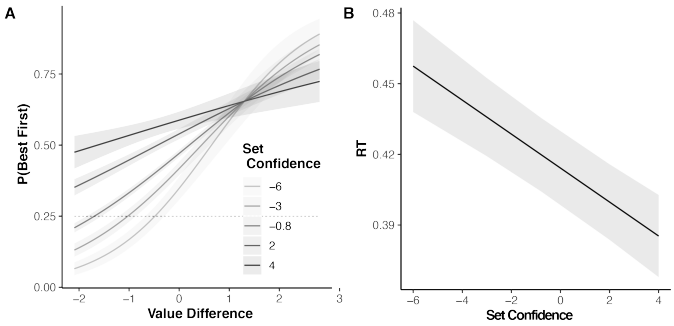


Figure 2. Set confidence effects on initial choice. (A) The probability of choosing the best item increases with increasing Value difference and for higher set confidence. **(B)** Reaction time decreases with higher set confidence.

3.2 The influence of confidence on continued option selection in non-exclusive choices.

Our task allows us to investigate the impact of choice context beyond these initial choices, for instance how confidence and value impact how many additional items were chosen in the nonexclusive context. As expected, participants selected more additional items as the overall value of the set increased (S1: $b = 0.63, p < .001$, S2: $b = 0.67, p < .001$; Fig. 3A), and fewer additional items the greater the difference in value between the first item and the rest of the set (S1: $b = -0.05, p = .001$, S2: $b = -0.08, p < .001$; Fig. 3B) [5]. Importantly, this value difference effect, reflecting that the only good item had already been selected, was amplified when participants were more confident in their option set values (S1: $b = -0.04, p = .003$, S2: $b = -0.02, p = .036$; Fig. 3B) - when set confidence was low, participants were less sensitive to value difference when selecting additional options. In Study 2, but not Study 1, we found that higher set confidence also led people to choose more additional options overall (S2: $b = .06, p < .001$, S1: $b = 0.02, p = .086$), and that it amplified the positive influence of overall value on option selection (S2: $b = 0.03, p = .011$, S1: $b = -.01, p = .552$; Fig. 3A).

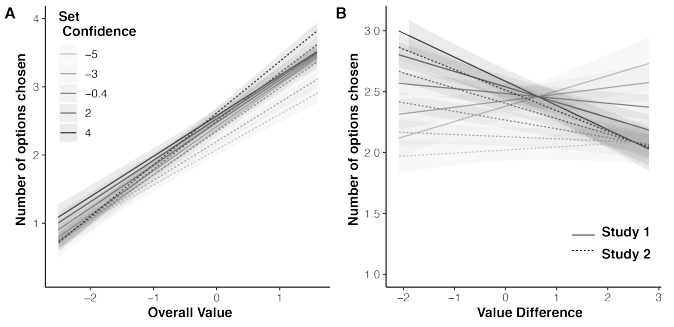


Figure 3. Set confidence effects on number of items chosen. (A) The number of items chosen increases with overall value, and more so for higher set confidence in Study 2. **(B)** Number of options chosen decreases with increasing value difference and more consistently so for higher set confidence.

To directly compare the decision process for initial relative to subsequent choices, we examined value and confidence in an item-wise manner (vs. at the level of the set) to see how these predict the likelihood of choosing a given item initially or subsequently. As expected, across both types of choice, participants were more likely to choose more valuable items (P(chosen first): S1: $b = 0.48, p < .001$, S2: $b = 0.45, p < .001$; P(chosen subsequent): S1: $b = 0.78, p < .001$, S2: $b = 0.85, p < .001$; Fig. 4) [5]. When choices were obligatory (initial choice), the influence of an item's value was diminished or amplified according to one's confidence in that value estimate (S1: $b = 0.08, p < .001$, S2: $b = 0.13, p < .001$; Fig. 4A). Participants treated low-confidence low-value items as more valuable and low-confidence high-value items as less valuable on average, consistent with Bayesian value estimation accounts [4, 7].

However, this pattern was qualitatively different when additional selection was voluntary (subsequent choices). In these cases, the effect of an item's value on choice was flatter when confidence was high than when it was low (S1: $b = -0.05, p = .121$, S2: $b = -0.11, p = .001$; Fig. 4B). In other words, participants in this context refrained from choosing low-confidence, low-value items, but were similarly if not more inclined to choose low-confidence, high-value items. This inverted confidence effect suggests that when allowed to forego all items, people no longer treat low-confidence, low-value items as more valuable, and low-confidence, high-value items as less valuable. Thus in initial choices, low-confidence's benefits for low value items and costs for high value items might be shaped by the dreaded alternative of selecting certainly bad options or forgoing potentially better options.

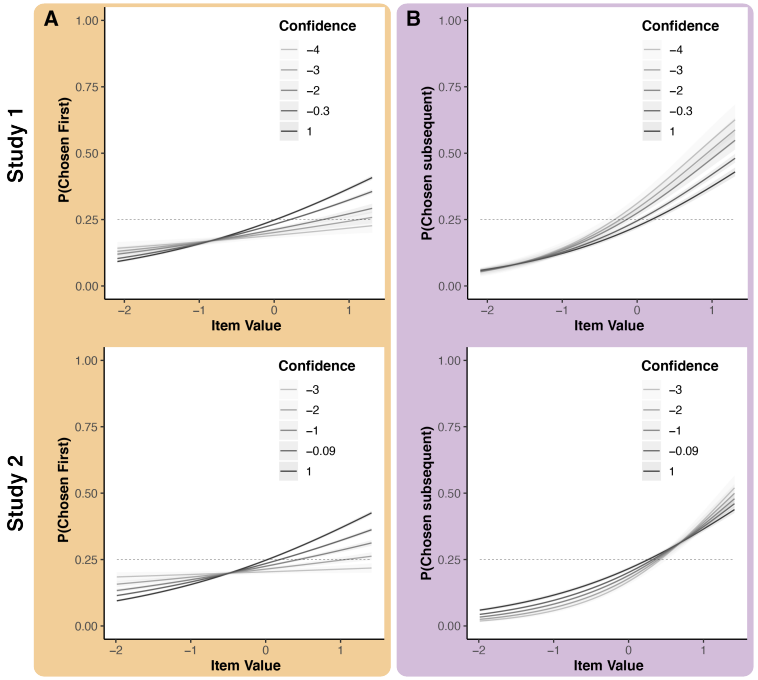


Figure 4. Confidence effects on initial vs. subsequent choice. (A) Lower value confidence reduces value effects on initial choice. **(B)** Unlike in initial choices, in subsequent choices value effect on choice are not reduced for lower value confidence.

3.3 The influence of confidence and choice exclusivity on experienced choice conflict.

As in previous studies [8, 9], participants reported experiencing the greatest conflict when choosing among especially high-value or especially low-value options (Linear; S1: $b = 16.46, p < .001$, S2: $b = -7.14, p < .001$, Quadratic; S1: $b = 16.01, p < .001$, S2: $b = 7.11, p < .001$). This finding was hypothesized to reflect reference-dependent valuation, resulting in avoid-avoid conflict when choosing among low value options and approach-approach conflict when choosing among high value options [9]. We found that this effect is modulated by set confidence, such that the U-shaped effect is strongest at higher levels of set confidence, and flattens with lower set confidence (S1: $b = 3.37, p = .008$, S2: $b = 4.13, p = .001$; Fig. 5).

This could reflect weaker avoid-avoid and approach-approach conflict when choosing among higher and lower values options, respectively [9], perhaps due to regression to the mean for lower confidence, as predicted by Bayesian value estimation. Participants also experienced less conflict overall when making non-exclusive relative to exclusive choices (S1: $b = -0.18, p < .001$, S2: $b =$

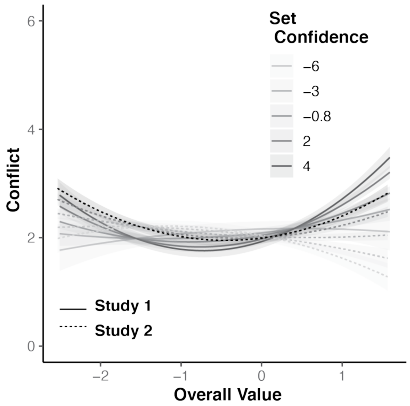


Figure 5. Experienced conflict in decision making. The U-shaped value effect (higher conflict experienced for more extreme overall value) is reduced for lower set confidence.

$-0.12, p < .001$; cf. [5]), but this did not robustly interact with value confidence across both studies (S1: $b = 0.05, p = .003$, S2: $b = -0.02, p = .101$).

4 Conclusion

We demonstrated that value confidence not only affects our choices but also the conflict we experience when making them. Notably, we show that the way in which value confidence affects choice depends on whether at least one item must be chosen or not. Taken together, these findings provide a crucial stepping stone for delving further into the specific mechanisms underlying choice dynamics and how they relate to the subjective experience of choice conflict. In doing so, we may identify strategies to make hard choices easier and feel better about making them.

References

[1] Rafael Polanía, Michael Woodford, and Christian C. Ruff. Efficient coding of subjective value. *Nature Neuroscience*, 22(1):134–142, 2019.

[2] M. Lebreton, R. Abitbol, J. Daunizeau, and M. Pessiglione. Automatic integration of confidence in the brain valuation signal. *Nat Neurosci*, 18(8):1159–67, 2015.

[3] Douglas G. Lee and Marius Usher. Value certainty in drift-diffusion models of preferential choice. *Psychological Review*, pages No Pagination Specified–No Pagination Specified, 2021.

[4] F. Callaway, A. Rangel, and T. L. Griffiths. Fixation patterns in simple choice reflect optimal information sampling. *PLoS Comput Biol*, 17(3):e1008863, 2021.

[5] X. Leng, R. Frömer, T. Summe, and A. Shenhav. A theoretical and experimental investigation of the role of mutual inhibition in shaping choice. in prep.

[6] R. Bogacz. Optimal decision-making theories: linking neurobiology with behaviour. *Trends Cogn Sci*, 11(3):118–25, 2007.

[7] R. Frömer, F. Callaway, T. Griffiths, and A. Shenhav. Considering what we know and what we don't know: Expectations and confidence guide value integration in value-based decision-making. in prep.

[8] A. Shenhav and R. L. Buckner. Neural correlates of dueling affective reactions to win–win choices. *Proceedings of the National Academy of Sciences*, 111(30):10978–10983, 2014.

[9] A. Shenhav, C. K. Dean Wolf, and U. R. Karmarkar. The evil of banality: When choosing between the mundane feels like choosing between the worst. *J Exp Psychol Gen*, 147(12):1892–1904, 2018.

Deep Conservative Reinforcement Learning for Personalization of Mechanical Ventilation Treatment

Flemming Kondrup*
McGill University
Montreal, Canada
flemming.kondrup@mail.mcgill.ca

Thomas Jiralerspong*
McGill University
Montreal, Canada
thomas.jiralerspong@mail.mcgill.ca

Elaine Lau*
McGill University
Montreal, Canada
tsoi.lau@mail.mcgill.ca

Nathan de Lara
McGill University
Montreal, Canada
nathan.delara@mail.mcgill.ca

Jacob Shkrob
McGill University
Montreal, Canada
jacob.shkrob@mail.mcgill.ca

My Duc Tran
McGill University
Montreal, Canada
my.d.tran@mail.mcgill.ca

Doina Precup
Mila, McGill University, DeepMind
Montreal, Canada
dprecup@cs.mcgill.ca

Sumana Basu
Mila, McGill University
Montreal, Canada
sumana.basu@mail.mcgill.ca

Abstract

Mechanical ventilation is a key form of life support for patients with pulmonary impairment. An important challenge faced by physicians is the difficulty of personalizing treatment and thus to offer the best ventilation settings for each patient. This leads to sub-optimal care which further leads to complications such as permanent lung injury, diaphragm dysfunction, pneumonia and potentially death. It is therefore essential to develop a decision support tool to optimize and personalize ventilation treatment.

We present DeepVent, the first deep reinforcement learning model to address ventilation settings optimization. Given a patient, DeepVent learns to predict the optimal values for the ventilator parameters Adjusted Tidal Volume (V_t), FiO_2 (Fraction of inspired O_2) and PEEP (Positive End-Expiratory Pressure) with the final objective of promoting 90 day survival. We use the MIMIC-III dataset, comprised of 19,780 patients under ventilation. We show that our use of Conservative Q-Learning addresses the challenge of overestimation of the values of out-of-distribution states/actions and that it leads to recommendations within safe ranges, as outlined in recent clinical trials. We evaluate our model using Fitted Q Evaluation, and show that it is predicted to outperform physicians. Furthermore, we design a clinically relevant intermediate reward to address the challenge of sparse reward. Specifically, we employ the Apache II score, a widely used score by physicians to assess the severity of a patient's condition, and show that it leads to improved performance.

Keywords: Reinforcement Learning, Conservative Q-Learning, Mechanical Ventilation, Personalized Healthcare

We would like to thank Bogdan Mazoure (Mila) and Eyal de Lara (University of Toronto) for reviewing the paper and sharing constructive feedback, Adam Oberman (Mila) for their advice on the intermediate reward and Andrew Bogecho (McGill) for helping us getting access to the McGill RL Lab compute.

*These authors contributed equally

1 Introduction

The COVID-19 pandemic has put enormous pressure on the healthcare system, particularly on intensive care units (ICUs). In cases of severe pulmonary impairment, mechanical ventilation assists breathing in patients and acts as the key form of life support. However, the optimal ventilator settings is individual specific and often unknown [1], leading to ventilator induced lung injury (VILI), diaphragm dysfunction, pneumonia and oxygen toxicity [2]. Previous work has approached ventilation optimization with RL using a tabular approach [3]. Our work makes three key contributions:

- We propose the first deep reinforcement learning approach to personalize mechanical ventilation settings
- We demonstrate the potential of Conservative Q-Learning [4], a recently proposed offline deep reinforcement learning algorithm, to address overestimation of the values of out-of-distribution states/actions, which is very important in a healthcare context, where data is limited and risk in decision making must be avoided
- We introduce an intermediate reward based on the Apache II mortality prediction score [5] to address the challenge of sparse reward

We compare DeepVent's decisions to those of physicians, as recorded in an existing standard dataset, as well as to those of an agent trained with Double Deep Q-Learning (DDQN) [6]. DeepVent is predicted to outperform physicians while avoiding the overestimation problems of DDQN, thus making safe recommendations.

2 Preliminaries

2.1 Double Deep Q-Networks (DDQN)

Overestimation occurs when the estimated value of a random variable is higher than its true value. Deep Q-Networks are known to overestimate the values of unseen state-action pairs. DDQNs were introduced as a solution by modifying the calculation of the target value [6]. At any point in time, one of DDQN's networks, chosen at random, is updated, by using as target the estimate from the other network. Although this partially solves overestimation, DDQNs can still suffer from it [6], particularly in offline RL where exploration is limited to the dataset used in training. This can lead to important overestimation in state-action pairs underrepresented in the dataset, or out-of-distribution (OOD), leading to sub-optimal action choices [4] which may translate to unsafe recommendations, putting patients at risk.

2.2 Conservative Q-Learning (CQL)

To address the challenge of overestimation in an offline setting, Conservative Q-Learning (CQL) was proposed [4]. It learns a conservative estimate on the Q-function by incorporating a regularizer $E_{\mathbf{s_t}\sim\mathcal{D},\mathbf{a_t}\sim A}[Q(\mathbf{s_t},\mathbf{a_t})]$ on top of the standard TD error to minimize the overestimated Q-values of unseen actions. In addition, the term $-E_{\mathbf{s_t},\mathbf{a_t}\sim\mathcal{D}}[Q(\mathbf{s_t},\mathbf{a_t})]$ is added to maximize the Q-values in the dataset, providing a lower bound in expectation of the policy. CQL minimizes the estimated Q-values for all actions while simultaneously maximizing the estimated Q-values for the actions appearing in the dataset. This prevents overestimation of OOD state-action pairs which are underrepresented in the dataset.

3 Datasets

We used the MIMIC-III database [7] containing data of 61,532 ICU stays at the Beth Israel Deaconess Medical Center. Standardized Query Language (SQL) was used to extract data for a total of 19,780 patients under ventilation. For features with $<$ 30% of the data was missing, KNN imputation was used [8]. If 30% to 95% of the data was missing, time-windowed sample-and-hold was used [8]. If $>$ 95% was missing, the variable was removed. An out-of-distribution (OOD) set of outlier patients was created with patients having at least one feature in the top or bottom 1% of the distribution at the start of their ICU stay.

4 Proposed Approach

Our MDP was defined similarly to [3], with episodes lasting from the time of the patient's intubation to 72 hours after.

State Space The state space $\mathcal{S}$ is composed of 37 variables[1]:

- Demographics: Age, gender, weight, readmission to the ICU, Elixhauser score
- Vital Signs: SOFA, SIRS, GCS, heart rate, sysBP, diaBP, meanBP, shock index, respiratory rate, temperature, spO2

[1]For variable definitions refer to the MIMIC-III paper [7] 316

- Lab Values: Potassium, sodium, chloride, glucose, bun, creatinine, magnesium, carbon dioxide, Hb, WBC count, platelet count, ptt, pt, inr, pH, partial pressure of carbon dioxide, base excess, bicarbonate
- Fluids: Urine output, vasopressors, intravenous fluids, cumulative fluid balance

Action Space The 3 ventilator settings of interest are:

- Adjusted tidal volume *or* Vt (Volume of air in and out of the lungs with each breath adjusted by ideal weight)
- PEEP (Positive End Expiratory Pressure)
- FiO2 (Fraction of inspired oxygen)

The action space $\mathcal{A}$ is the Cartesian product of the set of these three settings. Each setting can take one of seven values corresponding to ranges. We can therefore represent an action as the tuple $a = (v, o, p)$ with $v \in Vt, o \in FiO_2, p \in PEEP$.

Reward Function The main objective of our agent is to keep a patient alive in the long-term. Therefore, even if DeepVent only treats patients for 72 hours, it learns how to maximize their 90 day survival. We thus define a terminal reward $r(s_t, a_t, s_{t+1})$, which takes at the final state the value -1 if the patient passes away within 90 days and $+1$ otherwise.

The sole use of a sparse terminal reward is known to cause poor performance [9] in RL tasks. We therefore developed an intermediate reward based on the Apache II score [5], a widely used score in ICUs to assess the severity of a patient's disease. Apache II takes various physiological variables such as temperature, blood pressure etc. as input and returns a score based on how far each variable is from the healthy range. A higher score is associated with variables being far from the normal range, and thus a more severe state. The score was adapted to the variables present in MIMIC-III. In order to not simply define reward based on how well a patient was doing but rather their evolution through time, our intermediate reward consists of the change in Apache II score between s_{t+1} and s_t, which is normalized by dividing it by the total range of the score. Combining our intermediate and terminal rewards, we obtain our final reward function:

$$r(s_t^i, a_t^i, s_{t+1}^i) = \begin{cases} +1 & \text{if } t+1 = l_i \text{ and } m_{t+1}^i = 1 \\ -1 & \text{if } t+1 = l_i \text{ and } m_{t+1}^i = 0 \\ \frac{(A_{t+1}^i - A_t^i)}{\max_A - \min_A} & \text{otherwise} \end{cases}$$

where:
$\mathbf{A}_t^i$ is the modified Apache II score of patient i at t
$\mathbf{m}_t^i$ = 0 if patient i is dead at 90 days and 1 otherwise
$\mathbf{l}_i$ is the length of patient i's stay at the ICU
$\mathbf{max}_A, \mathbf{min}_A$ are the max. and min. values of our modified Apache II score

Off-Policy Evaluation (OPE) The performance of various OPE methods was recently evaluated in healthcare [10], and Fitted Q Evaluation (FQE) was found to consistently provide the most accurate results. We thus use FQE [11], which takes as input a dataset D and a policy π, and outputs a value estimate for each state in D, corresponding to an approximation of the cumulative discounted reward received by following a policy π starting at a given state. It is important to note that FQE outputs an approximation of true performance and not its exact value. Clinical trials would thus be required to confirm the results in section 5.1.

5 Results & Discussion

5.1 DeepVent Overall Performance

To begin, we compare the performance of DeepVent- (CQL without intermediate reward), DeepVent (CQL with intermediate reward), and the physician when applied to the patients in our test set (see Table 1).

Table 1: Mean initial state value estimates for physician, DeepVent- and DeepVent, with std. errors. DeepVent- significantly outperforms the physician. Adding the Apache II intermediate reward (DeepVent) further improves the estimate.

PHYSICIAN	DEEPVENT-	DEEPVENT
$\mathbf{0.502} \pm 0.00709$	$\mathbf{0.762} \pm 0.00402$	$\mathbf{0.797} \pm 0.00670$

The initial state of an episode represents the state of a given patient when ventilation is initiated. The performance of DeepVent- or DeepVent can be approximated by the value estimation output by FQE for the initial state of a patient. Although DeepVent was trained with intermediate rewards, FQE's value estimation only depends on the dataset $\mathcal{D}$ and the actions chosen by the policy π used to train FQE. Because we trained FQE using the dataset without intermediate rewards for both DeepVent- and DeepVent, the estimates are solely based on the terminal reward and can thus be used as a fair comparison between the two models. Since the physician policy effectively generates the episodes in our dataset, its value estimates for each initial state can be computed by taking the cumulative discounted reward for the episode starting at that state. We observe that DeepVent outperforms physicians by a factor of 1.52. The addition of the intermediate reward increases this factor to 1.59.

5.2 DeepVent and Safe Recommendations

We next evaluate DeepVent's action distributions and compare it with DDQN and the physician (see Figure 1).

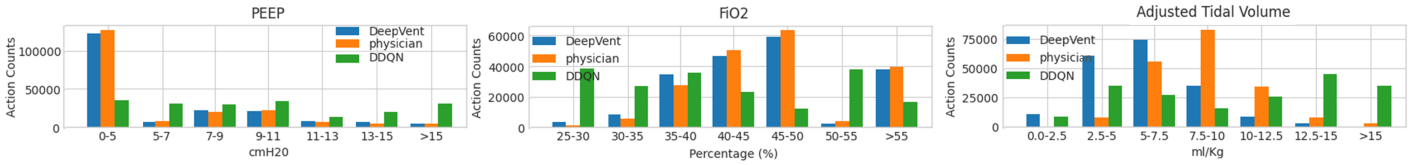


Figure 1: Distribution of actions across ventilator settings. Unlike DDQN, DeepVent makes recommendations in safe and clinically relevant ranges for each setting.

DeepVent was observed to suggest safe setting recommendations. The standard of care in terms of PEEP setting is commonly initiated at 5 cmH2O [12] which is supported by the high number of recommendations by physicians being in the range of 0-5 cmH2O in our dataset. DeepVent spontaneously chose to adopt this strategy by making most recommendations in the range of 0-5 cmH2O. In contrast, DDQN chose settings distributed along all the options, ranging up to 15 cmH2O, where physicians rarely went. High PEEP settings have been associated with higher incidence of pneumothorax [13], inflammation [14] and impaired hemodynamics [15], and should therefore be avoided.

In terms of FiO2 setting, DeepVent was once again found to follow clinical standards of care. More specifically, we observe that DeepVent often chose actions in the same ranges as physicians in our dataset, with many recommendations in the ranges of 35-50% and >55% and few recommendations below 35% and between 50-55%. In contrast, DDQN made few recommendations in ranges often suggested by physicians, and many in those that were rarely employed.

Finally, for the adjusted tidal volume, the optimal tidal volume is usually found in the 4-8 ml/kg range [16, 17]. DeepVent made a majority of recommendations in the range of 2.5-7.5 ml/kg, with an important amount of these being concentrated in the 5-7.5 ml/kg range. In contrast, DDQN made many recommendations in higher ranges, often even going above 15 ml/kg, a range rarely observed in clinical practice and associated with increased lung injury and mortality [18].

5.3 DeepVent in Out-Of-Distribution Samples

We next investigated whether the sub-optimal recommendations made by DDQN might be caused by value overestimation. To do so, we investigated the mean initial values for DeepVent and DDQN (as estimated by FQE). It is interesting to not only understand how well the model performs on data similar to that on which it was trained, but also on outlier data. We thus consider both an in-distribution (ID) and an out-of-distribution (OOD) setting (see Figure 2).

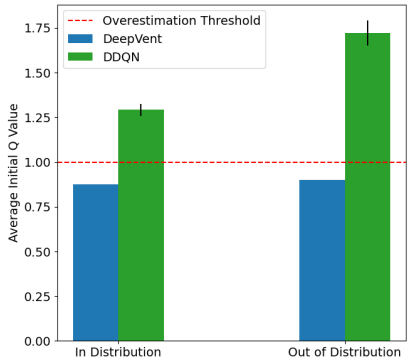


Figure 2: Mean initial Q-values for both in and out of distribution settings for DeepVent and DDQN (with variances - DeepVent's variance is not visible because of its small value). The horizontal line is the maximum expected return per episode. In contrast to DeepVent, DDQN clearly suffers from overestimation, which is aggravated in the OOD setting

Since the maximal expected return for an episode in our dataset is set at 1, values above this threshold should be considered as overestimated. We observe that DDQN overestimates policy values in both the ID and OOD settings. In addition, DDQN's overestimation is exacerbated in the OOD setting. This failure to accurately assess these OOD states may be the cause of the unsafe recommendations discussed above. DeepVent seems to avoid these problems, as its average initial state value estimate stays below the overestimation threshold of 1 in both settings, and barely changes in OOD.

6 Conclusion

In this work, we developed DeepVent, a decision support tool for personalizing mechanical ventilation treatment using offline deep reinforcement learning. We showed that our use of Conservative Q-Learning leads to settings in clinically relevant and safe ranges by addressing the problem of overestimation of the values of out-of-distribution state-action pairs. Furthermore, we showed using FQE that DeepVent achieves a higher estimated performance when compared to physicians, which can be further improved by implementing our Apache II based intermediate reward. We conclude that DeepVent intuitively learns to pick actions that a physician would agree with, while using its capacity to overview vast amounts of clinical data at once and understand the long-term consequences of its actions to improve outcomes for patients. Moreover, the fact that DeepVent is associated with low overestimation in out-of-distribution settings makes it much more reliable, and thus closes the gap between research and real-world implementation. Future work should aim to investigate the potential of the DeepVent methodology in other healthcare applications.

References

[1] H. Zein, A. Baratloo, A. Negida, and S. Safari. Ventilator Weaning and Spontaneous Breathing Trials; an Educational Review. *Emerg (Tehran)*, 4(2):65–71, 2016.

[2] T. Pham, L. J. Brochard, and A. S. Slutsky. Mechanical Ventilation: State of the Art. *Mayo Clin Proc*, 92(9):1382–1400, 09 2017.

[3] A. Peine, A. Hallawa, J. Bickenbach, G. Dartmann, L. B. Fazlic, A. Schmeink, G. Ascheid, C. Thiemermann, A. Schuppert, R. Kindle, L. Celi, G. Marx, and L. Martin. Development and validation of a reinforcement learning algorithm to dynamically optimize mechanical ventilation in critical care. *NPJ Digit Med*, 4(1):32, Feb 2021.

[4] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning, 2020.

[5] W.A. Knaus, E.A. Draper, D.P. Wagner, and J.E. Zimmerman. APACHE II: a severity of disease classification system. *Crit Care Med.*, 1985.

[6] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015.

[7] A. E. W. Johnson, T. J. Pollard, L. Shen, L. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 2016.

[8] C.M. Salgado, C. Azevedo, H. Proença, and S.M. Vieira. Missing data. in: Secondary analysis of electronic health records. *Springer, Cham.*, 2016.

[9] Maja J. Mataric. Reward functions for accelerated learning. 1994.

[10] S. Tang and J. Wiens. Model selection for offline reinforcement learning: Practical considerations for healthcare settings, 2021.

[11] Michita Imai Takuma Seno. d3rlpy: An offline deep reinforcement library. In *NeurIPS 2021 Offline Reinforcement Learning Workshop*, December 2021.

[12] GF Nieman, J Satalin, P Andrews, H Aiash, NM Habashi, and LA Gatto. Personalizing mechanical ventilation according to physiologic parameters to stabilize alveoli and minimize ventilator induced lung injury (vili). *Intensive Care Med Exp.*, 2017.

[13] J Zhou, Z Lin, X Deng, B Liu, Y Zhang, Y Zheng, H Zheng, Y Wang, Y Lai, W Huang, and X Liu et al. Optimal Positive End Expiratory Pressure Levels in Ventilated Patients Without Acute Respiratory Distress Syndrome: A Bayesian Network Meta-Analysis and Systematic Review of Randomized Controlled Trials. *Front. Med.*, 2021.

[14] A. Güldner, A. Braune, L. Ball, P. L Silva, C. Samary, A. Insorsi, R. Huhle, I. Rentzsch, C. Becker, L. Oehme, M. Andreeff, M. F Vidal Melo, T. Winkler, and P. Pelosi et al. Comparative Effects of Volutrauma and Atelectrauma on Lung Inflammation in Experimental Acute Respiratory Distress Syndrome. *Critical care medicine*, 2016.

[15] S. N. PROVE Network Investigators for the Clinical Trial Network of the European Society of Anaesthesiology, Hemmes, M. Gama de Abreu, P. Pelosi, and M. J. Schultz. High versus low positive end-expiratory pressure during general anaesthesia for open abdominal surgery (PROVHILO trial): a multicentre randomised controlled trial. *Lancet (London, England)*, 2014.

[16] A.M. Luks. Ventilatory strategies and supportive care in acute respiratory distress syndrome. *Influenza and other respiratory viruses, 7 Suppl 3*, 2013.

[17] O. Kilickaya and O. Gajic. Initial ventilator settings for critically ill patients. *Critical care*, 2013.

[18] A. Serpa Neto, S. O. Cardoso, J. A. Manetta, V. G. Pereira, D. C. Espósito, M. Pasqualucci, M. C. Damasceno, and M. J. Schultz. Association between use of lung-protective ventilation with lower tidal volumes and clinical outcomes among patients without acute respiratory distress syndrome: a meta-analysis. *JAMA*, 2012.

Regulating the Deadly Triad with Gradient Regularization

Saurabh Kumar
Department of Computer Science
Stanford University
szk@stanfor.edu

Shi Dong
Department of Electrical Engineering
Stanford University
sdong15@stanford.edu

Benjamin Van Roy
Department of Electrical Engineering
Stanford University
bvr@stanford.edu

Abstract

In reinforcement learning, the *deadly triad* refers to the combination of off-policy learning, function approximation, and bootstrapping. These three components, when used simultaneously in a reinforcement learning algorithm, can lead to training instability. We introduce a regularizer that favors functions with small gradients and by doing so stabilizes training, even in the face of off-policy learning and bootstrapping. We elucidate how and why this regularizer stabilizes training, and our experiments demonstrate that the proposed regularizer can indeed mitigate instability and improve performance.

Keywords: bootstrapping, function approximation, generalization, deadly triad, off-policy learning

Acknowledgements

SK is supported by an NSF Graduate Research Fellowship and the Stanford Knight Hennessy Fellowship.

1 Introduction

Deep reinforcement learning (RL) algorithms have demonstrated promise on a variety of complex domains, including game playing and robotics. The ability of high capacity function approximators such as neural networks to generalize predictions to unseen data has been essential to the success of RL on environments with complex high-dimensional state spaces. Many RL algorithms use temporal difference (TD) methods, which bootstrap a network's predictions to learn value functions. While TD methods are computationally efficient and can be applied with off-policy learning, these approaches often suffer from a phenomenon known as the *deadly triad*.

The deadly triad consists of three components: bootstrapping, function approximation, and off-policy learning. In *bootstrapping*, the immediate reward is added to the discounted value estimate at the subsequent state. *Function approximation* is used when a separate value estimate cannot be individually stored for each state. Finally in *off-policy learning*, an agent learns from policies different from the one it is using to act. Analysis of simple didactic examples establishes that, in the face of the the deadly triad, TD methods can exhibit divergence [1, 5], with value estimates growing unbounded.

A number of approaches have been proposed to prevent divergence, but they are limited to the linear function approximation setting. While divergence has been identified as a critical issue in prior analyses of the deadly triad, this phenomenon is not observed in practice when combining Q-learning with neural networks, and off-policy TD-based algorithms with neural networks have demonstrated empirical success [4, 3, 2]. However, prior work has found that the deadly triad can still lead to training instability, with TD methods producing unrealistic value estimates that may hurt performance [6]. In this work, we develop a mechanism that ameliorates this instability when the function approximation used is a high capacity neural network.

Our approach involves modifying how function approximation is done. To understand how the standard approach to function approximation can fuel instability, consider two sets of states: S_1 which contains states whose value estimates are updated often and S_2 which contains rarely updated states whose value estimates are bootstrapped to states in S_1. When high magnitude incorrect predictions in S_2 are not corrected due to off-policy learning, there is a cycle that emerges. First, bootstrapping reinforces the high magnitude predictions on S_2 to states in S_1. Second, the neural network's generalization from S_1 to S_2 further increases the magnitude of predictions on S_2.

We find that such instability can be addressed by encouraging a neural network to interpolate when between known values and to extrapolate only in a very conservative manner. This reduces the magnitude of predictions that can be produced via more aggressive generalization and thus helps break the second step in the cycle. Concretely, we propose a novel regularizer that penalizes gradients with respect to inputs, favoring neural network parameters that yield functions which vary slowly across the input space. This regularizer can be easily incorporated into TD methods, and we augment Approximate Value Iteration (AVI) in this way. Empirically, we show that enhancing AVI with our regularizer improves training stability on two didactic examples: a four state MDP and GridWorld environment.

2 Markov Decision Processes

We now introduce our Markov decision process (MDP) formulation and notation. Each MDP we consider is identified by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, R is the reward function, $P \in \mathbb{R}_+^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}|}$ is the transition matrix where $P_{sa,s'}$ represents the probability of transitioning to a next state $s' \in \mathcal{S}$ if the current state and action are $s \in \mathcal{S}$ and $a \in \mathcal{A}$, and $\gamma \in [0, 1)$ is the discount factor. A policy π assigns at each state $s \in \mathcal{S}$ a probability $\pi(a|s)$ to each action $a \in \mathcal{A}$. If this policy is executed when at state $s \in \mathcal{S}$, an action is sampled from the probability mass function $\pi(\cdot|s)$. The *action value* of a policy π, $Q^\pi(s, a)$, is the expected sum of discounted rewards received when starting from each state-action pair (s, a) and following π.

3 Gradient Penalty for Improving Stability

In this section, we present our approach for improving training stability in the presence of the deadly triad. We introduce a novel loss function which penalizes the gradient of a neural network's outputs with respect to its inputs. We then introduce this loss function as a regularization term in Approximate Value Iteration. The resulting method, Approximate Value Iteration + Gradient Penalty (AVI + GP), balances (1) learning a value function by bootstrapping with (2) favoring parameters which yield functions that vary slowly across the input space.

3.1 Generalization by Interpolation

The deadly triad can lead to instability due to the coupling of two problems. First, due to off-policy learning, incorrect value targets are repeatedly bootstrapped from infrequently updated states. Second, a neural network's generalization of value predictions across states further reinforces incorrect predictions (see Figures 2 and 3 for an illustration of this phenomenon). We address the second problem by regularizing how a neural network generalizes its value predictions

from frequently updated states to more sparsely updated states. Our approach is based on the following intuition: for a state of the latter type, if between states with known values, generalize via interpolating, and otherwise, extrapolate in a very conservative manner.

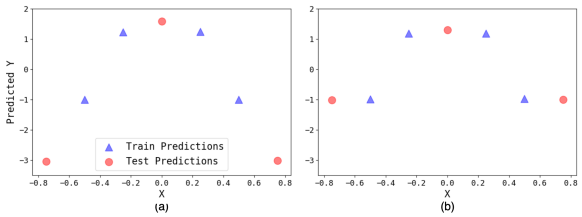


Figure 1: As an illustrative example, consider a supervised learning regression problem. Given the training dataset $\{(-0.5,-1),(-0.25,1),(0.25,1),(0.5,-1)\}$, we train a neural network to predict y given x. (a) In red, we show the network's predictions on the test x values $-0.75, 0.0$, and 0.75 when trained (a) without gradient penalty regularization and (b) with regularization.

In Figure 1, we perform a simple experiment to illustrate how a neural network naturally generalizes to unseen test data in a simple regression problem. The network tends to follow the slope of the training predictions when extrapolating. By taking a more conservative tack, our approach reduces the magnitude of test predictions.

3.2 Penalizing Gradients

Consider a neural network parameterized by a vector θ that outputs a scalar $f_\theta(x)$ for each input $x \in \mathbb{R}^d$. Given a set $\mathcal{D}$ of data pairs, a standard regression loss function takes the form $\mathcal{L}(\theta|\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}}(y - f_\theta(x))^2$. A basic version of gradient descent updates parameters according to

$$\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}(\theta|\mathcal{D}) \tag{1}$$

where α is a learning rate. To induce the sort of generalization we have discussed, we introduce a regularization penalty function

$$\mathcal{R}(\theta|\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}}\|\nabla_x f_\theta(x)\|_2^2.$$

This encourages the use of functions with small gradients, where the gradients are computed with respect to the inputs to the network. With this regularizer, the gradient descent update formula becomes

$$\theta \leftarrow \theta - \alpha\nabla_\theta(\mathcal{L}(\theta|\mathcal{D}) + \beta\mathcal{R}(\theta|\mathcal{D})). \tag{2}$$

where β is a hyperparameter that controls the degree of regularization.

We used the gradient descent updates 1 and 2 to produce the results in Figure 1 (a) and (b), respectively. Both versions produce correct predictions on the training dataset but generalize to the test data differently. By penalizing the squared norm of the gradient, the network's prediction at the test data point 0.0 locally interpolates between the predictions at the two closest training examples. Without gradient penalty, the network's prediction at 0.0 appears to be influenced by the farther training examples as well. Additionally, when generalizing to test data points -0.75 and 0.75, smaller gradients result in more conservative extrapolation.

3.3 Extension to TD Methods

This form of regularization extends naturally to TD updates. In particular, consider now a neural network that outputs action-value estimates $Q_\theta(s,a)$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. Assume that states are embedded in a Euclidean space $\mathbb{R}^d$ and that, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $Q_\theta(s,a)$ is differentiable with respect to s.

Algorithm 1 AVI + GP

Inputs: $\theta, \mathcal{D}, \alpha, \beta$
$\bar{\theta} \leftarrow \theta$
for value_iter $= 1$ to num_value_iter **do**
  for gradient_iter $= 1$ to num_gradient_iter **do**
    $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}((\theta|\bar{\theta},\mathcal{D}) + \beta\mathcal{R}(\theta|\mathcal{D}))$
  end for
  $\bar{\theta} \leftarrow \theta$
end for

Given a set $\mathcal{D}$ of transition tuples (s,a,r,s'), where $s \in \mathcal{S}$, $a \in \mathcal{A}$, r is a reward obtained by executing action a in state s, and s' is a subsequent state, the TD loss function is defined by

$$\mathcal{L}(\theta|\bar{\theta},\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{(s,a,r,s')\in\mathcal{D}}\delta_{\text{TD}}(s,a,r,s')^2, \tag{3}$$

where $\delta_{\text{TD}}(s,a,r,s')$ is the TD error $\delta_{\text{TD}}(s,a,r,s') = r + \gamma \cdot \max_{a'\in\mathcal{A}} Q_{\bar{\theta}}(s',a') - Q_\theta(s,a)$, and $\bar{\theta}$ is the weight vector of a target network. A standard TD update takes the form

$$\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}(\theta|\bar{\theta},\mathcal{D}). \tag{4}$$

To induce the sort of generalization we have discussed, we use a modified update rule

$$\theta \leftarrow \theta - \alpha\nabla_\theta(\mathcal{L}(\theta|\bar{\theta},\mathcal{D}) + \beta\mathcal{R}(\theta|\mathcal{D})), \tag{5}$$

where

$$\mathcal{R}(\theta|\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{(s,a,r,s')\in\mathcal{D}}\|\nabla_s Q_\theta(s,a)\|_2^2. \tag{6}$$

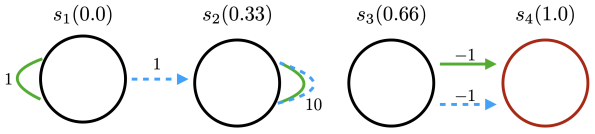


Figure 2: A four state MDP. The states have a single dimension, and there are two actions in each state. The states are $s_1 = 0.0$, $s_2 = 0.33$, $s_3 = 0.66$, and $s_4 = 1.0$. Actions a_1 and a_2 are indicated with green (solid) and blue (dashed) arrows, respectively. No transitions are seen from state s_4. The MDP has a discount factor of $\gamma = 0.9$. The optimal policy is to take action a_2 from state s_1.

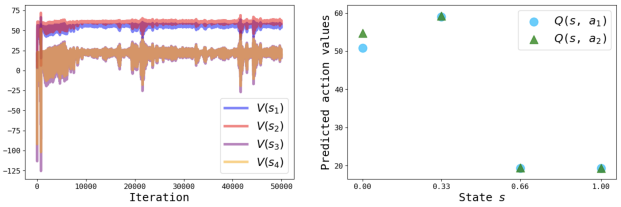


Figure 4: AVI + GP results on the 4 state MDP. (a) Predicted values as training proceeds, where $V(s) = \text{argmax}_{a \in \mathcal{A}} Q(s, a)$. (b) Final action values after 50,000 training iterations. Training is more stable and the resulting policy is optimal ($Q(s_1, a_1) < Q(s_1, a_2)$).

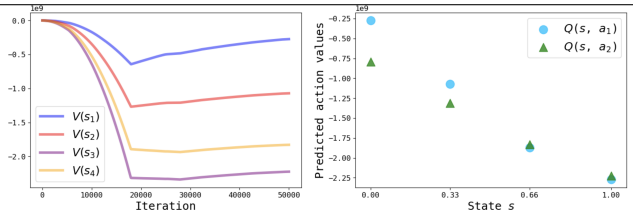


Figure 3: AVI results on the 4 state MDP. (a) Predicted values during training, where $V(s) = \text{argmax}_{a \in \mathcal{A}} Q(s, a)$. (b) Final action values after 50,000 training iterations. The value function magnitude becomes large and the resulting policy is sub-optimal ($Q(s_1, a_1) > Q(s_1, a_2)$).

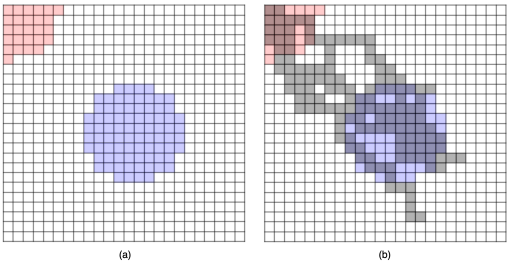


Figure 5: (a) GridWorld environment. State-action pairs for which the next state enters the red region and blue region have rewards -1 and 1, respectively. Otherwise, each transition has reward -0.05. (b) The shaded black states indicate the transitions that are collected by a fixed policy. We train AVI using this dataset.

We introduce our gradient penalty into Approximate Value Iteration (AVI). We provide pseudocode for our gradient penalty regularized AVI algorithm, AVI + GP, in Algorithm 1. In this algorithm, num_value_iter is the number of iterations and num_gradient_iter is the number of gradient updates in each iteration. At the end of an iteration, $\bar{\theta}$ is updated with the latest values of θ. Algorithm 1 highlights the ease with which we can augment TD methods with our approach: the only modification to the AVI update is the term colored in red.

4 Experiments

The goal of our experiments is to evaluate whether our gradient penalty regularizer improves (1) overall performance and (2) stability of value estimates when using TD methods in the presence of the deadly triad. To this end, we conduct experiments within a four state MDP and a GridWorld environment. The four state MDP and GridWorld serve as illustrative examples of the deadly triad. On these domains, we compare AVI with AVI + GP (Algorithm 1), specifically analyzing performance and stability. We set the regularization weight β to 0.01.

4.1 Four State MDP

As a didactic example of the deadly triad, consider a simple MDP with 4 states, each of which is a single dimension, and 2 actions in each state (Figure 2). Importantly, we do not see transitions or rewards from state s_4 to emulate the fact that in off-policy learning, certain regions of the state space may not be either visited or updated. Using AVI, we attempt to train a single hidden layer neural network with 1000 hidden units and RELU activations to learn the optimal value function of this MDP.

The predictions of the network with AVI and AVI + GP after $50,000$ training iterations (num_value_iter $= 50,000$ in Algorithm 1) are shown in Figures 3 and 4. We perform 1000 gradient steps in each iteration before updating value targets (num_gradient_iter $= 1000$). The action value predictions at states s_1 and s_2 are inaccurate with high magnitude after training is completed, notably after $50,000$ training iterations which is quite large for such a simple problem. This problem arises because of the following cycle. First, the network's action value estimates at state s_4 impact the those at state s_3 via bootstrapping. Second, due to the network's generalization across states, when the prediction at s_2 is larger than the prediction at s_3, this further reduces the prediction at s_4. Finally, the now negative predictions at s_3 and s_4 reduce the prediction at s_2, also from generalization. To break this cycle, AVI + GP prevents high magnitude

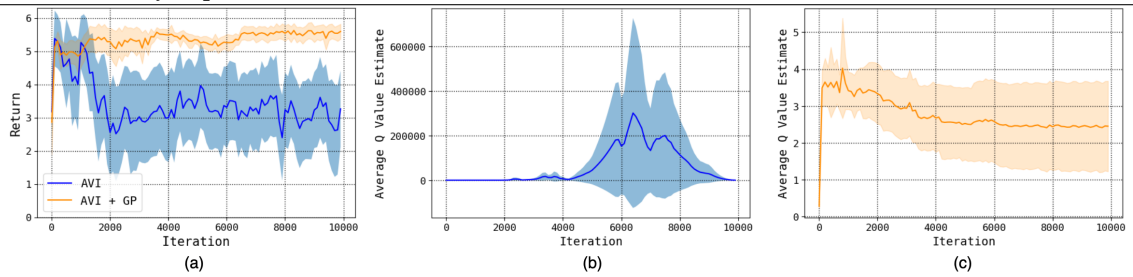


Figure 6: Return (a) and action value estimates throughout training ((b) and (c)) for AVI and AVI + GP. AVI + GP enjoys improved training stability and higher return than AVI. Reporting mean with one standard deviation as shaded region over 10 different seeds.

negative predictions at s_4 occurring in the cycle's second step by performing conservative extrapolation, which in turn prevents high magnitude negative predictions at s_3 from bootstrapping. When the predictions at both s_3 and s_4 have lower magnitude, the accuracy is improved at states s_1 and s_2 with known transitions and rewards.

4.2 GridWorld

The GridWorld is a 25×25 grid with each grid cell represented as an (x, y) coordinate, where x and y are normalized to be between 0 and 1. There are four actions in each state, and transitions are deterministic. One region of the state space has negative reward, another has positive reward, and otherwise the reward is a small step cost. Using a fixed behavior policy, we collect a fixed dataset of transitions shown in Figure 5. We train AVI and AVI+GP using a neural network with two hidden layers, each containing 20 nodes, and RELU activations. We train both algorithms for $10,000$ iterations with 100 gradient steps in each iteration ($\text{num\_value\_iter} = 10,000$ and $\text{num\_gradint\_iter} = 100$).

We plot (1) the average return of the greedy policy derived from action value estimates and (2) the average estimated action value over state-action pairs in the dataset throughout training. AVI achieves relatively low average return and large variability of return between seeds as shown in Figure 6(a). Lower performance correlates with unstable value estimates, as action value predictions attain unreasonably high magnitude during training (Figure 6(b)). In contrast, with AVI + GP the return achieved is consistently higher throughout training, the standard deviation of return between seeds is much smaller, and the action value estimates remain stable (Figure 6(a), (c)).

5 Conclusion

In this work, we investigate the function approximation component of the deadly triad. We introduce a regularizer that penalizes large gradients of a neural network's outputs with respect to its inputs. Our gradient penalty approach can be easily incorporated into TD methods such as AVI, resulting in practical algorithms which bootstrap value predictions and generalize across states by interpolation. An empirical study on several domains illustrates that augmenting TD methods with gradient penalty improves training stability. An analysis of this approach on more challenging domains and in conjunction with RL algorithms such as Q-learning is left for future work.

References

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[2] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[3] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[5] John N Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, 1996.

[6] H van Hasselt, Y Doron, F Strub, M Hessel, N Sonnerat, and J Modayil. Deep reinforcement learning and the deadly triad. arxiv. 2018.

Planning to plan: a Bayesian model for optimizing the depth of decision tree search

Ionatan Kuperwajs
Center for Neural Science
New York University
New York, NY, United States
ikuperwajs@nyu.edu

Wei Ji Ma
Center for Neural Science and Department of Psychology
New York University
New York, NY, United States
weijima@nyu.edu

Abstract

Planning, the process of evaluating the future consequences of actions, is typically formalized as search over a decision tree. This procedure increases expected rewards but is computationally expensive. Past attempts to understand how people mitigate the costs of planning have been guided by heuristics or the accumulation of prior experience, both of which are intractable in novel, high-complexity tasks. In this work, we propose a normative framework for optimizing the depth of decision tree search via Bayesian inference. Specifically, we model a metacognitive process where tree search is represented as continuously sampling noisy measurements of the value of a given state-action pair. This statistical approximation is then combined with any available prior experience to compute optimal planning depth. In the absence of retrospective information, our model makes intuitive predictions over a range of parameters. Meanwhile, integrating past experiences into our model produces results that are consistent with the transition from goal-directed to habitual behavior over time and the uncertainty associated with prospective and retrospective estimates.

Keywords: sequential decision-making; planning; Bayesian inference

Acknowledgements

This work was supported by Graduate Research Fellowship number DGE183930 from the National Science Foundation, grant number IIS-1344256 from the National Science Foundation, and grant number R01MH118925 from the National Institutes of Health.

1 Introduction

Planning involves the mental simulation of future actions and their consequences in order to make a decision. Such problems have typically been formalized as search over a decision tree in both cognitive science (Daw et al., 2005; Huys et al., 2015; van Opheusden et al., 2021) and artificial intelligence (Shannon, 1950; Silver et al., 2016). However, evaluating every possible course of action in complex environments is simply intractable. A growing body of literature has focused on solutions for approximating the values of choices without fully expanding a search tree (Snider et al., 2015) or efficiently allocating limited computational resources during planning (Callaway et al., 2018). Additionally, other models of human planning that achieve similar goals have been predominantly guided by researcher-specified heuristics (Dasgupta et al., 2018; Huys et al., 2012; Kool et al., 2017). A notable exception to this computes the value of information gained by planning in a principled manner (Sezener et al., 2019), but relies on a habitual system to estimate values at the frontier of the search tree.

Here, we propose an alternative: a normative model for optimizing the depth of decision tree search. Our framework operates at a metacognitive level, where in a given state, an agent thinks about how far into the future they should plan before any planning takes place. This model sacrifices the specificity of a tree search algorithm in favor of a simpler, task-general statistical description that facilitates the optimization of planning parameters. Our simulation results show that this framework derives intuitive principles about the depth to which planning is beneficial as a function of the cost per measurement, the total number of actions the agent must evaluate, the amount of accumulated retrospective experience, and the uncertainties associated with both prospective and retrospective samples.

2 Model

We assume the agent is in state s with actions $\{a_1, ..., a_n\}$ available to them, and that each state-action pair has a theoretical long-running expected reward under a tree search policy, $Q(a)$. This value is unknown to the agent, and the agent builds a posterior probability distribution over each $Q(a)$ while balancing the costs of deeper planning. This distribution is computed by considering how a one-step, myopic measurement will change under a certain depth of search, and is combined with any retrospective measurements that the agent has already accumulated for each state-action pair.

2.1 Generative model

To simulate the measurements that the agent has available to them prior to planning, we assume that the true value Q for a given state-action pair is normally distributed:

$$Q \sim \mathcal{N}\left(\mu_0, \sigma_0^2\right). \tag{1}$$

We omit the dependence of the parameters on the action a from the notation until the last step, where we will compute the value of the state across all actions. A retrospective experience with a is modeled as a noisy measurement, $q_{\text{retro},i}$, drawn from a normal distribution centered at the true Q:

$$q_{\text{retro},i} \sim \mathcal{N}\left(Q, \sigma_{\text{retro}}^2\right). \tag{2}$$

The retrospective measurements form a vector $\mathbf{q}_{\text{retro}} \equiv (q_{\text{retro},1}, \ldots, q_{\text{retro},n})$, where n is the number of past experiences with action a in state s. Similarly, the agent can perform a one-step look-ahead to obtain another noisy measurement, q_1, of Q:

$$q_1 \sim \mathcal{N}\left(Q, \sigma^2\right). \tag{3}$$

The core of our framework is a statistical model maintained by the agent of the effects of prospective tree search without actually performing the search. We assume that each iteration of the tree search algorithm works on a branch that starts with action a and produces a new, independent measurement of Q, q_t. Therefore, after T iterations of tree search, the agent has another vector of measurements $\mathbf{q} \equiv (q_1, \ldots, q_T)$.

2.2 Inference

The overarching goal of this framework is to solve for the optimal number of iterations to plan for, T^*. We take a normative approach, meaning that we assume the agent makes this decision by maximizing expected reward given costs. T is optimized independently of any action, which includes the possibility of choosing $T = 1$, or no planning at all. One way to conceptualize this is that, before planning, the agent approximates a breadth-first search algorithm by evaluating the future expected reward at each action to the same depth.

The inference scheme works as follows: (1) the agent considers different futures for each action after T planning iterations, (2) this future distribution is integrated with any retrospective information to form a posterior distribution,

and (3) the agent marginalizes over all possible futures by combining across actions and computing the maximum over the distribution of the posterior's expected value. The final output of this inference procedure is the mean of the max distribution, which we call the value of planning for T iterations.

Formally, the posterior is the normalized product of the prior, the retrospective likelihood, and the prospective likelihood, all of which we assume to be independent:

$$p(Q|\mathbf{q}_{\text{retro}}, \mathbf{q}) \propto p(Q)p(\mathbf{q}_{\text{retro}}|Q)p(\mathbf{q}|Q). \tag{4}$$

Each of the likelihoods is over Q based on the retrospective or prospective measurements available to the agent:

$$p(\mathbf{q}_{\text{retro}}|Q) = \mathcal{N}\left(Q; \bar{q}_{\text{retro}}, \frac{\sigma^2_{\text{retro}}}{n}\right) \tag{5}$$

$$p(\mathbf{q}|Q) = \mathcal{N}\left(Q; \bar{q}, \frac{\sigma^2}{T}\right), \tag{6}$$

where $\bar{q}_{\text{retro}} \equiv \sum_{i=1}^{n} q_{\text{retro},i}$ and $\bar{q} \equiv \sum_{t=1}^{T} q_t$. This allows us to rewrite the posterior as the normal distribution $\mathcal{N}\left(Q; \mu_T, \sigma_T^2\right)$, where we define the mean and variance as

$$\mu_T = \frac{J_0\mu_0 + J_{\text{retro}} n \bar{q}_{\text{retro}} + JT\bar{q}}{J_T} \tag{7}$$

$$\sigma_T^2 = \frac{1}{J_T}, \tag{8}$$

along with the precision quantities $J_{\text{retro}} \equiv \frac{1}{\sigma^2_{\text{retro}}}$, $J \equiv \frac{1}{\sigma^2}$, $J_0 \equiv \frac{1}{\sigma_0^2}$, and $J_T \equiv J_0 + J_{\text{retro}} n + JT$. We then write $\bar{q}$ to indicate that the myopic values, q_1, are known while the remaining values are defined as $\bar{q}_{>1} = \frac{1}{T-1}\sum_{t=2}^{T} q_t$. Now, we calculate the distribution over $\bar{q}_{>1}$ given q_1 and $\mathbf{q}_{\text{retro}}$ by marginalizing over the current possible values of Q:

$$p(\bar{q}_{>1}|q_1, \mathbf{q}_{\text{retro}}) = \int p(\bar{q}_{>1}|Q)p(Q|q_1, \mathbf{q}_{\text{retro}})dQ \tag{9}$$

$$= \mathcal{N}\left(\bar{q}_{>1}; \mu_1, \frac{\sigma^2}{T-1} + \sigma_1^2\right). \tag{10}$$

This distribution is over future prospective measurements T time steps into the future given the first myopic measurement. On average, the mean of the resultant distribution stays at the mean of the first measurement while the variance becomes narrower. The variance of this distribution at T is

$$\text{Var}[\mu_T] = \frac{\sigma_1^2}{1 + \frac{\sigma^2}{(T-1)\sigma_1^2}}. \tag{11}$$

Intuitively, the variance monotonically increases as a function of T because the future measurements are unknown and will cumulatively pull the posterior mean away from μ_1. Note that when $T = 1$, the variance is 0 as μ_1 is known and that when $T \to \infty$, the variance saturates as σ_1^2. This is because the future measurements will have fully pulled the posterior mean to the true value of Q.

The value of planning is the maximum of the future posterior mean of Q across all actions, $M_T = \max_a \mu_T(a)$. We can evaluate the expected value of this quantity, $E[M_T]$, by computing the max distribution and taking its mean. Within our framework, the mathematical reason why planning is beneficial is that the expected value of a maximum is greater than the maximum of expected values.

2.3 Optimization

After computing the benefit of planning for a range of T values, this expected reward is compared against the cost of planning. We assume a fixed cost per evaluation c such that the utility of planning for T iterations is given by:

$$U_T = (E[M_T] - E[M_1]) - cNT, \tag{12}$$

where N is the total number of actions considered in the state. In this way, our cost function takes into account the depth and breadth of the tree being approximated by the model. The first term increases sublinearly with T, while the second one increases linearly, meaning that their difference will have an optimum. This optimum is the best number of steps to plan ahead:

$$T^* = \underset{T}{\text{argmax}}\ U_T. \tag{13}$$

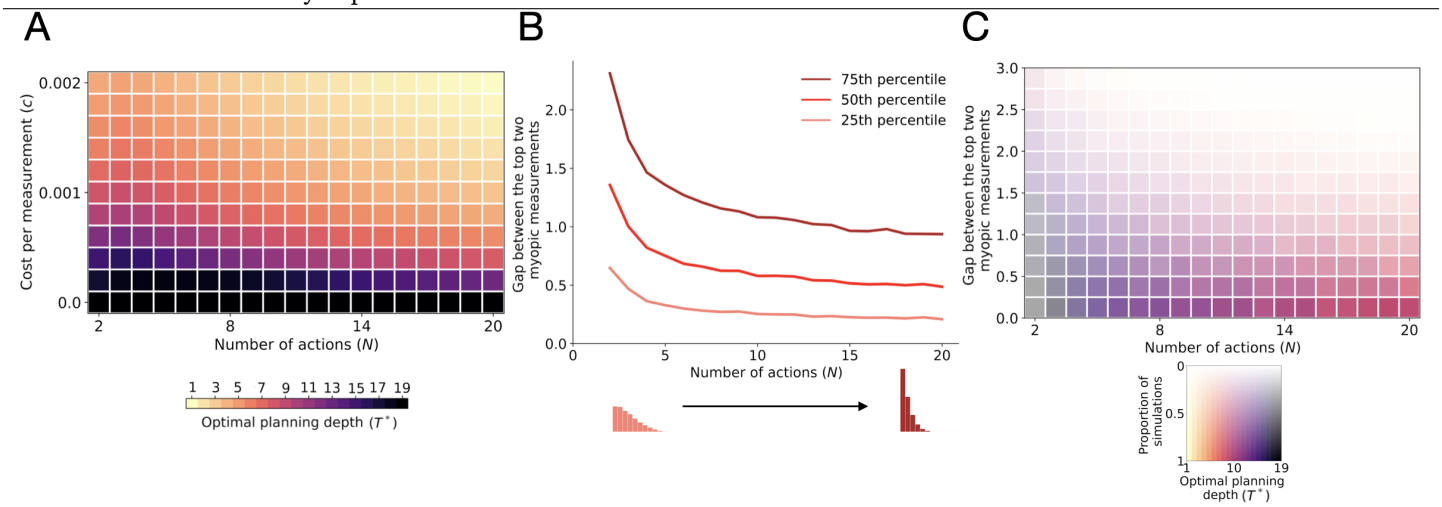


Figure 1: The model makes intuitive predictions about planning depth using only myopic estimates. **(A)** Optimal planning depth (T^*) as a function of the cost per measurement (c) and the number of actions available to the agent (N). **(B)** 75th, 50th, and 25th percentiles for the gap between the top two myopic measurements as a function of number of actions for a single cost ($c = 0.001$). **(C)** Optimal planning depth as a function of the gap between the top two myopic measurements and number of actions, for the same cost as in (B). The color code is two-dimensional: the hue represents optimal planning depth and the saturation proportion of total simulations for the combination of top gap size and N.

3 Results

3.1 Myopic model predictions

We first consider the case where the agent has no prior experience and relies only on myopic evaluations to decide how far into the future to plan. This mimics real-world planning environments where an agent has uninformed priors over their retrospective system, such as in novel tasks or tasks in which states may not repeat often.

Our cost function is dependent on the number of evaluations that the agent must make during inference, or the product of the cost per measurement (c) and the total number of actions available to the agent in the state (N). Over this parameter space, our model predicts that deeper planning is beneficial at lower costs and with less alternatives (Figure 1A). The reasoning behind this effect is based on the value of the myopic measurements. If the gap between two myopic values is small, should the agent plan further ahead or avoid wasting valuable resources planning? And conversely, if the gap between two myopic evaluations is large, should the agent plan more or less?

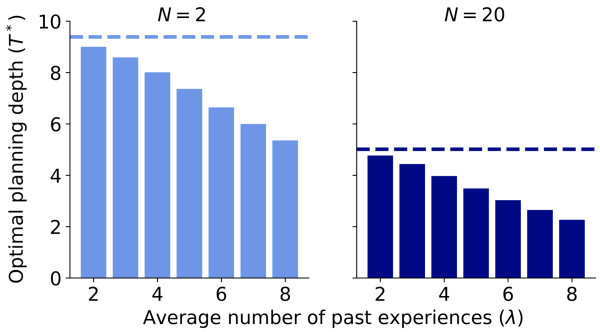


Figure 2: The effect of experience on depth of planning. Optimal planning depth for 2 actions (left) and 20 actions (right) as a function of the average number of retrospective measurements per action (modeled as a Poisson rate) for a single cost ($c = 0.001$). The dashed line in each panel indicates the optimal planning depth without retrospective experience.

In Figure 1B, we show that as the number of actions increases, the distribution of the difference between the top two myopic measurements becomes more right-skewed. In other words, small top gap sizes are more common when there are more actions to consider, and the size of the top gap is varied at a lower number of alternatives. Figure 1C investigates the relationship between top gap size and optimal planning depth: T^* is higher with smaller top gap sizes and with less available actions. This suggests that smaller gap sizes do increase optimal planning depth, but that this effect is diminished with more actions where cost grows and ultimately outweighs the added benefit of planning more deeply.

3.2 Incorporating retrospective information

Next, we examine environments where the agent does have prior experience. Planning depth should be modulated by the total amount of retrospective experience as well as the uncertainty of those estimates. These correlate to well-studied mechanisms in the planning literature: the transition from model-based to model-free control over time and uncertainty-based arbitration between prospective and retrospective systems.

We simulate total experience by using a variable Poisson rate (λ) to determine n for each action. The model predicts a shallower optimal planning depth as more experience is accumulated, and this trend holds irrespective of the number of alternatives that the agent considers (Figure 2). The rationale behind this is that environments with low amounts of retrospective information require similar planning depths compared to when the model relies only on myopic estimates. Optimal planning depth then decreases as the agent gains more experience. In these cases, the agent can spend less resources planning and instead relies more heavily on its cost-effective retrospective experiences. Additionally, we varied the amount of uncertainty for both the retrospective and myopic estimates to investigate their joint effect on planning depth (Figure 3). This is straightforward to implement, since our model directly takes the variance associated with each type of sample as part of its generative model. With a low number of alternatives, increased uncertainty with either or both sources of information leads to deeper planning. With more alternatives, however, high amounts of prospective uncertainty results in lower optimal planning depths. Intuitively, planning more deeply is generally beneficial in gaining high-value estimates under uncertainty, but if the uncertainty attached to planning is too high then it is no longer worthwhile to obtain these costly measurements.

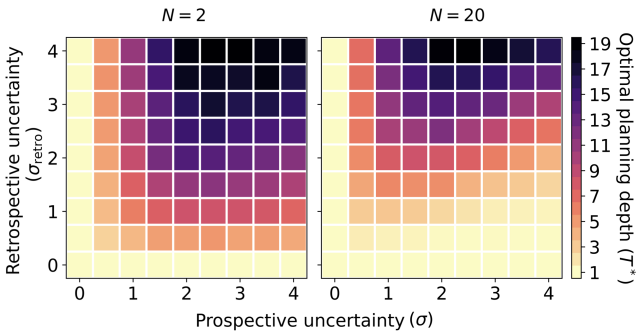


Figure 3: The effect of uncertainty on depth of planning. Optimal planning depth for 2 actions (left) and 20 actions (right) as a function of the retrospective and prospective variance in the generative model for a single cost ($c = 0.001$).

4 Discussion

We presented a normative framework for optimizing the depth of decision tree search based on Bayesian inference to compute the value of planning from a combination of retrospective samples and myopic estimates. We showed that this model makes intuitive depth predictions in the absence of retrospective information, primarily driven by cost per measurement and the number of actions. Then, we introduced retrospective experience and found that planning depth decreases as the agent gains experience and increases with the uncertainty of the model's evaluations unless prospective uncertainty is so high that deeper planning is no longer worthwhile. Ultimately, our framework approximates planning via an optimal information sampling algorithm that doesn't construct a full search tree. A more sophisticated variant of our model would determine online which action to expand the search frontier for, thereby optimizing both the depth and direction in which planning is beneficial. In future work, we plan to investigate how our model and its variants interact with forward search in complex planning tasks.

References

Callaway, F., Lieder, F., Das, P., Gul, S., Krueger, P. M., & Griffiths, T. L. (2018). A resource-rational analysis of human planning. *Proceedings of the 40th Annual Conference of the Cognitive Science Society*.

Dasgupta, I., Schulz, E., Goodman, N. D., & Gershman, S. J. (2018). Remembrance of inferences past: Amortization in human hypothesis generation. *Cognition*, *178*, 67–81.

Daw, N. D., Niv, Y., & Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, *8*(12), 1704–1711.

Huys, Q. J. M., Eshel, N., O'Nions, E., Sheridan, L., Dayan, P., & Roiser, J. P. (2012). Bonsai trees in your head: How the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLOS Computational Biology*.

Huys, Q. J. M., Lally, N., Faulkner, P., Eshel, N., Seifritz, E., Gershman, S. J., Dayan, P., & Roiser, J. P. (2015). Interplay of approximate planning strategies. *Proceedings of the National Academy of Sciences*, *112*(10), 3098–3103.

Kool, W., Gershman, S. J., & Cushman, F. (2017). Cost-benefit arbitration between multiple reinforcement-learning systems. *Psychological Science*, *28*, 1321–1333.

Sezener, C. A., Dezfouli, A., & Keramati, M. (2019). Optimizing the depth and direction of prospective planning using information values. *PLOS Computational Biology*.

Shannon, C. E. (1950). Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), 256–275.

Silver, D. et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Snider, J., Lee, D., Poizner, H., & Gepshtein, S. (2015). Prospective optimization with limited resources. *PLoS Comput Biol*, *11*(9), e1004501.

van Opheusden, B., Galbiati, G., Kuperwajs, I., Bnaya, Z., Li, Y., & Ma, W. J. (2021). Revealing the impact of expertise on human planning with a two-player board game. *PsyArXiv*.

Understanding human decision-making in a complex planning task with large-scale behavioral data

Ionatan Kuperwajs
Center for Neural Science
New York University
New York, NY, United States
ikuperwajs@nyu.edu

Wei Ji Ma
Center for Neural Science and Department of Psychology
New York University
New York, NY, United States
weijima@nyu.edu

Abstract

Large-scale data sets in cognitive science allow researchers to test how established results generalize beyond constrained laboratory settings while leading to novel scientific questions. In this work, we present a data set of $1,234,844$ participants playing $10,874,547$ games of a challenging variant of tic-tac-toe. This task is at an intermediate level of complexity, providing rich behavior for which modeling is still tractable. We utilize this task and data set in order to examine fundamental components of human decision-making, namely the interaction between dropout and learning as well as the balance between prospection and retrospection. We find a correlation between task performance and total experience, and independently analyze participants' dropout behavior and learning trajectories. We uncover that stopping patterns are driven by increases in playing strength, and investigate the factors underlying playing strength increases with experience using a set of metrics derived from a best-first search model with a heuristic value function. Finally, we explain discrepancies between the planning model's predictions and observed data in early game choices with retrospective behavioral patterns. Our results provide examples of how massive behavioral data sets paired with complex tasks provide unique opportunities for understanding human decision-making.

Keywords: planning; reinforcement learning; decision-making; behavioral modeling

Acknowledgements

This work was supported by Graduate Research Fellowship number DGE183930 from the National Science Foundation, grant number IIS-1344256 from the National Science Foundation, and grant number R01MH118925 from the National Institutes of Health.

1 Introduction

In recent years, massive data sets collected through online experiments have become more common in cognitive science (Stafford and Dewar, 2014; Steyvers et al., 2019). The purpose of this methodology is to obtain rich data in participants' real-world environments rather than isolating a single variable and controlling for sources of variation. In turn, this data can be used to study a wide array of cognitive mechanisms from learning to decision-making to planning (Schulz et al., 2019; Steyvers and Schafer, 2020), or even for leveraging methods from machine learning to construct computational models (Peterson et al., 2021). These data sets have additional value in that they can clarify whether results or models derived from more constrained settings generalize while simultaneously allowing for entirely new research questions.

Here, we present a massive data set to investigate fundamental components of human decision-making in a combinatorial game of intermediate complexity. In contrast to traditional laboratory experiments, people initiate, continue, and stop play completely at their own discretion. We begin by characterizing the endpoint of participants' learning trajectories, establishing that there is a correlation between final playing strength and overall experience. Then, we characterize features that drive dropout behavior and learning trajectories in our task. Finally, we establish that people engage in prospective planning by fitting a computational model to their choices, and illustrate two behavioral patterns that the model fails to predict but are consistent with retrospective decision-making.

2 Task and data set

Our task is a variant of tic-tac-toe, in which two players alternate placing tokens on a 4-by-9 board (Figure 1). The objective is to get four tokens in a row horizontally, vertically, or diagonally. The game, which we call 4-in-a-row, has approximately $1.2 \cdot 10^{16}$ non-terminal states, and therefore is at a level of complexity which far exceeds tasks commonly used in cognitive science (van Opheusden and Ma, 2019). However, 4-in-a-row retains computational tractability, making it an ideal candidate for studying the interplay between dropout and learning as well as prospection and retrospection. Additionally, we partnered with Peak, a mobile app company, to implement a visually enriched version of 4-in-a-row on their platform (`https://www.peak.net`), which users play at their leisure in their daily environment. We are currently collecting data at a rate of approximately 1.5 million games per month, and here we analyze a subset consisting of $10,874,547$ games from $1,234,844$ unique users collected between September 2018 and April 2019. These users each play a wide range of games, but the data set includes thousands of users who have played upwards of 20 or even 100 games. In this version of the task, users always move first against an AI agent implementing a planning algorithm, with parameters adapted from fits on previously collected human-vs-human games (van Opheusden et al., 2021). AI agent playing strength is modulated by changing model parameters based on game outcomes.

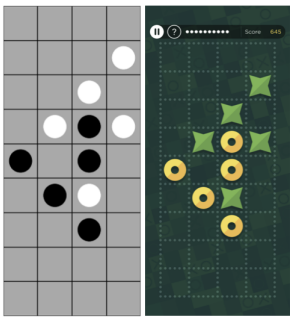


Figure 1: Example board position in 4-in-a-row. The left is the laboratory version of the task, while the right is the gamified version used on the Peak platform.

3 Results

3.1 Dropout and learning functions

In order to motivate the rest of our findings on learning, we first investigate the relationship between task performance and total experience. We measure users' task performance using Elo ratings (Elo, 1978), a standard method for calculating the relative skill levels of players in zero-sum games. Ratings calculated for relatively few games can be statistically unreliable, so we exclude players with less than 20 total games played from our analysis and group each user's experience into blocks of 20 games. This results in $115,968$ unique users, and we use a common baseline to compute Elo ratings across all experimental data. In Figure 2A, we correlate this final playing strength with the total number of games played by each user ($\rho = 0.27$). Due to the size of the data set, our p-values are below the minimum representable float ($2.0 \cdot 10^{-308}$) unless reported otherwise. This result illustrates that, at the endpoint of their learning trajectories, users are more likely to have higher task performance if they have accumulated more total experience with the task.

One novel feature of a data sets like ours is temporal: participants have complete autonomy over when and for how long to engage in the given task. In practice, this means that any investigation into learning using these data sets must also consider dropout behavior. We hypothesize that dropout behavior in this game is strongly influenced by two factors: current playing strength and current number of games played. In order to test this hypothesis, we examine the effect of each user's Elo rating in a given window of 20 games on the probability that they stop playing in the next block of 20 games, and further bin these probabilities by number of games that the user has played so far (Figure 2B). We find that as users play more games they have a lower stopping probability (logistic regression: $\beta = -0.02 \pm 5.8 \cdot 10^{-5}$), and their stopping probability increases with higher Elo ratings (logistic regression: $\beta = (6.31 \pm 0.10) \cdot 10^{-4}$). This finding suggests

that as users gain more experience, they are more likely to continue playing when the game is still challenging and they have lower Elo ratings. Conversely, users tend to quit when their playing strength is high. This aligns with the literature on intrinsic motivation in humans: if users devalue the task due to lack of a challenge or information gain, they will stop participating (Schmidhuber, 2010). However, we do not find evidence for the inverse trend that users find the task too difficult and are therefore more likely to stop playing with lower Elo ratings. One possible explanation for this is that the AI agents that users were matched with did not make the task difficult enough.

We then validate that a reliable increase in playing strength over time occurs at the population level. In Figure 2C, we show that average Elo ratings increase as users gain more experience, and that this trend occurs irrespective of the total number of games each user ends up playing (linear regression: $\beta = 1.5 \pm 8.9 \cdot 10^{-3}$). We also find a reliable, albeit smaller, effect of initial Elo ratings on current playing strength (linear regression: $\beta = 0.66 \pm 1.6 \cdot 10^{-3}$). Note that the correlation from Figure 2A can be observed by connecting the endpoints of each learning curve. Importantly, we find no evidence for changes in the slopes of users' average learning trajectory when conditioned on either total number of games played or initial playing strength. This suggests that learning rates in this task are independent of these two factors, and primarily captured by current playing strength, current experience level, and individual differences.

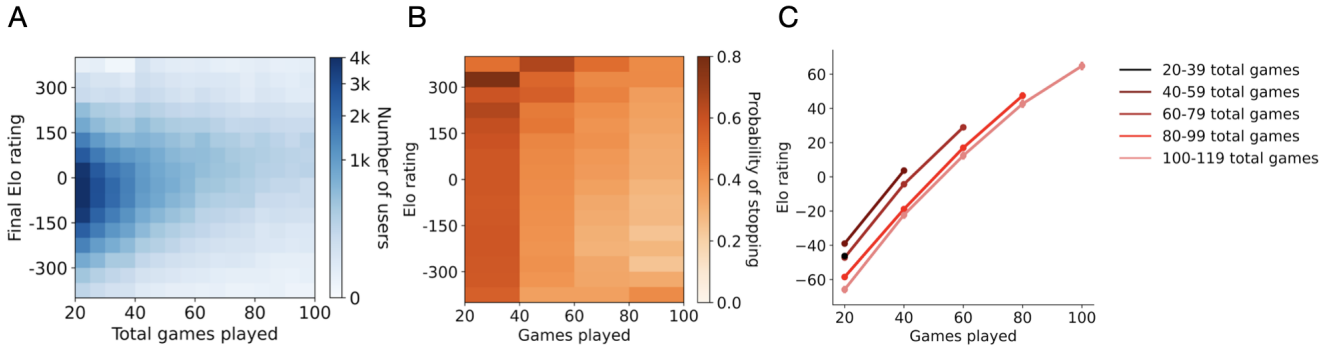


Figure 2: Dropout and learning are driven by playing strength and experience. **(A)** Two-dimensional histogram of the final Elo rating and total number of games played for users in the data set. The visualization is limited to $104,681$ users who had a final Elo rating between -400 and 400 and played at least 20 and less than 100 total games. **(B)** Probability of stopping during the next block of 20 games as a function of the Elo rating and number of games played so far in the current block, again limited to the same ranges as in (A). **(C)** Average user Elo rating as a function of current experience level conditioned on total number of games played for $107,769$ users who played at least 20 and less than 120 total games.

3.2 Prospective planning

Given our insights into the factors driving dropout and learning functions in this task, we fit a computational model to users' choices in 4-in-a-row to demonstrate that they engage in prospective planning. This algorithm combines a heuristic function, which is a weighted linear combination of board features (Campbell et al., 2002), with the construction of a decision tree via best-first search. Best-first search iteratively expands nodes on the principal variation, or the sequence of actions that lead to the best outcome for both players given the current decision tree (Dechter and Pearl, 1985). To allow the model to capture variability in human play and make human-like mistakes, we add Gaussian noise to the heuristic function and include feature dropout. For each move the model makes, it randomly omits some features from the heuristic function before it performs search. Such feature omissions can be interpreted cognitively as lapses of selective attention (Treisman and Gelade, 1980). During search, the model also prunes the decision tree by removing branches with low heuristic value (Huys et al., 2012).

When fitting the computational model to behavioral data, we estimate the log probability of a user's move in a given board position with inverse binomial sampling (van Opheusden et al., 2020), optimize the log-likelihood function with Bayesian adaptive direct search (Acerbi and Ma, 2017), and account for potential overfitting by reporting 5-fold cross-validated log-likelihoods. To test whether the tree search component is necessary to fit human choices, we compared the model's log-likelihood per move with that of a model where tree search terminates after a single iteration. Model fitting and comparison is computationally taxing, so we ran this analysis on 50 pseudo-randomly selected users. The cross-validated log-likelihood per move of the computational model is higher across users than that of the myopic model (Figure 3A), demonstrating that tree search is indeed necessary to predict user's moves.

We convert the model parameters inferred for a set of $1,000$ users who had played at least 100 games from the learning analysis to three metrics: planning depth, feature drop rate, and heuristic quality. Planning depth roughly corresponds to the number of steps people think ahead, feature drop rate measures the frequency of attentional lapses, and heuristic quality measures the "correctness" of the feature weights. Figure 3D-E show that depth of planning increases with

experience (linear regression: $\beta = 0.011 \pm 8.0 \cdot 10^{-4}$), while feature drop rate decreases (linear regression: $\beta = (-2.58 \pm 0.36) \cdot 10^{-4}$, $p = 4.5 \cdot 10^{-13}$). We verify that users' response times decrease across blocks, signifying that the planning depth increase is not a result of slower play. Figure 3F also shows a reliable increase in heuristic quality with experience (linear regression: $\beta = (6.11 \pm 0.29) \cdot 10^{-4}$). However, the heuristic quality in the first 20 games of this data set is much lower than in previously collected laboratory data, suggesting that users have more opportunity to improve their feature weights rather than starting at ceiling. These results demonstrate that users' learning trajectories are underscored by deeper planning, fewer lapses of attention, and bounded improvement of feature weights.

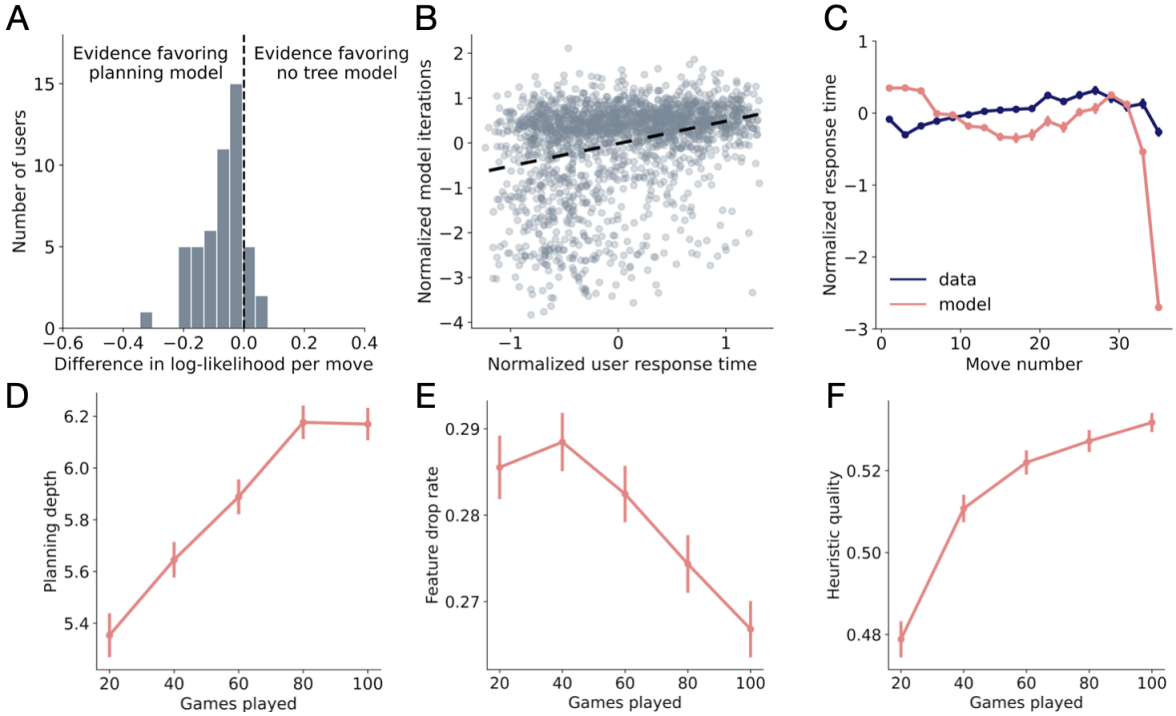


Figure 3: Evidence for prospective planning. **(A)** Histogram of the difference in cross-validated log-likelihood per move for the planning and no tree models across 50 users (mean: -0.075 ± 0.011). **(B)** Scatterplot of the number of model iterations and user response time on each move for the same users as in (A). To normalize, we first divide the number of iterations or response time by the mean and then take the logarithm. **(C)** Average user response time (blue) and number of model iterations (pink) taken to make a move during gameplay. **(D)** Average depth to which $1,000$ users plan as they gain experience, as estimated by the model. **(E)** Same as (D) for feature drop rate. **(F)** Same as (D) for heuristic quality.

3.3 Retrospective decision-making

One major difference between the model and our observed data is predicted response times. We do find that people's response times correlate on individual trials with the number of model iterations ($\rho = 0.30$, $p = 6.0 \cdot 10^{-41}$, Figure 3B), but their average trend over the course of a game differs considerably (Figure 3C). Early in gameplay, the model predicts that people search larger decision trees and thus have longer response times, but the data shows the opposite. Therefore, it is likely that in situations where the board is fairly empty and no player can immediately win the game, there is a faster retrospective process that takes place before prospective planning begins. In the middle and late game, response time trends roughly follow model predictions.

Due to the size of our data set, we were able to uncover clear evidence for retrospective decision-making. We find that response times in early stages of a game are mediated by previous game outcome: user response times across the first 7 moves are, on average, longer after losses rather than wins (Figure 4A). Furthermore, third move response times decrease significantly when users encounter repeated 2-piece board states. In terms of action selection, users are significantly more likely to repeat their opening moves following wins rather than losses, and these moves are primarily distributed in the center or corners of the board (Figure 4B). This effect continues on the third move, where users most often elect to play in the center positions closest to the two pieces already on the board (Figure 4C). However, the proportion of move repetitions based on game outcome decrease further into gameplay. These population-wide trends suggest that users make decisions partially based on whether or not an opening strategy was successful in previous games, and transition to alternative strategies, such as thinking ahead, in subsequent moves when board positions are more likely to be unique.

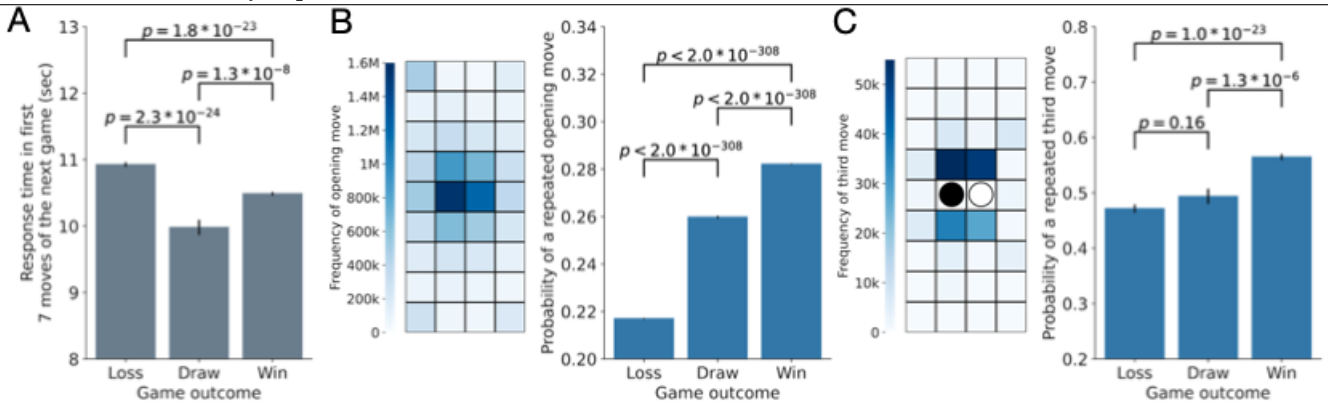


Figure 4: Evidence for retrospective decision-making. **(A)** Average response times across the first 7 moves of a game directly following a loss, draw, or win. **(B)** Probability that users repeat their opening move directly after a loss, draw, or win. **(C)** Same as (B), but for the most frequent 2-piece board state. All panels are computed on the full data set.

4 Discussion

We leveraged a large-scale data set of human participants playing a two-player combinatorial game in order to interrogate their decision-making process. We first established that playing strength and overall experience are correlated before characterizing the factors that underlie dropout and learning. For dropout, we found that current playing strength and experience level drive stopping probabilities. For learning, we demonstrated that playing strength increases over time, and that these improvements can be attributed to increased planning depth, decreased feature dropping, and bounded increase in heuristic quality. We then used a planning model to show that humans engage in prospective reasoning in the middle and late game, and that their early game moves as well as their response times are affected by the outcome of previous games in which they encountered the same board positions. Our findings showcase how massive behavioral data sets with complex tasks can be used to study novel aspects of human decision-making.

References

Acerbi, L., & Ma, W. J. (2017). Practical bayesian optimization for model fitting with bayesian adaptive direct search. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1834–1844.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, *134*(1-2), 57–83.

Dechter, R., & Pearl, J. (1985). Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)*, *32*(3), 505–536.

Elo, A. E. (1978). *The rating of chessplayers, past and present*. Arco Pub.

Huys, Q. J., Eshel, N., O'Nions, E., Sheridan, L., Dayan, P., & Roiser, J. P. (2012). Bonsai trees in your head: How the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology*, *8*(3), e1002410.

Peterson, J. C., Bourgin, D. D., Agrawal, M., Reichman, D., & Griffiths, T. L. (2021). Using large-scale experiments and machine learning to discover theories of human decision-making. *Science*, *372*(6547), 1209–1214.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, *2*(3), 230–247.

Schulz, E., Bhui, R., Love, B. C., Brier, B., Todd, M. T., & Gershman, S. J. (2019). Structured, uncertainty-driven exploration in real-world consumer choice. *Proceedings of the National Academy of Sciences*, *116*(28), 13903–13908.

Stafford, T., & Dewar, M. (2014). Tracing the trajectory of skill learning with a very large sample of online game players. *Psychological science*, *25*(2), 511–518.

Steyvers, M., Hawkins, G. E., Karayanidis, F., & Brown, S. D. (2019). A large-scale analysis of task switching practice effects across the lifespan. *Proceedings of the National Academy of Sciences*, *116*(36), 17735–17740.

Steyvers, M., & Schafer, R. J. (2020). Inferring latent learning factors in large-scale cognitive training data. *Nature Human Behaviour*, *4*(11), 1145–1155.

Treisman, A. M., & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive psychology*, *12*(1), 97–136.

van Opheusden, B., Acerbi, L., & Ma, W. J. (2020). Unbiased and efficient log-likelihood estimation with inverse binomial sampling. *PLoS computational biology*, *16*(12), e1008483.

van Opheusden, B., Galbiati, G., Kuperwajs, I., Bnaya, Z., Ma, W. J., et al. (2021). Revealing the impact of expertise on human planning with a two-player board game.

van Opheusden, B., & Ma, W. J. (2019). Tasks for aligning human and machine planning. *Current Opinion in Behavioral Sciences*, *29*, 127–133.

Comparing the Effects of Stress on Directed and Random Exploration

Kyle J. LaFollette
Case Western Reserve University
Cleveland, OH
kjlafoll@case.edu

Heath A. Demaree
Case Western Reserve University
Cleveland, OH
had4@case.edu

Abstract

Recent insights from artificial intelligence and computer science have proven invaluable for studying the explore-exploit dilemma, particularly in disentangling information-directed from random exploration strategies. Despite these advances, the algorithms used to solve this dilemma are largely devoid of affective parameters. In this study, we aimed to investigate the relationship between the cognitive computational substrates of the explore-exploit dilemma and a particularly powerful indicator of affective state: physiological stress reactivity. We first used a Bayesian generalized logistic regression model to evaluate propensities to choose more informative bandits over higher valued bandits. This revealed more directed exploration under stress, and no change in random exploration. We then considered three hierarchical Bayesian reinforcement learning models to determine how uncertainty associated with Kalman filter states can influence explorative behavior under stress versus no stress: a Thompson sampler, an upper confidence bound algorithm, and a hybrid of the two. We found that an upper confidence bound algorithm best explains exploration under stress, whereas a hybrid model may better capture exploration without stress. We discuss the implications of these findings for future modeling of the explore-exploit dilemma; models must be flexible to affective states such as stress if modeling of the explore-exploit dilemma is to progress beyond normative considerations.

Keywords: exploration-exploitation, reinforcement learning, decision making, stress

1 Introduction

Exploration versus exploitation decisions are fundamental to everyday human decision-making. From a computational perspective, these decisions arise from uncertainty in the value of multiple options. For example, when a person feels uncertain about whether s/he should continue working a demanding, perhaps unfulfilling job, s/he may choose to continue *exploiting* their current position, or *explore* alternative employers. However, not all exploration is value-driven: Some exploration is stochastic. A person may choose to explore new targeted job ads to gain information on other specific employers (i.e., *value-directed* exploration), or the choice may be reflexive of their overall uncertainty (i.e., *random* exploration).

Much of the explore-exploit dilemma has been studied under normative considerations, but what can we expect people to do under stress? Numerous animal models suggest that anxiety is negatively associated with exploration [6, 10]. In humans, Lenow and colleagues [5] discovered that both acute and chronic physiological stress – markers of environmental safety or quality – were associated with an increase in exploitation. Other studies have suggested that anxiety in humans is similarly inversely related to explorative behavior [2, 8]. Although it may appear from this literature that stress drives exploitation at the cost of exploration, few studies have considered disparate effects on directed versus random exploration. We hypothesized that directed exploration would be attenuated under stress, whereas random exploration would be unaffected. To test this hypothesis, we measured participants' preferences on a multi-armed bandit task immediately after an acute physiological and psychological stress manipulation.

2 Methods

Participants. We tested 29 participants in this study. Participants were recruited from a midwestern University and received partial course credit for their participation. All participants provided written informed consent in accordance with the university's institutional review board. In addition to course credit, participants were incentivized with up to $5 USD in performance bonuses.

Procedure. The study was conducted over two experimental sessions. The sessions were approximately 1.5-hours in duration and separated by 1-week. All participants completed a variation of the Maastricht Acute Stress Task (MAST; [7]) during each session. The MAST is a physiological and psychosocial stress manipulation which combines a cold pressor task (i.e. cold produced pain), serial subtraction task (i.e. social evaluation during arithmetic), and uncertainty (i.e. pseudorandom durations). During one session, participants alternated between immersing a foot into a bucket of 2-degrees Celsius ice water and counting backwards from 2043 in units of 17. During another control session, the same procedures were followed but with room temperature (17-degrees Celsius) water and serial subtractions of 2. Session order was within-subjects and counterbalanced.

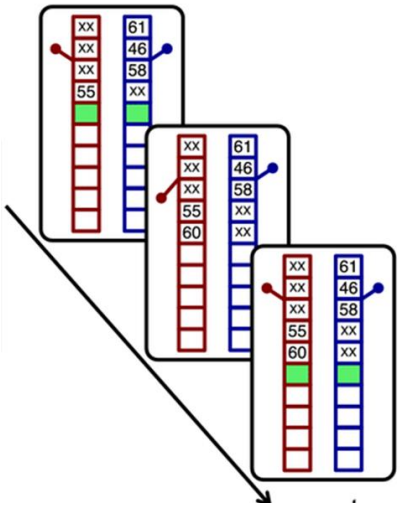


Figure 1. Horizon task design.

Participants completed 80 games of the Horizon Task (Figure 1 [9]) immediately after the MAST. On each game, participants chose between two one-armed bandits (i.e., slot machines), each of which provided points from different Gaussian distributions. The means of these two distributions ranged anywhere between 1 and 100 points, but their standard deviations were always equal to 8 points. Through sampling, participants were expected to learn the means of these two distributions and maximize their total reward. At the start of each game, participants were instructed to make four forced

336

options (i.e., they were bound to select a specific bandit without any volition of their own). These forced choices set up the game to have unequal information between the two bandits (e.g., one bandit was forcibly chosen three times while the other was chosen once). After the forced choices, participants were free to choose either bandit either one time (i.e., short horizon 1) or six times (i.e., long horizon 6; participants may switch the bandits chosen throughout the 6 trials). These two horizon conditions manipulated the relative value of exploration and exploitation.

Behavioral Analysis. We fit a Bayesian generalized logistic regression model (i.e., Richards' curve) to participant choice data immediately following the four forced trials. This determined the probability of choosing the lesser known bandit on the first free-choice trial as a function of the difference in average bandit values δ with Gaussian noise: $choice \sim N\left(a + \frac{a-b}{1+e^{-\beta\delta}}, \sigma^2\right)$. a, b, and β were free to vary with stress condition (stress versus no stress). Following the informal observations of Wilson and colleagues [9], the growth rate β reflects the degree to which choice is influenced by random exploration, and the point of inflection $a - b$ reflects the influence of directed exploration.

RL Modeling. We modeled choice behavior using three hierarchical Bayesian reinforcement learning models; one using a Thompson sampler, another an upper confidence bound algorithm, and the last a hybrid of the two. Bandit values were updated using Kalman filtering equations, $Q_{t+1}(a) = Q_t(a) + \kappa_t\big(r_t - Q_t(a)\big)$ where $Q_t(a)$ is the expected value of the chosen bandit and κ_t is the Kalman gain. Likewise, variance was updated as a proxy for uncertainty, $\sigma^2_{t+1}(a) = \sigma^2_t(a) - \kappa_t\sigma^2_t(a)$. The Kalman gain is dynamic with learned uncertainty and measurement variance, $\kappa_t = \frac{\sigma^2_t(a)}{\sigma^2_t(a) + \rho^2(a)}$, where $\rho^2(a)$ is the measurement variance. Initial $Q_0(a)$ was fixed at 0 whereas initial $\sigma^2_0(a)$ was free to vary across subject and stress conditions. Choice probabilities $p(a)$ for the first model were calculated using a Thompson sampling policy to gauge random exploration, $p(a_t) = \Phi((Q_t(a) - Q_t(b))/\sqrt{(\sigma^2_t(a) + \sigma^2_t(b)\,)})$, where stochasticity is proportional to total uncertainty $\sqrt{\sigma^2_t(a) + \sigma^2_t(b)}$ across bandits a and b. The second model used an upper confidence bound algorithm instead for directed exploration, $p(a_t\,) = \Phi(((Q_t(a) - Q_t(b)) + \gamma(\sigma^2_t(a) - \sigma^2_t(b)))/\lambda)$, where γ reflects an information bonus for bandit a's uncertainty relative to b, and λ controls stochasticity. Both γ and λ were free to vary across subject and stress conditions. Last, the hybrid model combined these policies to assess the simultaneous influence of random and directed exploration, $p(a_t) = \Phi\left(\frac{(1-\omega)\left(Q_t(a) - Q_t(b)\right)}{\sqrt{\sigma^2_t(a) + \sigma^2_t(b)}} + \omega\left(\sigma^2_t(a) - \sigma^2_t(b)\right)\right)$, where ω is the weight of directed exploration on choice relative to random exploration [4]. ω was also free to vary across subject and stress conditions.

Model fitting. All models were fit using a Hamiltonian Monte Carlo No-U-Turn sampler, a MCMC method provided in the Stan probabilistic programming language. We collected 10,000 samples for each parameter across 4 chains run in parallel. The first 5,000 samples of each chain were discarded as warm-up. We compared the relative fits of models by approximating their point-wise out-of-sample predictive accuracy using leave-one-out cross validation (LOO-CV). All Stan models and analysis scripts are available at: https://github.com/kjlafoll/stress-ee-rldm2022.

3 Results

3.1 Behavioral

First, we checked for main effects of stress and horizon on the propensity to choose the explorative or less-

informed bandit. These behavioral analyses did not account for learning and as such were restricted to the first free-choice trial of each game. A frequentist logistic regression model revealed more explorative choices with longer horizons, $\beta = 0.458$, $z = 7.162$, $p < 0.001$, as well as after stress manipulation, $\beta = 0.402$, $z = 6.291$, $p < 0.001$ (Fig 2A). Second, we considered the propensity to explore as a function of the mean difference between both bandits prior to the first free-choice. A Bayesian generalized logistic regression model revealed disparate effects of stress on model parameters a and β, the lower asymptote and slope of the sigmoid, respectively (Figure 2B-C). Participants had a much greater a after the stress session, meaning that they had a greater propensity to choose a much lower-valued explorative bandit when under stress than when not. This suggests more directed exploration under stress. Conversely, participants had near equivalent β after stress relative to control, indicating similar accelerations in preference with changes in value. This suggests no change in random exploration between stress conditions.

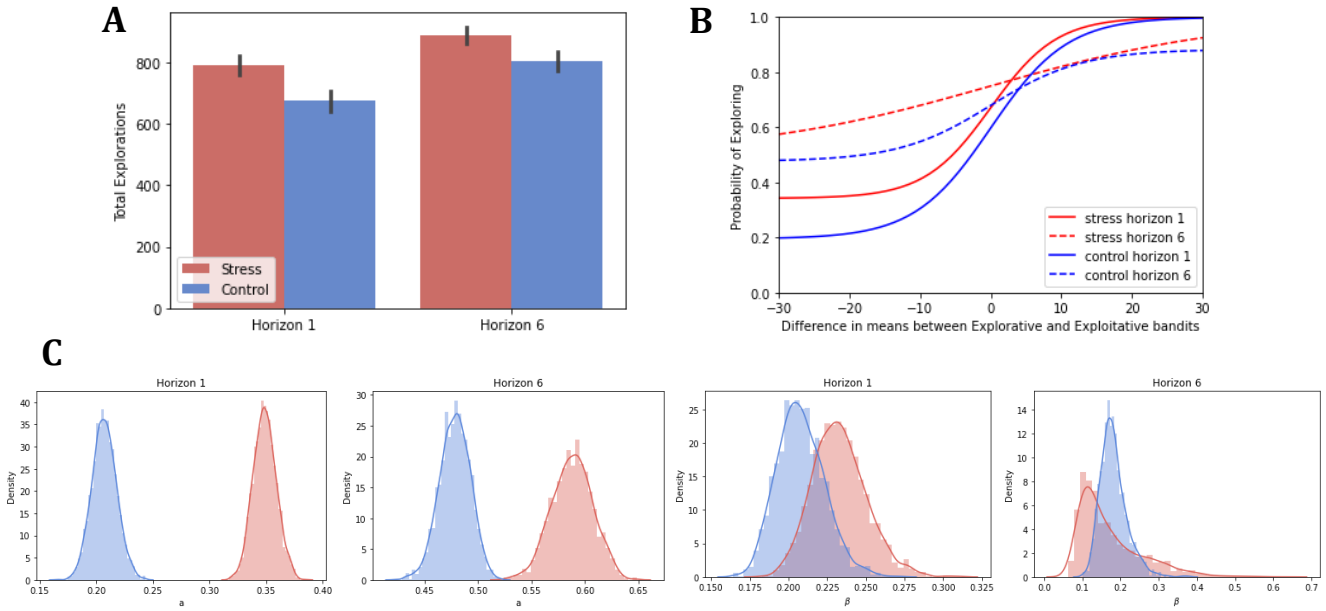


Figure 2. First free-choice behavior. (A) Total number of explorative choices made. (B) Logistic fits for probability of choosing explorative bandit as a function of mean value difference. (C) Posteriors over parameter means from Bayesian generalized logistic model for each stress condition.

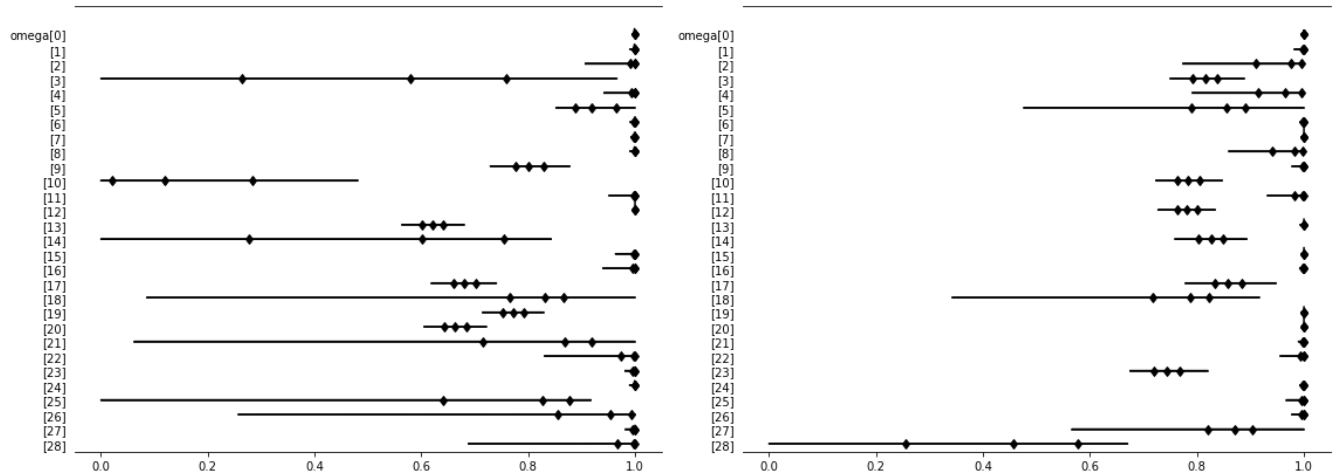


Figure 3. Results of hybrid model. Subject-level posteriors over ω weight for directed exploration for control (left) and stress (right) sessions. 338

3.2 Reinforcement Learning

The behavioral analyses were limited in that while they could elucidate horizon effects, they were restricted to the first free-choice and blind to learning processes. To extend our analysis to processes of reinforcement learning, we fit three hierarchical Bayesian model to all free-choice trials in long horizon 6 games (short games were excluded due to learning being a nonfactor). The winning model, a hybrid of a Thompson sampling policy and upper confidence bound algorithm, was selected for inference. Subject-level posteriors over the parameter weighting directed exploration relative to random exploration revealed substantially more directed exploration during the stress session than during control (Figure 3).

4 Discussion

In this study, we presented a two-factored approach to modeling the explore-exploit dilemma under acute physiological and psychological stress: a descriptive, generalized logistic regression model, and an explanatory hybrid reinforcement learning model. The models revealed that, counter to our hypotheses, directed exploration strengthened under stress. This finding conflicts with many behavioral analyses of exploration under stress. Nonetheless, it is a step forward toward disentangling directed and random exploration at a latent processing level, especially considering that some conflicts in the literature suggest that predicting explorative behavior is not so clear-cut. For example, Feldman-Hall and colleagues [3] reported that electrodermal activity – a gold-standard measure for stress reactivity – was associated with increased risk-taking behavior when probabilities were ambiguous or uncertain. A recent study by Bennett and colleagues [1] also concluded that anxiety correlates with informatic utility. Taken together, this suggests that stress may have a modulatory effect on explorative behavior dependent on context; it is important not to conflate threats of opportunity costs with elements of uncertainty when predicting exploration under stress. More research is needed on explorative behavior outside normative considerations such as stress in the face of uncertainty, and computational modeling can be a useful tool for formalizing new theoretical directions.

References

[1] Bennett, D., Sutcliffe, K., Tan, N. P.-J., Smillie, L. D., & Bode, S. (2021). Anxious and obsessive-compulsive traits are independently associated with valuation of noninstrumental information. Journal of Experimental Psychology: General, 150(4), 739–755.
[2] Biedermann, S. V., Biedermann, D. G., Wenzlaff, F., Kurjak, T., Nouri, S., Auer, M. K., Wiedemann, K., Briken, P., Haaker, J., Lonsdorf, T. B., & Fuss, J. (2017). An elevated plus-maze in mixed reality for studying human anxiety-related behavior. BMC Biology, 15(1), 125.
[3] Feldman-Hall, O., Glimcher, P., Baker, A. L., & Phelps, E. A. (2016). Emotion and decision-making under uncertainty: Physiological arousal predicts increased gambling during ambiguity but not risk. Journal of Experimental Psychology: General, 145(10), 1255-1262.
[4] Gershman, S. J. (2018). Deconstructing the human algorithms for exploration. Cognition, 173, 34-42.
[5] Lenow, J. K., Constantino, S. M., Daw, N. D., & Phelps, E. A. (2017). Chronic and acute stress promote overexploitation in serial decision making. Journal of Neuroscience, 37(23), 5681-5689.
[6] Prut, L., & Belzung, C. (2003). The open field as a paradigm to measure the effects of drugs on anxiety-like behaviors: A review. European Journal of Pharmacology, 463(1–3), 3–33.
[7] Smeets, T., Cornelisse, S., Quaedflieg, C. W. E. M., Meyer, T., Jelicic, M., & Merckelbach, H. (2012). Introducing the Maastricht Acute Stress Test (MAST): a quick and non-invasive approach to elicit robust autonomic and glucocorticoid stress responses. Psychoneuroendocrinology, 37(12), 1998-2008.
[8] Walz, N., Mühlberger, A., & Pauli, P. (2016). A human open field test reveals thigmotaxis related to agoraphobic fear. Biological Psychiatry, 80(5), 390–397.
[9] Wilson, R. C., Geana, A., White, J. M., Ludvig, E. A., & Cohen, J. D. (2014). Humans use directed and random exploration to solve the explore-exploit dilemma. Journal of Experimental Psychology: General, 143(6), 2074-2081.
[10] Zweifel, L. S., Fadok, J. P., Argilli, E., Garelick, M. G., Jones, G. L., Dickerson, T. M. K., Allen, J. M., Mizumori, S. J. Y., Bonci, A., & Palmiter, R. D. (2011). Activation of dopamine neurons is critical for aversive conditioning and prevention of generalized anxiety. Nature Neuroscience, 14(5), 620–626.

Memory-guided, goal-driven reinforcement learning explains subclinical depression

Gyubin Lee*
School of Computing
Korea Advanced Institute of Science and Technology
Republic of Korea, Daejeon 34141
gbstorm81@kaist.ac.kr

Minsu Abel Yang
Department of Bio and Brain Engineering
Program of Brain and Cognitive Engineering
Korea Advanced Institute of Science and Technology
Republic of Korea, Daejeon 34141
minsuyang@kaist.ac.kr

Sang Wan Lee†
Department of Bio and Brain Engineering
Program of Brain and Cognitive Engineering
Center for Neuroscience-inspired Artificial Intelligence
KAIST for Health Science Technology
KAIST for Artificial Intelligence
Korea Advanced Institute of Science and Technology
Republic of Korea, Daejeon 34141
sangwan@kaist.ac.kr

Abstract

Depression often leads to impairment in various cognitive functions. Despite earlier computational and neural studies, we have a limited understanding of how depression affects goal-driven decision-making, where the memory retrieval can guide the value-based decision-making processes. To investigate these issues, we first designed a computational model of reinforcement learning (RL) that incorporates memory-based control into the successor representation (SR), called goal-driven SR. Second, using behavioral and fMRI data acquired using a two-stage goal-directed decision-making task, we found evidence for goal-driven SR; SR prediction error was encoded in the neural activity of the ventrolateral prefrontal cortex (vlPFC) and right insula, whereas the conventional reward prediction error was encoded in the ventral striatum. These results suggest the potential role of the prefrontal cortex and insula in memory-based value learning. Third, through correlation analyses, we found effects of depression on value learning, memory-based control, and action selection. Lastly, we built a novel Predictor-Decoder system that estimates the severity of depression from fitted goal-driven SR parameters, with significantly higher accuracy compared to data-driven machine learning models, including support vector regression (SVR) and deep neural network (DNN).

Keywords: Depression, Value-based decision-making,
Memory-based control, Prediction of depression severity

Acknowledgements

This study was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2019M3E5D2A0106626721, Development of metacognitive AI for rapid learning), and Institute of Information & Communications Technology Planning& Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01371, Development of brain-inspired AI with human-like intelligence).

*First author
†Corresponding author

1 Introduction

Major depressive disorder, or depression is known to affect various aspects of human value-based decision-making. While recent studies have attempted to understand depression in the context of reinforcement learning (RL), whether and how depression affects goal-driven decision-making, where continuous memory storage and retrieval can support value-based decision-making. Although the recent study found that arbitration between model-based and model-free systems can characterize depression well [1], the existing theory does not fully accommodate memory involvement in goal-driven learning. This necessitates the understanding of depression in the composite viewpoint where we fully incorporate both RL theory and related memory functions.

This study investigates how subclinical depression affects goal-driven decision-making at behavioral and neural levels. First, we implemented a computational model incorporating memory-based control into successor representation (SR). We ran a model comparison analysis on behavioral data to show that our model explains human subjects' choice patterns qualitatively better than model-based and model-free RL. Second, we conducted general linear model (GLM) analysis on fMRI data. Next, through correlation analyses, we demonstrated that fitted model parameters can support sufficient information about depression scale of the corresponding subject. Finally, we built a framework to predict the severity of depression from fitted parameters of our model, demonstrating the possibilities of model-driven diagnosis of subclinical depression.

2 Result

2.1 Computational model of memory-guided goal-directed learning

To establish computational principles of memory-guided goal-directed learning, we constructed the computational model based on SR named 'goal-driven SR' (Figure 2 left) and fitted its parameter to behavioral data while subjects were performing the two-stage Markov decision task (Figure 1). The task involves a goal condition, in which a subject needs to make a choice leading to a specific goal that is assigned on a trial-by-trial basis. We constructed a "memory bank" in the computational model [4] to account for the suboptimal learning and decision-making process caused by maladaptive memory functions originated from depression. The memory bank keeps weight vectors for each goal condition in the task and returns the weight vector associated with the given goal condition. With support of the memory bank, the model can explicitly learn a goal-specific policy, whereas a previous RL algorithms maintained one policy to multiple goals, which update it whenever the goal changes by running backward planning [2, 3].

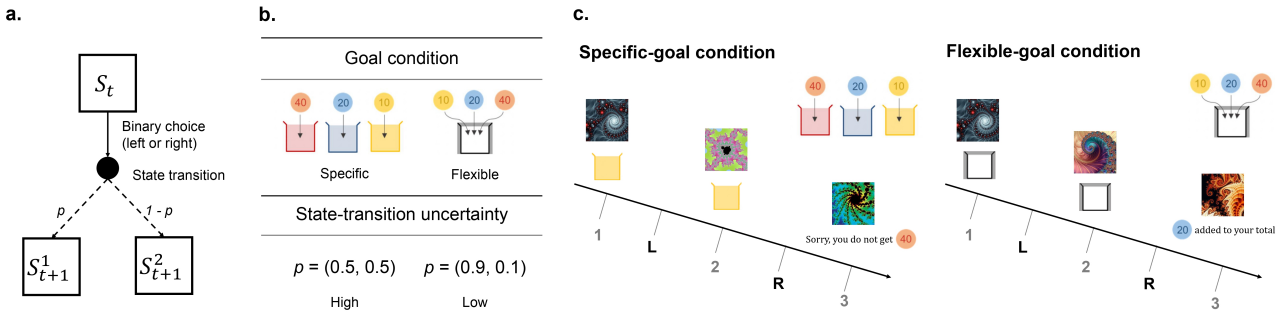


Figure 1: (a) Sequential two-stage Markov decision task proposed in [3]. Participants make binary choice (left or right) at each state and move to one of next two states with a certain state-transition probability (p, $1-p$) (b) Illustration of the task context. The goal condition consists of specific and flexible-goal conditions. In specific-goal condition, the color of the collecting box (red, blue or yellow) should match the color of the coin. In flexible-goal condition, all kinds of coins are allowed to be collected. The state-transition uncertainty consists of high and low uncertainty conditions. During state-transition uncertainty is high, state-transition probability was set to $p = 0.5$, whereas during the low uncertainty condition, state-transition probability was set to $p = 0.9$. (c) Illustration of the task. Each state is associated with a specific fractal image. After the fractal image for initial state was shown, participants made a binary choice (left or right). Then, they were transited to the next state determined by the state-transition probability. Again, subjects made a binary choice again to arrive to an outcome state. Each outcome state was associated with a colored coin (red, blue, yellow). In the specific-goal condition, participants should collect the coin that matches with the color of the collecting box. In the flexible-goal condition, they can receive the reward afterward regardless of the color of coin.

We compared the prediction performance of our computational model with that of the goal-directed RL model by utilizing Bayesian information criterion (BIC). As a result, we found that the BIC score of goal-driven SR model (M=614.22,

SEM=20.56) is significantly lower than the BIC score of arbitration control model (M=666.83, SEM=18.33) (Figure 2c-d, paired-sample permutation test, goal-driven SR model vs. arbitration control model: $p < 10^{-5}$, $d = -0.8524$, $n = 64$).

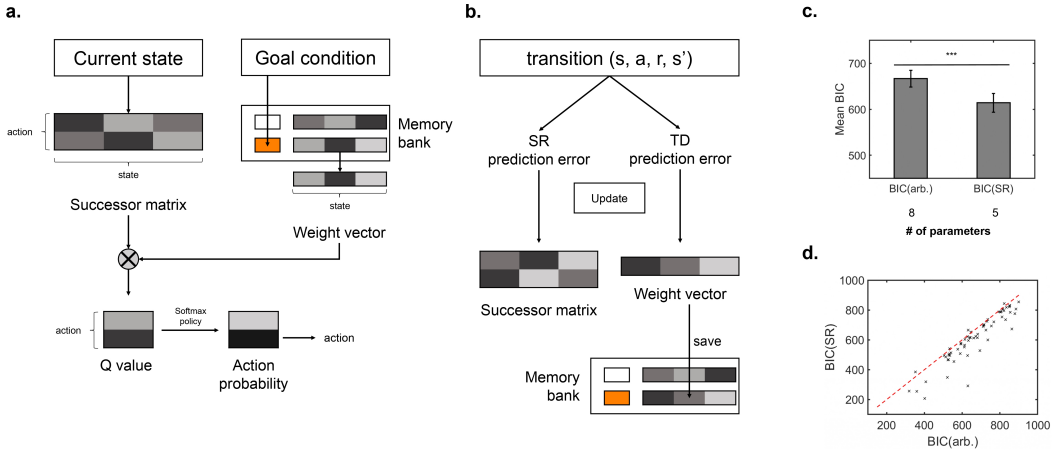


Figure 2: (a) The structure of goal-driven SR. In each state, the model returns a successor matrix that indicates expected cumulative future state occupancy for each action. When the new goal is assigned, the memory bank returns the saved weight vector associated with the goal condition. The state-action value (Q value) is given by the product of the successor matrix and the weight vector. Using the state-action value, the model makes a stochastic choice with the softmax function. (b) After transition (s, a, r, s'), two types of prediction error occur: SR prediction error and temporal difference (TD) prediction error. SR prediction error refers to the difference between actual and expected state occupancy, and TD prediction error refers to the difference between received and predicted reward. The model updates its successor matrix using SR prediction error and weight vector using TD prediction error. After updating the weight vector, the memory bank saves it, tagging the corresponding goal condition. (c-d) Model comparison analysis result. We used Bayesian information criterion (BIC); (c) The mean and standard error of the mean of BIC scores. Error bar stands for the standard error of the mean, (d) Individual comparison of BIC scores for each subject. X axis indicates BIC of arbitration control, and Y axis indicates BIC score of goal-driven SR. The dotted line in the figure indicates identity line as reference.

2.2 Neural evidence of goal-driven SR

To verify that our model explains the neural activity in the brain, we conducted general linear model (GLM) analysis to check whether the key signals from our model, SR and TD prediction errors, were represented within the fMRI data. We found that the SR prediction error was found in the right insula and the ventrolateral prefrontal cortex (vlPFC) (Figure 3 left, all $p < 0.05$, FWE corrected for multiple comparisons). On the other hand, the TD prediction error was found on both sides of ventral striatum (Figure 3 right, all $p < 0.05$, FWE corrected for multiple comparisons).

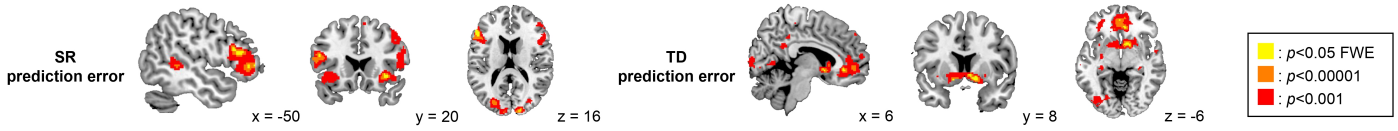


Figure 3: GLM analysis results. The SR prediction error were encoded in the neural activity of ventrolateral prefrontal cortex (vlPFC) and right insula, whereas the conventional TD prediction error were encoded in ventral striatum. All the effects reported were survived after adjusting for family-wise error (FWE $p < 0.05$).

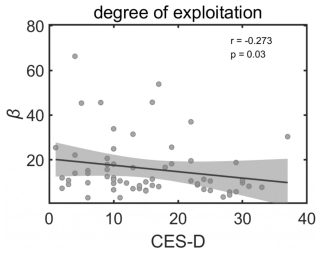
2.3 Model parameters explain depression severity

To verify that our model reflects the characteristics of the subclinical depression, we measured Spearman's correlation coefficients between each subject's estimated model parameter and their self-reported depression score (CES-D). First, we found that the degree of exploitation is negatively correlated ($r_S(62) = -0.273, p = 0.03, n = 63$) with CES-D score (Figure 4 left.) For more detailed investigation, we conducted partial correlation analyses and found that the learning rate of the weight vector ($r_S(62) = -0.360, p = 5.10 \times 10^{-3}, n = 63$), the discount factor of the successor matrix ($r_S(62) = -0.397, p = 1.90 \times 10^{-3}, n = 63$), and the degree of exploitation ($r_S(62) = -0.483, p = 1.00 \times 10^{-4}, n = 63$) are significantly negatively correlated with CES-D score (Figure 4 right.) 342

a.

Parameter	α_{TD}	α_{SR}	γ_{SR}	β	b_0
r	-0.1408	0.0509	-0.0389	-0.2729	-0.0327
p	0.2710	0.6920	0.7629	0.0304	0.7993

α_{TD}: learning rate of weight vector, α_{SR}: learning rate of successor matrix, γ_{SR}: discount factor of successor matrix, β: degree of exploitation, b_0: bias in action selection



b.

Parameter	α_{TD}	α_{SR}	γ_{SR}	β	b_0
r	-0.3598	-0.0315	-0.3967	-0.4828	-0.0114
p	0.0051	0.8127	0.0019	0.0001	0.9315

α_{TD}: learning rate of weight vector, α_{SR}: learning rate of successor matrix, γ_{SR}: discount factor of successor matrix, β: degree of exploitation, b_0: bias in action selection

Figure 4: (a) Simple correlation analysis between model parameters and CES-D score. Right graph shows the relationship between the individual CES-D score and the degree of exploitation, one of the model parameters. (b) Partial correlation analysis between model parameters and CES-D score.

2.4 Model-driven diagnosis of depression

To fully accommodate the observation that our model parameters convey information related to depression, we designed a framework that predicts the subject's CES-D score using model parameters. First, we attempted to directly predict subjects' CES-D score from their fitted parameters using data-driven approaches, including support vector regression (SVR) and deep neural network (DNN). However, their performances were poor because there mainly suffered from insufficient number of severely depressed subjects compared to the control case (Figure 5d top.)

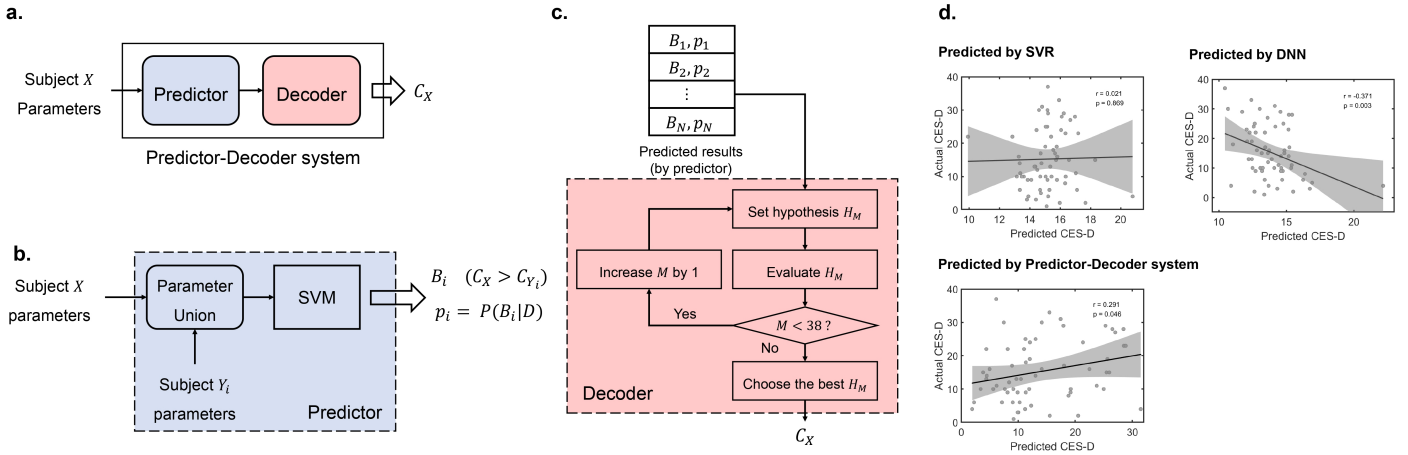


Figure 5: (a) The outline of the Predictor-Decoder system. The parameters of goal-driven SR for the subject X are fed into the system as an input. The model outputs the predicted CES-D score of subject X, C_X. (b) The structure of the predictor. Model parameters of subject X are combined with model parameters of subject Y_i, whose CES-D score is known. The concatenated parameter vector is passed to the binary SVM classifier that outputs Boolean value B_i (True if C_X is higher than CES-D score of subject Y_i, C_{Y_i}, False otherwise) and the posterior probability of the prediction $p_i = P(B_i|D)$. (c) Decoder is an algorithm that estimates a subject's CES-D score using the Predictor's output. (d) Prediction performance of support vector regressor (SVR), deep neural network (DNN), and our Predictor-Decoder system. The predictions from our Predictor-Decoder system has significantly positive correlation with the actual CES-D scores, whereas predictions from SVR and DNN have no significant correlation with actual CES-D scores.

To resolve this issue, we built a Predictor-Decoder system, which consists of the 'predictor' and 'decoder' modules (Figure 5a). The predictor receives the individually-fitted parameter set of goal-driven SR as input (Figure 5b). Then it compares the input parameter vector with the parameters of other subjects whose CES-D scores are known by the predictor. The outputs of the predictor are 1) a Boolean value set; each element indicates whether the subject's CES-D score is higher than that of the input subject, and 2) the posterior probability of the prediction. The decoder converts the output of the predictor into the subject's CES-D score by using the iterative hypothesis evaluation (Figure 5c). First, the decoder assumes the subject's CES-D score as an integer M that starts with 1 (Hypothesis H_M). Then it evaluates H_M by calculating the score of H_M. The decoder assumes C_X being in the range from 1 to 37 (= a maximum CES-D score in our dataset), each of which is considered to represent a different 'hypothesis'; the Decoder evaluates each hypothesis iteratively. The score of hypothesis H_M, S_{H_M} is

$$S_{H_M} = \sum_{i=1}^{N} a_i \times p_i + (1 - a_i) \times (1 - p_i), \tag{1}$$

where N is the number of classifications from the Predictor, and a_i is a Boolean value, which becomes 1 if B_i is a right prediction under H_M and 0 otherwise. After evaluating all hypotheses, the decoder accepts the hypothesis with the highest score. Notably, our Predictor-Decoder system can predict the CES-D score well (Figure 5d), compared to conventional regression analyses, including SVR and DNN.

3 Discussion

3.1 Neural evidence of goal-driven SR

Previous findings show the neural correlates of state prediction error (SPE) for the model-based learning in the insula and the lateral prefrontal cortex (lPFC), whereas association with reward prediction error (RPE) for the model-free learning appears in the ventral striatum [3]. We found that goal-driven SR is associated with neural representations of the two distinct prediction errors: SR and TD prediction error. The successor matrix keeps track of expected cumulative future state occupancies. In our task, the visited state cannot be re-visited in the same trial, so the successor matrix should be converged to the state-transition probability. Accordingly, the SR prediction error resembles SPE. In our analyses, SR prediction error was found in the insula and ventrolateral prefrontal cortex (vlPFC), consistent with previous findings of SPE. Taken together, the results expand the potential role of the lateral prefrontal cortex and insula to memory-guided goal-driven learning. The TD prediction error is associated with the ventral striatum like RPE of the model-free system. Although the updating processes are quite different, the concept of TD prediction error and RPE are similar, therefore it can be an evidence that the TD prediction error works like RPE.

3.2 Effects of depression on memory-guided goal-driven learning

For the first, the arbitration control [1] and our goal-driven SR have a parameter that indicates the tendency to make the most valuable choice. The corresponding parameter value is negatively correlated with the CES-D score in both the arbitration and our model. This result suggests that the depressed group makes more exploratory choices rather than exploiting the previous history of success than the healthy group. Also, we found that the learning rate of temporal difference learning is negatively correlated with the CES-D score. It indicates that the depression group takes more time to update state values than the healthy control group. In this case, if the subject happens to obtain more rewards than previously, the control group updates their state-action value accordingly, whereas the depressed group is less capable of improving their policy. This can be seen as habitual behavior. This result corroborates a previous finding that choice optimality is negatively correlated with CES-D score. One interesting result is that the discount factor for the successor matrix is negatively correlated with the CES-D score. The successor matrix provides information about the task structure. Therefore, this result shows that goal-driven SR can represent the effect of depression on memory-guided learning.

3.3 Predictor-Decoder system for predicting the depression score

We designed a Predictor-Decoder system to estimate CES-D score from model parameters, and it outperforms a few other data-driven models. The key idea is to learn the 'gradient' between two data points. The predictor is trained to compare subject's CES-D scores from the two subjects' model parameters. This strategy is intended to learn the difference in the CES-D score between two subjects instead of the CES-D score itself. Another advantage of the predictor design is that pairing subjects' model parameters can increase the number of training data. In addition, the decoder is intended to recover the subject's CES-D score correctly by obscuring the wrong predictions by the predictor. Although the most reasonable hypothesis may receive some fewer points ($(1 - P(B_i|D))$) because of the wrong prediction, the effects of incorrect prediction are counterbalanced after summing up the received points.

References

[1] Suyeon Heo, Yoondo Sung, and Sang Wan Lee. Effects of subclinical depression on prefrontal–striatal model-based and model-free learning. *PLoS computational biology*, 17(5):e1009003, 2021.

[2] Dongjae Kim, Jaeseung Jeong, and Sang Wan Lee. Prefrontal solution to the bias-variance tradeoff during reinforcement learning. *Cell reports*, 37(13):110185, 2021.

[3] Sang Wan Lee, Shinsuke Shimojo, and John P O'Doherty. Neural computations underlying arbitration between model-based and model-free learning. *Neuron*, 81(3):687–699, 2014.

[4] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836. PMLR, 2017.

Towards neoRL networks; the emergence of purposive graphs.

Per R. Leikanger

UiT – Norges Artigste Universitet
Per.Leikanger@uit.no

Abstract

The neoRL framework for purposive AI implements latent learning by emulated cognitive maps, with general value functions (GVF) expressing operant desires toward separate states. The agent's expectancy of reward expressed as learned projections in the considered space, allows the neoRL agent to extract purposive behavior from the learned map according to the reward hypothesis. We explore this allegory further, considering neoRL modules as nodes in a network with desire as input and state-action Q-value vectors as output; we observe how an action set with a Euclidean significance allows for interpreting state-action values as Euclidean projections of desire. *Autonomous desire* from neoRL nodes within the agent allows for deeper neoRL behavioral graphs. Experiments confirm the effect of neoRL networks governed by autonomous desire, verifying four identified principles for purposive networks for behavioral AI. The neoRL agent, governed by desire vectors formed from purposive graphs, can navigate Euclidean spaces in real-time while learning — exemplifying how modern AI still can profit from gathering inspiration from early psychology.

Keywords: Tolman, purposive AI, GVF, autonomous navigation, neoRL net

1 Behavioristic AI by neoRL nodes

Thorndike's *law-an-effect* in functionalist psychology has been reported as a prime inspiration for Reinforcement Learning (RL) in AI [6]. Thorndike considered the reinforcement of randomly encountered reflexes as a plausible explanation for simple behavior and acquired reflexes. The law-of-effect represents an essential first step toward the study of behavior becoming a natural science, quickly replaced by *behaviorism* for explaining advanced policies and human behavior. Edward C. Tolman (1866-1959) further considered the formation of behavior as being distinct from learning, based on the observation that an animal could express different behavior based on varying motivation [9]. B. F. Skinner expanded on Tolman's *latent learning*, proposing that acquired policies were *operant* toward an objective. Operant behaviors are activated on demand, thus allowing for complex policies.

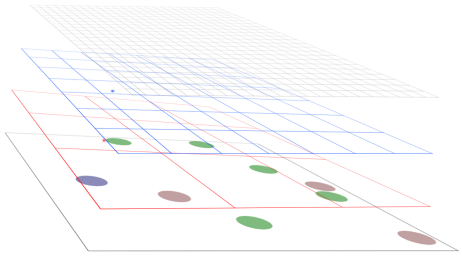


Figure 1: **The first multi-map neoRL agent.** The neoRL agent is capable of expressing latent learning across several representations of the same Euclidean space, forming agent value function as a weighted sum of operant GVFs across multiple NRES maps. (figure from [1])

The neoRL framework for behavioral AI combines Tolman's cognitive maps and Skinner's *operant conditioning* with results from modern neuroscience [1]. Via general value functions (GVF) [7], one can express latent learning as a distributed mechanism across numerous off-policy learners for separate operant state activation. Letting each GVF be guided by a separate *intent* signal, i.e., a pseudo-reward that reflects some external conditional, patterns of GVFs could be activated for accurate control in this space [1, 2]. Inspired by the 2014 Nobel Price in neuroscience, for the discovery of place cells and other mechanisms commonly referred to as neural representation of Euclidean space *(NRES)*, Leikanger (2019) demonstrated how NRES-oriented RL (*neoRL*) agents are capable of online autonomous navigation by intent signals analogous to NRES reported in modern neuroscience. Elements-of-interest, the agent's projections of reward in the considered navigational modality, facilitates a graded activation of GVFs for purposive activation of latent experience. See my thesis [4] for more on neoRL, scheduled to be presented in a public online[1] PhD defence in the weeks following RLDM-2022.

A neoRL learning module can be considered as a behavioral node in a purposive network. Early results for neoRL navigation explored the effect of considering multiple state spaces in parallel for the neoRL agent. In some ways analogous to the Hybrid Reward Architecture [10], the neoRL navigation agent combines several learners that establish GVFs toward separate concerns [2]. From applying the superposition principle in the value domain, the neoRL agent is capable of combining the value function from many learners in one state space [1], across multiple state representations of one information space [2], or across information represented by orthogonal Euclidean spaces [3]. The distributed neoRL value function thus allows a decomposed Markov state, facilitating online learning and autonomous navigation. Each behavioral node consists of three parts; first, the latently learned cognitive map formed by GVF on operant desires toward NRES cells. Operant desires are trained by off-policy GVF, expressing latent learning for how to accomplish separate conditionals in this environment representation. Second, the neoRL agent is governed by purpose – mental projections of parameter configurations associated with reward are expressed as elements-of-interest in the NRES map. These free-ranging representations of desire in the considered space are mapped to NRES nodes, activating GVFs corresponding to the associated valence – the element's expectancy of reward upon achievement. Note that the same neoRL node, containing latent knowledge in one NRES representation, can be harvested by different sets of elements-of-interest. Third, the value function can be extracted by elements-of-interest from the digital analogy to Tolman's cognitive map – resulting in an actionable Q-vector output of the neoRL node. When mutually exclusive NRES receptive fields are used for latent learning, the GVF components become operant toward that NRES cell – operant value function components. The singular (orthogonal) value component can be combined with others to form the full value function of the agent, further implying that separate navigational modalities can be learned in parallel and combined to one agent value function [3]. Figure 2 shows the aggregation of the value function for the neoRL agent in [2]. The location and valence of elements-of-interest can be considered as inputs to the neoRL behavioral node, and the superposition of weighted operant GVFs establishes an output of the neoRL node.

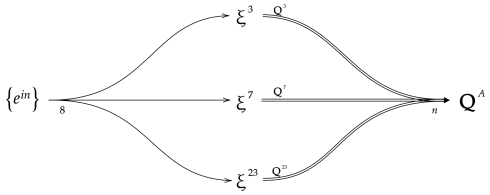


Figure 2: **A schematic representation of fig. 1,** the neoRL agent from [2]. Agent value function is formed from the combined value function Q^N from each of the three ξ^N neoRL nodes, where $N \in \{3, 7, 23\}$ [2].

[1]Information about the streamed PhD defence will be posted at *www.neoRL.net*

The value function of the neoRL agent is composed across multiple state representations[1], as illustrated in figure 2. The separate sub-agents ξ^3, ξ^7 and ξ^{23} are thus independent sources for the neoRL value function, each formed by three functional parts: the *input* to the node are comprised of elements-of-interest, $\{e^{in}\}$, each representing a parameter configuration associated with reward; the *latent map*, reflecting operant policies toward features in the considered state set, formed by latent learning as expressed as GVFs on the agent's stream of experience; and the actionable state-action value-vector as the *output* of the neoRL node, established as the integral across the operant desires activated by elements-of-interest. When considering an action set where choices have a Euclidean significance, as with $\mathbb{A} = \{N, S, E, W\}$ in [1], a state-action value vector could be interpreted a Euclidean *desire-vector*;

$$\vec{d} = \vec{\sum} \mathbf{Q}^{in} \quad ,$$

where $\vec{\sum}$ represents the vector sum. The valence of the desire vector $\vec{d}$ should express the combined valence across $\{e^{in}\}$.

$$e^{\vec{out}} = \vec{d}$$
$$e^{out}_{\psi} = \sum e^{in}_{\psi} \ ,$$

where $\vec{e^i}$ signifies the coordinate and e^i_{ψ} represents the valence of purposive element i. A functional schematic of the neoRL module is illustrated in figure 3; a single output-desire can be formed from a number of input elements $\{e^{in}\}$. Different sets of purposive elements-of-interest $\{e^{in}\}$ can establish different output desire e^{out} and actionable state-action values Q^N from a single cognitive map. The network expressed by figure 2 could be seen as a behavioristic analogy to the one-layered perceptron (from [5]). Likewise, deeper behavioristic networks are plausible by multi-layered neoRL graphs with purposive flow of desire. Tolman's purposive behaviorism and latent learning could be expressed by deep neoRL networks — allowing for autonomous navigation governed by the agent's own experience and expectancy of reward.

2 Experiments

A comprehensive environment for research on autonomous navigation is the PLE implementation [8] of Karpathy's WaterWorld environment. An agent controls acceleration of the self in the four Cardinal directions $[N, S, E, W]$, i.e., [up, down, right, left]. Three objects move around in the Euclidean plane according to predefined mechanics. Encountering an object replaces it with a new object with a random color, location, and speed vector. Green objects are desirable with a positive reward $\mathbb{R} = +1.0$, and red objects are repulsive with a negative valence of -1.0. Capturing the last green object resets all remaining (red) objects by the same reset mechanisms. The only reward comes from encountering objects, making the accumulation of $\mathbb{R}$ an objective measure for navigational capabilities and its time course an indication of real-time learning capabilities.

The sparse reward scheme in the WaterWorld environment, with discrete $+1.0$ or -1.0 rewards after encountering objects, makes measuring the immediate proficiency of the agent by $\mathbb{R}$ difficult. Capturing the transient time course of agent skill can be done by averaging independent runts; the enclosed experiments average 100 separate runs to measure transient navigational proficiency, i.e., across 100 separate agents. No pre-training or other precursors are available for the agents, making all navigation happen live as the agent gathers experience in the environment for the first time. Curves are presented with minutes along the x-axis, signifying the wall-clock time since the beginning of each run.

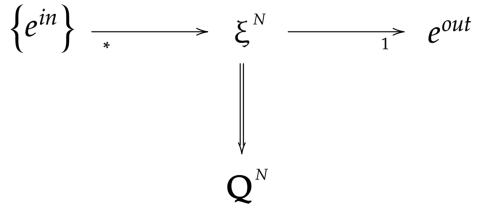
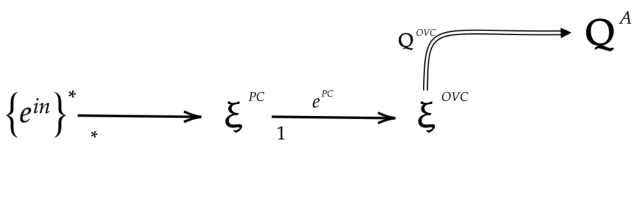
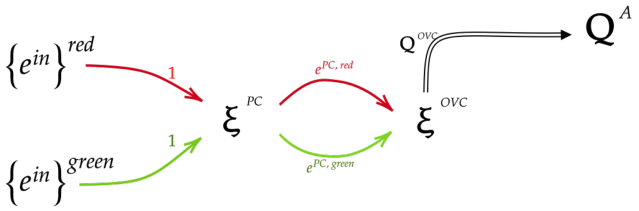


Figure 3: **A neoRL learning module with one input and two output;** actions with a Euclidean significance implies state-action vectors to be representable in the same Euclidean space; e^{out} can be used as input for compatible neoRL nodes.

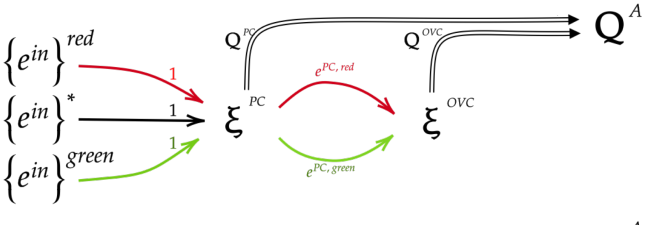
We shall explore four aspects of purposive graphs; first, we challenge the basic principle of propagating purpose by the layout illustrated in figure 4**a**. The first neoRL node, ξ^{PC}, forms a purposive desire based on all reported objects from WaterWorld; a single desire vector $\vec{d}$ with accompanying valence e_{ψ} propagates to the compatible neoRL node ξ^{OVC} as autonomous desire e^{PC}. *Experiment **[b]*** explores the effect of extracting separate desires from the same learned cognitive map. A purposive desire vector from the neoRL node comes from extracting latent knowledge from considered coordinates; the agent extracts two purposive vectors from ξ^{PC}, one from desirable objects $e^{in_{green}}$ and a separate from aversive objects $e^{in_{red}}$. The output from multiple neoRL nodes can be combined for the policy-forming value function in the neoRL framework; *experiment **[c]*** explores the effect of aggregating value function from multiple depths of the neoRL net. The output desire from one node ξ^N can give input to any compatible neoRL node ξ^M, including itself: *experiment **[d]*** explores recurrent connections for neoRL nodes. Collaborative experience is explored with and without recursive desires, as illustrated in figure 4c and 4d. All results are reported in figure 4e.



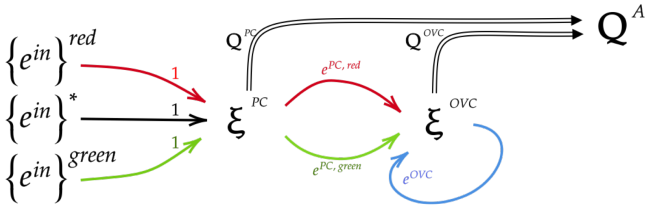
(a) Aspect 1: a single desire vector e^{PC} as input to ξ^{OVC}.



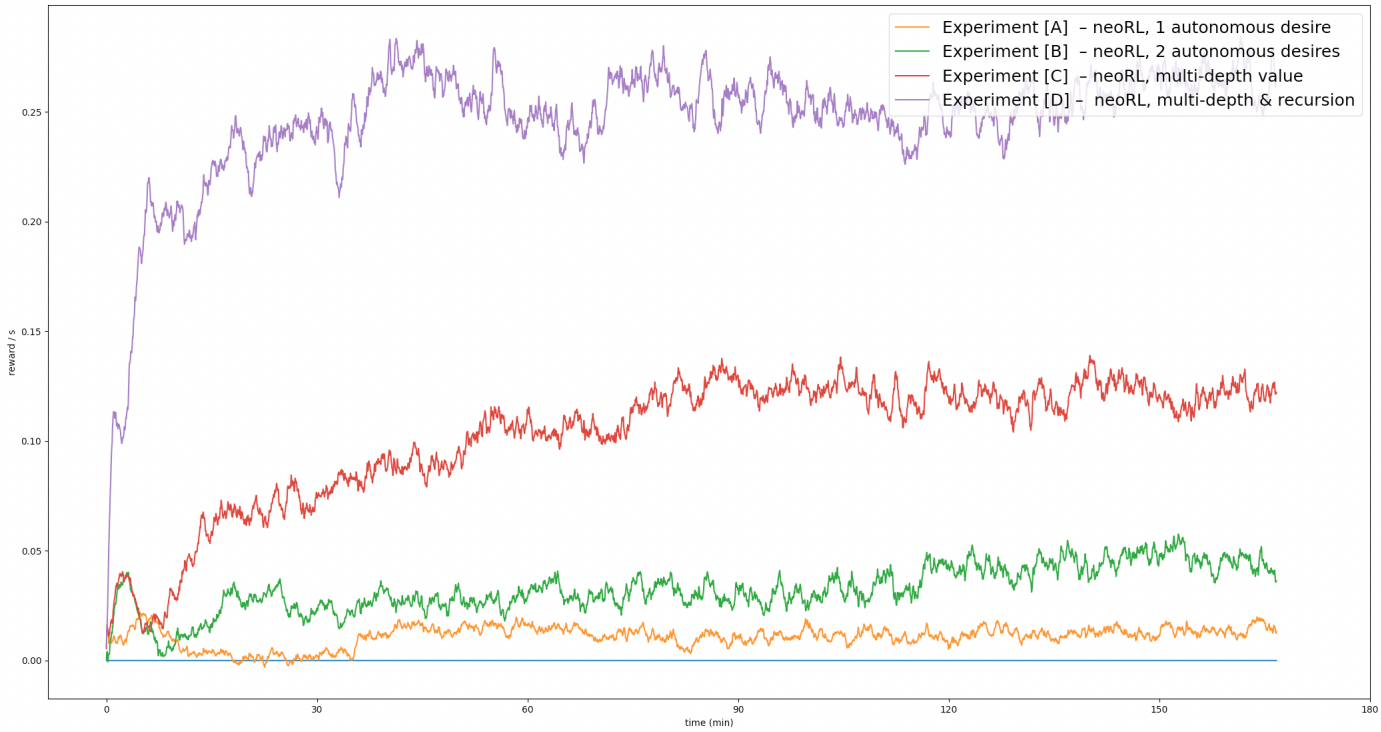
(b) Aspect 2: separate desire extraction; $e^{PC_{red}}$ and $e^{PC_{green}}$.



(c) Aspect 3: joint value function from sequential neoRL nodes.



(d) Aspect 4: recursive desires for neoRL autonomy.



(e) Transient proficiency of neoRL agent A-D.

Figure 4: [[**Top**]] Illustrations of the neoRL architecture tested in experiment A-D. [**a**] A first attempt on desire from experience; neoRL node ξ^{PC} forms a single desire e^{PC} for value-generating neoRL node ξ^{OVC}. [**b**] Latent knowledge can be extracted separately for separate classes for desire; experiment B forms two desire-vectors $e^{PC_{red}}$ and $e^{PC_{green}}$ from ξ^{OVC} – grouping according to valence. [**c**] The value function output from neoRL node ξ^{PC} and node ξ^{OVC} contribute equally to agent value function. [**d**] Recursive desires are possible for neoRL nodes: the ξ^{OVC} is governed by three elements-of-interest, $e^{PC_{red}}$, $e^{PC_{green}}$, and recurrent desire e^{OVC}. [[**Down**]] Results from the four experiments: [**e**] Purposive neoRL networks allows for purposive autonomy by deep and/or recurrent desires.

3 Discussion

The neoRL agent navigates continuous space by projections of desire, vectors of purpose associated with an agent's expectancy or reward. When actions have a Euclidean significance, purposive Q-vectors can form autonomous projections of desire, experience-based inferences that can establish input to deeper neoRL nodes. Experiments demonstrate how deeper or recurrent desires are crucial for navigational proficiency, suggesting purposive neoRL networks as a plausible approach to autonomous navigation.

This work explores four principles of purposive neoRL networks. *First*, experience-based desire vectors can shape purposive navigation in Euclidean space. The neoRL network from illustration 4a improves navigational proficiency over time; however, considering all objectives under one, i.e., forming an autonomous desire based on all red and green objects, appears to be too compressed to express proficient navigation. *Second*, a single neoRL node can generate different e^{out} desires by considering different sets of objectives $\{e^{in}\}$. Experiment **b** demonstrates the effect of separating desires according to valence, resulting in an increased performance by the neoRL navigational agent. The simplicity and clarity expressed by the separation of desires, as illustrated in figure 4b, facilitates the explainability of the trained solution. *Third*, the neoRL agent can base agent value function on any neoRL node in the network. Agents extracting purpose from multiple depths of the neoRL network, as illustrated in 4c, become better navigators than shallow agents. *Fourth*, desire vectors e^{out} from one neoRL node can form objectives for any compatible neoRL node – including itself. Experiment [d] explores recursive desires, where the output of ξ^{OVC} – desire vector e^{OVC} – establish an additional purposive input to neoRL node ξ^{OVC}. The substantial increase in navigational performance by the simple recursive connection from 4d indicates the potential of purposive flow in autonomous navigation. Results can be examined in figure 4e and in real-time video demonstrations published at *www.neoRL.net*

Note that no comparison has been made with alternative approaches for control; this work is only concerned with uncovering the basic principles of purposive neoRL nets for behavioral AI. Still, any attempt on finding RL or AI solutions capable of allocentric Euclidean navigation in real-time has failed. Further work could involve finding and comparing alternative approaches for real-time autonomous navigation in the WaterWorld environment. Likewise, this work involves no search for optimal parameters for neoRL navigation. Experiment [c] explores collaborative experience with 1:1 weight ratio between ξ^{PC} and ξ^{OVC}, and experiment [d] only explores the unitary feedback loop with $w_r = -1.0$. It is left for further work to explore purposive network theory or automate parameter search by data-driven methods from narrow AI. We have barely scratched the surface of autonomous navigation by purposive graphs, demonstrating four identified principles in neoRL networks. A deep or recursive flow of desire, here demonstrated for neoRL networks, might hold the key to genuine autonomy in decision making by digital agents.

References

[1] Per R. Leikanger. Modular RL for real-time learning. In *The 3rd Conference on Cognitive and Computational Neuroscience*. Cognitive and Computational Neuroscience, 2019.

[2] Per R. Leikanger. Decomposing the Prediction Problem; Autonomous Navigation by neoRL Agents. volume ALIFE 2021: The 2021 Conference on Artificial Life of *ALIFE 2021: The 2021 Conference on Artificial Life*, 07 2021. 30.

[3] Per R. Leikanger. Navigating conceptual space; a new take on AGI. In *International Conference on Artificial General Intelligence*, pages 116–126. Springer, 2021.

[4] Per R. Leikanger. *Autonomous navigation in the (animal and the) machine*. PhD thesis in review, UiT, 2022.

[5] Frank Rosenblatt. The perceptron, a perceiving and recognizing automaton project para. Technical report, 1957.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[8] Norman Tasfi. Waterworld in pygame learning environment. https://github.com/ntasfi/PyGame-Learning-Environment.

[9] Edward Chace Tolman and Charles H Honzik. Degrees of hunger, reward and non-reward, and maze learning in rats. *University of California Publications in Psychology*, 1930.

[10] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5392–5402, 2017.

Heterogeneous Representations of Variables in Task-Optimized DRL Agents Depend on Task-Relevance

Dongyan Lin
McGill University
Mila
Montreal, QC, Canada
lindongy@mila.quebec

Ann Zixiang Huang
McGill University
Mila
Montreal, QC, Canada
zixiang.huang@mail.mcgill.ca

Blake A. Richards
McGill University
Mila
Montreal, QC, Canada
blake.richards@mila.quebec

Abstract

In biological organisms, the ability to integrate information about space, time, and sensory stimuli is crucial for many cognitive functions, including episodic memory, working memory, and decision making. Neurophysiological studies have found that these external variables are heterogeneously represented in the brain by neurons whose firing rates are conjunctively modulated by these variables. But why do these representations emerge in a biological neural network, and how do they contribute to higher-order cognitive functions? In this work, using deep reinforcement learning agents trained on simulated working memory tasks, we provide a normative model in which heterogeneous representations of external variables naturally emerge in agents optimized on cognitive tasks. Moreover, by altering the task demand, we show that the conjunctive modulation of neural activities by these variables depends on their task-relevance. Our findings link cognitive models of neuroscience with normative models of reinforcement learning, and provide concrete experimental predictions for future studies.

Keywords: Neural representations, Working memory, Place cells, Time cells

Acknowledgements

This research is supported by grants to B.A.R from NSERC (Discovery Grant: RGPIN-2020-05105; Discovery Accelerator Supplement: RGPAS-2020-00031), Healthy Brains, Healthy Lives (New Investigator Award: 2bNISU-8), and CIFAR (Canada AI Chair; Learning in Machine and Brains Fellowship). Additionally, D.L. was supported by an IVADO Excellence Scholarship (MSc-2020-0435588519) and a Healthy Brains, Healthy Lives PhD Fellowship. We thank all members of the LiNC Lab for their support.

1 Introduction

The ability to remember "what happened when and where" is central to many cognitive functions of biological intelligence, such as episodic memory, working memory, and decision making. Such ability relies on robust neural representations that can integratively encode sensory stimulus, time, and location. In mammalian brains, neuroscientists have discovered neurons that consistently fire when the animal occupies a certain location in space (i.e., "place cells") [1] or at a particular time (i.e, "time cells") [2], which are theorized to construct a cognitive "map" to help scaffold the representations of sensory stimuli. As evidence for this, both place cells and time cells representations have been shown to be modulated by sensory context: the firing rate of place cells at a particular position has been shown to be modulated by where the animal has been [3], and the firing rate of time cells at a particular time has been shown to be distinct following the presentation of different sensory stimuli to the animal [4].

Given these biological observations, many studies have used computational modeling of time cells and place cells to better understand their contributions to higher-order cognitive functions. Earlier models focused on reproducing these spatiotemporal representations by either handcrafting properties of these representations into the network design [5][6] or replicating the anatomical structures that give rise to these representations [7]. One caveat of these models is that they are uninformative about how the brain uses these representations to perform downstream tasks. To address this caveat, more recent works have shifted their focus to training the network end-to-end on simulated cognitive tasks to demonstrate the natural emergence of these spatiotemporal patterns [8][9]. While these types of models have shown competence in linking neural representations to cognitive functions, they have focused on single types of representation and overlooked the interactions between the modulations of multiple different variables on the same neural population.

Given these limitations in the current literature, in this study, we trained deep reinforcement learning (DRL) agents on a simulated working memory task that required the agent to integrate information about time, space, and sensory stimuli, and analyzed the hidden-state activities of recurrent units using neuroscience-inspired methods. We hypothesized that the heterogeneous representations of "what happened when and where" can simultaneously emerge in a parallel distributed processing system optimized on simulated cognitive tasks. Moreover, we hypothesized that the emergence of representations of behavioral variables will depend on the relevance of such variables to the task.

2 Methods

We simulated the delayed nonmatch-to-sample (DNMS) task (Figure 1A), an episodic working memory task commonly used in cognitive neuroscience studies. We embedded the DNMS task in a 2D simulation chamber that resembled the gridworld environment. Agents could move up, down, left, or right, and could also "nose-poke" to interact with elements of the simulated chamber. A complete episode of the task consists of five stages: 1) At the beginning of each episode, the agent encounters an initiation signal, which it must navigate to and poke with in order to proceed to the sample phase. 2) During the sample phase, the sample will be displayed in one of two possible locations: either on the left or right corner of the triangular arena, and the agent must poke the sample. 3) Upon poking the sample, all signals are turned off, and the delay period of 40 simulation time steps starts. The end of the delay period is indicated by the onset of the initiation signal again, with which the agent must poke in order to proceed to the choice phase. 5) During the choice phase, both left and right sample locations are presented. If the agent pokes the location not displayed during the sample phase (i.e. "nonmatch"), it will receive a reward and the episode ends. This task requires the agent to remember the location of the sample throughout the delay period in order to choose correctly during the choice phase after the delay. To maximize the reward, the agent must not only remember the sample, but also navigate to desired locations in the shortest path possible without taking redundant actions. As such, to optimize the reward, the agent must learn to integrate information about location, time, and sensory stimuli (i.e., sample).

The DRL agents used in our study (Figure 1B) were composed of a visual module, a memory module, and an actor-critic module. We used a deep convolutional neural network as the visual module to generate a latent representation of the visual input, a three-channel RGB image of the environment state as described in the previous section. The output of the visual module was passed to the memory module, which consisted of an LSTM layer with 512 hidden units and a linear layer with 512 hidden units. The output of the linear layer was then fed forward to a value network and a policy network, which generated an estimate of state value $\hat{V}(S_t, \theta)$ and a stochastic policy $\pi(a_t|S_t, \theta)$ from which the action will be sampled, respectively. We used an actor-critic algorithm, in which the network parameters were adjusted to minimize the loss $L = L_\pi + L_V$ where

$$L_\pi = \sum_{t=0}^{T-1} -log[\pi(a_t|S_t, \theta)] * [R_t - \hat{V}(S_t, \theta)] \tag{1}$$

$$L_V = \sum_{t=0}^{T-1} l_1[\hat{V}(S_t, \theta), R_t] \tag{2}$$

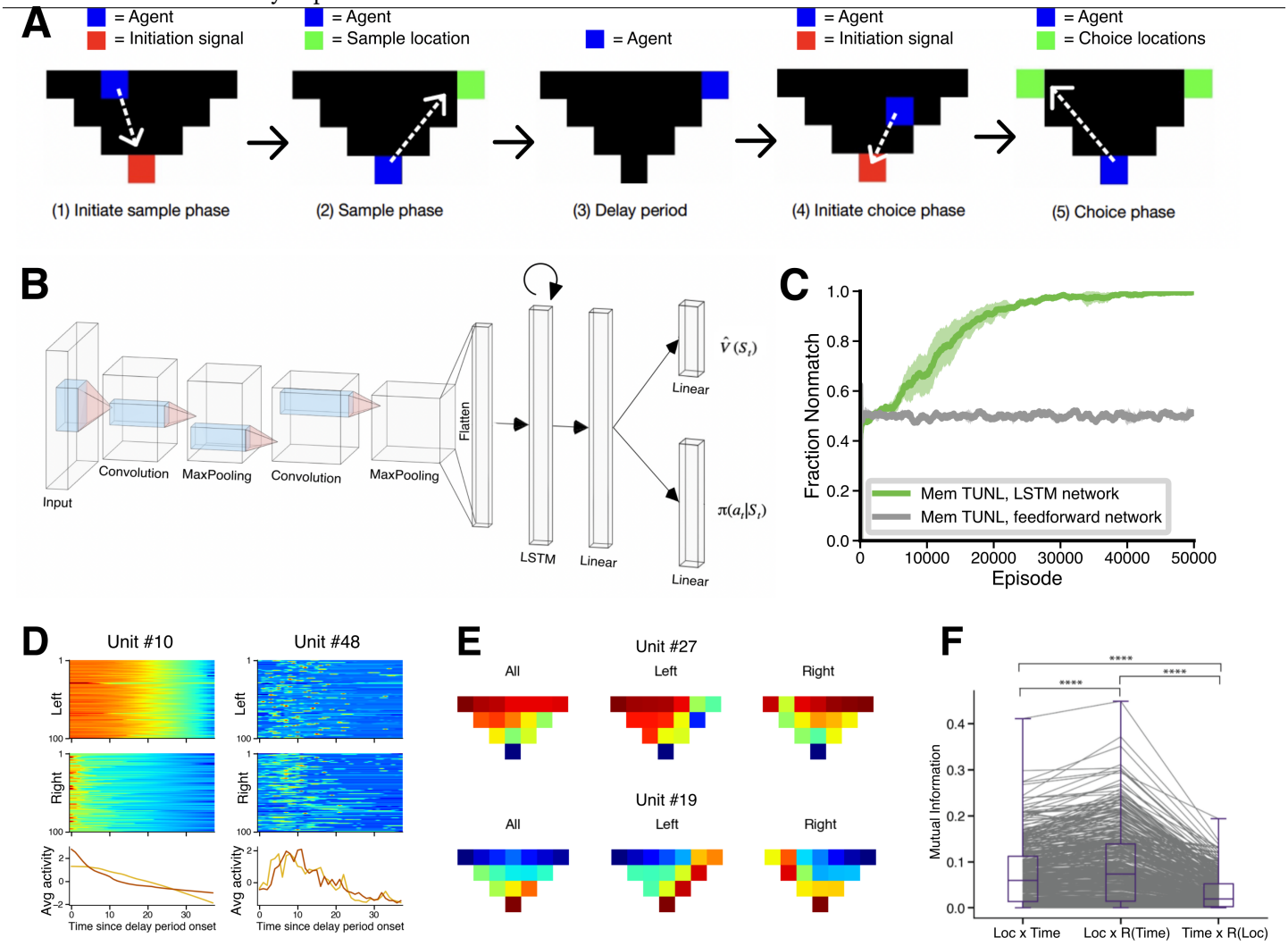


Figure 1: **DRL agents trained on working memory task exhibited conjunctive coding of time, space, and stimulus. A)** Schematic illustration of the task structure in one trial of DNMS task. **B)** Architecture of the DRL agent. **C)** The agent achieves almost perfect performance on the DNMS task in approximately 30000 episodes (green). In contrast, a feed-forward agent without any recurrent connections chooses nonmatch at random (grey). Performance is measured as the fraction of choices that are nonmatch to sample. Solid line and shaded area represent the average and standard deviation of performance over 4 seeds, respectively. **D)** Two example LSTM units that exhibited stimulus-specific temporal tuning as a conjunctive representation of stimulus and elapsed time. Each panel shows, from top to bottom, a heatmap of z-normalized hidden state activities during the delay period in 100 left-sample trials and 100 right-sample trials (blue and red represent the lowest and highest hidden state activities during the delay period of each trial, respectively), and the temporal tuning curves averaged over the 100 left-sample trials (yellow) and 100 right-sample trials (brown). **E)** Two example LSTM units whose hidden state activity exhibited stimulus-specific spatial tuning as a conjunctive representation of stimulus and location. Each panel shows the unit's trial-averaged activity when the agent is at different locations under all trials, left-sample trials, or right-sample trials. **F)** The amount of mutual information between the hidden unit activities and occupancy in a two-dimensional location-by-time (Loc x Time) space (left column), compared to the amount of mutual information if the units encode location (middle column) or time (right column) independently. R(Time) and R(Loc) indicate randomized temporal and spatial dimensions, respectively. Only units that contain significant amount of mutual information in the Loc x Time space were included in the analysis (N=421).

where $t = 0, 1, ..., T-1$ index the time steps in an episode with T experiment steps, $R_t = \sum_{i=0}^{t} \gamma^i r_{t-i}$ denotes the discounted return at t, and l_1 is the smooth L1 loss implemented in PyTorch.

For all tasks, we used a discount factor γ=0.99. The model parameters were updated using gradient descent and Adam optimizer with β_1=0.9, β_2=0.999, ϵ=1e-8, batch size=1, and learning rate of 10^{-5}.

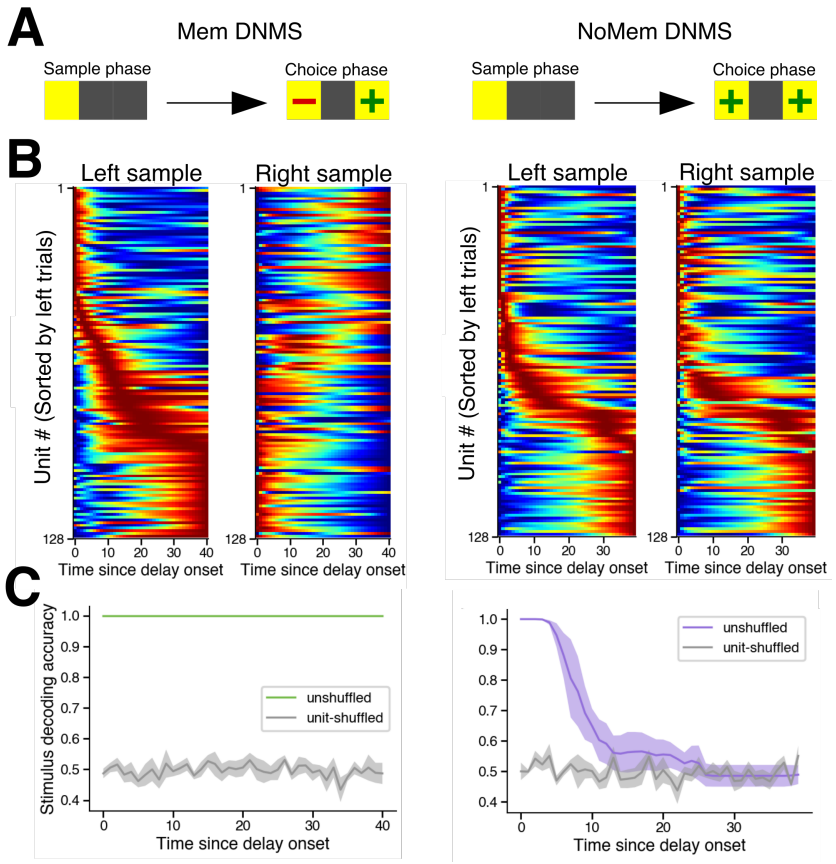


Figure 2: **Task-irrelevant sensory stimuli minimally modulate temporal representations.** **A)** The schematic illustration of mnemonic DNMS task ("Mem DNMS", left panel) and non-mnemonic DNMS task ("NoMem DNMS", right panel). **B)** Ensemble neural activity during the delay period for left- or right-sample trials under the mnemonic (left panel) or non-mnemonic (right panel) condition. Each heatmap shows the trial-averaged hidden state activities of all LSTM cells, normalized to each cell's minimum (blue) and maximum (red) activity. In all heatmaps, cells are sorted by the latency to peak hidden state activity during the left-sample trials under the corresponding task condition. **C)** SVM decoding of the sample displayed prior to the delay period from population activities at each time step during the delay period of Mem DNMS (left panel, green) and NoMem DNMS (right panel, purple). Decoding accuracy is measured by the fraction of trials decoded correctly. Decoding accuracies from unit-shuffled population activities are plotted in grey and serve as chance baseline. Solid lines and shaded areas represent mean and standard deviation of accuracies across 5 cross-validation folds.

3 Results

3.1 DRL agents trained on working memory task exhibited conjunctive representations of time, space, and stimulus

After training, the DRL agent was able perform the DNMS task well (Figure 1C). To examine the representations of task-relevant variables by the hidden state activities of LSTM cells, we adopted several analyses commonly used to analyze the neuronal activities in the brain. First, inspired by the observation of time cells, we investigated the how the elapsed time since delay onset can modulate the hidden state activity of LSTM units. We found that the temporal receptive fields of individual LSTM "time cells" were not only trial-reliable, but also different under the left- versus right-sample conditions, suggesting a mechanism by which LSTM cells conjunctively encode time and sensory stimuli (Figure 1D). Second, using analysis methods commonly used in place cell studies, we also found that LSTM cells have preferred location of firing in the simulated chamber, and that their spatial encoding is in conjunction with the encoding of sensory stimuli, as suggested by the differential spatial tuning following different samples (Figure 1E). We further confirmed the conjunctive coding of time and space using mutual information, where we showed that shuffling the spatial dimension (Time x R(Loc)) significantly reduces the mutual information, indicating the joint encoding of location and time in the LSTM unit activities. Interestingly, we see an increase in mutual information when the temporal dimension is randomized (Loc x R(Time)) (Figure 1F), which suggests that space is a stronger regulator of neural activity than time. Therefore, by optimizing the DRL agent on the cognitive task, we showed that the same population of LSTM units can simultaneously encode space, time, and sensory information.

3.2 Neural representations of sensory stimuli depend on task-relevance

Next we asked: how do these representations contribute to the cognitive function? Specifically, when the sensory stimulus is irrelevant to working memory, is it still represented in the same way by the neural activities? To answer these questions, we examined the sensory modulation of temporal and spatial representations separately. First, to investigate whether time representations are still stimulus-specific when the stimulus is task-irrelevant, we simulated a non-mnemonic version of the DNMS task wherein, after the delay, the agent can choose either nonmatch or match location to receive a reward (Figure 2A), in a location-fixed environment to avoid confounding by spatial variables. We then trained new DRL agents on the NoMem DNMS task and contrasted how the identity of the stimulus (i.e., left or right) is

represented by the activities of LSTM cells under the mnemonic and non-mnemonic conditions. As expected, under the mnemonic condition, the identity of the sensory stimuli are represented by the LSTM hidden state dynamics throughout the delay, as suggested by the different orders of sequential activation of the neural ensemble during the delay period (Figure 2B, left). In contrast, when the task does not require the agent to remember the identity of sensory stimulus (i.e., non-mnemonic), different sensory stimuli were represented by almost identical neural dynamics, suggesting a lack of neural representation for the sensory stimulus when it is task-irrelevant (Figure 2B, right). We further quantified this by constructing a binary support vector machine (SVM) to decode the identity of the sample in each trial from hidden state activities of LSTM cells at each single time step during the delay period from that trial. We found that the decoding accuracy gradually decreased to chance level when the agent does not need to remember the sensory sample, suggesting that the representation of sensory stimuli is gradually lost when they are irrelevant to the task (Figure 2C, right). In contrast, when the agent is required to remember the sample across the delay, the sensory representation remained intact (Figure 2C, left).

Having established that temporal representations are not modulated by the sensory context when the context does not contribute to working memory, we next asked whether it is the same for spatial representations. To this end, we embedded the NoMem DNMS task in the simulated chamber where the agent can freely move, and we compared the location at which each LSTM hidden unit activity peaked in left-sample trials vs. right-sample trials as an estimate of stimulus-specificity of place coding. We found that the percentage of LSTM units whose place coding is modulated by the sensory stimulus is similar for the mnemonic (104/321, or 32.4%) and non-mnemonic (76/259, or 29.3%) conditions. This suggests that the sensory scaffolding of place cells may arise naturally and independently of the task requirements.

4 Discussion & Future Work

In this work, we present a normative model using DRL to demonstrate that heterogeneous representations of space, time, and stimulus can simultaneously emerge in the same population of recurrent units in agents trained to optimize a single reward function. Moreover, by simulating a control task without the cognitive demand of working memory, we show that the temporal and spatial representations are differentially modulated by the sensory stimuli, and such modulation depends on the working memory demand. We note that there are a few limitations to our model. First, our network is not a physiological model; it is inspired by, but does not replicate, the anatomy and function of the hippocampus. Second, unlike animals, our network is trained from scratch without any pre-wired, highly structured brain connectivity, which explains the large number of training episodes needed for the agent to learn the task compared to training animals on the same task. Third, the DNMS tasks in this study are simulated toy models of the real experiments and therefore may not fully capture the complexity of cognitive tasks. Nonetheless, our study successfully reproduced the heterogeneous representations commonly observed in the real brain, and provided a concrete prediction for future neuroscience experiments: the task-relevance of sensory information will determine the extent to which they modulate the representation of other variables. Moreover, it provided a DRL training scheme that, in the future, could afford the opportunity to perform in-silico lesion experiments to evaluate the causal roles of these heterogeneous representations in performing cognitive tasks.

References

1. O'Keefe, J. & Nadel, L. *The hippocampus as a cognitive map* en (Clarendon Press ; Oxford University Press, Oxford : New York, 1978).
2. MacDonald, C. J., Lepage, K. Q., Eden, U. T. & Eichenbaum, H. Hippocampal "Time Cells" Bridge the Gap in Memory for Discontiguous Events. en. *Neuron* **71,** 737–749 (Aug. 2011).
3. Wood, E. R., Dudchenko, P. A., Robitsek, R. J. & Eichenbaum, H. Hippocampal Neurons Encode Information about Different Types of Memory Episodes Occurring in the Same Location. en. *Neuron* **27,** 623–633 (Sept. 2000).
4. Taxidis, J. *et al.* Differential Emergence and Stability of Sensory and Temporal Representations in Context-Specific Hippocampal Sequences. en. *Neuron* **108,** 984–998.e9 (Dec. 2020).
5. Howard, M. W. *et al.* A Unified Mathematical Framework for Coding Time, Space, and Sequences in the Hippocampal Region. en. *Journal of Neuroscience* **34,** 4692–4707 (Mar. 2014).
6. Rajan, K., Harvey, C. D. & Tank, D. W. Recurrent Network Models of Sequence Generation and Memory. eng. *Neuron* **90,** 128–142 (Apr. 2016).
7. Solstad, T., Moser, E. I. & Einevoll, G. T. From grid cells to place cells: a mathematical model. eng. *Hippocampus* **16,** 1026–1031 (2006).
8. Orhan, A. E. & Ma, W. J. A diverse range of factors affect the nature of neural representations underlying short-term memory. en. *Nature Neuroscience* **22.** Number: 2 Publisher: Nature Publishing Group, 275–283 (Feb. 2019).
9. Banino, A. *et al.* Vector-based navigation using grid-like representations in artificial agents. en. *Nature* **557.** Number: 7705 Publisher: Nature Publishing Group, 429–433 (May 2018).

Comparing Machine and Human Learning in a Planning Task of Intermediate Complexity

Xinlei (Daisy) Lin*
Center for Neural Science
New York University
New York, 10003, NY, USA
xl1005@nyu.edu

Zheyang (Sam) Zheng*
Center for Neural Science
New York University
New York, 10003, NY, USA
zz737@nyu.edu

Jake Topping*
University of Oxford
United Kingdom Imperial College London
jake.topping@nyu.edu

Wei Ji Ma
Center for Neural Science and Department of Psychology
New York University
New York, 10003, NY, USA
weijima@nyu.edu

*** These authors contributed equally to this work**

Abstract

Deep reinforcement learning agents such as AlphaZero have achieved superhuman strength in complex combinatorial games. By contrast, the cognitive science of planning has mostly focused on simple tasks, for experimental and computational tractability. To allow for direct comparisons between humans and artificial intelligence, we need tasks with intermediate complexity. Therefore, we trained AlphaZero on 4-in-a-row, a variant of tic-tac-toe in which human planning has previously been modeled. We characterize AlphaZero's performance using three derived metrics: planning depth, policy quality and value function quality; the latter two are derived based on the objective value of a state. We find that AlphaZero agents improve in value function quality and planning depth through learning, similar to human in previous modeling work. In addition, these metrics reflect causal contributions to AlphaZero's playing strength, and the strongest contributor is the policy quality. The decrease in policy entropy also drives the increase in planning depth. The contribution of planning depth to performance is lessened in late training. These results contribute to a joint understanding of machine and human planning, providing an interpretable way of understanding the learning and strength of AlphaZero, while generating novel hypothesis on human planning.

Keywords: Human-DNN comparison; Planning; Learning; Interpretable Machine Learning;

1 Introduction

There has long been a positive mutual influence between research on artificial intelligence (AI) and research on human intelligence (Turing, 2009; Lake, Ullman, Tenenbaum, & Gershman, 2017). However, such mutual influence is less common in the field of planning, defined as a cognitive process in which the decision-maker mentally simulates future states, actions or outcomes in a decision tree (van Opheusden & Ma, 2019). AI research has focused on solving complex tasks like Chess (Silver et al., 2018) and Go (Silver et al., 2017), while cognitive science and psychology have favored detailed modeling using simple tasks like the two-step task (Daw, Gershman, Seymour, Dayan, & Dolan, 2011), and the Solway and Botvinick task (Solway & Botvinick, 2015).

The tasks used in studies comparing AI and humans still lie on either extreme (Wang et al., 2018; Tian et al., 2019; McGrath et al., 2021). We aim to complement the comparative work between AI and humans by using a task that is challenging enough for both humans and artificial agents while being computationally tractable. We train AlphaZero agents to learn 4-in-a-row. Using a combination of observation and causal manipulations, we show that planning metric improvements mediate the increase in playing strength. We also find phenomena that have not been reported in human study. Policy quality contributes the most to performance. The increase in planning depth is mediated through a decrease in the entropy of the policy, reflecting a more concentrated search process. The contribution of planning depth to playing strength diminishes at later stages of learning. These discrepancies might either reflect mechanistic differences between machine and human planning, or point to phenomena only observable in the more extreme end of the spectrum in human expertise, or highlight alternative hypotheses about human planning. Our result not only provides a cognitive account for the playing strength and learning of AlphaZero, but also inspires future research directions in human planning.

2 Methods

2.1 Task

The four-in-a-row task is a generalization of tic-tac-toe. Two players alternate placing pieces on a 4-by-9 board; black moves first. The goal is to have four pieces of one's color in a row horizontally, vertically, or diagonally.

2.2 Agents

The deep RL agents trained to play 4-in-a-row are slightly modified versions of AlphaGo Zero (Silver et al., 2017) and AlphaZero (Silver et al., 2018). AlphaZero type deep RL agents use a deep neural network (DNN) to guide its Monte-Carlo tree search (MCTS). Given the input (a board and optionally the player's color), the DNN returns an immediate value v of a board, indicating how likely the current player will win/lose from this board. The DNN also returns a policy $\mathbf{p}$ (or $P(s,a)$ for a board s and move a), a prior probability of selecting an action before doing any tree search. MCTS simulates future actions and states in a way that balances exploration and exploitation. It then integrates those results to inform current decision.

For each training iteration, we use the current best agent to play 100 games against itself to generate the training examples. Each training example contains a tuple of (board positions s, MCTS output $\pi(s,a)$, game outcome r). The DNN outputs $(\mathbf{p},v)$ are trained to match the game result r under a mean-squared error loss and the action probabilities π under a cross-entropy loss, with $l2$ weight regularization. The DNN parameters are optimized by the Adam Optimizer. The updated network will play 30 games against the current best network. If the updated network can win more games than it loses, it will be accepted and become the new current best network for data generation and network comparison. We use this looser selection criterion compared to AlphaGo Zero to encourage easier update of networks. Because if the updated network is not accepted, we revert it back to the current best network, forgoing the parameter update (different from AlphaGo Zero).

Using thirteen sets of hyperparameters, we produce thirteen Networks that show a diversity in the playing strength and planning metrics (value function quality and planning depth) (1). We call models sharing the same hyperparameter set and initialization during training a "Network", to distinguish the concept from an individual iteration saved during training, which we call an "agent" or a "model."

2.3 Measuring playing strength

We hold a tournament in which each agent plays against every other agent once as both colors. There are 789 agents in total, including all accepted iterations from the thirteen Networks, as well as the agents whose N_{MCTS} have been modified, and those whose value or quality functions have been swapped. The temperature is fixed at 0.1. Playing strength is quantified by Elo ratings, computed by the BayesElo program (Coulom, 2008). The Elo ratings are computed such that the difference between the Elo ratings of two players maps monotonically to the probability that one player will defeat the other.

2.4 Probe boards and game-theoretic values

The probe boards are all positions (5482 positions) which occurred in human-vs-human experiments conducted by van Opheusden et al. (2021). The game-theoretic values of these boards are defined as game outcomes in which both sides play perfectly. van Opheusden et al. (2021) approximated the game-theoretic values by searching each board for 200,000 iterations using the cognitive model. The result for most boards converges to a game-theoretic value, while the undetermined ones are assigned a 0 value, indicating a draw. We used these pre-computed game-theoretic values for our value quality calculation.

2.5 Policy quality

After computing the game-theoretic values for each child board of all probe boards (used in value quality and depth calculation), we applied softmax to the values of those children boards to get an "optimal" policy for each probe board. We then concatenate these optimal policies of all probe boards, and correlate the long vector with the concatenated policy vector returned by a DNN.

3 Results

3.1 Playing strength increases

AlphaZero's playing strength increases over training across all Networks (Figure 1A; left). To obtain a human benchmark, we have the strongest human player we could find play 4 games each against 8 selected agents, with Elo ratings ranging from 140 to 242 (the best). Agents at middle training iterations already start to surpass the human bench mark, with later agents lying above the 95-confidence interval. Human learning curve from the previous study is recreated here (1A; right). Our question is what aspects of the agents' capacity have improved to enable such an improvement in playing strength.

3.2 Evidence for smarter trees

For each AlphaZero agent, we compute the value function quality by calculating the Pearson correlation between the DNN-returned immediate values of the probe boards and their game-theoretic values obtained in previous work (see Methods). We measure planning depth by having each agent make a move at probe boards, and average across the probes the length of the deepest branch of each resulting MCTS tree. Both value function quality and planning depth increase as training progresses (Figure 1B and C; left). We plot previous human results here to aid comparison (Figure 1B and C; right) (van Opheusden et al., 2021). Compared to human learning, the increase in planning metrics are more drastic in AlphaZero, which is expected given that human is not a blank slate to begin with.

The increase of planning depth in human learning has been attributed only to an increase in the number of search iterations (van Opheusden et al., 2021). By contrast, we discover that planning depth of AlphaZero increases over training despite the number of MCTS searches, N_{MCTS}, being fixed. This suggests either a potential difference between humans and machines in how their decision trees change during learning, or a potential alternative hypothesis for the mechanism of the depth increase in humans. Here we only provide a mechanism of depth increase in AlphaZero.

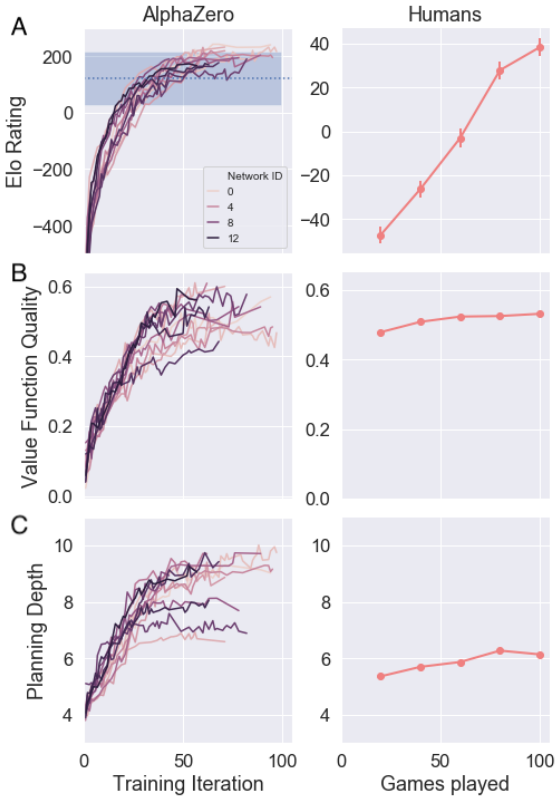


Figure 1: Elo and planning metric comparison between AlphaZero and human. Playing strength (Elo rating, A), value function quality (B) and planning depth (C) of both AlphaZero and human increase with training. Solid lines represent Networks. Dotted line in (A) represents the Elo of a strong human player, and the shade reflects the 95-confidence interval of this Elo estimate. Human results are reproduced from data in van Opheusden et al. (2021). The scale of Elo ratings are different between AlphaZero (left) and human (right) and the numbers are not directly comparable because there is no tournament between human players from prior study with our AlphaZero agents.

3.3 Entropy of action prior mediates depth increase

When the total search budget is fixed, one possible mechanism for the increase in planning depth could be a more targeted and less scattered search process. In AlphaZero the targetedness of the search is largely modulated by the policy. The policy starts out uniform and evolves to match the post-MCTS action probabilities. Since the search process makes the action probabilities be less uniform, the policy should become less uniform and thus have a lower entropy over training, defined as $H(s) = -\sum_a p(a|s) \log p(a|s)$. A decrease in entropy over training is confirmed in Figure 2B. (Similar to planning depth, the entropy here is also averaged across probe boards.)

3.4 Policy quality improves

A more concentrated prior does not necessarily imply "smarter" searches. A bad prior can lead a deep but misguided search. We therefore develop a metric, policy quality, to quantify how good AlphaZero's policies are. Policy quality reflects the correlation between AlphaZero's policies and the optimal policies(see Methods). The policy quality improves over training for all Networks (Figure 2A. So not only are the priors more concentrated, but they also align better with optimal policies, leading the search in more promising directions.

3.5 The increase in playing strength is mediated by planning metrics

Playing strength of AlphaZero agents increases with training (Fig 1A; left), and we hypothesize that the effect of training on playing strength is mediated by planning metrics. Mediation analysis shows that policy quality, value function quality and planning depth all have a significant mediated effect on Elo ratings (Figure 3). 358

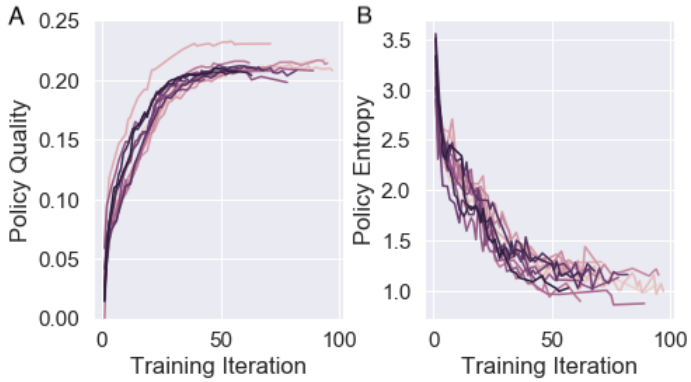


Figure 2: Policy quality and policy entropy. A) Policy quality of AlphaZero agents increases with training. B) Policy entropy of AlphaZero agents decreases with training.

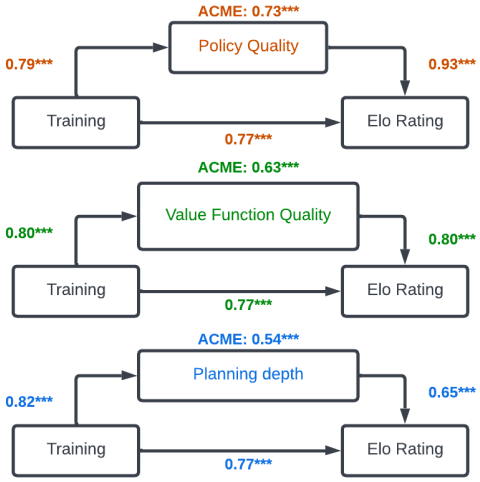


Figure 3: Mediation analysis: illustration and results. The effect of training on Elo ratings is mediated via three planning metrics: policy quality, value quality and planning depth. ACME is the average causal mediation effect. Numbers next to arrows represent the regression coefficients between variables. Asterisks denote statistical significance.

3.6 Policy quality matters the most

Mediation analysis on each planning metric shows that learning-induced Elo changes are mediated by planning metrics. However we cannot reliably conclude the relative contribution of each metric, since the metrics are correlated with each other

Policy quality, planning depth and value function quality together explain 0.95 of the variance in Elo in a linear regression, with weights: $\beta_{policy} = 0.92$, whereas $\beta_{depth} = 0.09$ and $\beta_{value} = -0.03$. We also test the dominance of policy quality by first regressing out all the other confounders from Elo. Policy quality explains the residuals significantly well ($F = 29.11$, $p < 10^{-6}$). By contrast, neither a similar residual regression for value function quality ($F = 0.18$, $p = 0.668$) nor one for depth ($F = 2.69$,$p = 0.101$) is significant.

3.7 Causal manipulations reveal contributions from all planning metrics

To arbitrate the role of planning depth in playing strength, we causally manipulate planning depth for each iteration in the best Network, while holding everything else about an agent constant. To do this, we replicate an agent and then set its number of MCTS searches (N_{MCTS}) to four different levels. For the same iteration (dots connected by the same line in Figure 4A.), a higher N_{MCTS} induces a high planning depth, which correlates positively with Elo in all iterations. As training progresses, the positive effect of planning depth on Elo diminishes, as seen from the decreasing of the slopes of the lines in Figure 4A. We perform a linear regression (Elo $\sim$ depth) for each iteration within the Network to obtain its "depth efficiency" (Figure 4B). The depth efficiency decreases as training progresses. One possible explanation for why the benefit of depth diminishes is that the good action priors of well-trained models are sufficient to guide actions.

For the value and policy manipulation, we select eleven models from a Network that spans early, middle and late epochs of training. We swap either the value or the policy function of a model with the value/policy function of low, middle or the best quality. These swap targets come from the initial, a middle (iter 22) or the final iteration of the model within the Network, respectively. Models from one training epoch do not have swaps with models from the same training epoch.

The result shows that both value and policy contribute to performance, as equipping early models with a well-trained policy or value function improves their Elo (Figure 5). The gain is larger with the policy swap initially, but the value swap catches up during the middle epoch (20-35 iters), suggesting value and policy quality can complement each other in this intermediate range, i.e. a good performance does not require both value and policy to be really high, but only one to be high and the other intermediate. Swapping the policy function of a well-trained model to a naive policy is unambiguously more disruptive than swapping the value function, again echoing the the overall dominance of policy. Swapping the components of the late models to those of the middle ones produces qualitatively similar but quantitatively less drastic reduction, compared to swapping to early ones.

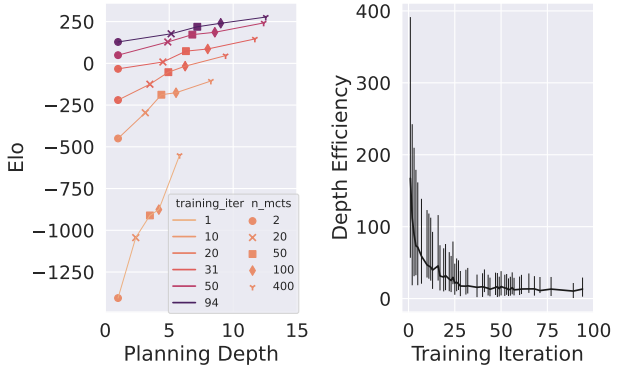


Figure 4: The effect of N_{MCTS} manipulation on depth and Elo. A) Elo vs planning depth for selected iterations and all N_{MCTS} manipulation of the chosen Network. Color indicates training iteration, and marker style indicates the number of MCTS searches. Agents with the same training iteration are connected by a line. B) Depth efficiency vs Training iteration for all agents. Depth efficiency is defined as the slope of each line in A (as well as the lines for other iterations not shown in A), which represents the efficiency of depth increase in increasing Elo. Error bars reflects the 95% confidence intervals.

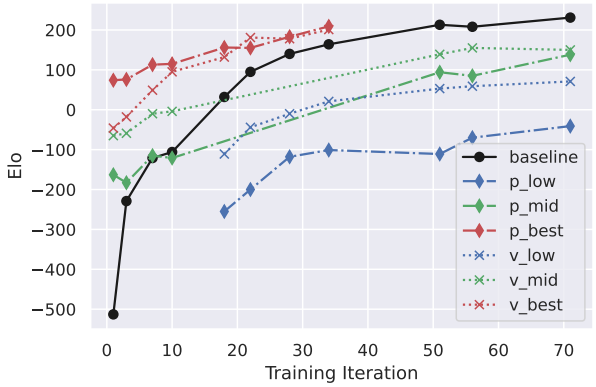


Figure 5: Elo ratings as a result of the value or policy function manipulation. Black markers (solid line) are the original agents across training. Colored diamond markers (half solid line) are agents with policy functions swapped. Colored cross markers (dotted lines) are agents with value functions swapped. Color reflects the quality of the target of the swap: low-blue, middle-green, best-red.

4 Discussion

We analyzed what capacity changes underlie AlphaZero's learning in a game of intermediate complexity. We found that although value function quality, planning depth and policy quality all improve during training, the improvement in performance is driven the most by policy quality. The intermediate complexity of 4-in-a-row allowed us to compute the game-theoretical values of board positions and therefore the value function quality and policy quality metrics, which are difficult to obtain in complex games.

The distinction between value and policy is worth emphasizing. Value provides a context independent common scale for measuring all possible states, and can be updated during the planning/reinforcement learning process. Policy provides a context dependent relative scale for comparing available actions at one state. The machine learning community has noted the distinct and crucial role of policy. Hessel et al. (2021) argued that action values alone are not enough for representing the best stochastic policy. Hamrick et al. (2020) showed that planning is most useful in the learning process, while post-learning, shallow trees are often as performant. The AlphaGo Zero study also showed strong play from a myopic policy network without MCTS (Silver et al., 2017). These results are consistent with our result of manipulating the N_{MCTS}, all of which point to the effect of a good policy in maintaining performance in the absence of many searches. The appropriate complexity of the task allows us to further quantify policy and value quality and demonstrate the dominant role of policy quality.

Our agents improve planning depth through a more concentrated policy induced by training. By contrast, in humans, a depth increase in 4-in-a-row seemed to be due to an increase in the number of searches (Van Opheusden, Galbiati, Bnaya, Li, & Ma, 2017). What is the origin of this difference? In the cognitive model for the human, after the current player selects the best leaf node and simulates an opponent move, that move would often block the current player's feature, turning the best node into the worst. Only after a thorough search through the alternative actions, can the initial best node regain its status. Only then can the search advance one level deeper. Perhaps aspects of MCTS, such as using a policy to guide tree search, could possibly help explain human behavior. It would be an interesting future direction in human planning research to arbitrate between "more searches" and "smarter searches" as mechanisms for depth increase.

The diminishing contributions of the planning depth on Elo as training progresses has not been reported in human 4-in-a-row studies. One possibility is that the human subjects recruited were not on the extreme end of expertise such that deeper searches stop providing marginal utilities. Intriguingly, the early work on chess by de Groot (1965) was unable to find gross differences in the number of moves considered, search heuristics, and depth of search, between masters and weaker players. But masters are good at coming up with the "right" moves to search further. This finding allows us to draw similarities between the masters and AlphaZero agents at a later learning stage. It also provides some confidence that AlphaZero could be valuable in showing a wider range of possible behaviors than those shown in the recruited human subjects. More importantly, the chess result indicates that something similar to the policy in AlphaZero might indeed play a crucial role in human decision making.

References

Coulom, R. (2008). Whole-history rating: A bayesian rating system for players of time-varying strength. In *International conference on computers and games* (pp. 113–124).

Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., & Dolan, R. J. (2011). Model-based influences on humans' choices and striatal prediction errors. *Neuron*, *69*(6), 1204–1215.

de Groot, A. D. (1965). *Thought and choice in chess*. The Hague, Mouton.

Hamrick, J. B., Friesen, A. L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., … Weber, T. (2020). On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*.

Hessel, M., Danihelka, I., Viola, F., Guez, A., Schmitt, S., Sifre, L., … Van Hasselt, H. (2021). Muesli: Combining improvements in policy optimization. In *International conference on machine learning* (pp. 4214–4226).

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, *40*.

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Hassabis, D., Kim, B., … Kramnik, V. (2021). Acquisition of chess knowledge in alphazero. *arXiv preprint arXiv:2111.09259*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., … others (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., … others (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354–359.

Solway, A., & Botvinick, M. M. (2015). Evidence integration in model-based tree search. *Proceedings of the National Academy of Sciences*, *112*(37), 11708–11713.

Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J., & Zitnick, L. (2019). Elf opengo: An analysis and open reimplementation of alphazero. In *International conference on machine learning* (pp. 6244–6253).

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the turing test* (pp. 23–65). Springer.

Van Opheusden, B., Galbiati, G., Bnaya, Z., Li, Y., & Ma, W. J. (2017). A computational model for decision tree search. In *Cogsci.*

van Opheusden, B., Galbiati, G., Kuperwajs, I., Bnaya, Z., Ma, W.-J., et al. (2021). Revealing the impact of expertise on human planning with a two-player board game.

van Opheusden, B., & Ma, W. J. (2019). Tasks for aligning human and machine planning. *Current Opinion in Behavioral Sciences*, *29*, 127–133.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., … Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature neuroscience*, *21*(6), 860–868.

Constrained Variational Policy Optimization for Safe Reinforcement Learning

Zuxin Liu[*]
Carnegie Mellon University
zuxinl@cmu.edu

Zhepeng Cen
Carnegie Mellon University
zcen@andrew.cmu.edu

Vladislav Isenbaev
Nuro Inc.
visenbaev@nuro.ai

Wei Liu
Nuro Inc.
w@nuro.ai

Zhiwei Steven Wu
Carnegie Mellon University
zstevenwu@cmu.edu

Bo Li
UIUC
lbo@illinois.edu

Ding Zhao
Carnegie Mellon University
dingzhao@cmu.edu

Abstract

Safe reinforcement learning (RL) aims to learn policies that satisfy certain constraints before deploying to safety-critical applications. Primal-dual as a prevalent constrained optimization framework suffers from instability issues and lacks optimality guarantees. This paper overcomes the issues from a novel probabilistic inference perspective and proposes an Expectation-Maximization style approach to learn safe policy. We introduce a novel Expectation-Maximization approach to naturally incorporate constraints during the policy learning: 1) a provable optimal non-parametric variational distribution could be computed in closed form after a convex optimization (E-step); 2) the policy parameter is improved within the trust region based on the optimal variational distribution (M-step). The proposed algorithm decomposes the safe RL problem to a convex optimization phase and a supervised learning phase, and we show its unique advantages by proving its optimality and policy improvement stability. A wide range of experiments on continuous robotic tasks show that the proposed method achieves significantly better performance in terms of constraint satisfaction and sample efficiency than primal-dual baselines.

Keywords: Safe Reinforcement Learning
Constrained Reinforcement Learning
Safe Exploration
RL as Inference

Acknowledgements

This project is funded in part by Carnegie Mellon University's Mobility21 National University Transportation Center, which is sponsored by the US Department of Transportation.

[*]Contact Author.

Note that due to the page limit, we are unable to provide the proofs of Theorem and Propositions. They are provided in the full version: https://arxiv.org/abs/2201.11927

1 Introduction

The past few years have witnessed great success of reinforcement learning (RL). However, deploying a trained RL policy to the real world is challenging. One of the major obstacles is to ensure the learned policy satisfies safety constraints. Safe RL studies the RL problem subject to certain constraints, where the agent aims to not only maximize the task reward return, but also limit the constraint violation rate to a certain level. However, learning a parametrized policy that satisfies constraints is not a trivial task, especially when the policy is represented by black-box neural networks.

Existing safe RL optimization approaches are mostly under the primal-dual framework, which transforms the original constrained problem into an unconstrained one by introducing the dual variables (i.e., Lagrange multipliers) to penalize constraint violations [6, 7]. However, the primal-dual iterative optimization may run into numerical *instability* issues and lacks *optimality* guarantee for each policy iteration [4, 8]. The instability usually comes from imbalanced learning rates of the primal and dual problem, and the optimality term means both *feasibility* (constraint satisfaction) and reward maximization. [2] and [9] propose to approximate the constrained optimization problem with low-order Taylor expansions such that the dual variables could be solved efficiently, but the induced approximations errors may yield poor constraint satisfaction performance in practice [7]. In addition, these policy-gradient algorithms are on-policy by design, and extending them to a more sample efficient off-policy setting is non-trivial. Note that in safe RL context, sample efficiency means using both minimum constraint violation costs and minimum interaction samples to achieve the same level of rewards. As a result, a constrained RL optimization method that is 1) **sample efficient**, 2) **stable** and has 3) **optimality guarantees** is absent in the literature.

To bridge the gap, this paper proposes the Constrained Variational Policy Optimization (CVPO) algorithm from the probabilistic inference view. The main contributions of this work are summarized as follows: 1) To our best knowledge, this is the first work that formulates safe RL as a *probabilistic inference* problem. We propose a novel two-step algorithm in an Expectation Maximization (EM) style to naturally incorporate safety constraints during the policy training; 2) We show that an optimal and feasible non-parametric variational distribution can be solved *analytically* during the E-step, and the parametrized policy could be trained via a *supervised learning* fashion during the M-step, which allows us to improve the policy with off-policy data and increase the sample efficiency; 3) We evaluate CVPO on a series of continuous control tasks. The empirical experiments demonstrate the effectiveness of the proposed method – more stable training, better constraint satisfaction, and up to 1000 times better sample efficiency than on-policy baselines.

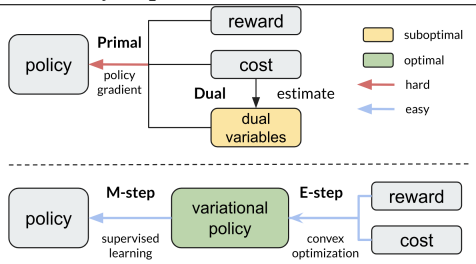
2 Constrained Variational Policy Optimization (CVPO)

Constrained Markov Decision Processes. Constrained Markov Decision Processes (CMDPs) provide a mathematical framework to describe the safe RL problem [3], where the agents are enforced with restrictions on auxiliary safety constraint violation costs. A CMDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0, c)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition kernel that specifies the transition probability $p(s_{t+1}|s_t, a_t)$ from state s_t to s_{t+1} under the action a_t, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \to [0, 1)$ is the discount factor, and $\rho_0 : \mathcal{S} \to [0, 1]$ is the distribution over the initial states, $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ is the cost for violating safety constraints. Denote $\pi(a|s)$ as the policy, and $\tau = \{s_0, a_0, ..., \}$ as trajectory, the discounted return of the reward is $J_r(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t]$, and similarly the discounted return of the cost is $J_c(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t c_t]$, where the initial state $s_0 \sim \rho_0$. The objective is to find the policy π^* that maximizes the expected cumulative rewards while limiting the costs incurred from constraint violations to a threshold $\epsilon_1 \in [0, +\infty)$: $\pi^* = \arg\max_\pi J_r(\pi), \quad s.t. \quad J_c(\pi) \leq \epsilon_1$.

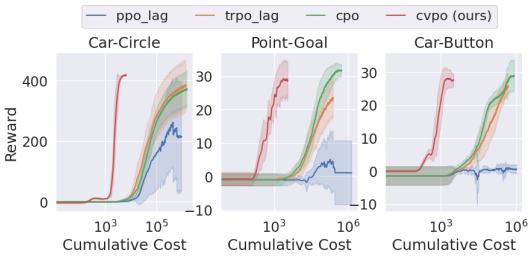
Constrained RL as Inference. Before presenting the inference view, we first introduce the standard primal-dual perspective to solve CMDP, which transforms the objective into a min-max optimization by introducing the Lagrange multiplier λ: $(\pi^*, \lambda^*) = \arg\min_{\lambda\geq 0} \max_\pi J_r(\pi) - \lambda(J_c(\pi) - \epsilon_1)$. The core principle of primal-dual approaches is to solve the min-max problem iteratively. However, the optimal dual variable $\lambda = +\infty$ when $J_c(\pi) > \epsilon_1$ and $\lambda = 0$ when $J_c(\pi) < \epsilon_1$, so selecting a proper learning rate for λ is critical. Approximately solving the minimization also leads to suboptimal dual variables for each iteration. In addition, the non-stationary cost penalty term involving λ will make the policy gradient step in the primal problem hard to optimize, just as shown in the upper diagram of Fig. 1a. To tackle the above problems, we view the safe RL problem from the probabilistic inference perspective – inferring safe actions that result in "observed" high reward in states. This is done by introducing an optimality variable O to represent the event of maximizing the reward. As shown in [5], the likelihood of being optimal given a trajectory is proportional to the exponential of the discounted cumulative reward: $p(O = 1 \mid \tau) \propto \exp(\sum_t \gamma^t r_t/\alpha)$, where α is a temperature parameter. Denote the probability of a trajectory τ under the policy π as $p_\pi(\tau)$, then the lower bound of the log-likelihood of optimality under the policy π is:

$$\log p_\pi(O = 1) = \log \int p(O = 1 \mid \tau) p_\pi(\tau) d\tau \geq \mathbb{E}_{\tau\sim q}[\sum_{t=0}^{\infty} \gamma^t r_t] - \alpha D_{\mathrm{KL}}(q(\tau) \| p_\pi(\tau)) = \mathcal{J}(q, \pi) \tag{1}$$

where $q(\tau)$ is an auxiliary trajectory distribution and $\mathcal{J}(q, \pi)$ is the evidence lower bound (ELBO). To ensure safety, we limit the choices of $q(\tau)$ within a feasible distribution family that is subject to certain constraints. Recall that



(a) Primal-dual view (upper) and inference view (lower) of safe RL problem.



(b) Reward versus cumulative cost (log-scale).

$c_t = c(s_t, a_t)$ is the cost for constraint violations. We define the **feasible distribution family** in terms of the threshold ϵ_1 as $\Pi_{\mathcal{Q}}^{\epsilon_1} := \{q(a|s) : \mathbb{E}_{\tau\sim q}[\sum_{t=0}^{\infty}\gamma^t c_t] < \epsilon_1, a \in \mathcal{A}, s \in \mathcal{S}\}$, which is a set of all the state-conditioned action distributions that satisfy the safety constraint. Afterwards, by factorizing the trajectory distributions $q(\tau) = p(s_0)\prod_{t\geq 0} p(s_{t+1}|s_t, a_t)q(a_t|s_t), \forall q \in \Pi_{\mathcal{Q}}^{\epsilon_1}$ and $p_{\pi_\theta}(\tau) = p(s_0)\prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)p(\theta)$, where $\theta \in \Theta$ is the policy parameters, and $p(\theta)$ is a prior distribution, we obtain the following ELBO by cancelling the transitions:

$$\mathcal{J}(q, \theta) = \mathbb{E}_{\tau\sim q}\left[\sum_{t=0}^{\infty}\left(\gamma^t r_t - \alpha D_{\mathrm{KL}}(q(\cdot|s_t)\|\pi_\theta(\cdot|s_t))\right)\right] + \log p(\theta), \quad \forall q(a|s_t) \in \Pi_{\mathcal{Q}}^{\epsilon_1}. \tag{2}$$

Optimizing the new lower bound $\mathcal{J}(q, \theta)$ w.r.t q within the feasible distribution space $\Pi_{\mathcal{Q}}^{\epsilon_1}$ (E-step) and π within the parameter space Θ (M-step) iteratively via an Expectation-Maximization (EM) fashion yields the Constrained Variational Policy Optimization (CVPO) algorithm. Viewing safe RL as a variational inference problem has many benefits. As shown in the lower diagram of Fig. 1a, we break the direct link between the inaccurate dual variable optimization and the difficult policy improvement and introduce a variational distribution in the middle to bridges the two steps, such that the policy improvement could be done via a much easier supervised learning fashion. The **key challenges** arise from the E-step because of the limitation of the variational distribution within a constrained set. However, we observe that the constrained q could be solved analytically, efficiently and with optimality and feasibility guarantee through convex optimization.

Constrained E-step. The objective of this step is to find the optimal variational distribution $q \in \Pi_{\mathcal{Q}}^{\epsilon_1}$ to improve the return of task reward, while satisfying the safety constraint. At the i-th iteration, we resort to perform a partial constrained E-step to maximize $\mathcal{J}(q, \theta)$ with respect to q by fixing the policy parameters $\theta = \theta_i$. We set the initial value of $q = \pi_{\theta_i}$ such that the return of task reward $Q_r^q(s, a)$ and cost $Q_c^q(s, a)$ could be estimated by: $Q_r^q(s, a) = Q_r^{\pi_{\theta_i}}(s, a) = \mathbb{E}_{\tau\sim\pi_{\theta_i}, s_0=s, a_0=a}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$, $Q_c^q(s, a) = Q_c^{\pi_{\theta_i}}(s, a) = \mathbb{E}_{\tau\sim\pi_{\theta_i}, s_0=s, a_0=a}\left[\sum_{t=0}^{\infty}\gamma^t c_t\right]$. We further optimize $q(\cdot|s)$ by the following KL regularized objective:

$$\max_q \quad \mathbb{E}_{\rho_q}\left[\int q(a|s)Q_r^{\pi_{\theta_i}}(s, a)da\right] \quad s.t. \quad \mathbb{E}_{\rho_q}\left[\int q(a|s)Q_c^{\pi_{\theta_i}}(s, a)da\right] \leq \epsilon_1, \quad \mathbb{E}_{\rho_q}\left[D_{\mathrm{KL}}(q(a|s)\|\pi_{\theta_i})\right] \leq \epsilon_2, \forall s \sim \rho_q \tag{3}$$

where $\rho_q(s)$ is the stationary state distribution induced by $q(a|s)$ and ρ_0. Intuitively, in E-step, we aim to find the optimal non-parametric distribution that 1) maximizes the task rewards, 2) belongs to the feasible distribution family $\Pi_{\mathcal{Q}}^{\epsilon_1}$, and 3) stays within the trust region of the old policy. To solve (3), we use a non-parametric form for $q(a|s)$. Namely, given a state s, we use $|\mathcal{A}|$ variables for $q(\cdot|s)$ if the action space is finite. Otherwise, we sample K particles within the action space to represent the variational distribution for the continuous action space. Then we can obtain the optimal analytical solution after solving a convex dual problem. We show the strong duality guarantee under the Slater's condition:

Assumption 1. *There exists a feasible distribution $\bar{q} \in \Pi_{\mathcal{Q}}^{\epsilon_1}$ within the trust region of the old policy π_{θ_i}: $D_{\mathrm{KL}}(\bar{q}\|\pi_{\theta_i}) < \epsilon_2$.*

Theorem 1. *If assumption 1 holds, then the optimal variational distribution within $\Pi_{\mathcal{Q}}^{\epsilon_1}$ for problem (3) has the form:*

$$q_i^*(a|s) = \frac{\pi_{\theta_i}(a|s)}{Z(s)}\exp\left(\frac{Q_r^{\theta_i}(s, a) - \lambda^* Q_c^{\theta_i}(s, a)}{\eta^*}\right), \tag{4}$$

where $Z(s)$ is a constant normalizer, and the dual variables η^* and λ^* are the solutions of the following convex optimization problem:

$$\min_{\lambda,\eta\geq 0} \quad g(\eta, \lambda) = \lambda\epsilon_1 + \eta\epsilon_2 + \eta\mathbb{E}_{\rho_q}\left[\log \mathbb{E}_{\pi_{\theta_i}}\left[\exp\left(\frac{Q_r^{\theta_i}(s, a) - \lambda Q_c^{\theta_i}(s, a)}{\eta}\right)\right]\right]. \tag{5}$$

Theorem 1 suggests the non-parametric variational distribution could be easily solved in close-form with optimality guarantee, since Assumption 1 ensures zero duality gap. One exciting property of Theorem 1 is that we prove the

convexity of the dual problem, which guarantees the optimality of the solution. Furthermore, when the Slater condition in Assumption 1 holds, at least one optimal solution exist, which ensures the dual problem could be solved efficiently via any convex optimization tools. We believe this finding is original and non-trivial.

M-step. After E-step, we obtain an optimal feasible variational distribution $q_i^*(\cdot|s)$ for each state by solving (3). In M-step, we aim to improve the ELBO (2) w.r.t the policy parameter θ. By dropping the terms in Eq. (2) that are independent of θ, we obtain the following M-step objective: $\bar{\mathcal{J}}(\theta) = \mathbb{E}_{\rho_q}\Big[\alpha \mathbb{E}_{q_i^*(\cdot|s)}\big[\log \pi_\theta(a|s)\big]\Big] + \log p(\theta)$. Since optimizing $\bar{\mathcal{J}}(\theta)$ is a supervised learning problem, we could use any prior $p(\theta)$ to regularize the policy. Note that the objective could be viewed as a weighted maximum a-posteriori problem, which is similar to MPO [1]. We adopt a Gaussian prior around the old policy parameter θ_i in this paper: $\theta \sim \mathcal{N}(\theta_i, \frac{F_{\theta_i}}{\alpha\beta})$, where F_{θ_i} is the Fisher information matrix and β is a positive constant. With this Gaussian prior, we obtain the following generalized M-step by converting the soft KL regularizer to a hard KL constraint, which is important to improve the training robustness and prevent the policy from overfitting:

$$\max_\theta \quad \mathbb{E}_{\rho_q}\Big[\mathbb{E}_{q_i^*(\cdot|s)}\big[\log \pi_\theta(a|s)\big]\Big], \quad s.t. \quad \mathbb{E}_{\rho_q}\big[D_{\mathrm{KL}}(\pi_{\theta_i}(a|s)\|\pi_\theta(a|s))\big] \le \epsilon. \tag{6}$$

Theoretical Analysis. CVPO updates the policy by maximizing the KL-regularized objective in an EM manner, which has it two advantages over primal-dual methods – the ELBO improvement guarantee $J(q_i, \theta_{i+1}) \ge J(q_{i-1}, \theta_i)$ and the worst-case constraint satisfaction bound.

Proposition 1. *Suppose $\pi_{\theta_i} \in \Pi_{\mathcal{Q}}^{\epsilon_1}$. $\pi_{\theta_{i+1}}$ and π_{θ_i} are related by the M-step (6), then we have the upper bound: $J_c(\pi_{\theta_{i+1}}) \le \epsilon_1 + \frac{[(1-\gamma)+\sqrt{2\epsilon\gamma}]\delta_c^{\pi_{\theta_{i+1}}}}{(1-\gamma)^2}$, where $\delta_c^{\pi_{\theta_{i+1}}} = \max_s |\mathbb{E}_{a\sim\pi_{\theta_{i+1}}}[A_c^{\theta_i}(s,a)]|$, $A_c^{\theta_i}(s,a)$ is the cost advantage function of π_{θ_i}.*

We can see the worst-case episodic constraint violation upper bound for $\pi_{\theta_{i+1}}$ is related to the trust region size ϵ and the worst-case approximation error $\delta_c^{\theta_{i+1}}$. Though we inherit a similar bound as CPO, our method ensures the optimality of each update and could be done in a more sample-efficient off-policy fashion. In addition, we observe that by selecting proper KL constraints ϵ in the M-step, we have the following policy improvement robustness property:

Proposition 2. *Suppose $\pi_{\theta_i} \in \Pi_{\mathcal{Q}}^{\epsilon_1}$. $\pi_{\theta_{i+1}}$ and π_{θ_i} are related by the M-step. If $\epsilon < \epsilon_2$, where ϵ, ϵ_2 are the KL threshold in E-step and M-step respectively, then the variational distribution q_{i+1}^* in the next iteration is guaranteed to be feasible and optimal.*

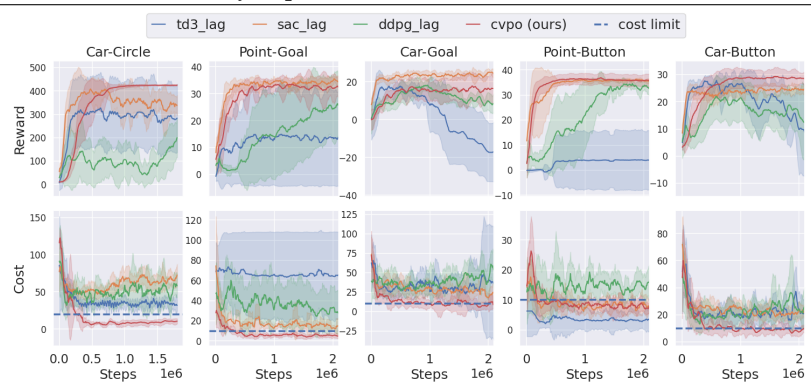
Proposition 2 provides us with an interesting perspective and a theoretical guarantee of the policy improvement robustness – no matter how bad the M-step update in one iteration is, we are still able to recover to an optimal policy within the feasible region. Furthermore, We find that under the Gaussian policy assumption, multiple steps robustness could also be achieved. The monotonic reward improvement guarantee, the worst-case cost bound and the policy updating robustness ensure the **training stability** of CVPO. Formal proofs and implementation details are left out due to space constraints.

3 Experiments and Conclusion

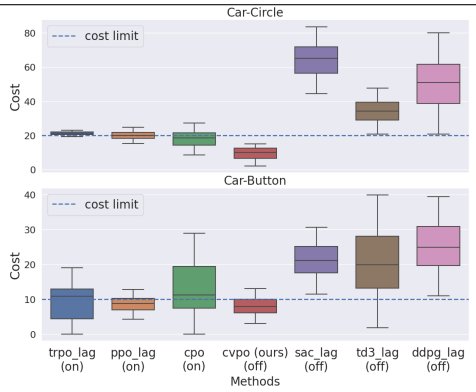
We designed 5 robotic control tasks on SafetyGym [7] with different difficulty levels to show the effectiveness of our method. In these tasks, the agent is supposed to obtain high reward by reaching the goal and avoid penalty or cost by keeping in the safe zones. To better understand the role of variational inference, we compare our method with three off-policy primal-dual-based baselines (i.e., `SAC-Lag`, `DDPG-Lag` and `TD3-Lag`) and three on-policy baselines (i.e., CPO [2], `TRPO-Lag` and `PPO-Lag`), where `Lag` denotes the PID-Lagrangian [8] method to update the dual variables and are thus stronger than naive Lagrangian approaches. We separate the comparison with off-policy and on-policy baselines because off-policy ones are more sample efficient, so the scales for them are different and thus making it hard to distinguish the plots. For fair comparison, we use the same network sizes of the policy and critics for all the methods, including CVPO (our method). The safety critics updating rule and the discounting factor are also the same for all off-policy methods.

Results. Fig. 2a shows the training curves for off-policy safe RL approaches. Two rows correspond to the episodic reward and cost while the dashed line is the target cost threshold. We can clearly see that CVPO outperforms the off-policy baselines in terms of the constraint satisfaction while maintaining high episodic reward, which validates the optimality and feasibility guarantee of our method. Note that all the off-policy baselines use the same network architecture and sizes for Q_r and Q_c critics, and the only difference between them is the policy optimization. The training curves also demonstrate better stability of our approach, which validates our theoretical analysis. Fig. 1b demonstrates the **efficacy** of utilizing each cost – how much task rewards we could obtain given a budget of constraint violations. The curves that approach to the upper left are better because fewer costs are required to achieve high rewards. We can clearly see that CVPO outperforms baselines with large margin among all tasks – we use **100 $\sim$ 1000 times less** cumulative constraint violations to obtain the same task reward. Note that the x-axis is on the log-scale.

Convergence cost comparison. Fig. 2b shows the box plot of each method's constraint satisfaction performance after convergence. We define convergence as the last 20% training steps. The box plot – also called whisker plot – presents



(a) Training curves for off-policy baselines comparison.



(b) Convergence cost of different methods

Figure 2: (a) Comparison between different off-policy baselines. Each column corresponds to an environment. The curves are averaged over 10 random seeds, where the solid lines are the mean and the shadowed areas are the standard deviation. (b) Box plot of the convergence cost. (on)/(off) denotes on-policy/off-policy method, respectively.

a five-number summary of the data. The five-number summary is the minimum (the lower solid line), first quartile (the lower edge of the box), median (the solid line in the box), third quartile (the upper edge of the box), and maximum (the upper solid line). As shown in the figure, on-policy baselines have better constraint satisfaction performance than off-policy baselines, which indicates that extending the primal-dual framework to off-policy settings is non-trivial. We find that our method meets the safety requirement better and with smaller variance than most baselines. Interestingly, we could observe that even **the third quartile of cost of our approach is below the target cost threshold**, which indicates that CVPO satisfies constraints state-wise, rather than in expectation as baselines. The reason is that we sample a mini-batch from the replay buffer to compute the optimal non-parametric distribution for these states, while baseline approaches directly optimize the policy to satisfy the constraints in expectation.

Conclusion. We show the safe RL problem can be decomposed to a convex optimization phase with a non-parametric variational distribution and a supervised learning phase. We show the unique advantages of constrained variational policy optimization: 1) high **sample-efficiency** from the off-policy variant of the approach; 2) **stability** ensured by the monotonic reward improvement guarantee, worst-case constraint violation bound and the policy updating robustness; and 3) with theoretical **optimality and feasibility guarantees** by solving a provable convex optimization problem in the E-step. We validate the proposed method with extensive experiments, and the results demonstrate the better empirical performance of our method than previous primal-dual approaches in terms of constraint satisfaction and sample efficiency. We believe our work will provide a new perspective in this field and inspire more exciting researches along this direction.

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[3] Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3):387–417, 1998.

[4] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.

[5] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[6] Qingkai Liang, Fanyu Que, and Eytan Modiano. Accelerated primal-dual policy optimization for safe reinforcement learning. *arXiv preprint arXiv:1802.06480*, 2018.

[7] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7, 2019.

[8] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

[9] Yiming Zhang, Quan Vuong, and Keith Ross. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 2020.

Q-Functionals for Efficient Value-Based Continuous Control

Sreehari Rammohan
Department of Computer Science
Brown University
Providence, RI 02906
sreehari@brown.edu

Bowen He
Department of Computer Science
Brown University
Providence, RI 02906
bowen_he@brown.edu

Shangqun Yu
Department of Computer Science
Brown University
Providence, RI 02906
shangqun_yu@brown.edu

Sam Lobel
Department of Computer Science
Brown University
Providence, RI 02906
samuel_lobel@brown.edu

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02906
gdk@cs.brown.edu

Abstract

We present an alternative architecture for continuous control deep reinforcement learning which we call *Q-functionals*: instead of returning a single value for a state and action, our network transforms a state into a *function* that can be rapidly evaluated in parallel for *many* actions, allowing us to efficiently choose high-value actions through sampling alone. This contrasts with the typical architecture of off-policy reinforcement learning, where a policy network is trained for the sole purpose of selecting actions from the Q-function. We represent our action-dependent Q-function as a weighted sum of basis functions (Fourier, Polynomial, etc) over the action space, where the weights are state-dependent and output by the Q-functional network. In addition to fast sampling, this representation of state-action value is helpful in imposing an inductive bias on learning (such as making the value function smooth over actions) which can lead to faster speed of convergence. We characterize our framework, describe two implementations of Q-functionals, and demonstrate promising performance on a suite of continuous control tasks.

Keywords: Reinforcement Learning, Value-Based Continuous Control, Q-Functionals

1 Introduction

The final product of a successful reinforcement learning system is a *policy* that performs well at interacting with the environment, as measured by expected cumulative discounted reward. A dominant paradigm in reinforcement learning is to derive policies from a *Q-function* [8], which summarizes the expected cumulative reward for taking a given action. When the number of actions available is finite, a strong policy can be derived by enumerating all action-values for a given state and choosing the one with the highest value. However, this brute enumeration is impossible when there are large or infinite possible actions, for example when actions are drawn from a continuous vector space. This is the natural description of, for example, robotic locomotion and control; as such, a significant amount of effort has gone into action-selection in these domains. A common framework for so-called *continuous control* problems is to train a *policy network* that selects actions according to some criteria of the Q-values.

The training signal from a policy network comes completely from the Q-function, and as such they essentially summarize information about the Q-function for a given state so as to efficiently produce high-value actions. For example, standard practice is to train a policy network to estimate the *maximum*-valued action for any given state[5]. The quantities policy networks estimate are difficult to calculate on a per-state basis, so policy networks can be thought of as *amortizing* the expensive computation of finding desirable actions: the training procedure is expensive, but it allows actions to be *sampled* with a single pass of a neural network. We highlight two potential pitfalls of this framework. First, since the policy network is not reinitialized after every Q-function update, at any given timestep most of the policy training has been done on *old* versions of the Q-function; as such, the policy may be maximizing a *stale* estimate of action-value. Second, policies are generally either deterministic or sampled from a tight distribution around some single valuable action. Therefore, it is unlikely for the policy to sample two high-value actions if they are too far apart from each other [7]. This limits exploration as well as the robustness of the value-function across all actions.

We take a different perspective on this problem: instead of training a network to amortize expensive action-value computations, we design a learning system such that these evaluations are cheap to do in parallel. This opens up a new avenue for action-selection: instead of using a policy network, we can evaluate many actions in parallel and choose between them. Since the policy is directly derived from Q-values at each timestep it always reflects the most current action-value estimates. Furthermore, random sampling can easily generate high-value actions from throughout the action-space.

In this paper, we introduce a class of Q-functions we call *Q-functionals* that allow for efficient sampling. A *functional* is a function that returns another function. Likewise, a Q-functional transforms a state into a function over actions, such that Q-values for many actions can be evaluated in parallel and with little overhead. We describe two implementations of this architecture, and demonstrate its speed and effectiveness at continuous control reinforcement learning.

2 Q-Functionals

In RL, we seek to learn a *policy* which results in high cumulative discounted reward. For a given policy π, we define the state-action value function, or Q-function, as:

$$Q^\pi(s_t, a_t) = E_\pi[R(s_t, a_t) + \gamma V^\pi(s_{t+1})].$$

where $V^\pi(s)$ is the expected value of following π from state s. A common method of iteratively improving a Q function parameterized by θ is through *bootstrapping* [6]:

$$Q_\theta(s, a) \leftarrow r(s, a) + \gamma V^\pi(s')$$

Q-functionals are a way of expressing Q-functions so that many action-values can be evaluated in parallel for a given state. Consider a traditional Q-function, perhaps represented by a neural network:

$$Q(s, a) : (\mathcal{S} \times \mathcal{A}) \to \mathcal{R}.$$

A standard design for such a Q-function in continuous control is to concatenate states and actions, and then pass this through a 2-hidden-layer neural network. For this standard architecture, evaluating two action-values for the same state takes twice as long as evaluating one. A Q-functional breaks the computation of action-values into two parts: first, a state is transformed into parameters that define a function over the action space. Then, this function is evaluated for the action(s) in question:

$$Q_{\text{FUNC}}(s, a) \to (\mathcal{A} \to \mathcal{R}) \tag{1}$$

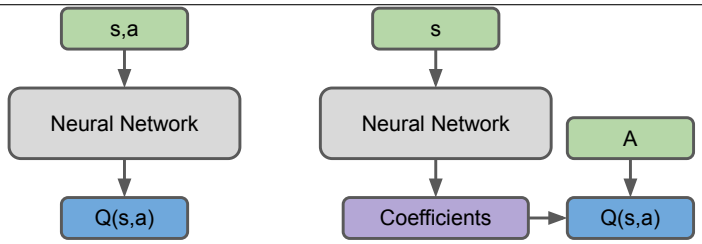


Figure 1: Network architecture of traditional Q-function (left) and Q-functional (right)

We design these functions such that while the per-state computation may be expensive, the per-action computation is relatively cheap. One simple way to represent these functions is by learning the coefficients of basis functions over the action space. An appropriate choice of basis function also adds an *inductive bias* to learning action-values: though we may expect values to have complicated dependence on state (especially when the state is something like an image), in general we expect values to be relatively smooth with respect to the action dimensions. In Figure 1, we visualize the difference between the two architectures as differing places that the actions enter the computation.

2.1 Fourier and Polynomial Basis Q-Functionals

We describe two implementations of Q-functionals using different basis functions. First, a Q-functional calculated using the Fourier basis is shown below in equation 2. The Fourier basis representation of *states* had great success when used for Q-learning in small state spaces [4]; we rely on the same inductive bias over the moderately-sized *action* spaces found in continuous control. L represents the difference between the max and min action space value allowed, $x_i(s), y_i(s)$ are the state-dependent coefficients output by the critic network. $\mathbf{C}$ is a matrix of frequencies for the Fourier basis. For a rank R Q-functional using the Fourier basis and operating in an environment with action dimension d, the full set of frequencies is represented by the cartesian product $\{0, 1, 2, ..., R\}^d$. In practice however, we find that limiting the set of frequencies to those where the sum of all elements in the frequency is less than or equal to the rank provided sufficient critic generalization. Formally speaking, $\mathbf{C} = \left\{C_i = \{0, 1, 2, ..., R\}^d | \sum_{j=0}^{d} C_{ij} \leq R\right\}$

$$Q_{\text{FUNC}}(s, a) = \sum_{C_i \in \mathbf{C}} x_i(s) \sin(\frac{\pi}{L} a \cdot C_i) + y_i(s) \cos(\frac{\pi}{L} a \cdot C_i). \tag{2}$$

The learned coefficients $x_i(s)$ and $y_i(s)$ are dependent solely on the state, allowing us to efficiently sample many actions at once without needing to pass this through the coefficient critic network again. We can define a polynomial basis similarly: we represent our action-value function such that the total polynomial-order is less than some R:

$$Q(s, a) = \sum_{C_i \in \mathbf{C}} x_i(s) \prod_{j=0}^{d} a_j^{C_{ij}}. \tag{3}$$

Previous work demonstrates that action-values can be analytically maximized for quadratic polynomials over actions ($R = 2$) [2]. Without a policy network, however, Q-functions defined as such are restricted to convex functions over actions which can be overly limiting. Our sampling framework can derive a strong policy from more complex, higher-order polynomial Q-functions. In addition, as described in the next section, sampling allows us to calculate state-values with various methods that work better than simple maximization.

2.2 Extracting policies and state-values from Q-Functionals

In standard continuous-control RL, the policy network is used in two distinct locations: calculating state-value functions (bootstrapping), and during action-selection (policy-evaluation). Here we describe how a variety of strategies for both can be implemented using sampling.

The most common strategy for both is *action maximization*: aiming to find the highest-value action for every state, and using this for both bootstrapping and action-selection:

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s, a) \quad ; \quad V(s) = Q(s, \pi(s))$$

Instead of training a policy network, we can directly estimate the maximum through sampling k actions:

$$\pi(s) = \arg\max_{a_i \in \mathcal{A}^k} Q(s, a_i) \quad ; \quad V(s) = \max_{a_i \in \mathcal{A}^k} Q(s, a_i)$$

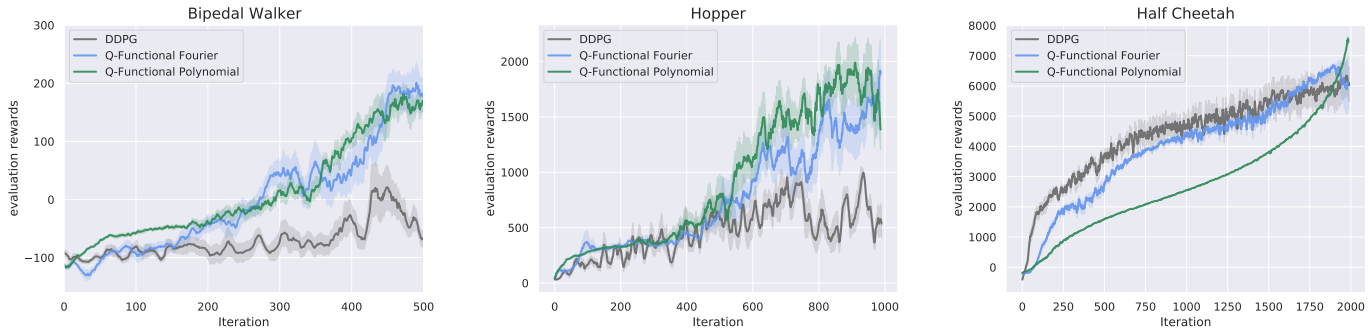


Figure 2: Q-functional Performance in comparison to DDPG. We perform 200 update steps every iteration with each iteration consisting of $\{1600, 1600, 2000\}$ interaction steps for Bipedal Walker, Hopper, and Half Cheetah respectively. Q-functionals all have "rank" 3. Results are averaged over 8 seeds, with the standard error shaded.

with accuracy bounds of $V(s)$ dependent on the dimension of $\mathcal{A}$, the Lipschitz-constant L of $Q(s,a)$, and k. For τ-entropy-regularized RL, the optimal policy is proportional to the Boltzmann distribution over Q-values. This simplifies to:

$$\pi(s) = \Pr(a|s) \leftarrow \frac{\exp\{Q(s,a)/\tau\}}{\int_{\mathcal{A}} \exp\{Q(s,a)/\tau\}} \quad ; \quad V(s) = \int_{\mathcal{A}} \pi(s)Q(s,a) - \log(\Pr(a|s)) = \tau \log\Big(\int_{\mathcal{A}} \exp\{Q(s,a)/\tau\}\Big)$$

Due to limitations imposed by common representations of stochastic policies (often constrained to state-dependent normal distributions over actions), this relation between policy and value cannot be achieved by standard policy-parameterizations [3]. In contrast, we can consistently estimate these qualities through Monte Carlo sampling:

$$\pi(s) = \Pr(a|s) = \frac{\exp\{Q(s,a)/\tau\}}{\frac{1}{k}\sum_{a_i \in \mathcal{A}^k} \exp\{Q(s,a)/\tau\}} \quad ; \quad V(s) = \tau \log\Big(\frac{1}{k} \sum_{a_i \in \mathcal{A}^k} \exp\{Q(s,a)/\tau\}\Big)$$

with accuracy bounds on both $\pi(s)$ and $V(s)$ computable from $Q_{\max} - Q_{\min}$, k, and τ.

In our experiments, we derive a robust policy $\pi(s)$ and value function $V(s)$ through a "top-n of k" approach: we evaluate k random actions, and represent our policy and value by sampling from the best n actions during interaction with the environment and averaging these n values during bootstrapping updates for stability.

$$\pi(s) \sim \text{top-n}\Big\{Q(s,a_1), ...Q(s,a_k)\Big\} \quad ; \quad V(s) = \text{mean}\Big[\text{top-n}\Big\{Q(s,a_1), ...Q(s,a_k)\Big\}\Big] \tag{4}$$

This is a similar strategy to TD3 [1], which computes $V(s)$ as a single-sample mean of the Q-values surrounding the policy-networks output. We find that averaging over *many* values, instead of simply choosing one, leads to decreased noisiness of $V(s)$ for the Bellman target in the update, and superior performance at maximizing reward.

3 Experimental Results

Our experimental results are included in Figure 2. All experiments are averaged over 8 seeds, with shading representing standard error confidence intervals. Both Fourier and Polynomial Q-functionals outperform Deep Deterministic Policy Gradients (DDPG) [5] on Bipedal Walker (4 dim action space), Hopper (3 dim action space), and Half Cheetah (6 dim action space). For action-selection and bootstrapping, we sample $k = 1,000$ actions for each state (which takes roughly equivalent computation to a single policy + value evaluation for DDPG). For $V(s)$ and our interaction policy $\pi(s)$ we anneal our top-n sampling in the range $[\frac{k}{2}, 1]$ over the course of training. We find this improves exploration when using Q-functionals; early in training the agent can try a broader range of actions. During evaluation, we choose the highest-value action ($n = 1$) sampled at each state.

As described earlier, standard feedforward Q-functions can be evaluated for multiple actions by simply evaluating the network multiple times. We quantify the speedup of the Q-functional paradigm compared to a standard feedforward Q function in Figure 2. A rank-3 Fourier critic is roughly 8 times faster than a standard neural network at this task on an Nvidia 2080ti GPU. Furthermore, the lightweight action-evaluation of Q-functionals allows for much larger sampling size: the standard architecture runs out of GPU memory with 5,000 samples, while the Fourier critic can evaluate up to 10,000 actions in a single pass. In the low-sample regime (up to 100), Q-functionals evaluate in near-constant time because the entire computation can be processed in parallel on a consumer GPU.

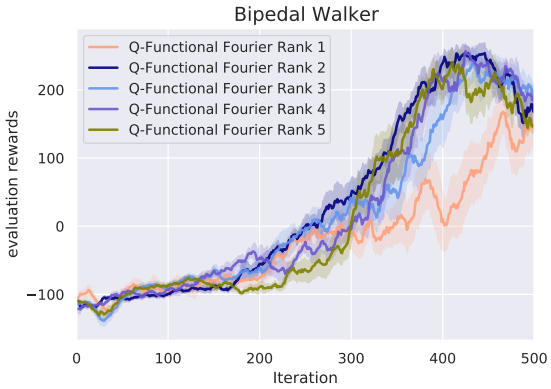


Figure 3: Rank Sweep for Bipedal Walker using Fourier Basis

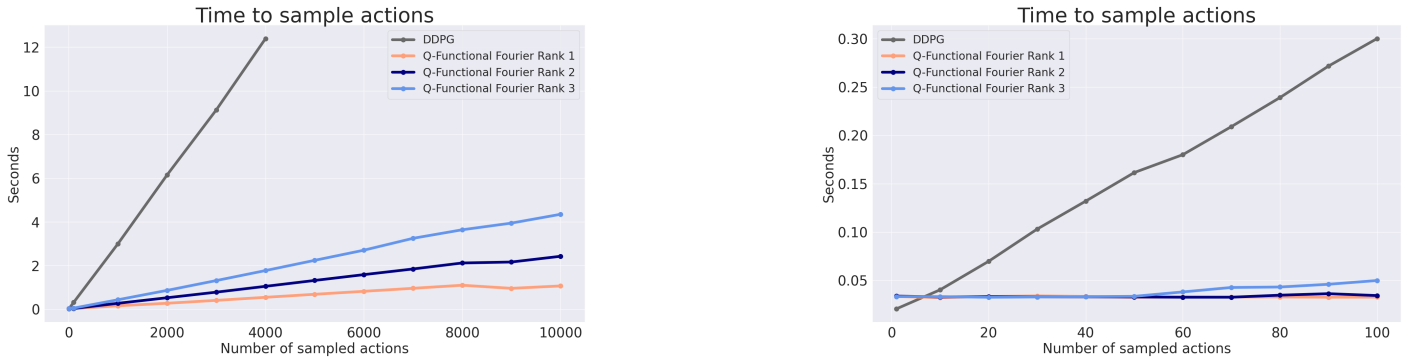


Figure 4: Speed comparison of 100 sets of action-evaluations of the same states (batch size 1024), for varying number of action-samples. **Left:** samples up to 10,000. **Right:** zoomed in sampling up to 100.

4 Discussion

We describe the framework of *Q-functionals*, a class of architectures for continuous-control RL that allow for efficiently calculating actions through *sampling*, without the need for training a large policy network. Besides requiring roughly half the parameters, our method naturally leads to more thorough exploration, represents the most *current* estimates of the Q-value, and enforces a smooth inductive bias over the action-space. We find that implementations of this framework consistently outperform baseline policy gradient methods on a range of continuous control tasks.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv e-prints*, page arXiv:1802.09477, February 2018.

[2] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv e-prints*, page arXiv:1603.00748, March 2016.

[3] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[4] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv e-prints*, page arXiv:1509.02971, September 2015.

[6] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[7] Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Flipping Coins to Estimate Pseudocounts for Exploration in Reinforcement Learning

Sam Lobel *
Department of Computer Science
Brown University
Providence, RI 02906
samuel_lobel@brown.edu

Akhil Bagaria *
Department of Computer Science
Brown University
Providence, RI 02906
akhil_bagaria@brown.edu

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02906
gdk@cs.brown.edu

Abstract

Count-based exploration can lead to optimal reinforcement learning in small tabular domains. But, it is challenging to keep track of visitation counts in environments with large state spaces. Previous work in this area has converted the problem of learning visitation counts to that of learning a restrictive form of a density model over the state-space. Rather than optimizing a surrogate objective, our proposed algorithm *directly* regresses to a state's visitation count. Compared to previous work, we show that our method is significantly more effective at deducing ground truth visitation frequencies; when used as an exploration bonus for a model-free reinforcement learning algorithm, our method outperforms existing approaches.

Keywords: Exploration, Reinforcement learning

*Equal Contributors

1 Introduction

Effective exploration is key to solving long-horizon problems using reinforcement learning. When the number of possible states is small, an agent can keep track of how many times it has visited each state. This count can then be used as an exploration bonus to train an optimal policy [Strehl and Littman, 2008]. When the environment has a large number of states, the agent may not visit the same state twice. To facilitate count-based exploration in such domains, the notion of visitation counts are generalized to that of "pseudocounts". Previous methods have equated the problem of estimating pseudocounts to the canonical machine learning problem of *density estimation*: the lower the probability density (under the learned model) of a given state, the higher the reward for reaching it [Bellemare et al., 2016, Ostrovski et al., 2017].

While providing the first way to estimate pseudocounts, Bellemare et al. [2016]'s relationship between counts and probability densities exists only when the density model meets the following restrictions [Ostrovski et al., 2017]:

- it must allow computing normalized probability densities, which precludes many powerful density models
- it must be *learning-positive*, which means that the probability density of a state must increase when it is encountered by the density model again,
- it must be updated on a state exactly once per visitation, precluding common techniques such as batching and replay.

Rather than designing a density model under such a restrictive setting, we propose to directly learn the exploration bonus associated with different states. Our proposed method does not place any restrictions on the type of function approximator used and can be trained using any and all of the common deep learning practices (such as using optimizers with adaptive learning rates, batching, experience replay and so on) [Goodfellow et al., 2016].

Given all the restrictions imposed by Bellemare et al. [2016]'s pseudocounts, it is tempting to forego count-based exploration in favor of other novelty estimates that have fewer theoretical guarantees. Random Network Distillation (RND) [Burda et al., 2019] is one such popular method that has achieved state-of-the-art performance on some hard exploration problems in the Arcade Learning Environment (ALE). RND assigns an exploration bonus to a state using a simple, elegant heuristic: the novelty of a state is directly proportional to how the accurately a learned network can mimic a randomly-initialized network's projection. If s is a state, f_θ is a neural network being learned and f_{θ^*} is a fixed randomly initialized neural network, the RND exploration bonus is given by $r_i(s) = ||f_\theta(s) - f_{\theta^*}(s)||^2$.

The output of a random network f_{θ^*} can have different scales depending on the random initialization of θ^* or the input state s. This makes RND sensitive to hyperparameter settings. Furthermore, unlike visitation counts, RND's exploration bonus does not have an intuitive interpretation—it is an unnormalized distance in a neural network's latent space. Normalization tricks are often used to stabilize training in practice, but they introduce their own unintended artifacts.

Another shortcoming of RND is that it conflates *encountering* a state and *updating* a model on that state. An RL agent may encounter a state exactly once, but the novelty associated with that state will fall every time RND's prediction model f_θ is updated on it. This is not ideal because a single gradient update is often insufficient for learning effective representations in high-dimensional spaces. As we will see in the next section, our procedure gets counts from the fixed point of an optimization procedure, allowing us to train our model to convergence.

Our core insight is that a state's visitation count can be derived from the sampling distribution of Bernouli trials made every time a state is encountered. When we use a neural network to predict this sampling distribution, we inadvertently learn the inverse of the state's visitation count. We test this procedure on a visual grid world and show that our method can recover the ground truth counts while other pseudocount methods cannot. When used as an exploration bonus, it causes a model-free RL agent to rapidly explore its environment and outperform strong model-free baselines.

2 Coin Flip Network (CFN)

Consider the fair coin flip distribution of -1 and +1. Imagine you flip this coin n times, and average the results into y_n. In expectation the average is 0, but any given time you run this experiment you likely get a non-zero value for y_n. For example, if you flip the coin once you will get $Pr(y_1 = -1) = 0.5$, and $Pr(y_1 = 1) = 0.5$. If you flip it twice, you get $Pr(y_2 = -1) = 0.25$ and $Pr(y_2 = 0) = 0.5$ and $Pr(y_2 = 1) = 0.25$. More generally, for all n the second moment of y_n is related to the inverse-count:

$$\mathcal{M}_2(y_n) = E[y_n^2] = \sum_i Pr(y_n = i) * i^2 = 1/n. \quad (1)$$

2.1 Bonuses from states

We construct a dataset of m encountered states $\{s_1, ..., s_m\}$, and pair each state with a random vector of d coin flips $c_i \sim \{-1, 1\}^d$. The mean of n random coin flip vectors is a vector of d samples from y_n:

$$\Big(\frac{1}{n}\sum_{i=1}^{n} c_i\Big)_j \sim y_n \tag{2}$$

Consider the class of functions $\mathcal{F}$ such that $f \in \mathcal{F} : R^{|S|} \to R^d$. Our goal is to find $f^* \in \mathcal{F}$ that best predicts each c_i:

$$f^*(s) = \arg\min_f \sum_{i=1}^{m} \|c_i - f(s_i)\|^2 = \arg\min_f \sum_{i=1}^{m}\sum_{j=1}^{d} (c_{ij} - f(s_i)_j)^2 \tag{3}$$

If each state s_i is encountered exactly once, the optimization problem above will simply learn the mapping from states to their associated random vector. However we are interested in cases where there are multiple instances of some states. In that case, the best you can do is to learn the *mean* random vector for all instances of a given state. Let's look at one such state s, of which there are n occurrences, and each c_i is a d-dimensional sample from the coinflip distribution:

$$\begin{aligned}
f^*(s) &= \frac{1}{n}\sum_{i=1}^{n} c_i \\
\implies E\Big[\frac{1}{d}\|f^*(s)\|^2\Big] &= \frac{1}{d}\sum_{j=1}^{d} E\Big[\Big(\sum_{i=1}^{n}\frac{c_{ij}}{n}\Big)^2\Big] && \text{[Taking expectation of the vector norm]} \\
&= \frac{1}{d}\sum_{j=1}^{d} E\Big[y_n^2\Big] && \text{[Using Eq 2]} \\
&= \frac{1}{d}\sum_{j=1}^{d} \frac{1}{n} = \frac{1}{n} && \text{[Using Eq 1]}
\end{aligned}$$

This suggests that in expectation, $f^*(s)$ contains an unbiased and consistent estimator of the inverse-count. **We can now use this property of f^* to map states to bonuses.** We train a neural network $f_\theta(s)$ to approximately solve the optimization problem described in Equation 3, and obtain exploration bonuses using:

$$\mathcal{B}(s) := \sqrt{\frac{1}{d}\|f_\theta(s)\|^2} \approx \frac{1}{\sqrt{N(s)}} \tag{4}$$

2.2 Bonuses from many states

$f_\theta(s)$ is a deterministic function that maps each input to a single output: that's why the best we can do is learn the *mean* of a state's random vectors. But, we can learn a different mean for each state. A learning architecture with infinite capacity would learn the exact mean for each unique state. However, finite capacity and training time imply that the network will not learn this mapping exactly. Next, we will discuss ways in which we can control how our network trades off prediction errors among states in the data set and generalizes to unseen states during inference.

2.3 Prioritizing Novel States

Training f_θ to convergence at every time step is not feasible. To maintain reasonable training time, we update it once every time step on a single mini-batch of states drawn from its replay buffer. Revisiting the optimization target from Equation 3, we note that a *unique* state s visited n times will appear in uniform sampling n times more than a state visited only once. This would make f_θ disproportionately better at predicting the bonus corresponding to high-count states. To remedy this problem, we assign more weight to low-count states. Note that we can re-weight the optimization target from Equation 3 by scaling each state's contribution to the loss by its *true* count, without changing the ideal fixed point $f^*(s)$:

$$f^*_{\text{prioritized}}(s) = \arg\min_f \sum_{i=1}^{m} \frac{1}{\text{True Count}(s_i)}\|c_i - f(s_i)\|^2.$$

Of course, we do not have access to the true count during training; so, we approximate this procedure by prioritizing by our current *estimate* of inverse-count:

$$\text{priority}(s) = \frac{1}{d}\|f_\theta(s)\|^2.$$

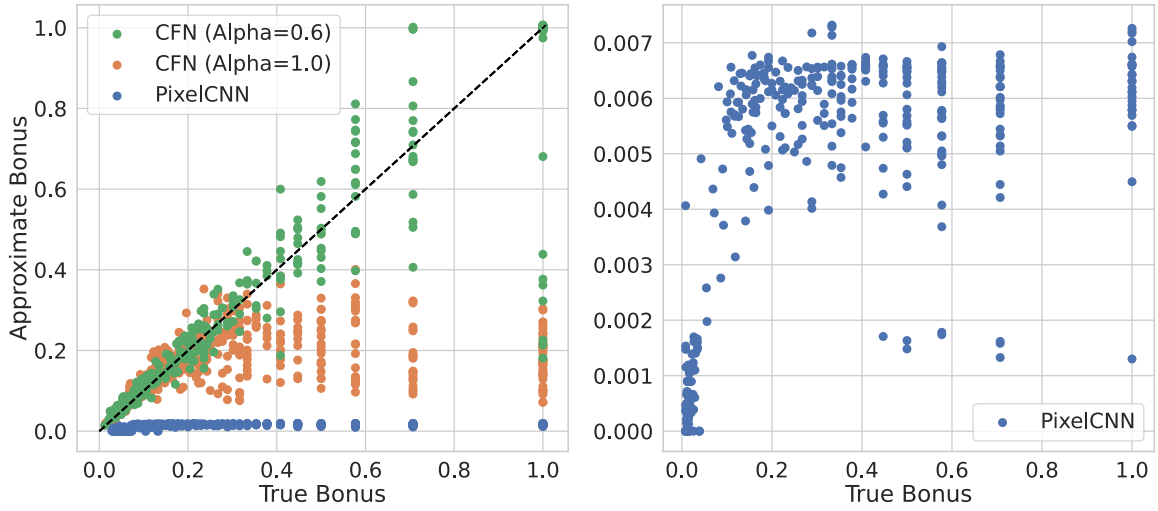


Figure 1: *(left)* Comparing bonus estimation accuracy for different variants of CFN and PixelCNN on the Visual Gridworld task (end of training). *(right)* Zoomed in version of PixelCNN's bonus prediction accuracy from the left subplot.

Prioritizing in this way introduces another difficulty: if a state has been recently added to the replay buffer, it has not appeared in many gradient updates and thus we cannot trust our estimate of its count. To combat this we also prioritize sampling by the number of times, $n_{\text{updates}}(s)$, we have sampled s in the past. We combine both these prioritization schemes using an α-weighted sum:

$$\text{priority}(s) = \alpha \left(\frac{1}{n_{\text{updates}}(s)} \right) + (1-\alpha)\frac{1}{d}\|f_\theta(s)\|^2.$$

2.4 Temporal Consistency for Better Generalization

The training scheme described so far ignores temporal structure present in sequential decision problems; to encourage meaningful generalization, we want our training scheme to reflect the temporal relations between states. We encourage sequential states to have similar latent representations by only "replacing" each random coin flip with a probability p_r:

$$c_{i+1,j} = \begin{cases} \{-1, 1\} & \text{with probability } p_n \\ c_{i,j} & \text{with probability } (1-p_n) \end{cases}$$

This makes the targets for sequential states similar, which in turn encourages them to have similar latent representations.

3 Experiments

We test our algorithm on a "visual grid world" task [Allen et al., 2021]. In this task, the agent gets an 84×84 image observation of the grid. The controllable block starts every episode at the bottom-left cell; it gets a terminating reward of 1 when it reaches the goal at the top-right of the grid and a reward of 0 everywhere else. For all experiments we use Rainbow DQN [Hessel et al., 2018] as a base agent and modify it with various exploration bonuses.

3.1 Evaluating Bonus Prediction Accuracy

We test CFN's ability to correctly predict the inverse visitation count in a 21×21 visual grid world. The learning agent gets image-based observations as input, but we keep track of the ground truth state (the agent's x, y location in the grid) to evaluate how well different methods can reconstruct the true bonus (or equivalently, $1/\sqrt{n_s}$). Figure 1 *(left)* shows the importance of α (as described in section 2.3)—when the ground-truth bonus is high (i.e, when the visitation count is low), CFN with $\alpha < 1$ outperforms CFN with $\alpha = 1$ because $\alpha < 1$ up-weights prediction errors on novel states. Figure 1 also shows that PixelCNN is unable to learn a meaningful count—not only are its predictions very close to 0, it assigns very similar exploration bonus to states that have been seen 25 times (which should have a true bonus of 0.2) and states that have been seen only once (which should have a true bonus of 1.0). In other words, PixelCNN does not recover the true counts in this domain *and* it dramatically underestimates the exploration bonus for truly novel states.

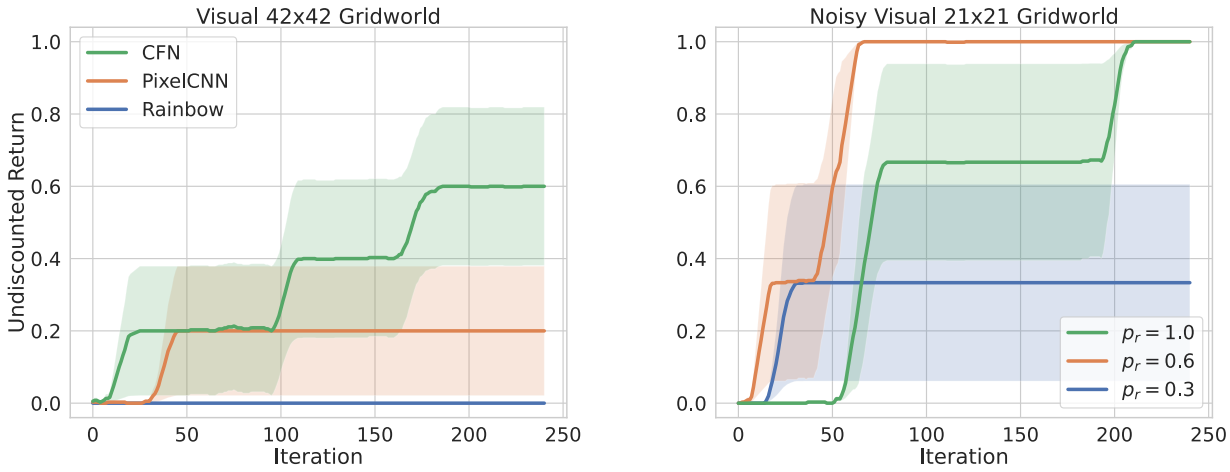


Figure 2: *(left)* Learning curves comparing CFN with Rainbow and PixelCNN on a 42×42 grid. *(right)* Comparing CFN with different values of p_r on 21×21 grid with noisy observations. Each line denotes the mean undiscounted return over 5 random seeds, shaded regions denote standard error. Each iteration is 1000 action executions in the environment.

3.2 Reinforcement Learning with CFN Bonus

RL in Visual Gridworld. We test if CFN's exploration bonus can lead to sample-efficient reinforcement learning. For this experiment, we consider a 42×42 visual gridworld. As shown in Figure 2 *(left)*, a Rainbow agent [Hessel et al., 2018] is unable to explore this grid and get any positive reward. When augmented with the CFN exploration bonus, the same agent rapidly explores the state-space and gets high return. We also compare our CFN agent to the most popular way of computing pseudocounts, PixelCNN [Ostrovski et al., 2017]; we find that CFN comfortably outperforms PixelCNN.

RL in Noisy Visual Gridworld. We now test if CFN is robust to observation noise. For this experiment, we consider a 21×21 visual grid world and add speckled noise to the visual observations. Figure 2 *(right)* shows that p_r (described in Section 2.4) helps CFN generalize to similar inputs and be robust to observation noise.

4 Conclusion

Though pseudocount-based exploration methods are a principled way to perform thorough exploration in sparse-reward reinforcement learning problems, they have not been the dominant algorithm used in practice; we aim to remedy that. We point to restrictions on the functional form, difficulty of implementation, and inability to use with a wealth of deep-learning training tricks, as a few reasons for this theory-practice divide. In contrast, CFNs estimate counts through a simple optimization procedure over the data set of encountered states, and can be used with most standard deep learning architectures. We find that CFNs explore more rapidly and represent ground-truth counts better than existing pseudocount methods. In this paper we focus on a simple environment where we can directly compute the counting accuracy; in future work, we plan to scale to more complicated domains like the hard exploration games in the ALE.

References

Cameron Allen, Neev Parikh, Omer Gottesman, and George Konidaris. Learning markov state abstractions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1lJJnR5Ym`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

376

Harnessing the wisdom of an unreliable crowd for autonomous decision making

Tamlin Love, Ritesh Ajoodha and Benjamin Rosman
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1438243@students.wits.com, ritesh.ajoodha@wits.ac.za and benjamin.rosman1@wits.ac.za

Abstract

In Reinforcement Learning there is often a need for greater sample efficiency when learning an optimal policy, whether due to the complexity of the problem or the difficulty in obtaining data. One approach to tackling this problem is to introduce external information to the agent in the form of domain expert advice. Indeed, it has been shown that giving an agent advice in the form of state-action pairs during learning can greatly improve the rate at which the agent converges to an optimal policy. These approaches typically assume a single, infallible expert. However, it may be desirable to collect advice from multiple experts to further improve sample efficiency. This may introduce the problem of multiple experts offering conflicting advice. In general, experts (especially humans) can give incorrect advice. The problem of incorporating advice from multiple, potentially unreliable experts is considered an open problem in the field of Assisted Reinforcement Learning.

Contextual bandits are an important class of problems with a broad range of applications such as in medicine, finance and recommendation systems. To address the problem of learning with expert advice from multiple, unreliable experts, we present CLUE (Cautiously Learning with Unreliable Experts), a framework which allows any contextual bandit algorithm to benefit from incorporating expert advice into its decision making. It does so by modelling the unreliability of each expert, and using this model to pool advice together to determine the probability of each action being optimal.

We perform a number of experiments with simulated experts over randomly generated environments. Our results show that CLUE benefits from improved sample efficiency when advised by reliable experts, but is robust to the presence of unreliable experts, and is able to benefit from multiple experts. This research provides an approach to incorporating the advice of humans of varying levels of expertise in the learning process.

Keywords: Assisted Reinforcement Learning, Interactive Reinforcement Learning, Agent Teaching, Contextual Bandits, Expert Advice

1 Introduction

Sample efficiency is often an issue of great concern in Reinforcement Learning (RL). This is often due to the complexity of a problem, which may consist of a large number of states and actions. It may also be due to the difficulty in acquiring data. The Assisted Reinforcement Learning (ARL) framework seeks to improve sample efficiency by incorporating external information in the learning process [1]. For example, a domain expert could advise an expert on which action it should perform in a given state. This advice could be given throughout the learning process, in response to the agent's behaviour; an approach termed Interactive Reinforcement Learning (IRL). The types of advice offered by an expert may differ between approaches. In this work, we consider policy-shaping advice in the form of a single action offered for a state, as it is often easier to elicit from a domain expert and is more robust to infrequent and inconsistent feedback [7].

It is often assumed that advice is coming from a single, infallible expert. These assumptions are not always practical, however. Experts, especially humans, can give suboptimal advice due to misunderstandings (e.g. advice is given for the wrong state), erroneous domain knowledge or on purpose with the intent of sabotaging the agent. Restricting advice to a single expert also limits the amount of information the agent can receive. Multiple experts can potentially have different perspectives or areas of expertise, and the contradictions and consensus between experts may reveal additional information.

Our aim is to build on the IRL approach in the specific context of contextual bandits (CBs), which are in essence RL problems whose episodes are a single timestep in length, where we attempt to tackle the open problem of incorporating the advice of multiple, potentially unreliable experts in policy-learning. Our main contribution in this regard is CLUE (Cautiously Learning with Unreliable Experts). This framework uses a model of the reliability of the experts to augment any CB action-selection algorithm with the ability to incorporate advice from multiple experts.

Related Work: There have been some approaches to tackling the problems of multiple experts and of unreliable experts, though the experts in these approaches often provide other forms of advice than the action advice we consider. Gimelfarb, Sanner, and Lee [6] combine reward-shaping advice from multiple experts as a weighted sum of potential functions, where the weights are updated as the agent learns. The decision-making rule in Section 2.2 is directly inspired by this Bayesian combination of advice. Griffith et al. [7] account for incorrect advice by modelling the probability of an expert giving correct advice with a single, static parameter $C \in (0, 1)$. Such a model of reliability is expanded on in Section 2.1. Other approaches include the adversarial bandit algorithms EXP4 and EXP4.P, whose experts provide advice in the form of probability vectors [10], and the probabilistic policy reuse algorithm, in which experts' entire policies are transferred and weighted against the agent's own policy using the reward [5].

2 Methodology

In this section we describe the CLUE framework and the problem setting, which is composed of three actors: an environment, an agent and a panel E of one or more experts. The environment is a standard Contextual Bandit (CB) environment. For each trial t, it samples state s_t, accepts action a_t from the agent and returns reward r_t. At the end of the trial, each expert e in panel E receives $\langle s_t, a_t, r_t \rangle$ and may independently offer their own advice, $(s_t, a_t^{(e)})$ on what action the agent should have taken this trial. How and when an expert decides to offer advice may differ between experts. In our formulation of the problem, we assume each expert to be a domain expert with consistent reliability across the breadth of the problem. It is worth noting here that, although we choose to have the expert give advice at the end of the trial in this work, this can occur instead at the start of a trial without requiring any change to the CLUE algorithm.

The agent is composed of three components, the first of which is a learning algorithm, which uses the information $\langle s_t, a_t, r_t \rangle$ to learn a policy, such as the action-value update rule $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t - Q(s_t, a_t))$, where $Q(s, a)$ is the action-value function and $\alpha \in [0, 1]$ is a step size parameter.

The second component, and one of the contributions of this work, is a model of the reliability of each expert (see Section 2.1). This model is necessary for learning which pieces of advice are to be followed and which are to be ignored. When an expert utters a piece of advice at the end of a trial, the agent uses its own information about the environment (such as an action-value function) to evaluate the advice and update the model. The third component, and another contribution of this work, is a decision making process which uses the information learned by the learning algorithm and the models of each expert to select an action for a state while exploring, given any advice it has previously received for that state (see Section 2.2).

2.1 Modelling Experts

Intuitively, we can think of an expert's reliability as the probability of them giving optimal advice [7]. We can therefore model it as $\rho \in [0, 1]$, where $\rho = 1$ is an infallible expert and $\rho = 0$ is an expert that always gives suboptimal advice. Given the range of values, a natural choice is to model the probability distribution $P(\rho)$ as a Beta distribution $Beta_\rho[\alpha, \beta]$,

where α, $\beta > 0$ can be thought of as counts recording the number of times the expert gave correct and incorrect advice respectively.

At the end of trial t, the agent must update this distribution for each expert that gave advice for s_t sometime in the past. To do this, the agent can evaluate the advice as either optimal or suboptimal, given its own information. In this work, we set $x_t = 1$ if $Q(s_t, a_t^{(e)}) = max_a Q(s, a)$, and $x_t = 0$ otherwise, where $a_t^{(e)}$ denotes the advice received from expert e. Let $x_t = 1$ denote an optimal evaluation, and $x_t = 0$ denote a suboptimal evaluation. In order to allow for inconsistent experts (e.g. an expert whose performance degrades over time), we update an estimate χ of the expected value $\mathbb{E}[\rho]$ using a recency-weighted moving average with weight parameter $\delta \in [0, 1]$,

$$\chi_{t+1} = (1 - \delta)\chi_t + \delta x_t, \tag{1}$$

where $\chi_0 = \mathbb{E}_0[\rho] = \frac{\alpha_0}{\alpha_0 + \beta_0}$, with prior counts α_0 and β_0.

2.2 Making Decisions

Suppose that, at the start of trial t, the agent observes state s_t and recalls any advice that some subset $E_t \subseteq E$ of experts offered for state s_t in trials $[0, ..., t - 1]$. The agent must now use its model of the reliability of each expert to decide which advice (if any) to follow. In order to allow the agent to surpass the performance of the experts advising it, we only allow the agent to consider expert advice when exploring. Determining when the agent is exploring depends on the underlying action-selection algorithm. As CLUE can augment any CB action-selection algorithm, we consider the Epsilon-Greedy, Adaptive Greedy, Explore-then-Exploit (ETE) and Upper Confidence Bound (UCB) algorithms, representing several families of CB algorithms [4]. For the first three algorithms, whether or not the agent is exploring is explicitly determined by the algorithms' parameters. For UCB, the agent can be said to be exploring if $argmax_a Q(s, a) \neq argmax_a(Q(s, a) + c\sqrt{\frac{2ln(t)}{N(s,a)}})$, where $N(s, a)$ counts the number of times action a has been selected for state s and c is a parameter that balances exploration and exploitation.

If exploring, the agent must choose between the action suggested by the underlying action-selection algorithm or between following advice it has received for s, in which case it must choose which advice to follow. If $E_t = \varnothing$, no advice has been offered, such as may happen at the beginning of the learning process, and the agent must act without advice according to its underlying action-selection algorithm. If $|E_t| \geq 1$, at least one expert has offered advice. In order to take advantage of the information provided by consensus and contradiction among experts, we employ a Bayesian method of pooling advice, inspired by similar approaches in potential-based reward shaping [6] and in crowd-sourced data labelling [2]. Let a^* denote the optimal action for state s_t and $v_t^{(e)}$ denote the advice utterance given by expert e for s_t, with V_t denoting the set $\{v_t^{(e)} | e \in E_t\}$. Our aim, therefore, is to calculate $P(a_j = a^* | V_t)$ for each $a_j \in A$. To do this, we employ Bayes' rule, coupled with the assumptions that each expert gives advice independently of every other expert and that each action has a uniform prior probability of being optimal,

$$P(a_j = a^* | V_t) = \frac{\prod_{e \in E_t} P(v_t^{(e)} | a_j = a^*)}{\sum_{k=0}^{|A|} \prod_{e \in E_t} P(v_t^{(e)} | a_k = a^*)}. \tag{2}$$

Note that, if for a particular domain one can reasonably assume a non-uniform prior distribution of $P(a = a^*)$, this distribution can be incorporated into Equation 2 without fundamentally changing this decision-making process.

All that remains is to calculate $P(v_t^{(e)} | a_j = a^*)$. Recalling that the probability of the advice being correct is estimated by $\chi^{(e)} \approx \mathbb{E}[\rho^{(e)}]$ and assuming that, if the advice is incorrect, the expert is equally likely to advise any suboptimal action, then $P(v_t^{(e)} | a_k = a^*) = \chi^{(e)}$ if the expert advised a_k and $P(v_t^{(e)} | a_k = a^*) = \frac{1 - \chi^{(e)}}{|A| - 1}$ otherwise. Substituting this into Equation 2, we can calculate the probability of each action in A being optimal, and can set $a_{best} = \arg\max_a P(a = a^* | V_t)$. In an approach reminiscent of both Epsilon Greedy and probabilistic policy reuse [5], the agent selects action a_{best} with probability $P(a_{best} = a^* | V_t)$, and otherwise acts as if $E_t = \varnothing$. This allows for a trade-off between following advice and exploring as normal, where the former is more likely if the agent is confident that a_{best} is optimal.

In the above formulations, we have assumed that the estimated $\chi^{(e)}$ accurately represents the underlying reliability of the expert e. Early in the learning process however, this will not be the case. Erring on the side of caution, we can compensate for the over-estimation of the reliability of particularly bad experts by introducing a threshold parameter $T \in [0, 1]$, such that if $P(a_{best} = a^* | V_t) < T$, the agent acts without advice. This approach ensures that the agent will only follow advice if it is sufficiently confident that the advice is correct. 379

3 Experiments and Results

In this section, we present a number of experiments to demonstrate that CLUE benefits from improved sample efficiency when being advised by a reliable expert but is robust to the presence of suboptimal advice. Furthermore, we aim to show that CLUE can benefit from advice from a panel of multiple experts with varying degrees of reliability.

These experiments are performed using a number of randomly generated Contextual Bandit environments, specified by Influence Diagrams with a great diversity of randomly generated graph structures and parameters [8]. Experts are simulated, and are limited in how much advice they can give, thus only giving advice if the agent is underperforming within some degree of tolerance [9]. In order to simulate reliability, each expert is controlled by a *true reliability parameter* ρ_{true}. When offering advice, the expert will advise the optimal action a^* with probability ρ_{true}, or else will randomly advise any other action. Thus an expert with $\rho_{true} = 1$ is reliable, while one with $\rho_{true} = 0$ never advises the optimal action.

3.1 Panel Comparisons

In this set of experiments, we compare the reward obtained in $80,000$ trials, averaged across 100 random environments ($|S| = 1024$, $|A| = 8$). LOWESS smoothing is employed for legibility [3], with the standard deviation represented by the shaded areas. We compare the performance of each agent with three panels of experts. The first, a *Single Reliable Expert*, consists of one expert that always gives correct advice ($\rho_{true} = 1$). The second, a *Single Unreliable Expert*, consists of one expert that always gives incorrect advice ($\rho_{true} = 0$). The third, a *Varied Panel*, consists of seven experts with varying degrees of unreliability ($P_{true} = \{0, 0.1, 0.25, 0.5, 0.75, 0.9, 1\}$). Agents tested include an unassisted *Baseline Agent*, a *Naïve Advice Follower* (NAF), which follows any advice it has received for a state (choosing randomly between contradicting advice) otherwise acting as the Baseline Agent, and CLUE ($\alpha_0 = 1 = \beta_0$, $T = \frac{2}{|A|}$, $\delta = 0.5$), which augments the Baseline Agent. The four tested baselines are Epsilon Greedy (ϵ decays from 1 to 0 across 80% of trials), Adaptive Greedy (z decays from 1 to -1 across 80% of trials), Explore-then-Exploit (ETE, threshold of $20,000$) and Upper Confidence Bound (UCB, $c = 0.25$), all of which employ the Q update rule in Section 2 ($Q_0 = 0, \alpha = \frac{1}{k(s,a)}$). Results are shown in Figure 1.

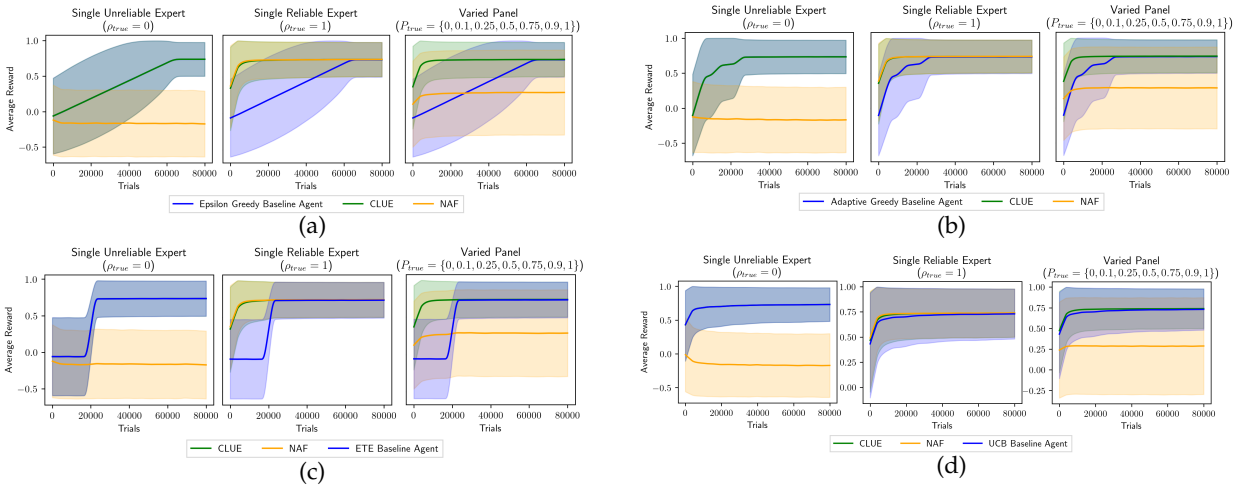


Figure 1: Panel comparisons for (a) Epsilon Greedy, (b) Adaptive Greedy, (c) ETE and (d) UCB. Note that CLUE and the Baseline are nearly identical for $\rho_{true} = 0$.

For $\rho_{true} = 1$, both CLUE and NAF converge faster than all Baselines as they quickly benefit from the optimal advice provided by the reliable expert. A demonstration of the robustness of CLUE comes when $\rho_{true} = 0$. In this scenario, NAF exclusively follows sub-optimal advice and thus is unable to converge to the optimal policy. CLUE on the other hand is able to identify that the expert is unreliable and defaults to its underlying action-selection algorithm, performing identically to the Baselines. For the varied panel, the performance of NAF lies somewhere between the two single expert cases, as it receives a mix of advice including optimal and suboptimal actions, and cannot discern which advice is advantageous to follow. CLUE is able to differentiate between reliable and unreliable experts and benefits from the former despite the presence of the latter. In all cases, CLUE either converges faster than the Baseline when good advice is available, or otherwise converges at the same rate as the Baseline.

3.2 Reliability Estimates

To investigate the results obtained in Section 3.1, we plot the value of $\chi^{(e)}$ over time for the same panels of experts and with an Epsilon Greedy Baseline. Results are plotted in Figure 2.

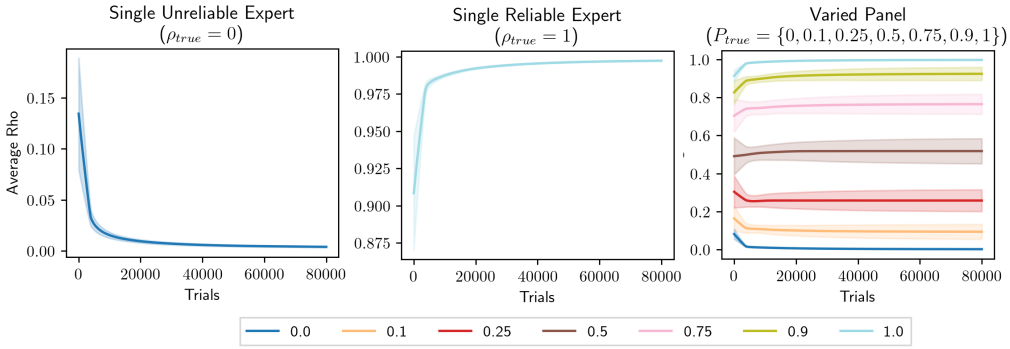


Figure 2: A comparison of $\chi^{(e)}$ for each panel with an Epsilon Greedy Baseline. The Legend denotes the value of ρ_{true}

For the single expert cases, the value of χ converges towards the correct value of ρ_{true} (1 and 0 respectively), with the final estimates being $\chi = 0.995$ for the single reliable expert and $\chi = 0.005$ for the single unreliable expert. For the varied panel, each expert is correctly ranked according to their reliability and the value of $\chi^{(e)}$ for each expert e correctly converges towards the true value of $\rho^{(e)}_{true}$, even faster than the single expert cases. This accuracy in the estimates of reliability explains the performance obtained in Section 3.1. As is to be expected, the variance in the final estimate is larger for experts that randomly choose between suboptimal and optimal advice ($\rho_{true} = 0.5$) than for experts that more consistently offer one or the other.

4 Conclusion

Our results show that CLUE is able to incorporate expert advice in such a way that it benefits from improved sample efficiency when advised by a reliable expert, but is robust to advice from unreliable experts. Furthermore, by modelling the reliability of the experts, CLUE is able to incorporate advice from multiple experts, even when these experts contradict each other. When multiple experts are present, CLUE is able to rank them by their reliability and exploit the information revealed by consensus and contradiction between experts. This work may allow for easier integration of external information in the learning process, ultimately contributing towards tackling more complex problems with greater sample efficiency.

References

[1] Adam Bignold et al. “A conceptual framework for externally-influenced agents: an assisted reinforcement learning review”. In: *Journal of Ambient Intelligence and Humanized Computing* (2021), pp. 1–24.

[2] Pierce Burke and Richard Klein. “Confident in the Crowd: Bayesian Inference to Improve Data Labelling in Crowdsourcing”. In: *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE. 2020, pp. 1–6.

[3] William S Cleveland. “LOWESS: A program for smoothing scatterplots by robust locally weighted regression”. In: *American Statistician* 35.1 (1981), p. 54.

[4] David Cortes. “Adapting multi-armed bandits policies to contextual bandits scenarios”. In: *arXiv preprint arXiv:1811.04383* (2018).

[5] Fernando Fernández and Manuela Veloso. “Probabilistic policy reuse in a reinforcement learning agent”. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 720–727.

[6] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. “Reinforcement learning with multiple experts: A Bayesian model combination approach”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 9528–9538.

[7] Shane Griffith et al. “Policy shaping: Integrating human feedback with reinforcement learning”. In: Georgia Institute of Technology. 2013.

[8] Ronald A Howard and James E Matheson. “Influence diagrams”. In: *Decision Analysis* 2.3 (2005), pp. 127–143.

[9] Craig Innes and Alex Lascarides. “Learning Structured Decision Problems with Unawareness”. In: *International Conference on Machine Learning*. 2019, pp. 2941–2950.

[10] Li Zhou. “A survey on contextual multi-armed bandits”. In: *arXiv preprint arXiv:1508.03326* (2015).

Challenges in Finetuning from Offline RL: An Empirical Study

Yicheng Luo UCL
Jackie Kay UCL, DeepMind
Edward Grefenstette UCL
Marc Peter Deisenroth UCL

Abstract

Offline reinforcement learning allows the training of competent agents from offline datasets without any interaction with the environment. However, depending on the dataset's quality, finetuning may be desirable to improve performance further. While offline RL algorithms can, in principle, be used for online finetuning, the online performance improves slowly in practice. We show that it is possible to use standard online off-policy algorithms to achieve competitive finetuning performance. However, this approach suffers from issues with initial training instability known which we referred to as policy collapse. We study the issue of policy collapse empirically and investigate approaches to stabilize online off-policy finetuning from offline RL. We show that conservative policy optimization is a promising solution that can stabilize online off-policy algorithms for finetuning. We analyze when conservative policy optimization is useful and discuss alternative strategies to stabilize finetuning when they fail.

Keywords: reinforcement learning, offline reinforcement learning, finetuning, transfer learning

Acknowledgements

This work is supported by the UKRI [grant number EP/S021566/1].

1 Introduction and Preliminaries

Offline reinforcement learning [8, 11] considers the problem of learning policies from fixed datasets without requiring additional interaction with the real environment. By construction, offline RL algorithms rely heavily on the quality of the data, and it may be impossible to reach optimal performance with offline data only; online data is still needed to further improve the policy's performance. While offline RL algorithms can, in theory, be utilized for sample-efficient online learning, in practice, they suffer from low sample efficiency due to distributional shifts from the fixed dataset to the online environment [13, 10]. In this paper, we study how to improve offline pre-trained policies with additional online finetuning. We found empirically that offline RL algorithms converge more slowly than online off-policy algorithms, possibly because the conservative nature of the offline RL algorithms hinders effective online finetuning. Previous work [13, 10] showed that conservative offline RL, such as Conservative Q Learning (CQL) [7], is not well-suited for finetuning. Our result complements their work by demonstrating similar findings with TD3-BC [3]. We also show that sometimes simply switching to a standard off-policy RL algorithm works well for online finetuning. However, we observe that finetuning with online off-policy algorithms suffers from *policy collapse*, where the policy degrades severely during initial training when the offline dataset has low diversity. We hypothesize that effective online finetuning from offline RL may be achieved with more robust online policy optimization and present a partial solution, a constrained policy update on top of the TD3 algorithm, which we call conservative TD3 (TD3-C), that empirically helps stabilize online finetuning. Finally, our results suggest that better evaluation protocols should be considered when evaluating finetuning performance of RL algorithms.

Offline RL with Online Finetuning We consider reinforcement learning (RL) in a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma)$. In addition to the standard reinforcement learning setting, we assume having an additional dataset $\mathcal{B} = \{(s_i, a_i, r_i)\}_{i=1}^N$ of experience collected by an intractable behavior policy μ in the same MDP. We are interested in utilizing the offline fixed dataset $\mathcal{B}$ to improve the agent's online sample efficiency. Concretely, we consider the setting where we first pretrain a policy using offline RL and then further finetune it online, similar to the scenario considered in [6, 10, 13].

Off-policy reinforcement learning Off-policy RL algorithms learn a policy π_θ with experience generated by a different policy μ. Example algorithms include Deep Deterministic Policy Gradient (DDPG) [12] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [5]. These deep off-policy algorithms learn an approximate state-action value function Q_ϕ and deterministic policy π_θ by alternating policy evaluation and improvement. During policy evaluation, we learn an approximate state-action value function Q_ϕ by minimizing the Bellman error

$$\phi^* = \min_\phi \mathbb{E}_{s,a,r,s' \sim \mathcal{B}} \left[(Q_\phi(s,a) - (r + \gamma Q_{\phi'}(s', \pi_{\theta'}(s')))^2 \right], \tag{1}$$

where $Q_{\phi'}$ and $\pi_{\theta'}$ are the target critic and policy networks used to stabilize TD learning with function approximation. During policy improvement, the policy is updated to maximize the current state-action value function

$$\theta^* = \max_\theta \mathbb{E}_{s \sim \mathcal{B}} \left[Q_\phi(s, \pi_\theta(s)) \right]. \tag{2}$$

In principle, we can apply off-policy algorithms, such as DDPG or TD3, to learn from a fixed dataset; however, in practice, they are ineffective when learning from fixed offline datasets due to extrapolation errors [4]. Recently, many deep offline RL algorithms [6, 7, 15, 4] have been proposed to reduce the extrapolation error. These algorithms modify standard deep off-policy algorithms and constrain policy learning to be supported by the dataset, which helps minimize extrapolation error. However, when offline RL algorithms are used for finetuning, they typically improve more slowly compared to their off-policy backbone [13, 10]. For example, [10] found that online finetuning with the offline RL algorithm CQL results in little improvement with addition of online data.

In this paper, we show that, instead of offline RL algorithms, online finetuning by standard off-policy algorithms is a surprisingly strong baseline on the D4RL MuJoCo benchmark [2]. However, we also found that this simple approach suffers from varied degrees of training instability and performance degradation on different D4RL MuJoCo datasets. Finally, we propose a solution that improves the training instability while enjoying fast online improvement. We also perform ablations on our proposed method and discuss further useful findings that we found to stabilize online finetuning.

2 Evaluation

In this section, we empirically analyze the challenges in performing online finetuning after pretraining with offline RL. Our analysis builds on top of MuJoCo tasks in the D4RL benchmark suite [2]. We consider datasets from the walker2d, halfcheetah and hopper tasks. For each task, we perform finetuning given the corresponding medium, medium-replay, and expert datasets. The medium datasets consist of transitions collected by an early-stopped, suboptimal agent. medium-replay datasets refer to the transitions stored in the replay buffer of an early-stopped agent, and expert refers to the transitions collected by an expert agent after a complete training run.

For offline pre-training, we use TD3-BC [3]. TD3-BC is a simple offline RL algorithm that extends the TD3 algorithm by including an additional behavior cloning (BC) term in the policy improvement step to encourage the policy to stay close to the behaviors in the offline dataset. We selected TD3-BC because of its strong empirical performance on the MuJoCo benchmarks.

During online finetuning, we load the weights for the neural networks obtained from offline training and use either TD3 [5] or TD3-BC as the finetuning algorithm. In both cases, we pretrain the actor and the critic offline for $500K$ iterations and then train online for $1M$ environment steps. Figure 1 shows the learning curve for both set-ups.

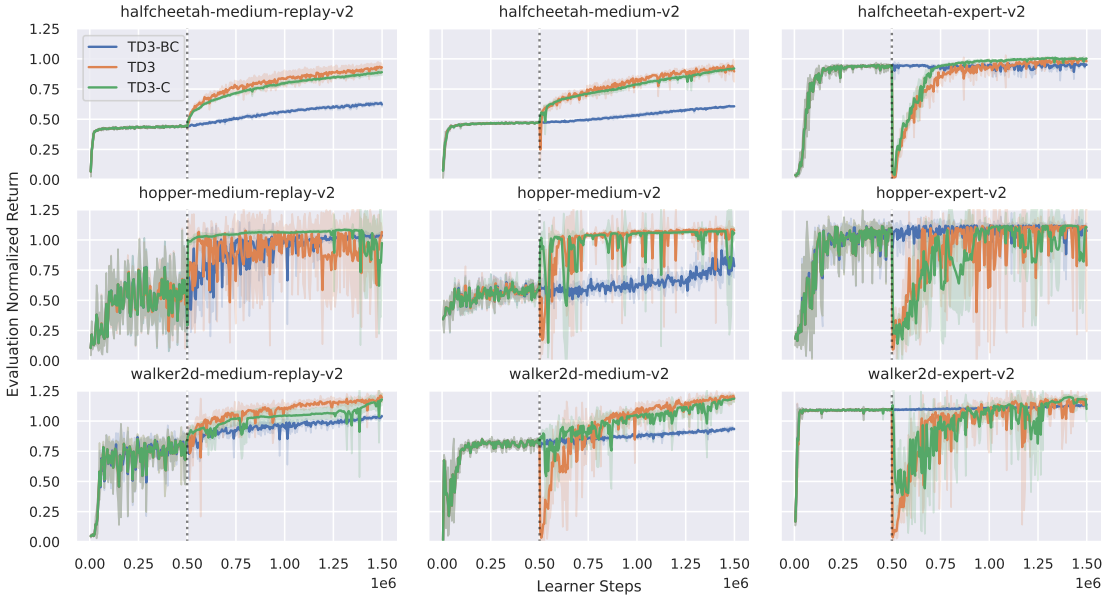


Figure 1: Comparison between TD3, TD3-BC and our proposed conservative TD3 (TD3-C) for online finetuning on the D4RL benchmark suite. Dashed lines indicate the end of offline learning and the beginning of finetuning. Results are averaged with three random seeds. The result is not smoothed, which accurately shows the effect of collapse.

2.1 Empirical Findings

Offline RL algorithms improve more slowly compared to their online counterparts. Figure 1 reveals a few interesting findings. First, we can see that finetuning online with TD3-BC improves slowly compared to using TD3. This suggests that offline RL algorithms that constrain the target policy to be close to behavior policy may improve more slowly than their standard off-policy counterparts. While we restrict our comparison to using the TD3 algorithm as the base RL algorithm, previous work [10, 13] shows similar findings for other offline RL algorithms.

Online finetuning with off-policy algorithms suffers from policy collapse. While online finetuning with TD3 achieves a better evaluation score compared to TD3-BC, there is noticeable training instability for some datasets at the beginning of online finetuning. This phenomenon is sometimes referred to as *policy collapse*. We hypothesize that collapse happens as the critic is inaccurate when finetuning starts and is over-optimistic on novel states encountered early in finetuning. Nevertheless, the asymptotic performance of finetuning with TD3 after 1M environment steps always surpasses TD3-BC.

Policy collapse is more severe when the diversity of dataset is low. The extent of instability varies across domains and dataset qualities and is more noticeable as the diversity of the dataset decreases. Although the offline performance on the medium and medium-replay datasets are comparable, finetuning from agents pretrained with TD3 on the medium datasets is more unstable. Notice that the rate at which TD3 recovers its original performance also varies across the datasets, and it is more difficult to recover the full performance when pretrained on expert datasets. This suggests that it is difficult to establish a standardized benchmark for studying finetuning: if we choose a small sampling budget for online finetuning, then TD3-BC is actually preferable despite having lower performance when more online interactions are allowed.

Policy collapse happens even with mild change in the online sampling distribution. We investigate the effect of discarding the offline data for performing online updates. To do this, we compare the online performance of agents finetuned with TD3-BC by loading or not loading the offline dataset into the online agents' replay buffer, similar to the

experiments considered in [14]. The result is illustrated in fig. 2. TD3-BC remains stable during the initial period of finetuning. This holds even in the absence of offline data in the online replay buffer. On the other hand, TD3 collapses immediately *independently* of whether we include the offline dataset in the online buffer. This is perhaps surprising since when we keep the offline data for online sampling, during the initial period of training, there is very little data from the online collection. In this case, we would expect that the sampled transitions not to be drastically different from the offline training distribution, and the transferal to online finetuning would therefore be more stable. However, this experiment suggests that offline RL algorithms are sensitive to even mild changes in the sampling distribution.

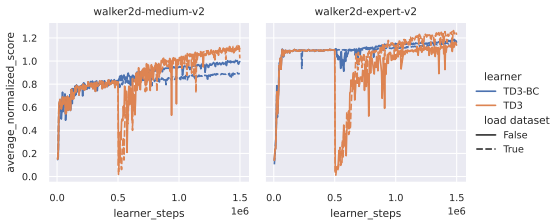


Figure 2: Effect of using offline datasets for online finetuning.

2.2 Negative Results

The critic suffers from significant extrapolation error when first deployed online. Therefore, we considered many approaches that we thought will mitigate this issue prior to the approach that we will discuss in section 2.3. However, we have not been able to succeed with any of these ideas. This section shows that bootstrap error even under mild distribution shift is severe and that standard regularization is insufficient to ensure stable online finetuning.

We tried running the policy evaluation only in the first 500K online steps before improving the policy and still observed significant policy collapse as soon as we start updating the policy. This suggests that it is not sufficient to avoid collapse by having a more accurate critic with online policy evaluation. We tried regularizing the policy and critic optimization with weight decay, clipping the gradients in the policy and/or the critic, using a larger critic network but none seems to mitigate this issue. The policy collapse suggests that the offline trained critic overfits, so we tried improving generalization with data augmentation [9], but this is ineffective for preventing policy collapse. We tried a distributional critic or using an ensemble of critic networks, i.e., instead of using two critics as in TD3, we trained ten critics in parallel in the hope of further reducing overestimation bias. We also tried using uncertainty in the critic ensemble to penalize policy improvements, but we have not managed to take advantage of it to prevent policy collapse.

2.3 Conservative Policy Improvement in TD3

We hypothesize that the policy collapse happens as the distribution shift in online learning causes the critic to produce erroneous estimates of the Q values. When the offline dataset is more diverse, the error is mild, similar to the errors encountered in online off-policy learning, thus it is not sufficient to collapse the good initial policy. However, when the dataset consists of high quality trajectories, the critic may have never seen the outcome of bad transitions. In this case, over-fitting to successful transitions in the offline dataset causes larger extrapolation error that overrides the good policy initialization.

While offline RL algorithms constrain the policy to be close to the empirical data distribution, such constraint is inadequate for finetuning since it may be too conservative to allow for fast online learning. On the other hand, constraining policy optimization to not deviate too much from a historically good policy may be beneficial since it may limit influence of an inaccurate critic.

Therefore, we propose to improve the online TD3 algorithm by changing the unconstrained policy improvement step to a constrained update that penalizes large policy updates. Concretely, we propose to use the following constrained policy improvement step in place of the original TD3 policy optimization step

$$\max_{\theta} \quad \mathbb{E}_{s\sim\mathcal{B}}[Q_\phi(s,a)|_{a=\pi_\theta(s)}] \qquad \text{s.t.} \quad \mathbb{E}_{s\sim\mathcal{B}}[\ell_2(\pi_\theta(s) - \pi_{\theta'}(s))] \leq \epsilon, \tag{3}$$

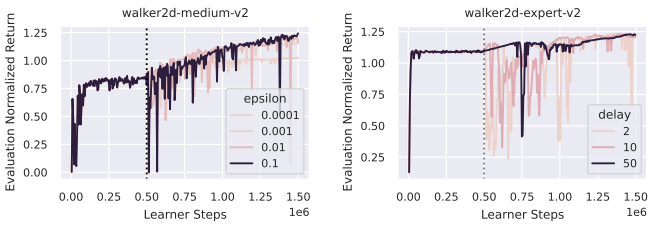
where θ is the online policy network parameter, θ' is the target policy network parameter, ℓ_2 is the L^2 norm and ϵ is a hyper-parameter that controls the degree of the constraint. The constraint regularizes the online policy to not deviate too much from the moving target policy. This formulation resembles the constrained optimization used in MPO [1], except that we are working with a deterministic policy and using the ℓ_2 norm as the constraint. We optimize the objective by formulating the Lagrangian with dual variables λ. The constrained optimization now becomes

$$\max_{\theta}\min_{\lambda\geq 0} \quad \mathbb{E}_{s\sim B}[Q_\phi(s,a) - \lambda[\epsilon - \ell_2(a - \pi_{\theta'}(s))]], \quad a = \pi_\theta(s), \tag{4}$$

where the primal θ and dual variables λ can be jointly optimized by stochastic gradient descent.

Figure 1 shows our proposed constrained policy improvement step is effective in stabilizing training on the medium and medium-replay datasets where online finetuning with TD3 suffers from policy collapse. At the same time, the constrained improvement enjoys comparable sample efficiency and improves significantly faster compared to finetuning online with TD3-BC. In fig. 1 we use $\epsilon = 0.001$. Figure 3a ablates on different choices of ϵ, and larger ϵ results in more unstable learning.

Improving training stability with more delayed policy improvements. The policy improvement constraint penalty stabilizes learning for the medium datasets. However, when only expert demonstrations are used in offline learning, the critic suffers from severe overfitting, and constrained updates alone are insufficient to prevent policy collapse. In this case, we found that updating the critic more frequently stabilizes finetuning. Note that reducing update frequency is different from constraining policy updates: constraining the policy optimization has the added benefit of making policy optimization more robust to critic error. Our concrete implementation takes advantage of the delay parameter in TD3, which determines the frequency of policy optimization. Figure 3b compares different delayed step values with evaluation performance. We see that increasing the delay helps prevent policy collapse during initial finetuning. This ablation suggests that when adopting standard off-policy algorithms for finetuning, reducing the ratio of policy improvement to policy evaluation steps may help.



(a) (b)

Figure 3: (a) Effect of increasing the number of delay steps in TD3-C. Constrained updates alone do not prevent policy collapse, but reducing the amount of policy improvement steps mitigates it. (b) Effect of varying ϵ. Smaller ϵ leads to more stable online finetuning but slower improvement.

3 Conclusion

In this work, we studied the difficulty in leveraging offline RL as pretraining for online RL. We found that pretrained policies and critics from offline RL are sensitive to distribution shifts due to online finetuning. While conservative policy optimization is a promising approach for stabilizing finetuning from offline RL, it may be insufficient for stable online improvement when the offline dataset lacks diversity. We found that changing the ratio between policy evaluation and policy optimization stabilizes online finetuning. Future work will focus on stablizing online critic learning to take full advantage of pretraining with offline RL.

References

[1] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a Posteriori Policy Optimisation. In *ICLR*, 2018.

[2] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning, 2021, arXiv:2004.07219.

[3] S. Fujimoto and S. Gu. A Minimalist Approach to Offline Reinforcement Learning. In *NeurIPS*, 2021.

[4] S. Fujimoto, D. Meger, and D. Precup. Off-Policy Deep Reinforcement Learning without Exploration. In *ICML*, 2019.

[5] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *ICML*, 2018.

[6] I. Kostrikov, A. Nair, and S. Levine. Offline Reinforcement Learning with In-sample Q-Learning. In *ICLR*, 2022.

[7] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-Learning for Offline Reinforcement Learning. In *NeurIPS*, 2020.

[8] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*. Springer, 2012.

[9] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement Learning with Augmented Data. In *NeurIPS*, 2020.

[10] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin. Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble. In *CoRL*, 2021.

[11] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems, 2020, arXiv:2005.01643.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous Control with Deep Reinforcement Learning. In *ICLR*, 2016.

[13] A. Nair, A. Gupta, M. Dalal, and S. Levine. AWAC: Accelerating Online Reinforcement Learning with Offline Datasets, 2021, arXiv:2006.09359.

[14] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards, 2018, arXiv:1707.08817.

[15] Z. Wang, A. Novikov, K. Zolna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, and N. de Freitas. Critic Regularized Regression. In *NeurIPS*, 2020.

Versatile Offline Imitation from Observations and Examples via State Occupancy Matching

Yecheng Jason Ma
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
jasonyma@seas.upenn.edu

Andrew Shen
University of Melbourne, Melbourne, Australia
andrewshen2913@gmail.com

Dinesh Jayaraman
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
dineshj@seas.upenn.edu

Osbert Bastani
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
obastani@seas.upenn.edu

Abstract

Offline reinforcement learning (RL) is a promising framework for sample-efficient, scalable, and practical data-driven decision-making. However, offline RL assumes that the offline dataset comes with reward labels, which may not always be possible. To address this, offline imitation learning (IL) leverages a small amount of expert demonstrations to provide supervision for policy learning from an offline dataset of unknown quality. Expert demonstrations, however, are often much more expensive to acquire than offline data; thus, offline IL benefits significantly from minimizing assumptions about the expert data.

To this end, we propose **S**tate **M**atching **O**ffline **DI**stribution **C**orrection **E**stimation (SMODICE), a novel and versatile algorithm for offline imitation learning (IL) via state-occupancy matching. Without requiring access to expert actions, SMODICE can be effectively applied to three offline IL settings: (i) imitation from observations (IfO), (ii) IfO with dynamics or morphologically mismatched expert, and (iii) example-based reinforcement learning, which we show can be formulated as a state-occupancy matching problem. We show that the SMODICE objective admits a simple optimization procedure through an application of Fenchel duality, reducing a nested optimization problem to a sequence of stable supervised learning problems. We extensively evaluate SMODICE on both gridworld environments as well as on high-dimensional offline benchmarks. Our results demonstrate that SMODICE is effective for all three problem settings and significantly outperforms prior state-of-art.

Keywords: Offline Imitation/Reinforcement Learning, DICE, Example-based RL, Convex Optimization

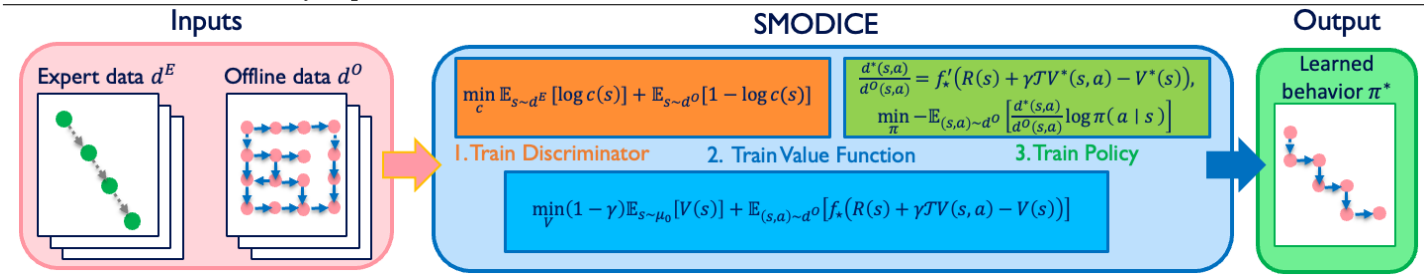


Figure 1: Diagram of SMODICE. First, a state-based discriminator is trained using the offline dataset d^O and expert observations (resp. examples) d^E. Then, the discriminator is used to train the Lagrangian value function. Finally, the value function provides the importance weights for policy training, which outputs the learned policy d^*.

1 Introduction

The offline reinforcement learning (RL) framework [11, 13] aims to use pre-collected, reusable offline data—without further interaction with the environment—for sample-efficient, scalable, and practical data-driven decision-making. However, this assumes that the offline dataset comes with reward labels, which may not always be possible. To address this, offline *imitation* learning (IL) [20, 1, 7] has recently been proposed as an alternative where the learning algorithm is provided with a small set of expert demonstrations and a separate set of offline data of unknown quality. The goal is to learn a policy that mimics the provided expert data while avoiding test-time distribution shift [18] by using the offline dataset.

Expert demonstrations are often much more expensive to acquire than offline data; thus, offline IL benefits significantly from minimizing assumptions about the expert data. In this work, we aim to remove two assumptions about the expert data in current offline IL algorithms: (i) expert action labels must be provided for the demonstrations, and (ii) the expert demonstrations are performed with identical dynamics (same embodiment, actions, and transitions) as the imitator agent. These requirements preclude applications to important practical problem settings, including (i) imitation from observations, (ii) imitation with mismatched expert that obeys different dynamics or embodiment (e.g., learning from human videos), and (iii) learning only from examples of successful outcomes rather than full expert trajectories [2]. For these reasons, many algorithms for *online* IL have already sought to remove these assumptions [19, 14, 17, 2], but extending them to offline IL remains an open problem.

We propose **S**tate **M**atching **O**ffline **DI**stribution **C**orrection **E**stimation (SMODICE), a general offline IL framework that can be applied to all three problem settings described above. At a high level, SMODICE is based on a state-occupancy matching view of IL; in particular, it optimizes a tractable offline upper bound of the KL-divergence of the state-occupancy d between the imitator π and the expert E:

$$\min_\pi \mathrm{D}_{\mathrm{KL}}(d^\pi(s) \| d^E(s)). \tag{1}$$

This state-occupancy matching objective allows SMODICE to infer the correct actions from the offline data in order to match the state-occupancy of the provided expert demonstrations. This naturally enables imitation when expert actions are unavailable, and even when the expert's embodiment or dynamics are different, as long as there is a shared task-relevant state. Finally, we show that example-based RL [2], where only examples of successful states are provided as supervision, can be formulated as a state-occupancy matching problem between the imitator and a "teleporting" expert that is able to reach success states in one step. Hence, SMODICE can also be used as an offline example-based RL[1] method without any modification.

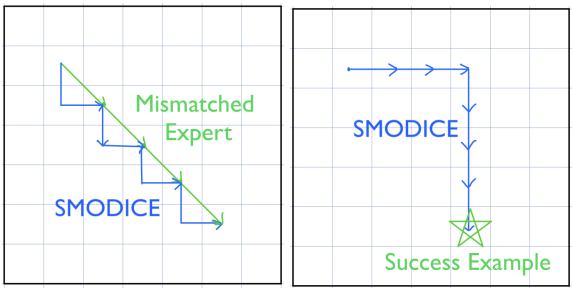
Naively optimizing the offline upper bound on (1) would result in an actor-critic style IL algorithm akin to prior work [6, 8, 9]; however, these algorithms suffer from training instability in the offline regime [10, 12, 7] due to the entangled nature of actor and critic learning, leading to erroneous value bootstrapping [13]. SMODICE bypasses this issue and achieves "actor-free" training by directly estimating the importance weight ratio of the occupancy measures between the optimal policy and the empirical behavior policy of the offline data, leveraging the stationary distribution correction estimation [15, 16] (DICE) paradigm. Specifically, by formulating the policy optimization problem via its dual (i.e., optimizing over the space of valid state-action occupancy distributions) and applying Fenchel duality, SMODICE obtains an unconstrained convex optimization problem over a value function arising from Lagrangian duality, which admits closed-form solutions in the tabular case and can be easily optimized using stochastic gradient descent (SGD) in the deep RL setting. Then, SMODICE projects the optimal value function onto the dual space to extract the optimal importance weights, and learns the optimal policy via weighted Behavior Cloning. Note that SMODICE does not learn a policy until the value function has converged. As such, SMODICE performs a sequence of *disjoint* supervised learning problems to

[1]We refer to this problem as "offline imitation learning from examples" to unify nomenclature with the other two problems.

achieve stable policy learning. The method overview is illustrated in Figure 1, and we omit all other method details and derivations in this extended abstract; the full paper is publicly available: https://arxiv.org/abs/2202.02433

Through extensive experiments, we show that SMODICE is effective for all three problem settings we consider and outperforms all state-of-art methods in each respective setting. Altogether, our proposed method SMODICE can serve as a versatile offline IL algorithm that is suitable for a wide range of assumptions on expert data.

Pedagogical examples. To illustrate SMODICE's versatility, we have applied it to two gridworld tasks, testing offline IL from mismatched experts and examples, respectively. Figure 2(a) shows an expert agent that can move diagonally in any direction, whereas the imitator can only move horizontally or vertically. In Figure 2(b), only a success state (the star) is provided as supervision. An offline dataset collected by a random agent is given to SMODICE for training in both cases. As shown, SMODICE recovers an optimal policy (i.e. minimum state-occupancy divergence to that of the expert) in both cases.



(a) Mismatched experts (b) Learning via examples

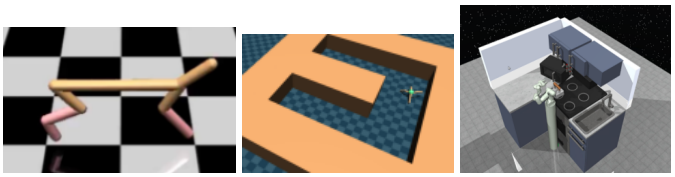
Figure 2: **Illustrations of tabular SMODICE for offline imitation learning from mismatched experts and examples.**

2 Experiments

We experimentally demonstrate that SMODICE[2] is effective for offline IL from observations, mismatched experts, and examples. We briefly describe the tasks and datasets for each setting, and provide all remaining experimental details in the full paper. SMODICE videos are on the project website:https://sites.google.com/view/smodice/home

Tasks and Datasets. We utilize the D4RL [3] offline RL dataset. We consider the following standard Mujoco environments: **Hopper, Walker2d, HalfCheetah**, and **Ant**. For each, we take a single expert trajectory from the respective "expert-v2" dataset as the expert dataset and omit the actions. For the offline dataset, following [7], we use a mixture of small number of expert trajectories ($\leq$ 200 trajectories) and a large number of low-quality trajectories from the "random-v2" dataset (we use the full random dataset, consisting of around 1 million transitions). This dataset composition is particularly challenging as the learning algorithm must be able to successfully distinguish expert from low-quality data in the offline dataset.

We also include two more challenging environments from D4RL: **AntMaze** and **Franka Kitchen**. In AntMaze (Figure 3(b)), an Ant agent is tasked with navigating an U-shaped maze from one end to the other end (i.e., the goal region). The offline dataset (i.e., "antmaze-umaze-v2") consists of trajectories ($\approx$ 300k transitions) of an Ant agent navigating to the goal region from initial states; The trajectories are not always successful; often, the Ant flips over to its legs before it reaches the goal. We visualize this dataset on the project website. As above, we additionally include 1 million random-action transitions to increase the task difficulty. We take one trajectory from the offline dataset that successfully reaches the goal to be the expert trajectory. Franka Kitchen (Figure 3(c)), introduced by [5], involves controlling a 9-DoF Franka robot to manipulate common household kitchen objects (e.g., microwave, kettle, cabinet) sequentially to achieve a pre-specified configuration of objects. The dataset (i.e., "kitchen-mixed-v0") consists of *undirected* human teleoperated demonstrations, meaning that each trajectory only solves a subset of the tasks. Together, these six tasks (illustrated in Figure 2) require scalability to high-dimensional state-action spaces and robustness to different dataset compositions.



(a) Mujoco (b) AntMaze (c) Franka Kitchen

Figure 3: **Ilustrations of the evaluation environments.**

Finally, in the offline IL from examples setting, we include the **PointMass-4Direction** environment. Here, a 2D PointMass agent is tasked with navigating to the middle point of a specified edge of the square that encloses the agent. At training and evaluation time, we set the left edge to be the desired edge and collect success states from the offline data accordingly. This task is low-dimensional but consists of multi-task offline data, making it challenging for algorithms such as BC that do not solve the example-based RL objective.

A subset of the first six tasks are re-used for the three settings, and the only difference across the settings is the type of provided expert data. In the mismatched expert setting, we utilize HalfCheetah, Ant, and AntMaze. For the first two, the original expert data is replaced by demonstrations from a crippled version of the agent. For AntMaze, we use a PointAass agent to provide the demonstration. In offline IL from examples setting, we utilize AntMaze and Kitchen tasks

[2]Code is available at: https://github.com/JasonMa2016/SMODICE

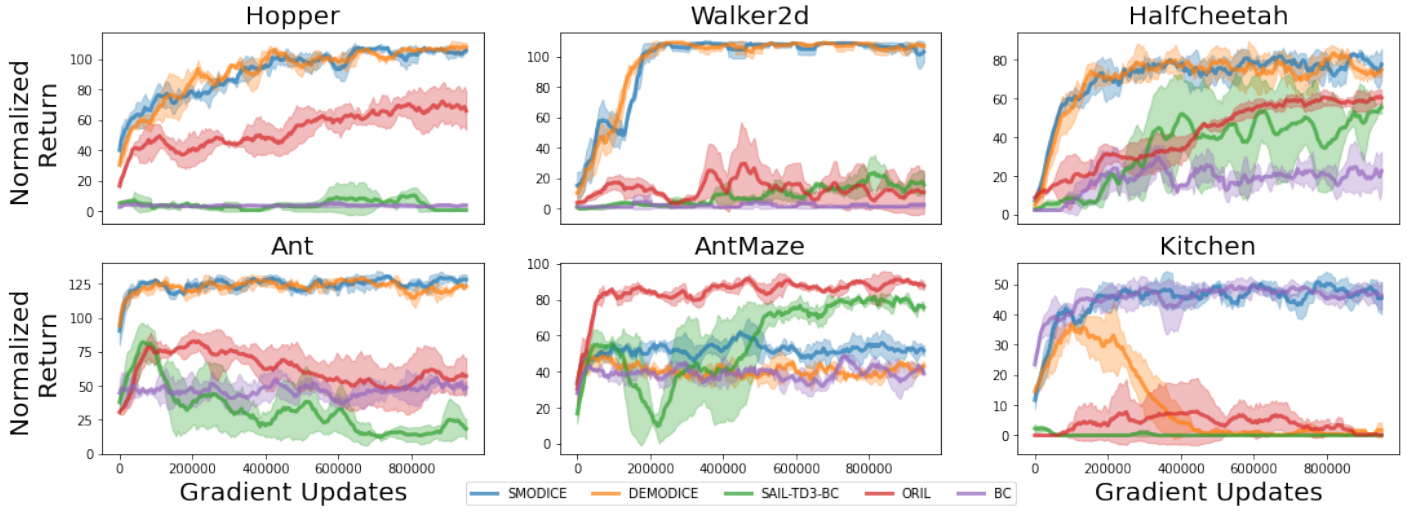


Figure 4: **Offline imitation learning from observations results.**

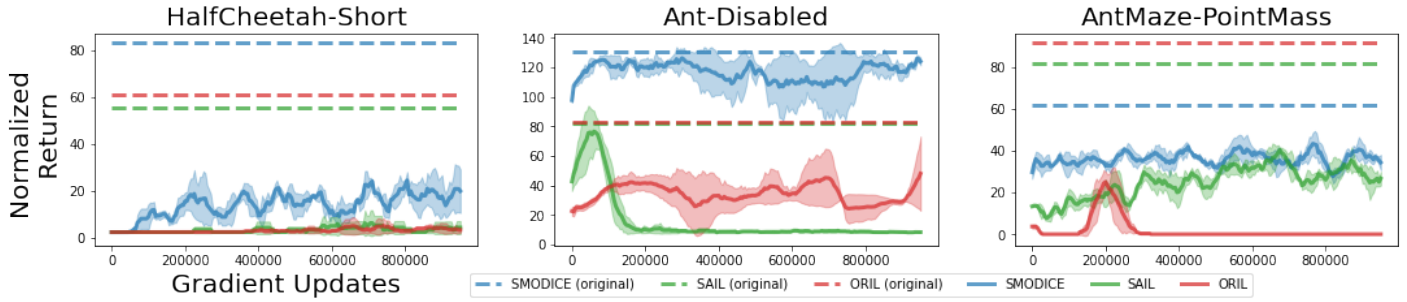


Figure 5: **Offline imitation learning from mismatched experts results.**

but replace expert demonstrations with just examples of success states; Kettle and Microwave are two sub-tasks needed to be completed in the original full task.

Baselines. We compare SMODICE to state-of-art methods in respective settings. In IfO, we consider **DEMODICE** [7], **Behavior Cloning (BC)**, **SAIL** [14], and **ORIL** [20]; DEMODICE uses expert-actions, while SAIL and ORIL[3] are state-based IL methods. For the mismatched expert setting, we consider SAIL and ORIL. For the offline IL from examples setting, we consider **RCE** [2], the state-of-art method in this setting, and ORIL. SAIL and RCE are online actor-critic methods, we replace their original RL method with TD3-BC [4], a state-of-art offline RL method, to make them compatible in the offline setting; we also implement ORIL using TD3-BC to make the comparison fair.

Results. The training curves are shown in Figure 4, 5, 6. As shown, SMODICE is the best performing method in all three settings. In Figure 4, we see that SMODICE matches the state-of-art method DEMODICE without access to expert actions and significantly outperforms it in the challenging kitchen task. Other baselines struggle to perform well in all six tasks due to different dataset compositions and high-dimensionality of the state space, among many other factors.

While ORIL and SAIL outperform SMODICE on AntMaze, when the expert supervision is provided by a morphologically different agent (i.e., PointMass), their performances greatly decline. In contrast, SMODICE exhibits robustness to mistmached expert in AntMaze as well as the other two tasks considered in Figure 5.

Finally, when we further reduce the expert supervision to mere examples of successful outcome (Figure 6), SMODICE is more stable and outperforms the current state-of-art RCE. Furthermore, in the most challenging Microwave task, SMODICE is the only method that makes non-trivial progress.

3 Conclusion

We have proposed SMODICE, a simple, stable, and versatile algorithm for offline imitation learning from observations, mismatched experts, and examples. Leveraging Fenchel duality, SMODICE optimizes a state-occupancy matching objective that enjoys closed-form tabular solution and stable optimization with deep neural networks. Through extensive experiments, we have shown that SMODICE significantly outperforms prior state-of-art methods in all three settings.

[3] we adapt ORIL to state-based IL method by using a state-based discriminator.

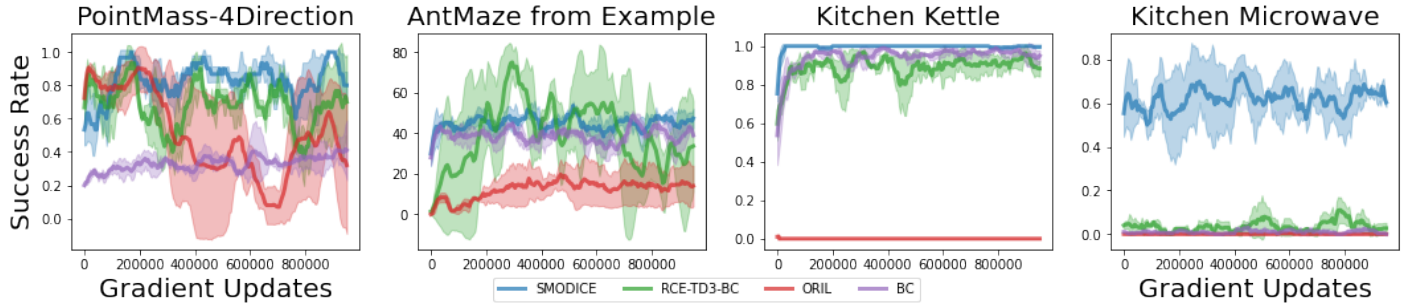


Figure 6: **Offline imitation learning from examples results.**

References

[1] Jonathan D. Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data without great coverage, 2021.

[2] Benjamin Eysenbach, Sergey Levine, and Ruslan Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. In Thirty-Fifth Conference on Neural Information Processing Systems, 2021.

[3] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021.

[4] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. arXiv preprint arXiv:2106.06860, 2021.

[5] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. arXiv preprint arXiv:1910.11956, 2019.

[6] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016.

[7] Geon-Hyeong Kim, Seokin Seo, Jongmin Lee, Wonseok Jeon, HyeongJoo Hwang, Hongseok Yang, and Kee-Eung Kim. DemoDICE: Offline imitation learning with supplementary imperfect demonstrations. In International Conference on Learning Representations, 2022.

[8] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. arXiv preprint arXiv:1809.02925, 2018.

[9] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. In International Conference on Learning Representations, 2020.

[10] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. arXiv preprint arXiv:1906.00949, 2019.

[11] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Reinforcement learning, pages 45–73. Springer, 2012.

[12] Jongmin Lee, Wonseok Jeon, Byung-Jun Lee, Joelle Pineau, and Kee-Eung Kim. Optidice: Offline policy optimization via stationary distribution correction estimation. arXiv preprint arXiv:2106.10783, 2021.

[13] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643, 2020.

[14] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. arXiv preprint arXiv:1911.10947, 2019.

[15] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. arXiv preprint arXiv:1906.04733, 2019.

[16] Ofir Nachum and Bo Dai. Reinforcement learning via fenchel-rockafellar duality, 2020.

[17] Ilija Radosavovic, Xiaolong Wang, Lerrel Pinto, and Jitendra Malik. State-only imitation learning for dexterous manipulation, 2020.

[18] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[19] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation, 2018.

[20] Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline learning from demonstrations and unlabeled experience. arXiv preprint arXiv:2011.13885, 2020.

Modeling Human Reinforcement Learning with Disentangled Visual Representations

Tyler Malloy
Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy, NY 12180
mallot@rpi.edu

Tim Klinger
IBM Research AI
T.J. Watson Research Center
Yorktown Heights, NY
tklinger@us.ibm.com

Chris R. Sims
Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy, NY 12180
simsc3@rpi.edu

Abstract

Humans are able to learn about the visual world with a remarkable degree of generality and robustness, in part due to attention mechanisms which focus limited resources onto relevant features. Deep learning models that seek to replicate this feature of human learning can do so by optimizing a so-called "disentanglement objective", which encourages representations that factorize stimuli into separable feature dimensions [4]. This objective is achieved by methods such as the β-Variational Autoencoder (β-VAE), which has demonstrated a strong correspondence to neural activity in biological visual representation formation [5]. However, in the β-VAE method, learned visual representations are not influenced by the utility of information, but are solely learned in an unsupervised fashion. In contrast to this, humans exhibit generalization of learning through *acquired equivalence* of visual stimuli associated with similar outcomes [7]. The question of how humans combine utility-based and unsupervised learning in the formation of visual representations is therefore unanswered. The current paper seeks to address this question by developing a modified β-VAE model which integrates both unsupervised learning and reinforcement learning. This model is trained to produce both psychological representations of visual information as well as predictions of utility based on these representations. The result is a model that predicts the impact of changing utility on visual representations. Our model demonstrates a high degree of predictive accuracy of human visual learning in a contextual multi-armed bandit learning task [8]. Importantly, our model takes as input the same complex visual information presented to participants, instead of relying on hand-crafted features. These results provide further support for disentanglement as a plausible learning objective for visual representation formation by demonstrating their usefulness in learning tasks that rely on attention mechanisms.

Keywords: Representation Modelling, Visual Learning, Attention

Acknowledgements

This research was supported by NSF grant DRL-1560829 and a grant from the RPI-IBM Artificial Intelligence Research Collaboration (AIRC). Empirical data was provided by Yael Niv from the Niv Lab Website.

1 Introduction

In the context of visual learning, a representation refers to the internal psychological state of an agent while perceiving the external information relevant to a task. In complex learning tasks, agents must form efficient representations of information due to limited cognitive resources by ignoring or abstracting away irrelevancies. An important phenomenon in studies of human representation learning is *acquired equivalence*, which describes increased generalization of dissimilar stimuli based on associations of similar outcomes [7]. The focus of this work is in modelling the visual representations formed by humans in utility-based learning tasks.

One method for modelling visual representation formation in humans is the Variational Auto-Encoder (VAE), a deep neural-network model that forms informationally limited representations of visual information. This is done by taking an input stimuli and reducing the dimensionality of the neural network layer by layer to form a constrained representation, which is then used to reconstruct the original stimuli. In VAEs the learned latent representations take the form of a multi-variate Gaussian distribution. This distribution can be sampled from to form lossy reconstructions of the original stimuli. VAE models are trained to produce reconstructions that are as close to the original stimuli as possible, and thus effectively learn useful low-dimensional representations.

Reinforcement Learning (RL) techniques for predicting human behaviour in utility-based learning have shown impressive accuracy, even in complex visual tasks that require attention mechanisms [8]. However, these methods typically rely on the use of tabular RL and hand-crafted features that represent complex visual information using low-dimension one-hot encodings. Deep Reinforcement Learning (DRL) has demonstrated human-level performance on a range of visual based tasks, such as video games [9]. These neural network-based techniques reach super human level performance on a variety of tasks by leveraging a large amount of experience, often the equivalent of thousands of hours on a single task [9]. The model [1] presented in this work connects deep learning techniques in representation learning with predictions of utility that can be trained using RL to model human behaviour and visual representations in learning tasks.

2 Background

2.1 Variational Autoencoders

VAE models seek to learn representations of visual stimuli that preserve statistical structure. The closely related β-VAE method adds to the VAE a controllable information bottleneck regulated by an additional β parameter, to encourage generalization and robustness. As β increases the information capacity of the latent representation decreases, while when set to 0, information is unconstrained. β-VAE models have previously been used to model human choice in a visual categorization task [2] and visual decision making [6]. Results from this experiment provide evidence that systematic biases of human categorization can be explained by the information-constrained representations formed by β-VAE models.

Relatively less attention has been focused on the usefulness of these latent representations for utility learning tasks. The Uβ-VAE model presented in this work seeks to account for the interaction between utility-based learning and visual representation learning. The result is a deep learning based approach that is able to reflect the high generalization and sample efficiency of human learning, while also producing predictions of psychological representations that have applications beyond predicting behaviour, detailed in section 4.

2.2 Feature Reinforcement Learning

Feature Reinforcement Learning has been suggested as a plausible account for the high sample efficiency of human learning in visual learning tasks [8]. This method predicts the value of a stimulus as the sum of its constituent features; for example, a red square is valued as the sum of the features "red" and "square" individually. This encourages compositionality, reducing the amount of experience required to make predictions of utility with novel stimuli, improving generalization and sample efficiency, and better modelling human behaviour.

The FRL model was trained on a task in which participants were presented with three objects, each composed of three unique features: *shape* (square, triangle, circle), *color* (red, yellow, green), and *pattern* (dotted, wavy, hatched), and asked to predict the highest-utility stimulus, based on an observed reward. Critically, during blocks of trials (∼20-25 stimuli), a single feature among the 9 possible features was indicative that a stimuli had a higher utility than the other two options. This single feature indicative of utility is the 'feature of interest', and the goal of a block of trials is to determine this feature and repeatedly select the option containing it. This allowed human participants to quickly generalize to stimuli pairs they had not seen before by leveraging the regularity of this utility function. Figure 1.A shows an example of the stimulus of red-wavy-square. A more complete description of the methodology is given in [8].

[1] Model code available at: github.com/TylerJamesMalloy/DisentangledVisualLearning

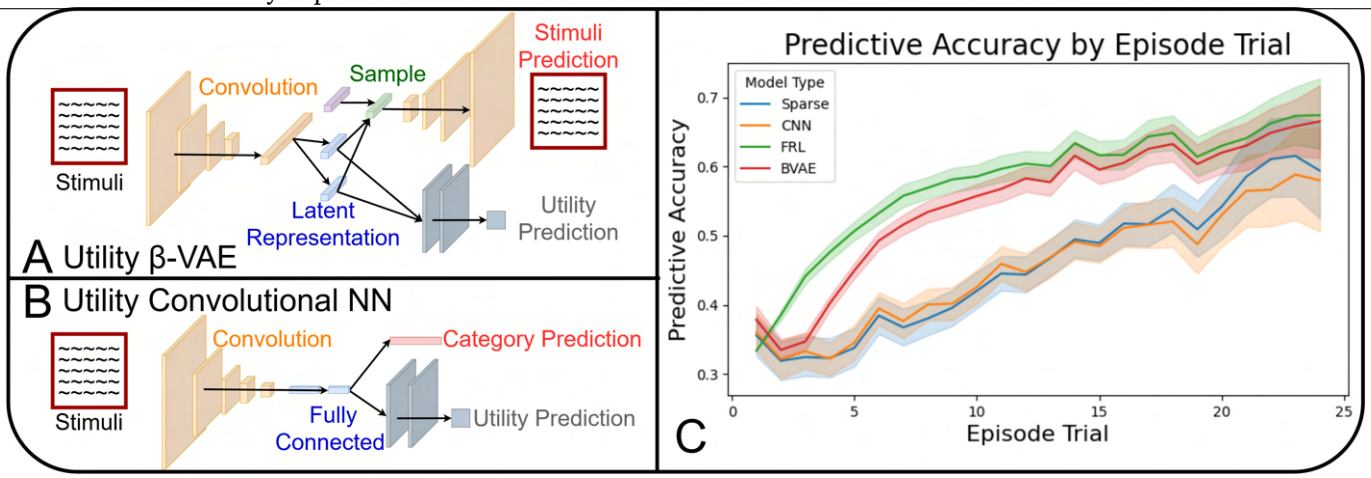


Figure 1: **A** Schematic of Utility β-VAE model with example input fed into convolutional layers to develop latent representations used both as input to utility prediction as well as sampled from to reconstruct original stimuli. **B** Schematic of CNN model used for comparison, structured similarly as the β-VAE based model, using the final fully connected layer to predict utilities. **C** Predictive accuracy of FRL, Utility β-VAE and 2 CNN models (sparse and standard).

3 Model

3.1 Utility β-VAE Model

To allow for utility-predictions based on visual stimuli, we developed a Utility β-VAE model (Uβ-VAE) that outputs both a stimulus reconstruction and utility prediction (see 1.A). This architecture is similar to previous methods that have been used in visual categorization tasks [2] and decision making [6] as well as deep RL methods that train based on abstract constrained representations of the state such as MuZero [9]. The Uβ-VAE model is trained using a loss function which balances the standard reconstruction error $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ and model complexity $\beta D_{KL}\big(q_\phi(z|x)||p(z)\big)$ with the addition of utility error $\upsilon(\mathcal{U}(z) - u(x))$ through the following loss function:

$$\mathcal{L}(\theta, \mathcal{U}, \phi; x, z, \beta) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}\big(q_\phi(z|x)||p(z)\big) + \upsilon\big(\mathcal{U}(z) - u(x)\big)^2$$

In the model shown in 1.A, $\mathcal{U}$ represents the utility prediction module which outputs the prediction $\mathcal{U}(z)$ based on the latent representation z, and $u(x)$ represents the predicted utility from the supplied RL algorithm. This utility prediction error is weighted by the value υ, which allows for flexibility in the relevance of utility. The remaining terms are identical to those defined in the traditional β-VAE model. ϕ represents the parameters of the encoder $q_\phi(z|x)$, which defines the probability distribution over latent representations z given the stimulus x and θ represents the parameters of the decoder $p_\theta(x|z)$ of the stimulus x given the representation z. For a more complete description, see [4].

As mentioned, updating the predictions of utility relies on an error term based on the difference between the model predictions and a separate RL algorithm. In the results presented here these training utilities are supplied by the FRL algorithm. Training the Uβ-VAE model on utility predictions from the FRL algorithm allows for a comparison of our model against alternative deep learning approaches which are described in the following section.

3.2 Model Comparison

To provide a comparison with alternative DRL techniques, our model is compared against a similarly structured convolutional neural network (CNN) that is trained to predict the category of the stimulus (shape, color, and texture). This network (Figure 1.B) makes utility predictions based on the final layer of the network before the output categorization prediction. The network is trained in the same utility-weighted manner as the Utility β-VAE model. We also compare our model against a 'Sparse' CNN, which is trained using the same dimensionality for the utility prediction input as our model (6 values). Additionally, to emulate the properties of the β-VAE, we use Dropout, randomly zeroing some percentage (60%) of the NN nodes during training, which has been related to the information bottleneck principle [1].

In Figure 1.B models are compared in their accuracy of predicting human behaviour according to *predictive accuracy*, the probability that the model assigns to the action that was selected by the human participant. Note that the upper limit for all deep learning methods (BVAE, CNN, Sparse) is the accuracy of the FRL model that produces utility values used in training all DNN models. The Uβ-VAE model closely approaches the predictive accuracy of the baseline FRL, indicating that it has learned to predict the utility, but has significantly higher accuracy than the other deep learning methods.

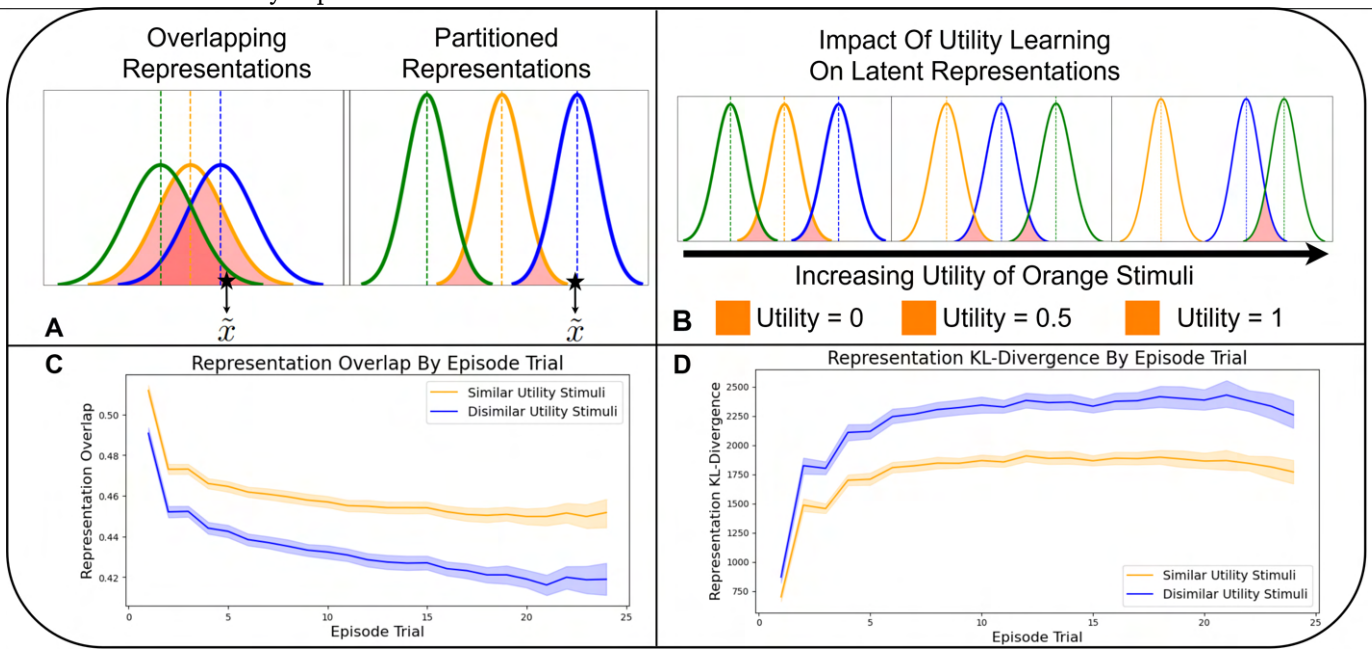


Figure 2: **A:** Demonstration of the relative likelihood that a data-point will be mislabeled (red area). Decreasing the likelihood of mislabelling results in a higher KL-divergence. The data-point $\tilde{x}$ drawn from the left overlapping blue distribution has a high likelihood of being mislabelled, this likelihood is decreased with the right partitioned distributions. **B:** Demonstration of the impact of learning utility values on latent representations. With all utilities as 0 (left) there is no preference for representation overlap. As the utility of the orange stimuli increases (middle) the model learns to prefer an overlap that reflects the acquired equivalence based on similar utility. Finally, as orange utility increases significantly (right), the model learns representations that only allow an overlap of stimuli with the same utility. **C:** Overlap of β-VAE latent representation distributions by episode trial, grouped by similar vs. dissimilar utility stimuli. **D:** Symmetric KL-divergence of β-VAE latent distribution overlap by episode trial, grouped by similar vs. dissimilar utility stimuli.

4 Modelling Results

The first method of analyzing the latent representations learned by Uβ-VAE uses information theoretic measures to explain the high generalization and robustness of the model. The β parameter in the β-VAE model limits informational complexity based on KL-Divergence. Overlapping latent representation distributions, such as those shown on the left side of Figure 2.A (based on a figure in [3]), have lower KL-Divergence scores and require less information to represent. However, a data point $\tilde{x}$ drawn from a distribution that overlaps with others is more likely to be incorrectly labelled as being drawn from a different distribution. This issue can be solved by partitioning representations, such as those shown on the right side of Figure 2.A, which increases the informational complexity of the learned representations. The loss function of the β-VAE model balances reconstruction accuracy and the complexity of learned representations. This has the effect of ensuring that neighboring stimuli in data space produce similar distributions in latent space [3].

In Uβ-VAE, representations are formed to balance reconstruction accuracy, informational complexity and utility-prediction error. This results in latent representations that depend on stimulus utility as well as visual structure. Previous research in acquired equivalence has demonstrated that stimuli representations can be updated based on categories within visual learning tasks [7]. The utility-based training of our model is motivated in part by a similar acquired equivalence, instead based on the utility that is associated with a visual stimuli in a learning task.

If this utility-based acquired equivalence exists in Uβ-VAE when predicting the behaviour of human subjects, it should be observable through a change in the similarity of latent representations. This expected change is demonstrated in Figure 2.B, which shows the hypothetical alteration of a learned latent representation as the utility associated with each stimuli is learned. As the utility associated with the orange stimuli increases, the cost associated with the orange stimuli being mistaken for either blue or green increases, and the overlap of these latent representation distributions changes. We test whether this effect takes place in Uβ-VAE by comparing the representation distribution overlap (Figure 2.C) and Kullback–Leibler divergence (Figure 2.D) between stimuli with similar utilities vs. stimuli with different utilities. The gap in overlap and distribution similarities based on utility provides evidence that the representations learned by our model have an acquired equivalence based on stimuli utility.

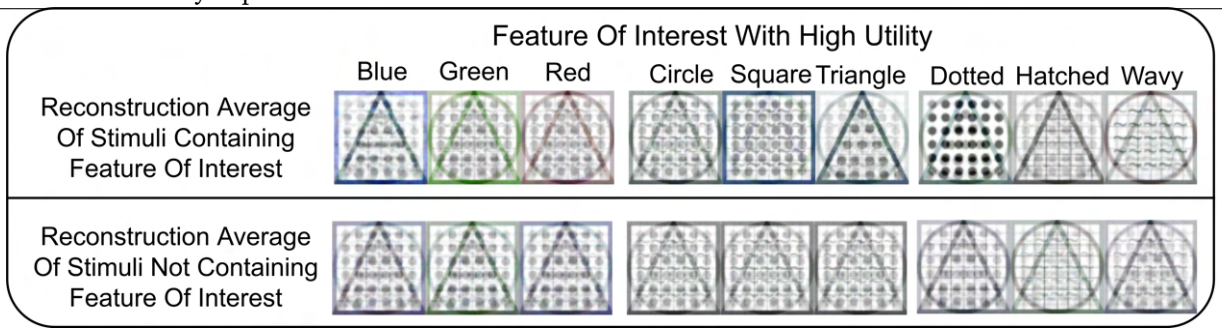


Figure 3: Average reconstructions from perturbed latent samples after training on each of the 9 features of interest, split into stimuli containing (top) and not containing (bottom) the given feature of interest. Note that the color features used (blue, green, red) differ from those used in [8] to maximize the reconstruction error between differently colored stimuli.

The final comparison of latent representations is done by visualizing the reconstructed images by first training our β-VAE model to predict a high utility for 1 of the 9 features of interest. After this, the learned latent representation for all stimuli is sampled from, with a small noise added. These noisy samples are used to reconstruct the stimuli, averaged over 1000 samples in the images shown in Figure 3. We compare the average reconstruction of stimuli that contain (top) and do not contain (bottom) the feature of interest. This visualization demonstrates that the learned representations of stimuli containing the feature of interest are more robust to noisy sampling. Specifically, when a particular color, shape, or texture is important for predicting utility, that feature tends to be better preserved in the stimulus reconstructions. This increased robustness is due to a partitioning of the representation space (as described in Figure 2.A) along the feature of interest, furthering the evidence of an acquired equivalence along the feature related to utility.

5 Conclusions

The Uβ-VAE model presented in this work quickly learns the stimuli utilities used by the FRL method to predict human performance in a complex visual task. This provides further support for the disentanglement objective as a plausible account of human visual representation formation, extending its application into visual utility learning. Compared with alternative deep learning methods, our approach better fits human behaviour while also producing latent representations that have been connected to biological visual representations in previous work.

Beyond predictive accuracy, the latent representations formed by our model can provide interesting insight into biological visual representations used in utility-based learning tasks. Analysis of both information theoretic and visualizations of these latent representations demonstrated evidence of a acquired equivalence of visually different stimuli based on their similar utility. This feature of Utility β-VAE representations opens up interesting possibilities for future research combining visual utility learning with additional tasks like change detection, categorization, and others.

References

[1] Alessandro Achille and Stefano Soatto. "Information dropout: Learning optimal representations through noisy computation". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2018), pp. 2897–2905.

[2] Christopher Bates and Robert Jacobs. "Efficient Data Compression Leads to Categorical Bias in Perception and Perceptual Memory." In: *CogSci*. 2019, pp. 1369–1375.

[3] Christopher P. Burgess et al. "Understanding disentangling in β-VAE". In: *CoRR* abs/1804.03599 (2018).

[4] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. 2017.

[5] Irina Higgins et al. "Unsupervised deep learning identifies semantic disentanglement in single inferotemporal face patch neurons". In: *Nature communications* 12.1 (2021), pp. 1–14.

[6] Tyler Malloy and Chris R. Sims. "Modelling visual decision making using a variational autoencoder". In: *19th International Conference on Cognitive Modelling, ICCM 2021, Virtual Conference, July 1-12, 2021*. 2021.

[7] M Meeter, D Shohamy, and CE Myers. "Acquired equivalence changes stimulus representations". In: *Journal of the experimental analysis of behavior* 91.1 (2009), pp. 127–141.

[8] Yael Niv et al. "Reinforcement learning in multidimensional environments relies on attention mechanisms". In: *Journal of Neuroscience* 35.21 (2015), pp. 8145–8157.

[9] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* 588.7839 (2020), pp. 604–609.

A Novel Inverse Reinforcement Learning Formulation for Sample-Aware Forward Learning

Giorgio Manganini
Department of Computer Science
Gran Sasso Science Institute
L'Aquila, Italy
giorgio.manganini@gssi.it

Angelo Damiani
Department of Computer Science
Gran Sasso Science Institute
L'Aquila, Italy
angelo.damiani@gssi.it

Alberto Maria Metelli
Artificial Intelligence and Robotic Laboratory
Politecnico di Milano
Milano, Italy
albertomaria.metelli@polimi.it

Marcello Restelli
Artificial Intelligence and Robotic Laboratory
Politecnico di Milano
Milano, Italy
marcello.restelli@polimi.it

Abstract

We propose a novel formulation for the Inverse Reinforcement Learning (IRL) problem, which jointly accounts for the compatibility with the expert behavior of the identified reward and its effectiveness for the subsequent forward learning phase. Albeit quite natural, especially when the final goal is apprenticeship learning (learning policies from an expert), this aspect has been completely overlooked by IRL approaches so far. We propose a new model-free IRL method that is remarkably able to autonomously find a trade-off between the error induced on the learned policy when potentially choosing a sub-optimal reward, and the estimation error caused by using finite samples in the forward learning phase, which can be controlled by explicitly optimizing also the discount factor of the related learning problem. The approach is based on a min-max formulation for the robust selection of the reward parameters and the discount factor so that the distance between the expert's policy and the learned policy is minimized in the successive forward learning task when a finite and possibly small number of samples is available. Differently from the majority of other IRL techniques, our approach does not involve any planning or forward Reinforcement Learning problems to be solved. After presenting the formulation, we provide a numerical scheme for the optimization, and we show its effectiveness on an illustrative numerical case.

Keywords: Inverse Reinforcement Learning, Off-line Reinforcement Learning, Sample Complexity, Min-Max Optimization

Acknowledgements

This work was partially supported by the PON - Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, AIM1880573 Smart, Secure and Inclusive Communities.

1 Introduction

Inverse Reinforcement Learning [IRL, 11] is the process of recovering, from (demonstrations of) an expert's policy, a reward function, which in many cases is the most parsimonious way to describe the behaviour of the expert, especially in complex problems. The learned reward is intended to be successively used in forward Reinforcement Learning (RL) to find new policies that could generalize over unseen states or even improve the expert's actions in new environments, accounting for transferability of the expert's intention, which is compactly described by its reward function.

In their quest for the expert's reward function, many IRL approaches implement an iterative process [1, 15, 4, 5], that alternates between solving a forward RL problem and updating a reward function estimate. In particular, consistency in terms of performance equivalence between the demonstrated trajectories and the ones induces by the learner's policy are enforced. Then, the learning of a generalized policy is performed using a forward RL procedure based on the current estimate of the reward. Another main common aspect in most IRL approaches is the assumption about the underlying MDP. Traditionally, the vast majority of IRL methods rely on the knowledge of the model (either given or accurately learnt from the demonstrated trajectories) [11, 15, 10], which sometimes is also used to perform the internal forward RL subroutines for finding/evaluating intermediate optimal policies. More recently model-free approaches have been proposed [5, 9], even though some of them still require continuous interactions with the environment [1, 4, 5].

In this work, we take a different point of view on IRL and focus on finding a reward function not only compatible with the expert's demonstrations, but that can make the next forward learning phase as efficient as possible, in terms of the sample complexity required to learn a near-optimal policy. We explicitly take into account *how* the recovered reward function will be employed, i.e., plugged into (a possibly different) environment and used to perform forward RL. In this spirit, among the compatible ones, we prefer the rewards that make the next forward learning phase as *efficient* as possible. This goal is *indirectly* pursued by many IRL algorithms, but, to the best of our knowledge, no algorithm performs the reward selection phase by *explicitly* quantifying the sample complexity of forward RL. The novel formulation we propose blends these ambitious goals together and results in an algorithmic procedure which *i)* is purely model–free, *ii)* does not need any interaction with the environment to collect new on-policy data for policy evaluation, and *iii)* does not require solving any forward problem (*i.e.*, finding an optimal policy given a candidate reward function).

2 Background

We define a Markov Decision Process as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are continuous state and action spaces, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition model $P(s'|s, a)$, $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ specifies the action density for each state $\pi(a|s)$. We denote with $Q^{\pi}_{r,\gamma} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the state-action value function of the policy π. Any policy $\pi \in \Pi$ satisfying $\int_{\mathcal{A}} \pi(a|s) Q(s, a) \mathrm{d}a = \max_{a \in \mathcal{A}} Q(s, a)$ for all states $s \in \mathcal{S}$ is named greedy w.r.t. the function Q. Given Q, we denote with $\mathcal{G}[Q] \subseteq \Pi$ the set of greedy policies w.r.t. Q.

The IRL problem aims at finding a reward function r that can explain the behaviour of an expert that follows a policy π_E, which is optimal w.r.t. some unknown reward r_E [11]. Formally, a reward r is *compatible* with the expert's policy π_E if $\pi_E \in \Pi$ is optimal under r, *i.e.*, $\pi_E \in \mathcal{G}[Q^{\star}_{r,\gamma}]$, where $Q^{\star}_{r,\gamma}$ is the optimal state-action value function under the pair (r, γ).

3 The IRL formulation for Efficient Forward Learning

The issue of efficient learning is related with the sample complexity of finding a good approximation of the optimal policy. In RL, the number of calls to the sampling model are generally a function of the problem parameters and, in particular, of the discount factor γ (linked to the effective number of decision epochs). The smaller is the discount factor, the smaller is the number of samples required to attain a near-optimal estimate of the optimal value-function, as shown in many sample complexity bounds depending on a power of $1/(1-\gamma)$ [*e.g.* 2]. Moreover, the recovered reward in IRL has to be compatible with expert's demonstration, which are known only within some accuracy ϵ and confidence δ, being estimated from a finite set of demonstrations.

The following novel IRL formulation blends all these elements together, and takes into direct consideration the effect of the learned IRL reward on the subsequent forward learning phase. Suppose we are given a forward RL algorithm $\mathfrak{A}$ that, provided with a reward function r, a discount factor γ, and a number of samples $M \geq 0$ is able to output an $\epsilon_{\mathfrak{A}}(M, \gamma)$-approximation of the expert Q-function, with probability at least $1 - \delta$. Then, the influence of the IRL reward and discount factor on the distance between the expert's policy and the learned policy in the successive forward learning task, when M samples are available, can be captured by the next adversarial min-max optimization program:

$$\min_{r,\gamma} \max_{\pi \in \mathcal{G}\left[\widehat{Q}^{\pi_E}_M\right]} \left\| Q^{\pi_E}_{r_E,\gamma_E} - Q^{\pi}_{r_E,\gamma_E} \right\| \tag{1a}$$

$$\text{s.t.} \left\| \widehat{Q}^{\pi_E}_M - Q^{\pi_E}_{r,\gamma} \right\| \leq \epsilon_{\mathfrak{A}}(M, \gamma), \quad r \in \mathcal{R},\ \gamma \in [0, 1), \tag{1b}$$

where $\mathcal{R}$ is a set of available reward functions and $\|\cdot\|$ is a suitably defined norm.

The formulation (1a) constitutes a worst-case guarantee on the sub-optimality of the learned policy π w.r.t. the expert's policy π_E, when evaluated under the true (and unknown) reward r_E and discount factor γ_E. This implies also the compatibility of the learned reward r with the expert's policy, which is the main requirement in IRL. Moreover, the explicit optimization of the learned discount factor γ allows to trade-off with the reward itself the optimality of the learned policy π, and hence tune the sample complexity in the subsequent forward RL task. To this end, we define in (1b) the confidence region of the estimated expert's Q-function $\widehat{Q}_M^{\pi_E}$ under the optimized reward r and discount factor γ. This set determines the feasible domain where we can seek for a greedy policy mimicking the expert's one, which will be known within some accuracy ϵ and confidence level δ varying with the number of data M available during the forward learning phase.

4 Construction of a Solvable IRL Formulation

This section is devoted to the construction of a numerically tractable optimization problem for the formulation (1), which is not readily solvable because of the unknown quantities r_E and γ_E in its objective function (1a). First, we consider linearly independent features $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to [0,1]^{d_\theta}$ and $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to [0,1]^{d_\omega}$ through which we linearly parametrize the reward function and the action-value function: $r_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\phi}(s,a)^\top \boldsymbol{\theta}$ and $\widehat{Q}^{\pi}_{\boldsymbol{\omega}}(s,a;\boldsymbol{\omega}) = \boldsymbol{\psi}(s,a)^\top \boldsymbol{\omega}$, where $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$ and $\boldsymbol{\omega} \in \mathbb{R}^{d_\omega}$. We restrict the search of the greedy policy over a class of parametric differentiable policies $\Pi_{\boldsymbol{\eta}} = \{\pi_{\boldsymbol{\eta}} : \boldsymbol{\eta} \in \mathbb{R}^{d_\eta}\}$. We manipulate the objective function and the constraint of the formulation (1) according to the following steps.

1) Wasserstein Distance on Expert's Policy π_E

We bypass the value-function representations in (1a), and hence remove the dependence on expert's reward r_E and discount factor γ_E, [1] by considering a surrogate objective function based on the notion of policy divergence. Specifically, we bound the Q-function distance with the policy distance, thanks to the following theorem.

Theorem 4.1. *[12] If the MDP and the policy π_E are Lipshitz continuous with constants (L_r, L_P) and L_π. Then, it holds that*

$$\left\|Q^{\pi_E}_{r_E,\gamma_E} - Q^{\pi}_{r_E,\gamma_E}\right\|_\mu \leq \frac{\gamma_E L_r L_\pi}{(1-\gamma_E)\left(1-\gamma_E L_P(1+L_\pi)\right)} \times \int_{\mathcal{S}\times\mathcal{A}} d^{\pi_E}_{\mu,\gamma_E}(s) W_2(\pi_E(\cdot|s), \pi(\cdot|s))\,\mathrm{d}s\,,$$

where W_2 is the L_2-Wasserstein distance and $d^{\pi_E}_{\mu,\gamma_E}$ is the γ_E-discounted state occupancy induced by policy π_E.

Since we deal with continuous actions and deterministic policies (the expert's policy is usually deterministic), the Wasserstein's distance is an appropriate distributional divergence. Given two deterministic policies $\pi_{\boldsymbol{\eta}}$ and π^E, and a state $s \in \mathcal{S}$, it can be computed as $W_2^2\big(\pi^E(s), \pi_{\boldsymbol{\eta}}(s)\big) = \big(\pi^E(s) - \pi_{\boldsymbol{\eta}}(s)\big)^2$.

2) Dealing with the Forward Q-function $\widehat{Q}_M^{\pi_E}$

We replace, in the constraint (1b), the forward Q-function $\widehat{Q}_M^{\pi_E}$, which is not available during the IRL problem, with one computable during the IRL task, say $\widehat{Q}_N^{\pi_E}$. To this end we start simplifying the greedy constraint from (1a):

$$\pi_{\boldsymbol{\eta}} \in \mathcal{G}\left[\widehat{Q}_M^{\pi_E}\right] \Rightarrow \widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) \geq \widehat{Q}_M^{\pi_E}(s,\pi_E(s)) \quad \forall s \in \mathcal{S} \Rightarrow \sum_{s\in\mathcal{D}_{\text{IRL}}} \widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi_E}(s,\pi_E(s)) \geq 0, \tag{2}$$

The first relaxation involves the transition from a greedy policy to all policy with at least some performance improvement, so as to have an explicit dependence of the learner policy $\pi_{\boldsymbol{\eta}}$ on $\widehat{Q}_M^{\pi_E}$. The second relaxation implies that the constraint should hold on average over a finite subset of selected states $\mathcal{D}_{\text{IRL}} \subseteq \mathcal{S}$, since it would be impossible enforce it in an infinite state space (a similar relaxation is operated for instance in [14] for the KL-divergence).

We now use, in the next proposition (see Appendix A for the proof), the original confidence interval (1b) in combination with the policy improvement inequality (2) to compute a looser constraint than (2) but that does not involve the unknown quantity $\widehat{Q}_M^{\pi_E}$.

Proposition 4.2. *Let $\widehat{Q}_M^{\pi_E}$ be a Q-function known with accuracy ϵ_M (with probability $1-\delta_M$) and let $\widehat{Q}_N^{\pi_E}$ be the Q-function estimated with N samples during IRL with accuracy ϵ_N (with probability $1-\delta_N$), i.e., :*

$$\left\|\widehat{Q}_M^{\pi_E}(s,a) - Q^{\pi_E}_{r,\gamma}(s,a)\right\| \leq \epsilon_M, \quad \left\|\widehat{Q}_N^{\pi_E}(s,a) - Q^{\pi_E}_{r,\gamma}(s,a)\right\| \leq \epsilon_N. \tag{3}$$

Then, with probability at least $1-\delta_M-\delta_N$, inequality (2) *is relaxed as:*

$$\sum_{s\in\mathcal{D}_{\text{IRL}}} \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) + 2\epsilon_M + 2\epsilon_N \geq 0. \tag{4}$$

[1] The Q-functions refer to the optimized pair (r,γ) which, for compactness, are removed from the subscripts.

As for the parameters ϵ_M and ϵ_N, we now consider the general structures $\epsilon_M = {}^{\gamma c_M}/_{(1-\gamma)\sqrt{M}}$, $\epsilon_N = {}^{\gamma c_N}/_{(1-\gamma)\sqrt{N}}$, which generalize most of the sample complexity bounds available in the literature [*e.g.* 2]. Parameters c_M and c_N are problem and algorithm-specific constants, that we will treat as hyperparameters.

Remark 4.3. We remark that the assumption in (3) is simply the constraint (1b) itself, which we directly use here to build the new relation involving only $\widehat{Q}_N^{\pi_E}$. In particular, inequality (4) comprises all the uncertainties related either with $\widehat{Q}_M^{\pi_E}$ and $\widehat{Q}_N^{\pi_E}$, and it depends on both the outer optimization variables γ (via parameters ϵ_M and ϵ_N, and the Q-function $\widehat{Q}_N^{\pi_E}$) and r (via the Q-function $\widehat{Q}_N^{\pi_E}$), as well as on the number of samples N available for the IRL task (rather then on the number of samples M that will be used in the forward RL problem). Dependence of (4) on the policy $\pi_{\boldsymbol{\eta}}$ is instead straightforward.

3) Expert's Policy Evaluation with $\widehat{Q}_N^{\pi_E}$

The final stage in our construction of a solvable IRL formulation is the estimation of the new Q-function $\widehat{Q}_N^{\pi_E}$. While, in principle, any policy evaluation algorithm may be used to this purpose, here we resort to the the least-squares temporal difference [LSTDQ, 7] algorithm, for which a confidence region of the form (3) is available via Finite-sample Analysis. In particular, the prediction error of the LSTDQ estimation $\widehat{Q}_N^{\pi_E}(s,a;\widehat{\boldsymbol{\omega}})$ w.r.t. the true value function $Q_{r,\gamma}^{\pi_E}(s,a)$ is provided by [8, Theorem 5], with the accuracy parameter ϵ_N asymptotic to ${}^{\gamma c_N}/_{(1-\gamma)\sqrt{N}}$.

5 Optimization Algorithm

Having introduced in the previous section all the necessary elements for the definition of our new IRL formulation, we discuss now the optimzation algorithm for the solution of the min-max optimization problem. We start by rewriting the final optimization problem in terms of the optimization variables $(\boldsymbol{\theta}, \gamma, \boldsymbol{\eta})$ as:

$$\min_{\substack{\boldsymbol{\theta}\in\mathbb{R}^{d_\theta}\\ \gamma\in[0,1)}} \max_{\boldsymbol{\eta}\in\mathbb{R}^{d_\eta}} \underbrace{\sum_{s\in\mathcal{D}_{\text{IRL}}} W_2\big(\pi^E(s), \pi_{\boldsymbol{\eta}}(s)\big)}_{\triangleq f(\boldsymbol{\eta})}, \quad \text{s.t.} \quad \underbrace{\sum_{s\in\mathcal{D}_{\text{IRL}}} \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) + 2\epsilon_M + 2\epsilon_N \geq 0}_{\triangleq -g(\boldsymbol{\theta},\gamma,\boldsymbol{\eta})}, \tag{5}$$

where the sample-based approximation on the dataset $\mathcal{D}_{\text{IRL}}$ is used for the computation of the Wasserstein distance $f(\cdot)$, as well as for the constraint $g(\cdot)$.

Solving problems as (5) could be extremely challenging in the non-convex setting, where there are no widely-accepted optimization algorithms. Here we look at the min-max optimization as a competitive game between two players and seek for a stationary solution of the problem. Specifically, we reformulate (5) via the potential function $F(\boldsymbol{\theta},\gamma) \triangleq \max_{\boldsymbol{\eta}: g(\boldsymbol{\eta},\boldsymbol{\theta},\gamma)\leq 0} f(\boldsymbol{\eta})$, obtaining $\min_{\boldsymbol{\theta}\in\mathbb{R}^{d_\theta},\,\gamma\in[0,1)} F(\boldsymbol{\theta},\gamma)$. If we assumed the concavity of $f(\cdot)$, we could compute, following [13], the gradient of $F(\cdot)$ as $\boldsymbol{\nabla}_{\boldsymbol{\theta},\gamma} F(\boldsymbol{\theta},\gamma) = \boldsymbol{\nabla}_{\boldsymbol{\theta},\gamma} f(\boldsymbol{\eta}^\star(\boldsymbol{\theta},\gamma))$, where $\boldsymbol{\eta}^\star(\boldsymbol{\theta},\gamma) = \arg\max_{\boldsymbol{\eta}: g(\boldsymbol{\eta},\boldsymbol{\theta},\gamma)\leq 0} f(\boldsymbol{\eta})$, and reach a stationary point of (5). Some caution should be exercised here, since $\boldsymbol{\eta}^\star$ is an implicit function of $(\boldsymbol{\theta},\gamma)$, as it is defined by the constraint $g(\cdot)$. In place of partial derivatives, we should then resort to the following total differential forms:

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial F}{\partial \boldsymbol{\eta}}\frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}\boldsymbol{\theta}}, \quad \text{with} \quad \frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}\boldsymbol{\theta}} = -\frac{\partial g}{\partial \boldsymbol{\theta}}\Big/\frac{\partial g}{\partial \boldsymbol{\eta}}, \qquad \frac{\mathrm{d}F}{\mathrm{d}\gamma} = \frac{\partial F}{\partial \boldsymbol{\eta}}\frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}\gamma}, \quad \text{with} \quad \frac{\mathrm{d}\boldsymbol{\eta}}{\mathrm{d}\gamma} = -\frac{\partial g}{\partial \gamma}\Big/\frac{\partial g}{\partial \boldsymbol{\eta}}, \tag{6}$$

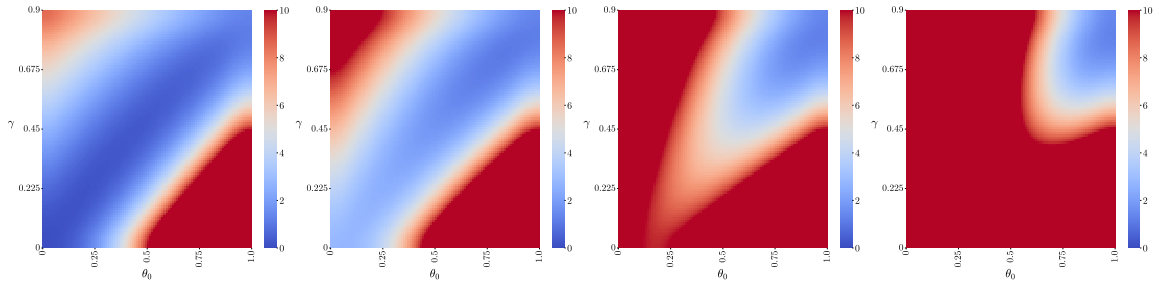
where the differentials and the divisions are to be intended component-wise, and the differentials of $\boldsymbol{\eta}$ are computed by applying to $g(\cdot) = 0$ the Implicit Function Theorem [6]. This iterative procedure would require to find the exact maximum solution of $\boldsymbol{\eta}^\star(\boldsymbol{\theta},\gamma)$, which can be computationally unfeasible if the function $f(\cdot)$ is not concave. Fortunately, we can substitute $\boldsymbol{\eta}^\star$ with an approximate value $\widetilde{\boldsymbol{\eta}}^\star$ so as to satisfy the condition $f(\widetilde{\boldsymbol{\eta}}^\star) \geq \max_{\boldsymbol{\eta}: g(\boldsymbol{\eta},\boldsymbol{\theta},\gamma)\leq 0} f(\boldsymbol{\eta}) - \epsilon$, and relax the concavity assumption. In this case, the algorithm is guaranteed [13] to find an approximate stationary point, where the accuracy level is given by the value of ϵ.

6 Experiments

As a proof of concept of the behaviour of our new IRL formulation, we run a set of experiments and investigate the main characteristics of the approach in the well-know Linear Quadratic Gaussian (LQG [3]) control problem. We consider the scalar case with nominal parameters, and compute in closed-form the expert policy π_E which is optimal for the reward $r_E(s,a) = -s^2 - a^2$ and $\gamma_E = 0.9$. The Q-function feature vector is $\boldsymbol{\psi} = [s^2, a^2, sa]$ so as to span the space of the exact Q-function $Q_{r_E,\gamma_E}^{\pi_E}$, while the reward features are set to $\boldsymbol{\phi} = [-s^2 - a^2, Q_{\bar{s}}^{\pi_E}(s,a)]$, where $Q_{\bar{s}}^{\pi_E}$ represent the Q-function of the expert in a shifted LQG problem with the goal in $\bar{s}$. In the following experiments, the policy is parametrized linearly in the state as $\pi_\eta(s) = \eta s$, and the reward weights $\boldsymbol{\theta}$ are normalized to sum to 1. Finally, we assumed to have an infinite number of samples to solve the forward learning problem, and set $M = \infty$.[2].

[2]We assumed to have a sufficiently high number of samples in the forward learning phase to reach the asymptotic behaviour $\epsilon_M \to 0$

For a complete numerical analysis of the new min-max formulation, we show in Figure 1 the values of the maximum Wasserstein distance $f(\boldsymbol{\eta}^\star)$ in (5) related to the change of the discount factor γ and the weights $\boldsymbol{\theta}$ of the reward $r_{\boldsymbol{\theta}}$, when we gradually increase the value of the goal state $\bar{s}$. As expected, when $\bar{s} = 0$, the formulation selects as the optimal min-max solution $\gamma^\star = 0$, thus minimizing the sample complexity for the forward learning phase, and $r_{\boldsymbol{\theta}^\star} = Q_{\bar{s}}^{\pi_E}$, which recovers the same behaviour of the expert's reward r_E. Interestingly, while the goal $\bar{s}$ moves from 0 (the expert's goal) to an higher value, the formulation trades off the sample complexity induced by a higher γ with the error induced on the learned policy when choosing a sub-optimal reward, moving towards the selection of the unbiased expert reward, selecting $\gamma^\star = \gamma_E$ and $r_{\boldsymbol{\theta}^\star} = r_E$ when the goal $\bar{s} = 0.4$ is too different (sub-optimal) w.r.t the expert goal 0.



(a) $\bar{s} = 0.0$ (b) $\bar{s} = 0.1$ (c) $\bar{s} = 0.2$ (d) $\bar{s} = 0.4$

Figure 1: Value of the objective function $f(\boldsymbol{\eta}^\star)$ in (5) related to the change of the outer variables $(\gamma, \boldsymbol{\theta})$, with $N = 200, c_N = 0.01$ and $M = \infty$. Each plot refers to different values of the goal $\bar{s}$.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, pages 1–8, 2004.

[2] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J. Kappen. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Mach. Learn.*, 91(3):325–349, 2013.

[3] Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear-quadratic control: an introduction*. Simon & Schuster, Inc., 1994.

[4] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4572–4580, 2016.

[5] Jonathan Ho, Jayesh K Gupta, and Stefano Ermon. Model-Free Imitation Learning with Policy Optimization. *International Conference on Machine Learning*, pages 2760–2769, 2016.

[6] Steven G Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012.

[7] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[8] Alessandro Lazaric, Mohammad Ghavamzadeh, and Remi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012.

[9] Alberto Maria Metelli, Matteo Pirotta, and Marcello Restelli. Compatible reward inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):2051–2060, 2017.

[10] Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. *arXiv preprint arXiv:1206.5264*, 2012.

[11] Stuart J. Ng, Andrew Y. and Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[12] Emmanuel Rachelson and Michail G. Lagoudakis. On the locality of action domination in sequential decision making. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2010, Fort Lauderdale, Florida, USA, January 6-8, 2010*, 2010.

[13] Meisam Razaviyayn, Tianjian Huang, Songtao Lu, Maher Nouiehed, Maziar Sanjabi, and Mingyi Hong. Non-convex min-max optimization: Applications, challenges, and recent theoretical advances. *arXiv:2006.08141 [cs, math, stat]*, Aug 2020. arXiv: 2006.08141.

[14] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[15] Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, page 1032–1039. ACM Press, 2008.

A Proof of Proposition 4.2

The idea of this proposition is to start from the constraint (2)

$$\sum_{s\in\mathcal{S}} \widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi_E}(s,\pi_E(s)) \geq 0, \tag{7}$$

which includes the unknown quantity $\widehat{Q}_M^{\pi_E}$, and to compute a new looser inequality that involves only the known quantity $\widehat{Q}_N^{\pi_E}$. To make the above constraint looser, we need to take a larger LHS, *i.e.*, we need to consider an upper bound to $\widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s))$ and a lower bound to $\widehat{Q}_M^{\pi_E}(s,\pi_E(s))$. Starting with the former, we can write:

$$\widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) = \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) + \left(\widehat{Q}\pi_E(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s))\right) + \left(\widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s))\right) \tag{8}$$

$$\leq \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) + \left|\widehat{Q}\pi_E(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s))\right| + \left|\widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s))\right| \tag{9}$$

$$\leq \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) + \epsilon_N + \epsilon_M, \tag{10}$$

where in the last step we applied the assumptions (3). Similarly, we can proceed with latter term, and derive

$$\widehat{Q}_M^{\pi_E}(s,\pi_E(s)) = \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) + \left(\widehat{Q}\pi_E(s,\pi_E(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_E(s))\right) + \left(\widehat{Q}_M^{\pi_E}(s,\pi_E(s)) - \widehat{Q}^{\pi_E}(s,\pi_E(s))\right) \tag{11}$$

$$\geq \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) - \left|\widehat{Q}\pi_E(s,\pi_E(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_E(s))\right| - \left|\widehat{Q}_M^{\pi_E}(s,\pi_E(s)) - \widehat{Q}^{\pi_E}(s,\pi_E(s))\right| \tag{12}$$

$$\geq \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) - \epsilon_N - \epsilon_M, \tag{13}$$

where again in the last step we applied the assumptions (3). Putting back together the computed upper and lower bound we obtain:

$$\sum_{s\in\mathcal{S}} \widehat{Q}_M^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi_E}(s,\pi_E(s)) \leq \sum_{s\in\mathcal{S}} \widehat{Q}_N^{\pi_E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_E}(s,\pi_E(s)) + 2\epsilon_N + 2\epsilon_M. \tag{14}$$

If the original constraint is satisfied, also the looser one holds.

B LQ case with a limited amount of samples

In order to highlight the effect of employing a possibly suboptimal reward when the forward RL phase has a limited number of samples at disposal, we design an additional experiment in the LQ setting. Specifically, we consider the two reward functions learned in the previous experiments $-s^2-a^2$ and $Q_0^{\pi_E}(s,a)$. In Figure 2 we plot the learned parameter (top row) and the average discounted return (bottom row), when performing RL with REINFORCE in two different LQ environments. On the left, we consider the very same environment in which we performed the IRL phase, while on the right we consider an LQ in which we change the dynamical matrix (multiplied by 0.85 compared to the original setting). Thus, on the left, as expected, we observe that both reward functions $Q_0^{\pi_E}(s,a)$ (Controller with IRL reward) and $-s^2-a^2$ (Controller with Real reward) are able to recover the optimal parameter, although $Q_0^{\pi_E}(s,a)$ requires a smaller number of samples. However, the interesting behavior is displayed on the right. While the original reward function $-s^2-a^2$ is able to recover the correct parameter, when the number of samples is limited the *biased* reward $Q_0^{\pi_E}(s,a)$ learned in a different environment is more effective to achieve a reasonable performance. Clearly, as the number of samples increase the effect of bias is more visible.

The effect of using the optimized IRL reward on the sample complexity of the forward learning problem is also depicted in Figure 3. After solving the problem (5) (with parameters $N=200, c_N=0.01, M=\infty, \bar{s}=0$), we employ the final pair $(\gamma^\star,\boldsymbol{\theta}^\star)$ to learn the optimal policy parameter as the number of available samples vary: in particular, we select 20 uniformly random initial states and then estimate the gradient direction in the REINFORCE algorithm by a Monte Carlo evaluation of the reward along trajectories of different length $H=\{1,2,6,10\}$. The plot clearly shows how the IRL reward and discount factor allow the RL algorithm to reach the optimal value of the policy parameters much faster then using the LQG reward, *i.e.*, the number of samples processed during the learning process is much lower if the solution of our proposed IRL formulation is used in place of the expert's (and exact) one (γ_E, r_E).

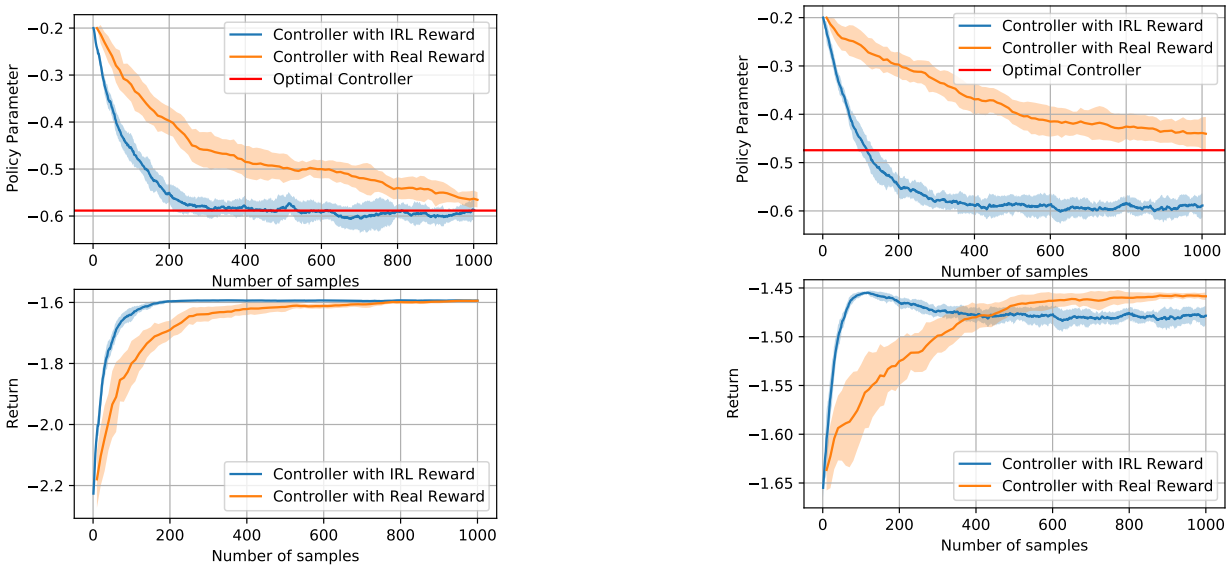


Figure 2: Comparison of the learned policy parameter and average return when learning in the same environment used for IRL (left) and when changing the environment (right) (10 runs, 95% c.i.).

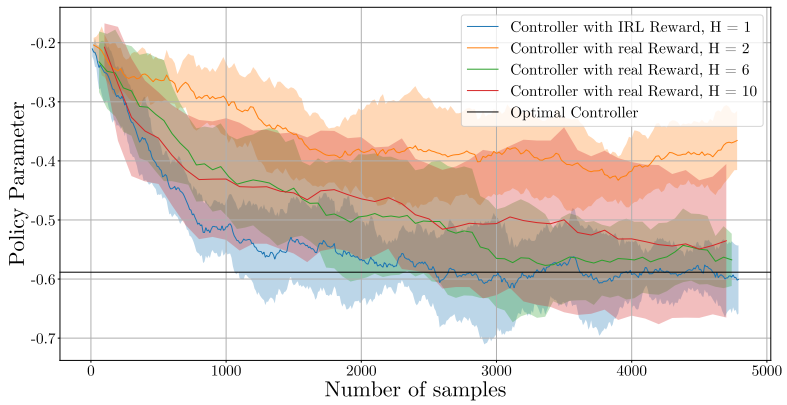


Figure 3: Impact of the optimized IRL reward on the sample efficiency of the forward learning task, and convergence to the expert's policy parameter (10 runs, 95% c.i.).

Adapting the Function Approximation Architecture in Online Reinforcement Learning*

John D. Martin†
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
jmartin8@ualberta.ca

Joseph Modayil†
DeepMind
Edmonton, AB, Canada
modayil@deepmind.com

Fatima Davelouis Gallardo
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
daveloui@ualberta.ca

Michael Bowling
DeepMind &
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
mbowling@ualberta.ca

Abstract

The performance of a reinforcement learning (RL) system depends on the computational architecture used to approximate a value function. Deep learning methods provide both optimization techniques and architectures for approximating nonlinear functions from noisy, high-dimensional observations. However, prevailing optimization techniques are not designed for strictly-incremental online updates. Nor are standard architectures designed to efficiently represent observational patterns from an *a priori* unknown structure: for example, light receptors randomly dispersed in space. This paper proposes an online RL algorithm for adapting a value function's architecture and efficiently finding useful nonlinear features. The algorithm is evaluated in a spatial domain with high-dimensional, stochastic observations. The algorithm outperforms non-adaptive baseline architectures and approaches the performance of an architecture given side-channel information about observational structure. These results are a step towards scalable RL algorithms for more general problem settings, where observational structure is unavailable.

Keywords: Online, Representation, Sparsity, Constructivism

Acknowledgements

The authors would like to acknowledge the support of their many colleagues. A special thanks goes to Tom Schaul, Zaheer Abbas, Brendan Englot, Paul Szenher, Dibya Ghosh, and Shruti Mishra for their comments on an early draft of this work; Parash Rahman for discussions on related work; Brian Tanner for his programming expertise; Patrick Pilarski, Rich Sutton, Adam White, and others at DeepMind for their comments and questions during the conceptual stages of this work.

*This extended abstract builds on the following article: "J. Martin, J. Modayil. Adapting the Function Approximation Architecture in Online Reinforcement Learning. CoRR abs/2106.09776, 2021"

†Equal contribution. Correspondence to John D. Martin

1 Introduction

Architectures for value function approximation typically impose sparse connections with prior knowledge of observational structure. When this structure is known, architectures such as convolutions, transformers, and graph neural networks can be inductively biased with fixed connections. However, there are times when observational structure will be unavailable or prohibitively difficult to encode as an architectural bias—for instance, relating sensors that are randomly dispersed in space. Even when observational structure is available, common biases may not always be the most useful. Yet in all of these situations it is still desirable to approximate value functions with a sparsely-connected architecture for computational efficiency. An important open question is whether equally-useful representations can be constructed when observational structure is unknown—particularly in the incremental, online setting without access to a replay buffer.

Prior work has viewed observational structure as a hidden aspect of the environment [1] or a fixed architectural element that relates inputs. Instances of the latter type examined small input spaces, a combinatorially-large space of graphs [6], or offline methods that learn to reduce connections of a dense architecture [2]. Early work treated observational structure as the learning target: first positing a family of smooth topologies then selecting one to minimize a reconstruction loss [4].

This paper is concerned with how a reinforcement learning system could construct a value function approximation architecture in the absence of observational structure. We propose an online algorithm that adapts connections of a neural network using information deriving strictly from the learner's experience stream, using many parallel auxiliary predictions. Auxiliary predictions are specified as General Value Functions (GVFs) [11], and their weights are used to relate inputs and form subsets we call *neighborhoods*. These represent the input of fully-connected, random subnetworks that provide nonlinear features for a main value function. The algorithm is validated in a synthetic domain with high-dimensional stochastic observations. Results show the algorithm can adapt an approximation architecture without incurring substantial performance loss, while also remaining computationally tractable. Code has been made publicly available at `https://github.com/jdmartin86/frogseye`. Our key contributions are as follows.

Online adaptive architecture: Using many parallel auxiliary learning objectives, our architecture dynamically connects observations to a set of filter banks to form useful nonlinear features.

Useful neighborhoods: The proposed algorithm is shown to compute sparse neighborhoods that perform comparably well to neighborhoods formed from side-channel distance information, and it substantially outperforms static baseline architectures.

Reduced architectural bias with conventional data and computational resources: In a domain of noisy observations with thousands of dimensions, useful neighborhoods are found within five-million time steps of experience, after running on a single GPU for two hours.

2 Problem Setting

This work considers the standard prediction setting for reinforcement learning [10]. The return at time $t \in \mathbb{N}$ is the discounted sum of future rewards, $G_t \equiv R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$. The *value function* gives the expected return from a state: $v(s) \equiv \mathbb{E}[G_t | S_t = s]$. Instead of experiencing states directly, the learner receives a stream of observation vectors $\mathbf{o}_t \in \mathbb{R}^d$ and rewards. The learner's only knowledge of the environment state S_t and dynamics comes from this single stream of experience. With no direct access to the environment state, the learner forms an approximate value function to estimate the expected return. Under a linear approximation, this is defined as a function of a feature vector $\mathbf{x}_t \in \mathbb{R}^\ell$, where

$$\hat{v}(\mathbf{x}_t; \mathbf{w}_t) \equiv \mathbf{w}_t^\top \mathbf{x}_t, \qquad \hat{v}(\mathbf{x}_t; \mathbf{w}_t) \approx v(S_t). \tag{1}$$

In this work the learner incrementally updates its weights $\mathbf{w}_t$ online with a temporal difference algorithm.

2.1 An approximation architecture for the online setting:

We consider an architecture that computes nonlinear features from a sparsely-connected neural network with one hidden layer of random weights. A *neighborhood* is the set of inputs connected to a given nonlinear feature in the network; here it contains a sparse subset of the input, similar to an image patch used with convolutional architectures for vision. Nonlinear features from the i-th neighborhood are computed as a composition of three functions, $\mathbf{y}_t^i \equiv \mathbf{f}(\mathbf{A}\mathbf{M}^i\mathbf{o}_t + \mathbf{a})$. First is a neighborhood selection matrix $\mathbf{M}^i \in \{0,1\}^{k \times d}$, then a linear projection $\mathbf{A} \in \mathbb{R}^{n \times k}$ shared between all the neighborhoods, and finally a nonlinearity $\mathbf{f}\colon \mathbb{R}^n \to \mathbb{R}^n$. The neighborhood selection matrix $\mathbf{M}^i$ is an orthogonal rank-k matrix with one-hot columns—used to mask out an ordered selection of k elements of the observation. The linear projection $\mathbf{A}$ can be thought of as a set of filters, and $\mathbf{a} \in \mathbb{R}^n$ is a bias. The function $\mathbf{f}$ applies a fixed nonlinearity $f\colon \mathbb{R} \to \mathbb{R}$ to each element of its n-dimensional input: $\mathbf{f}(\mathbf{z}) = (f(\mathbf{z}_1), ..., f(\mathbf{z}_n))$. The full feature vector, $\mathbf{x}_t$, contains nonlinear features from m neighborhoods, $\mathbf{M}^i\mathbf{o}_t$, and the current observation, $\mathbf{x}_t \equiv \text{concatenate}(\mathbf{o}_t, \mathbf{y}_t^1, \dots, \mathbf{y}_t^m)$. The m neighborhoods encode the architecture's graph topology.

3 Prediction Adapted Neighborhoods

Instead of constructing neighborhoods with prior knowledge of observational structure, we propose using information that derives from auxiliary RL predictions, specified as general value functions [11]. The resulting collections of observation subsets are called *prediction adapted neighborhoods*. This idea stems from the insights of previous works; one of which suggests that learning many GVF predictions in parallel can provide problem-relevant statistics [8]; another that shows predictions can be sufficient for representing state [5], and a third line of work showing how informative spatial embeddings can be defined from statistics between different observation components [9].

A GVF is defined as the expected return of some auxiliary reward signal $\bar{R}^i_{t+1}$, also known as a *cumulant*. Here auxiliary rewards are given by observation components, and their returns are predicted under the same discount and policy as the main value function (1): $\bar{G}^i_t \equiv \bar{R}^i_{t+1} + \gamma \bar{R}^i_{t+2} + \gamma^2 \bar{R}^i_{t+3} + \ldots$. A selector function $c(i)$ returns an index into the observation vector to determine the i-th cumulant $\bar{r}^i_{t+1} \equiv \mathbf{o}_{t+1}[c(i)]$, for $i = 1, \cdots, m$. In this work, GVFs are approximated by linear functions of the observation $\bar{\mathbf{x}}_t \equiv \mathbf{o}_t$, with weights $\bar{\mathbf{w}}^i$: $\bar{\mathbf{w}}^{i\top} \bar{\mathbf{x}}_t \approx \mathbb{E}[\bar{G}^i_t | S_t = s]$.

Algorithm 1 Online Value Estimation with Prediction Adapted Neighborhoods

1: Initialize: $\mathbf{w}$, $\mathbf{z}$, $\mathbf{A}$ (fixed), $\mathbf{a}$ (fixed), $\mathbf{M}^{1:m}$, $\bar{\mathbf{w}}^{1:m}$, $\bar{\mathbf{z}}^{1:m}$.
2: Receive observation $\mathbf{o}_1$ from the environment.
3: $\mathbf{x}_1 \leftarrow$ ComputeFeatures$(\mathbf{o}_1, \mathbf{M}^{1:m}, \mathbf{A}, \mathbf{a})$
4: **for** $t = 1, 2, 3, \cdots$ **do**
5: Receive $r_{t+1}, \mathbf{o}_{t+1}$ from the environment.
6: $\bar{\mathbf{x}}_j \leftarrow \mathbf{o}_j$ **for** $j \in \{t, t+1\}$
7: **parallel for** $i \in \{1, \cdots, m\}$
Update GVF weights with TD(λ).
8: $\bar{r}^i_{t+1} \leftarrow \mathbf{o}_{t+1}[c(i)]$
9: $\delta \leftarrow \bar{r}^i_{t+1} + \gamma \bar{\mathbf{w}}^{i\top} \bar{\mathbf{x}}_{t+1} - \bar{\mathbf{w}}^{i\top} \bar{\mathbf{x}}_t$
10: $\bar{\mathbf{z}}^i \leftarrow \gamma\lambda \bar{\mathbf{z}}^i + \bar{\mathbf{x}}_t$
11: $\bar{\mathbf{w}}^i \leftarrow \bar{\mathbf{w}}^i + \bar{\alpha}\delta \bar{\mathbf{z}}^i$
Construct top-k selection matrix.
12: $\boldsymbol{\ell} \leftarrow$ Top$(k, \bar{\mathbf{w}}^i)$, where $|\bar{\mathbf{w}}^i_{\boldsymbol{\ell}_1}| \geq \cdots \geq |\bar{\mathbf{w}}^i_{\boldsymbol{\ell}_k}|$
13: **parallel for** $j, l \in \{1, \ldots, k\} \times \{1, \ldots, d\}$
14: $\mathbf{M}^i_{j,l} \leftarrow \mathbf{1}\{l = \boldsymbol{\ell}_j\}$
15: $\mathbf{x}_{t+1} \leftarrow$ ComputeFeatures$(\mathbf{o}_{t+1}, \mathbf{M}^{1:m}, \mathbf{A}, \mathbf{a})$
Update main prediction weights with TD(λ).
16: $\delta \leftarrow r_{t+1} + \gamma \mathbf{w}^\top \mathbf{x}_{t+1} - \mathbf{w}^\top \mathbf{x}_t$
17: $\mathbf{z} \leftarrow \gamma\lambda \mathbf{z} + \mathbf{x}_t$
18: $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta \mathbf{z}$

Algorithm 2 ComputeFeatures

1: *input:* $\mathbf{o}, \mathbf{M}^{1:m}, \mathbf{A}, \mathbf{a}$
2: **parallel for** $i \in \{1, \cdots, m\}$
3: $\mathbf{y}^i \leftarrow \mathbf{f}(\mathbf{A}\mathbf{M}^i\mathbf{o} + \mathbf{a})$
4: **return** concatenate$(\mathbf{o}, \mathbf{y}^1, ..., \mathbf{y}^m)$

One of our key discoveries is that observations can be related when used as auxiliary rewards for linear GVFs. Recall a GVF in this work predicts the future discounted sum of an observation signal, i.e. the auxiliary reward. Also recall that a GVF is approximated with a linear combination of all observation components, and that linear representations are insensitive to the input ordering. This latter point means an accurate GVF can be obtained without knowledge of the observational structure, which is typically encoded in the relative ordering of inputs. Observations that closely relate to an auxiliary reward tend to have high-magnitude prediction weights. Based on these principles, prediction adapted neighborhoods are formed with observations with high absolute GVF weights $\bar{\mathbf{w}}^i$.

Algorithm 1 outlines how to compute prediction adapted neighborhoods in the online prediction setting (lines 6–14), with TD(λ) and accumulating traces $\mathbf{z}$. The algorithm constructs each neighborhood with the k observations whose absolute GVF weights are largest (lines 12–14), and it encodes them with the selection matrices $\mathbf{M}^i$.

In contrast to prior work that regularizes the internal architecture weights $\mathbf{A}$ with auxiliary prediction losses [3], our proposed algorithm uses auxiliary GVFs to impose sparse connections with a predictive structure. This information derives entirely from the observation stream, with no a priori knowledge of the observational structure. Furthermore, our proposed algorithm continually adapts the architecture's connections in response to patterns of the observation stream.

4 Empirical Results in the Frog's Eye Domain

Drawing inspiration from the arrangement of light receptors in a frog's eye, we introduce an environment for studying continual prediction in the absence of observational structure (Figure 1). In our environment, light receptors have a uniformly-irregular spatial distribution, with neither the concentrated fovea of a mammalian eye nor the regular grid of a silicon imaging chip. A simulated frog needs to anticipate the arrival of an insect without any knowledge of how its light receptors relate to one another.

The full observation vector $\mathbf{o}_t \in \{0, 1\}^{4000}$ is given from a random ordering of 4000 proximity receptor outputs. The observation's ordering is scrambled at the start of learning and then held fixed. Observations are corrupted with fifty-percent uniform binary noise. The reward is

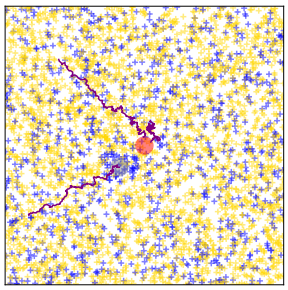
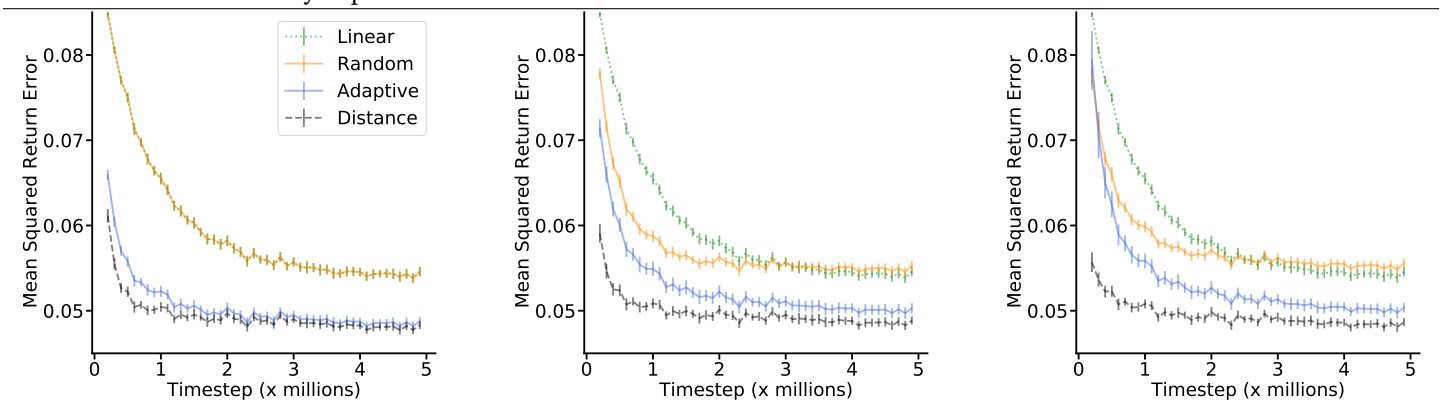


Figure 1: **The Frog's Eye domain:** An insect (gray circle) is detected by irregularly-distributed proximity receptors (blue: on, gold: off). Three different insect trajectories are shown: two prior, one active. A reward of +1 is received upon entering the circular red region.



(a) Majority (b) LTU (c) ReLU

Figure 2: **Prediction adapted neighborhoods are useful:** For each feature type, the Adaptive architecture (using prediction adapted neighborhoods) approaches the performance of Distance (biased with knowledge of observational structure). Adaptive also performs better than fixed architectures using random neighborhoods (Random) or none at all (Linear).

$+1$ whenever the insect enters a circular region at the center of the observable space, and it is zero otherwise. An insect entering the circular region will disappear and respawn at a random location. This process continues indefinitely.

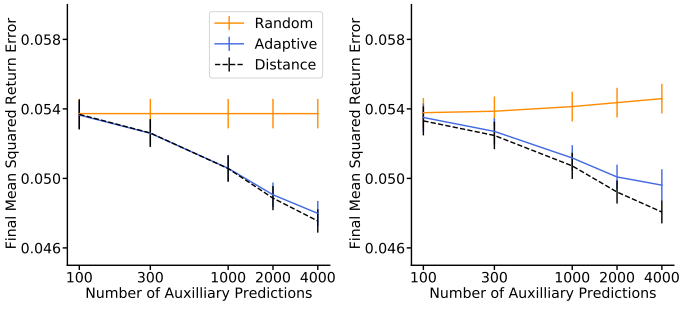
Methodology: Our experiments compare prediction accuracy of different value function architectures while specifically controlling for the effects of neighborhood selection. We use empty neighborhoods with $m = 0$ (denoted as Linear), sparse neighborhoods with k randomly-selected observation components (Random), prediction adapted neighborhoods from fixed randomly-selected cumulants (Adaptive), and sparse neighborhoods containing the k-nearest sensors to each cumulant using side-channel distance information (Distance). The three different nonlinearities considered are shown in Table 1. We report the average and standard error confidence intervals from 30 trials, which were run to 5 million steps and observed to complete in under two hours on a V100 GPU. Results are shown in Figures 2, 3, and 4. More experimental details including information about hyperparameter selection are provided in the full-length paper [7].

Features	$f(z)$	**A** shape and values
Majority	$\mathrm{I}(z > \frac{2k}{3})$	$(1 \times k)$ filled with one
LTU	$\mathrm{I}(z > 4)$	$(n \times k)$ from $\mathcal{N}(0, 1)$
ReLU	$\mathrm{ReLU}(z - 4)$	$(n \times k)$ from $\mathcal{N}(0, 1)$

Table 1: Nonlinear feature types defined on a neighborhood.

4.1 Are Prediction Adapted Neighborhoods Useful?

Our first experiment asks: *do prediction adapted neighborhoods provide measurable utility for approximating the main value function?* Figure 2 shows learning curves of prediction accuracy. The Adaptive architecture—using prediction adapted neighborhoods—leads to little performance loss compared to the Distance architecture when evaluated across three types of activation functions. Recall the Distance architecture uses neighborhoods that contain the k-nearest sensors to each cumulant. The small performance gap between Adaptive and Distance suggests that prediction adapted neighborhoods are just as useful in this domain as neighborhoods biased with side-channel distance information.



(a) Majority (b) LTU

Figure 3: **Auxiliary Learning Effect**: Increasing the number of auxiliary predictions leads to performance improvements, as measured with final error. These observations relate to two well-known results in RL; the idea that states can be represented as a collection of predictions [5], and that a representation can improve with more auxiliary predictions [3].

4.2 Does Performance Scale with more GVFs?

Our second experiment examined whether prediction accuracy of Adaptive monotonically improves with an increasing number of auxiliary predictions m. Indeed, in Figure 3 we see that Adaptive's performance improves as the number of GVFs increase. The best performance is with $m = 4000$ predictions: one for each sensor in the simulated frog's eye. These observations relate to two well-known results in RL. Namely, the idea that states can be represented as a collection of predictions [5], and that a representation can improve with more auxiliary predictions [3]. Across the range of m values, the Adaptive architecture's performance is comparable to the Distance baseline.

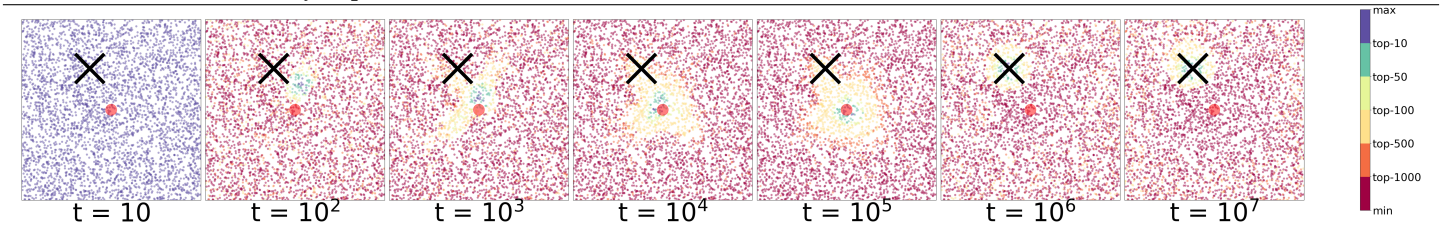


Figure 4: **Prediction adapted neighborhoods appear to encode a temporally-stable spatial structure:** A spatial distribution of auxiliary weights is shown for a random GVF with a cumulant marked by an $\times$. The top ten neighborhood use converges and remains centered around the cumulant's location.

4.3 The Spatial Structure of Adapted Neighborhoods

A final inspection examined whether the prediction weights of a GVF contained any spatial structure. Figure 4 shows one set of auxiliary weights as learning progresses for a randomly-selected GVF. Clearly there are GVFs whose weights encode a local degree of spatial structure. Furthermore, this structure appears temporally stable over the extended regime of ten million time steps. These two points highlight that even without prior knowledge of the observation's spatial structure, auxiliary GVFs are able to relate observations in a similar way—ultimately one that is useful for the main prediction.

5 Conclusion

This paper addressed how an RL system could construct a value function architecture, specifically in the incremental online setting for prediction, and in the absence of observational structure. One of our key discoveries was that weights of auxiliary predictions could be used to relate observations and impose useful sparse connections in a random neural network approximating a value function. We believe this work could be useful for designing general RL systems that acquire knowledge from sensory inputs whose observational structure is unknown.

References

[1] R. Evans, J. Hernández-Orallo, J. Welbl, P. Kohli, and M. Sergot. Making sense of sensory input. *Artificial Intelligence*, 293:103438, 2021.

[2] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[3] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[4] N. Le Roux, Y. Bengio, P. Lamblin, M. Joliveau, and B. Kégl. Learning the 2-d topology of images. *Advances in Neural Information Processing Systems*, 20:841–848, 2007.

[5] M. L. Littman, R. S. Sutton, and S. P. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561, 2001.

[6] A. R. Mahmood and R. S. Sutton. Representation search through generate and test. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[7] J. D. Martin and J. Modayil. Adapting the function approximation architecture in online reinforcement learning. *arXiv preprint arXiv:2106.09776*, 2021.

[8] J. Modayil, A. White, and R. S. Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 2014.

[9] D. Pierce and B. J. Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92(1-2):169–227, 1997.

[10] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

Decentralized Shield Decomposition for Safe Multi-Agent Reinforcement Learning

Daniel Melcer
Northeastern University
Boston, MA 02115
melcer.d@northeastern.edu

Chris Amato*
Northeastern University
Boston, MA 02115
c.amato@northeastern.edu

Stavros Tripakis*
Northeastern University
Boston, MA 02115
stavros@northeastern.edu

Abstract

Learning safe solutions is an important but challenging problem in multi-agent reinforcement learning (MARL). Shielded reinforcement learning is one approach for preventing agents from choosing unsafe actions. Current shielded reinforcement learning methods for MARL make strong assumptions about communication and full observability. In this work, we extend the formalization of the shielded Reinforcement Learning problem to multi-agent environments without any communication assumptions. We then identify a subset of safety specifications and centralized shields which allow for decentralization. An algorithm is presented for the construction of a decentralized shield, given a centralized shield that meets the requirements for decentralization. Our preliminary results show that this method of decentralization does not significantly decrease performance for most variations in a standard benchmark task, compared to a method where agents are able to communicate with each other to avoid taking unsafe actions. We conclude by presenting a number of additional research directions which we are actively investigating.

Keywords: Multi-Agent Reinforcement Learning; Shielding; Safety

Acknowledgements

This work has been supported by NSF SaTC Award CNS-1801546 and NSF CAREER Award IIS-2044993. This work was completed in part using the Discovery cluster, supported by Northeastern University's Research Computing team.

*Equal Advising

1 Introduction & Related Work

Recently, the advent of deep reinforcement learning has enabled the application of classic reinforcement learning techniques to highly complex domains such as Atari [8], and the game of Go [9]. However, the neural networks which power these methods are opaque, and it is difficult to verify their properties. This limits the real-world application of reinforcement learning in safety-critical systems.

Therefore, the subject of Safe Reinforcement Learning has been the focus of extensive study. Perhaps the most straightforward approach to this is the idea of shielding [1]. A shield monitors the agents and environment during both training and execution. When an agent proposes an action, the shield evaluates whether this action is safe. If the action is unsafe, or if it could lead to a future state in which no safe actions are available, the shield substitutes a known-safe action, and potentially assigns a punishment for the attempted unsafe action. Alternatively, the shield disables all unsafe actions at a given state, allowing the agent to only choose from a set of safe actions. This work draws from the idea of Runtime Enforcement [2] from the Formal Methods community. The idea of shielding has also been extended to continuous action spaces [3], in which the necessity and extent of action modifications are obtained as the solution to a set of linear equations. Additionally, shielding has been extended to multi-agent environments [4]. However, in previously proposed multi-agent shielding methods, all agents must be able to communicate with each other during action selection.

Therefore, we present a method of shield decentralization, allowing agents to avoid unsafe actions during training and testing, without any communication with each other. In Section 2, we introduce the setting and define several structures which are useful for the rest of the paper. In Section 3, we formalize the decentralized shielding problem. Finally, in Section 4, we show that decentralized shielding maintains the safety benefits of centralized shielding, without sacrificing performance in the majority of environments that we tested.

2 Preliminaries

Let $\mathbb{R}$ denote the set of real numbers. Let $\Delta(X)$ be the set of all probability distributions over set X. We denote $\mathbb{1}_x$ to be the indicator variable, equalling 1 when x holds and 0 otherwise.

Definition 1 (MMDP) *A* Multi-agent Markov Decision Process (MMDP) *is a tuple* $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$ *where* $\mathbb{D} = \{1, \ldots, n\}$ *is a set of agents,* $\mathbb{S}$ *is a set of states,* $\mathbb{A} = \prod_{i \in \mathbb{D}} \mathbb{A}_i$ *is a finite joint action space, factorizable into* n *individual action spaces,* $\mathcal{E} : \mathbb{S} \to \mathcal{P}(\mathbb{A})$ *represents the set of actions available to the agents at a given state,* $T : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{S})$ *is the transition probability distribution function,* $\gamma \in [0, 1]$ *represents the discount factor for future rewards,* $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ *is the reward given to each agent after a transition, and* $b_0 \in \Delta(\mathbb{S})$ *is the distribution of initial states.*

We call a sequence of states and joint actions $((s_0, s_1, \ldots, s_{n+1}), (a_0, a_1, \ldots, a_n))$ an *environment trace* of MMDP $\mathcal{M}$ if every pair (s_i, a_i, s_{i+1}) is allowed by the transition function of the MMDP.

$\mathcal{M}$ is said to be in a *deadlock* at state $s \in \mathbb{S}$ if $\mathcal{E}(s) = \emptyset$. $\mathcal{M}$ is *deadlock-free* if there are no environment traces of $\mathcal{M}$ that end in a deadlock state.

We will often have some prior knowledge of how to abstract the states in a meaningful way for safety properties. For our purposes, these properties are agnostic to the choice of reward function. For example, in a domain where we avoid collisions between agents, the relative positions of the agents may be important, even if the absolute positions of all agents is not known. We call a function $f : \mathbb{S} \to L$ which translates MMDP states into members of a given *label set* an *abstraction function*.[1] All agents in the MMDP are assumed to have access to the corresponding label for the current state. Every environment trace through a MMDP has a corresponding *label trace* $((l_0, a_0), \ldots, (l_n, a_n))$, where $\forall i \in \{0, \ldots, n\}, l_i = f(s_i)$.[2] Note that several unique environment traces may correspond to the same label trace.

Definition 2 (DFA) *A* Deterministic Finite Automaton (DFA) *is a tuple* $\phi = (Q, \Sigma, \delta, s_0, F)$ *where* Q *is a set of states,* $s_0 \in Q$ *is the initial state,* Σ *is an alphabet,* $\delta : Q \times \Sigma \to Q$ *is the transition function, and* $F \subseteq Q$ *is the set of accepting states.*

Given a word $(w_0, w_1, \ldots, w_n) \in \Sigma^*$, the corresponding *DFA trace* $(q_0, q_1, \ldots, q_{n+1}) \in Q^*$ is obtained by stepping through the transition function. A word is *accepted* by the DFA iff the last state of its corresponding DFA trace is a member of F.

For a given MDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$ and label set L, we define a *safety specification* over $\mathcal{M}$ and L to be a DFA $\phi^s = (Q^s, (L \times \mathbb{A}), \delta^s, s_0^s, F^s)$ which accepts the empty word and does not contain any transitions from $Q \setminus F$ to F. $\mathcal{M}$ satisfies ϕ^s ($\mathcal{M} \vDash \phi^s$) iff all possible label traces of $\mathcal{M}$ are accepted by ϕ^s.

ϕ^s induces the *accepting action function* $\mathcal{C}^\phi : Q \times L \to \mathcal{P}(\mathbb{A}) = (q, l) \to \{a \in \mathbb{A} | \delta(q, (l, a)) \in F\}$; given a DFA state and label, output the actions which transition the DFA to an accepting state.

[1] Prior work refers to f as the *MDP observer function* [1, 4].

[2] We intentionally omit $f(s_{n+1})$ to allow label traces to be a word of $(L \times \mathbb{A})^*$.

Definition 3 (DFA-MMDP Composition) *Given an MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$, label set L, an abstraction function f over $\mathcal{M}$ and L, and DFA $\phi^g = (G, (L \times \mathbb{A}), \delta^g, s_0^g, F^g)$ where $s_0^g \in F^g$, the* composition *of M and ϕ^g is a new MMDP denoted as $\mathcal{M}||\phi^g = (\mathbb{D}^c, \mathbb{S}^c, \mathbb{A}^c, \mathcal{E}^c, T^c, \gamma^c, R^c, b_0^c)$, where $\mathbb{D}^c = \mathbb{D}, \mathbb{S}^c = \mathbb{S} \times G, \mathbb{A}^c = \mathbb{A}, \mathcal{E}^c((s, q^g)) = \mathcal{E}(s) \cap \mathcal{C}^{\phi^g}(q^g, f(s)), T^c((s', q'^g)|(s, q^g), a) = T(s'|s, a) \cdot \mathbb{1}_{q'^g = \delta^g(q^g, (f(s), a))}, \gamma^c = \gamma, R^c((s, q^g), a) = R(s, a)$, and $Pr[b_0^c = ((s, q^g))] = Pr[b_0 = s] \cdot \mathbb{1}_{q^g = s_0^g}$.*

Definition 4 (Centralized Shield) *A DFA ϕ^g is a* centralized shield[3] *for MMDP $\mathcal{M}$ that enforces safety specification ϕ^s iff $\mathcal{M}||\phi^g \vDash \phi^s$ and $\mathcal{M}||\phi^g$ is deadlock-free.*

In practice, the DFA-MMDP composition is never explicitly constructed; rather, a shielded agent only keeps track of the current DFA state, and uses the accepting action function to determine which actions are safe. However, the use of a centralized shield requires the implicit assumption that agents are able to instantaneously communicate with each other. For example, suppose we had a MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$ where $\mathbb{D} = \{1, 2\}, \mathbb{A} = \{b, c\} \times \{x, y\}$. There may exist a centralized shield ϕ^g, such that $\mathcal{M}||\phi^g = (\mathbb{D}^g, \mathbb{S}^g, \mathbb{A}^g, \mathcal{E}^g, T^g, \gamma^g, R^g, b_0^g)$ where for some state $s \in \mathbb{S}^g$, $\mathcal{E}^g(s) = \{(b, x), (c, y)\}$. Without knowing which action agent 1 will pick, agent 2 has no way of picking an action while ensuring that the resulting joint action is safe, unless it can communicate with agent 1.

The problem of synthesizing a centralized shield DFA has previously been explored in [1] for the single-agent case, and [4] for the multi-agent case. Our focus in this paper is on creating shields without the implicit communication requirement mentioned above.

3 Decentralized Shielding

Definition 5 (Individual Specification) *Given a MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A} = \prod_{i \in \mathbb{D}} \mathbb{A}_i, \mathcal{E}, T, \gamma, R, b_0)$ and a label set L, a DFA of the form $\phi^{s_i} = (Q^{s_i}, (L \times \mathbb{A}_i), \delta^{s_i}, s_0^{s_i}, F^{s_i})$ is called an* Individual Specification *for agent $i \in \mathbb{D}$.*

Definition 6 (Action Product) *Given a MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A} = \prod_{i \in \mathbb{D}} \mathbb{A}_i, \mathcal{E}, T, \gamma, R, b_0)$, a label set L, two agents $i, j \in \mathbb{D}$, and two individual specifications $\phi^{s_i} = (Q^{s_i}, (L \times \mathbb{A}_i), \delta^{s_i}, s_0^{s_i}, F^{s_i}), \phi^{s_j} = (Q^{s_j}, (L \times \mathbb{A}_j), \delta^{s_j}, s_0^{s_j}, F^{s_j})$, the* Action Product[4] *of ϕ^{s_i} and ϕ^{s_j} is a new DFA denoted as $\phi^{s_i} \boxtimes \phi^{s_j} = (Q^{s_p}, (L \times \mathbb{A}_p), \delta^{s_p}, s_0^{s_p}, F^{s_p})$ where $Q^p = Q^{s_i} \times Q^{s_j}, \mathbb{A}_p = \mathbb{A}_i \times \mathbb{A}_j, \delta^{s_p}((q_i, q_j), (l, a_i, a_j)) = (\delta^{s_i}(q_i, (l, a_i)), \delta^{s_j}(q_j, (l, a_j))), s_0^{s_p} = (s_0^{s_i}, s_0^{s_j})$, and $F^{s_p} = F^{s_i} \times F^{s_j}$*

Definition 7 (Decentralized Shield) *Given a MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$ and a safety specification ϕ^s, a list of $|\mathbb{D}|$ DFAs $\{\phi^{s_1}, \ldots, \phi^{s_n}\}$ is a* Decentralized Shield *which enforces ϕ^s iff for every $i \in \mathbb{D}$, ϕ^{s_i} is an individual specification for agent i, and $\phi^{s_1} \boxtimes \ldots \boxtimes \phi^{s_n}$ is a centralized shield which enforces ϕ^s.*

We would like to synthesize a decentralized shield which enforces a given safety specification. As a first step, we attempt to synthesize a centralized shield—if this (decidable) process fails, we can assume that decentralized shield synthesis will be impossible.

Problem 1 (Dec-A Shield Decomposition) *Given a MMDP $\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$, a label set L, an abstraction function over $\mathcal{M}$ and L, a safety specification ϕ^s, and a centralized shield $\phi^g = (G, (L \times \mathbb{A}), \delta^g, s_0^g, F^g)$ that enforces ϕ^s, synthesize a decentralized shield which enforces ϕ^s.*

This problem statement admits a trivial solution: for every pair $(q, l) \in (F^g \times L)$, choose a single joint action $A^{q,l} = (a_1^{q,l}, \ldots, a_n^{q,l})$ in advance such that $\delta^s(q, (l, A^{q,l})) \in F^g$. Such an action is guaranteed to exist due to the deadlock-free nature of a centralized shield. The decentralized shield for each agent i has the same set of states. When the shield state equals q and the current environment label is l, the decentralized shield forces the agent to take action $a_i^{q_l}$. While this prescriptive approach technically results in a decentralized shield that enforces ϕ^s, it doesn't allow the agents any freedom to explore actions and optimize for expected future rewards.

To address this, we add the constraint that the decentralized shields must be *maximally-permissive*. Shield ϕ_X^g is at least as permissive as shield ϕ_Y^g iff every label trace which is accepted by ϕ_Y^g is also accepted by ϕ_X^g. A shield is maximally permissive iff there are no shields which are strictly more permissive than it. Note that there may exist multiple incomparable maximally permissive shields.

We propose Algorithm 1 to construct a Maximally Permissive Dec-A Shield Decomposition for the subset of centralized shields $(G, (L \times \mathbb{A}), \delta^g, s_0^g, F^g)$ which are *stateless*—centralized shields where $F^g = \{s_0^g\}$.[5] Each agent's individual shield

[3] Some prior formulations define the shield as a reactive system which prescribes a specific action in response to a proposed action and environment label [2]; for now, we use a definition based on allowed actions for simpler analysis.

[4] This is similar to the standard DFA *Synchronous Product* [6], except that the inputs only synchronize on the label, not the actions. Note that the action product is associative, and therefore can naturally be extended to more than two DFAs.

[5] There may still exist an arbitrary number of states in $G \setminus F^g$.411

Algorithm 1 Decompose a set of actions

Input
$\mathcal{M} = (\mathbb{D}, \mathbb{S}, \mathbb{A}, \mathcal{E}, T, \gamma, R, b_0)$ A MMDP
$\phi^g = (G, (L \times \mathbb{A}), \delta^g, s_0^g, F^g)$ Stateless centralized shield
$l \in L$ A label
$\mathcal{C}^l : \mathcal{P}(\mathbb{A})$ Joint actions where $\delta^g(s_0^g, (l, a)) = s_0^g$.
Output
$\mathcal{D}_i^l : \mathcal{P}(\mathbb{A}_i), i \in \mathbb{D}$ Set of actions where $\prod_{i \in \mathbb{D}} \mathcal{D}_i^l \subseteq \mathcal{C}^l$
procedure DECOMPOSESINGLESTATESHIELD($\mathcal{C}^l$)
Choose arbitrary $a = (a_1, a_2, \ldots, a_n) \in \mathcal{C}^l$
$\forall i \in \mathbb{D} : \mathcal{D}_i^l = \{a_i\}$
for $i \in \mathbb{D}$ **do** ▷ Optionally permute agents
for $a_i' \in \mathbb{A}_i$ **do**
if $\{a_i'\} \times \prod_{j \in \mathbb{D}, j \neq i} \mathcal{D}_i^l \subseteq \mathcal{C}^l$ **then**
$\mathcal{D}_i^l \leftarrow \mathcal{D}_i^l \cup \{a_i'\}$
end if
end for
end for
return $\mathcal{D}_i^l \forall i \in \mathbb{D}$
end procedure

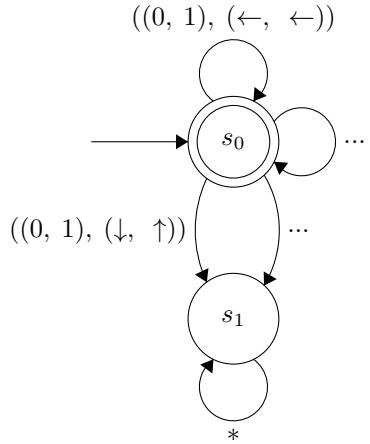


Figure 1: A partial view of the safety specification and centralized shield for the two-agent gridworld collision domain. Based on the relative positions of the two agents with respect to each other (the label), and the joint actions taken by the agents, the automaton either stays in the accepting state or transitions to a trapping non-accepting state, representing a collision having occurred.

will also be stateless. For every label, each agent can take any individual action in $\mathcal{D}_i^l$ to stay in the accepting state. This is safe because Algorithm 1 maintains the invariant that $\prod_{i \in \mathbb{D}} \mathcal{D}_i^l \subseteq \mathcal{C}^l$; all combinations of individual actions from the decentralized shield would have been deemed a safe joint action by the centralized shield. If the input is a maximally permissive centralized shield, the output shield is maximally permissive among all decentralized shields. The full proof is omitted due to space, but the innermost `if` statement in the algorithm implies that no agent can take any other individual actions and guarantee safety of the joint action.

4 Experiments

4.1 The Gridworld Collision Domain

We adapt the 2-agent gridworld maps from Melo and Veloso [7] to test our method. Each agent has five actions available: movement in the four cardinal directions, or a no-op. If all agents reach their goal positions, they each receive a +100 reward. Any agents which hit a wall receive a -10 reward. The safety specification is that a collision between two agents should never occur (including agents crossing over each other). If an agent violates this specification, all agents receive a -10 reward. Agents receive a -1 reward at all other time steps. We use the four maps "MIT," "ISR," "SUNY," and "Pentagon," both with and without randomized start positions. These maps range in size from 8x10 to 10x23.

These maps have previously been used to test multiagent shielding, with stateful shields and instantaneous local communication [4]. We use an abstraction function that calculates the relative position of the agents; the label set is all relative positions of the agents with respect to each other. The centralized shield in this environment, which was manually generated, happens to be identical to the safety specification—the two-state DFA shown in Figure 1. All joint actions are allowed by this centralized shield automaton, except for those which result in a collision. We decentralize the shield using the procedure described above. The resulting shields are more conservative when the agents are near each other, ensuring that no more than one agent moves into a potentially contested area.

4.2 Agents and Training

We train independent tabular Q-learning agents using ϵ-greedy exploration, with a linear ϵ annealing schedule from 0.2 to 0.01. The discount factor is 0.9. Agents are trained with a centralized shield, a decentralized shield, and with no shield. When an agent attempts to take an action a which is not allowed by the shield, the shield substitutes the action with a known-safe no-op action a' and observes the environment transition (s, a', r, s'). The agent is trained on this real transition, plus an additional synthetic transition $(s, a, r + r_p, s')$ where r_p is a penalty reward of -10. Each configuration is run with 10 random seeds for one million steps each. We record the average number of safety specification violations during training.

4.3 Testing and Results

After training completes, we run 100 episodes for each configuration, using a greedy policy, and average the rewards. Preliminary results are shown in Table 1. In nearly all the configurations, the decentralized shield performs similarly to the centralized shield, and both avoid the safety violations which are occasionally encountered by the unshielded agent.

Map Name	Pentagon		ISR		SUNY		MIT	
Randomized Starts	No	Yes	No	Yes	No	Yes	No	Yes
Centralized Shield	91.5 (0)	88.9 (0)	92.3 (0)	89.3 (0)	89.4 (0)	-291.7 (0)	76.5 (0)	78.4 (0)
Decentralized Shield	90.8 (0)	84.2 (0)	92.7 (0)	86.9 (0)	88.8 (0)	-307.6 (0)	19.6 (0)	84.3 (0)
No Shield	89.4 (0)	90.5 (0)	92.9 (0.04)	83.4 (0)	88.2 (0)	-675.9 (15.6)	14.8 (1.63)	80.4 (0)

Table 1: Average reward during testing over 10 seeds, and safety violations per episode encountered during the last 5000 time steps of training. Due to a logging issue, we could not calculate the total number of safety violations for all configurations; in a small sample, this value is 0 for the centralized and decentralized shield cases, and several thousand for the no shield case.

5 Conclusion

Preliminary results show that our shield decomposition method elicits comparable agent performance to a centralized shield in the majority of environments, without any communication between agents. We are currently working to extend our results in several ways. First, we are implementing additional agents and training schemes, including individual Deep Q Networks with convolutional layers [8], and Centralized-Critic Decentralized-Actor methods [5]. Second, we are currently investigating what shields can enforce which specifications in partially-observable environments, and how to apply our shield decomposition method in such environments. Third, we plan to test our method in several other environments, such as the cooperative navigation domain introduced in [10]. Lastly, we aim to generalize our shield decomposition method to work with a broader set of centralized shields and safety specifications.

References

[1] ALSHIEKH, M., BLOEM, R., EHLERS, R., KÖNIGHOFER, B., NIEKUM, S., AND TOPCU, U. Safe reinforcement learning via shielding, 2017.

[2] BLOEM, R., KOENIGHOFER, B., KOENIGHOFER, R., AND WANG, C. Shield synthesis: Runtime enforcement for reactive systems, 2015.

[3] DALAL, G., DVIJOTHAM, K., VECERIK, M., HESTER, T., PADURARU, C., AND TASSA, Y. Safe exploration in continuous action spaces, 2018.

[4] ELSAYED-ALY, I., BHARADWAJ, S., AMATO, C., EHLERS, R., TOPCU, U., AND FENG, L. Safe multi-agent reinforcement learning via shielding. *CoRR abs/2101.11196* (2021).

[5] FOERSTER, J., FARQUHAR, G., AFOURAS, T., NARDELLI, N., AND WHITESON, S. Counterfactual multi-agent policy gradients, 2017.

[6] HOLZMANN, G. J. *Explicit-State Model Checking*. Springer International Publishing, 2018, p. 153–171.

[7] MELO, F. S., AND VELOSO, M. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Richland, SC, 2009), AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, p. 773–780.

[8] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing atari with deep reinforcement learning, 2013.

[9] SILVER, D., SCHRITTWIESER, J., SIMONYAN, K., ANTONOGLOU, I., HUANG, A., GUEZ, A., HUBERT, T., BAKER, L., LAI, M., BOLTON, A., CHEN, Y., LILLICRAP, T., HUI, F., SIFRE, L., VAN DEN DRIESSCHE, G., GRAEPEL, T., AND HASSABIS, D. Mastering the game of Go without human knowledge. *Nature 550*, 7676 (Oct. 2017), 354–359.

[10] YANG, J., NAKHAEI, A., ISELE, D., FUJIMURA, K., AND ZHA, H. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning, 2020.

Query-Reward Tradeoffs in Multi-Armed Bandit Problems

Nadav Merlis
Technion – Institute of Technology
merlis@campus.technion.ac.il

Yonathan Efroni
Microsoft Research, New York
yefroni@microsoft.com

Shie Mannor
Technion – Institute of Technology
Nvidia Research, Israel
shie@ee.technion.ac.il

Abstract

We consider a stochastic multi-armed bandit setting where reward must be actively queried for it to be observed. We provide tight lower and upper problem-dependent guarantees on both the regret and the number of queries. Interestingly, we prove that there is a fundamental difference between problems with a unique and multiple optimal arms, unlike in the standard multi-armed bandit problem. We also present a new, simple, UCB-style sampling concept, and show that it naturally adapts to the number of optimal arms and achieves tight regret and querying bounds.

Keywords: Multi-Armed Bandits

Acknowledgements

This work was partially funded by the Israel Science Foundation under ISF grant number 2199/20. Yonathan Efroni is partially supported by the Viterbi scholarship, Technion.

1 Introduction

In the stochastic multi-armed bandit (MAB) problem, an agent repeatedly selects actions ('arms') from a finite set and obtains their rewards, generated independently from arm-dependent distributions, with the goal to minimize the regret. This setting has been extensively studied and its extensions are ubiquitous. In particular, many of its variants suggest different feedback models, but once the feedback model is fixed, rewards are always observed through this model.

Nevertheless, such models ignore a key element in sequential decision-making: whatever the feedback model is, asking to observe rewards usually comes at some cost. In some cases, the cost is evident from the setting, e.g., if experimentation is required or the reward is manually labeled by an expert. In other settings, the feedback cost is more subtle. For example, when humans supply the reward feedback, incessantly asking for it may aggravate them. Thus, it is natural to allow agents to decide whether they ask for feedback and design algorithms to ask for feedback with care.

One approach to regulate feedback requests is to limit reward querying by a hard querying-budget constraint [Efroni et al., 2021]. However, this approach is very wasteful, as it encourages exhausting the budget whenever possible, regardless of the querying costs. Instead, we should aim to query only when we gain valuable information and avoid querying otherwise. A goal of our work is to develop an algorithm that performs well, while not violating querying constraints and not querying for reward unnecessarily. The importance of such a goal is illustrated in the following example.

Example 1 (restaurant recommendation problem)**.** Consider a restaurant recommendation problem that learns from user-rankings ('feedback'). Repeatedly asking for rankings will annoy users, so it is natural to cap the ranking requests ('budget') with an initially low cap that gradually increases to allow learning. Yet, even when the agent is allowed to ask for user feedback, such queries harm the user experience, so we should avoid asking for feedback when not needed.

In this work, we study tradeoffs between feedback querying and reward gain in stochastic MAB problems. To do so, we derive both lower and upper bounds to this problem (see Table 1 for a summary of our main results). *Lower bounds.* In Section 2, we study asymptotic and finite-sample lower bounds for reward querying. These reveal fundamental tradeoffs between the querying profile and regret. Interestingly, the lower bounds highlight a clear separation – absent in the usual MAB setting – between problems where the optimal (highest rewarding) arm is unique and problems with multiple optimal arms. *Upper Bounds.* In Section 3, we present and analyze a simple algorithm for efficient reward querying – the BuFALU algorithm – and study its problem-dependent behavior. Notably, unlike prior work, we show that BuFALU naturally adapts to problems with a unique optimal arm and avoids wasting its reward queries unnecessarily. We conclude with a numerical comparison of BuFALU to other alternatives, which highlights its advantages.

1.1 Setting

At each round $t \geq 1$, an agent (*bandit strategy*) plays a single arm $a_t \in [K] \triangleq \{1, \ldots, K\}$. Then, the arm generates a reward $R_t \sim \nu_{a_t}$, of expectation μ_{a_t}, independently at random of other rounds. However, to observe this reward, the agent must actively query ('$q_t = 1$'); otherwise, the reward is not observed ('$q_t = 0$'). We denote the number of times an arm was *played* up to round t by $n_t(a)$, and the number of times it was *queried* by $n_t^q(a)$. For an arm to be queried, it must first be played, and thus $n_t^q(a) \leq n_t(a)$. We similarly denote the total number of queries up to t by $B^q(t)$ and sometimes limit it by a querying budget $B(t)$. The optimal reward is $\mu^* = \max_{a \in [K]} \mu_a$, and the set of optimal arms is $\mathcal{A}_* = \{a : \mu_a = \mu^*\}$. The suboptimality gap of an arm a is $\Delta_a = \mu^* - \mu_a$, and we denote the maximal and minimal gaps by $\Delta_{\max} = \max_a \Delta_a$ and $\Delta_{\min} = \min_{a:\Delta_a > 0} \Delta_a$, respectively. Finally, we define the empirical mean of an arm a, based on *observed* samples up to round t, by $\hat{\mu}_t(a)$. In this work, we evaluate agents by two metrics: the *expected number of reward queries*, $\mathbb{E}[B^q(T)]$, and the *regret*, $\mathrm{Reg}(T) = \mathbb{E}\left[\sum_{t=1}^{T}(\mu^* - \mu_{a_t})\right]$. Particularly, we analyze the effect of reducing the querying on the regret.

1.2 Failures of Existing Approaches

Failure of best-arm identification approach. The simplest approach to incorporate reward querying to sequential decision-making is to query rewards using a best-arm identification (BAI) algorithm [Kalyanakrishnan et al., 2012] and an explore-first-then-exploit scheme. However, this is nontrivial in an anytime setting, where the interaction horizon is unknown. For simplicity, assume that a hard querying budget is given at its whole at the beginning of the interaction ($B(t) = B$ for all $t \geq 1$). For interactions of length $T \gg B$, we would like to use all the budget for exploration, while for $T < B$, we should save rounds for exploitation. In general, the number of exploration rounds must be adaptive, and it is unclear how to do so with off-the-shelf BAI algorithms, especially with time-dependent adversarial budgets.

Failure of confidence-budget-matching (CBM). Alternatively, one might use the confidence-budget matching mechanism [CBM, Efroni et al., 2021], which is designed for anytime settings and time-dependent budgets. However, CBM wastefully expends its querying budget. For example, when $B(t) \gg t$ the algorithm reduces to be UCB1 and queries every round. In contrast, when the optimal arm is unique, $\mathcal{O}\left(\frac{K \ln t}{\Delta_{\min}^2}\right)$ queries are sufficient to identify it. Thus, CBM *should not* query every round (even though allowed), but rather on a logarithmic number of rounds, and fails to conserve queries.

Table 1: Summary of the main results. For simplicity, the lower bounds are stated for Gaussian arms of fixed variance. $\epsilon(t) > 0$ is a sequence that affects the querying and for simplicity is assumed to be strictly decreasing to 0. $n_t^q(a)$ is the number of queries from arm a before time t and $B^q(T)$ is the total queries before T.

Regime	**Scenario**	**Lower Bounds**	**Upper Bounds** (BuFALU)
Asymptotic Bounds	Suboptimal Arms	$\forall a \notin \mathcal{A}^*, \mathbb{E}[n_T^q(a)] = \Omega\Big(\frac{\ln T}{\Delta_a^2}\Big)^{\dagger}$	
	Unique Optimal	$\mathbb{E}[n_T^q(a^*)] = \Omega\Big(\frac{\ln T}{\Delta_{\min}^2}\Big)^{\dagger}$	$E[B^q(T)] = \mathcal{O}\Big(\sum_{a\notin\mathcal{A}^*} \frac{\ln T}{\Delta_a^2} + \frac{\ln T}{\Delta_{\min}^2}\Big)$ $\mathrm{Reg}(T) = \mathcal{O}\Big(\sum_{a\notin\mathcal{A}^*} \frac{\ln T}{\Delta_a}\Big)^{\ddagger}$
	Multiple Optimal	$\mathbb{E}[n_T^q(a^*)] = \omega(\ln T)^{\dagger}$	$E[B^q(T)] = \mathcal{O}\Big(\sum_{a\notin\mathcal{A}^*} \frac{\ln T}{\Delta_a^2} + \lvert\mathcal{A}^*\rvert \frac{\ln T}{\epsilon(T)^2}\Big)$ $\mathrm{Reg}(T) = \mathcal{O}\Big(\sum_{a\notin\mathcal{A}^*} \frac{\ln T}{\Delta_a}\Big)^{\ddagger}$
Scarce Querying Bounds	$\mathbb{E}[n_T^q(a^*)] \le B_a(T)$	$\mathrm{Reg}(T) = \Omega\Big(\sum_{a\notin\mathcal{A}^*} \frac{\Delta_a}{K} B_a^{-1}\Big(\frac{1}{K\Delta_a^2}\Big)\Big)^{\star}$	$B_a(t) \triangleq \mathbb{E}[n_T^q(a)] = \mathcal{O}\Big(\frac{\ln T}{\epsilon(T)^2}\Big)$ $\mathrm{Reg}(T) = \mathcal{O}\Big(\sum_{a\notin\mathcal{A}^*} \Delta_a B_a^{-1}\Big(\frac{\ln T}{\Delta_a^2}\Big)\Big)^{\S}$

† Theorem 2.1 ‡ Theorem 3.1, $\bar{N}(T,\Delta)$ term * Proposition 2.2 § Theorem 3.1, $L_\epsilon(T,\Delta)$ term

2 Lower Bounds

For simplicity, we state our results for Gaussian arms of unit variance, but they can be extended for general distributions.

2.1 Asymptotic Lower Bounds

Probably the most common assumption for a bandit strategy is *consistency*, namely, asymptotically sub-polynomial regret.

Definition 1. A bandit strategy is called *consistent* if for any bandit instance, any suboptimal arm $a \notin \mathcal{A}_*$ and any $\alpha \in (0,1]$ it holds that $n_T(a) = o(T^\alpha)$.

Theorem 2.1. *Let* $n_\infty^q(a) = \liminf_{T\to\infty} \frac{\mathbb{E}[n_T^q(a)]}{\ln T}$ *for any* $a \in [K]$. *If the bandit strategy is consistent, then:*

1. *For any suboptimal arm* $a \notin \mathcal{A}_*$, $n_\infty^q(a) \ge 2/\Delta_a^2$.
2. *Assume that* a^* *is the unique optimal arm and denote the maximal suboptimal reward by* $\mu^s = \max_{a\neq a^*} \mu_a$. *Then* $n_\infty^q(a^*) \ge 2/\Delta_{\min}^2$. *Moreover, for all suboptimal arms* $a \notin \mathcal{A}_*$ *it holds that* $n_\infty^q(a) + n_\infty^q(a^*) \ge \min\{8/\Delta_a^2, 2/\Delta_{\min}^2\}$.
3. *If there are at least two optimal arms, then* $n_\infty^q(a) = \infty$ *for some optimal arm* $a \in \mathcal{A}_*$.

These bounds are *asymptotic*; for large enough T, an action a is roughly queried $n_\infty^q(a) \cdot \ln T$ times. Also, recall that $n_t^q(a) \le n_t(a)$, so all results also hold for the number of plays. The theorem is divided into three parts. The first part is a natural extension of the classical lower bound for MABs [Lai and Robbins, 1985] and emphasizes that it is not enough to sufficiently *play* suboptimal arms, but we rather must sufficiently *query* them. The second and third parts discuss the querying requirements from *optimal arms* when there is a unique or multiple optimal arms, respectively. This comes in stark contrast to the classical lower bounds that disregard querying, as playing optimal arms does not incur regret and can thus be ignored. When there is a *unique optimal arm* a^*, the result first states that it must be distinguished from the highest suboptimal arm a. Yet, the second part of the theorem implies that by itself, this does not suffice. Instead, for any suboptimal arm a, both a and a^* must be sufficiently queried to *separate* them; namely, identifying that a^* is better than a. Concretely, for near-optimal arms, both a and a^* should roughly be sampled up to a precision of $\Delta_a/2$, which effectively separates their confidence intervals. Finally, the last part of Theorem 2.1 treats problems with *multiple optimal arms* and prove that in the Gaussian case, they must always be queried super-logarithmically.

2.2 Lower Bounds for Scarce Querying

Finally, we discuss the best possible performance in the limit of scarce querying.

Definition 2. A strategy is called *better-than-uniform* if for any bandit instance and any $T \ge 1$, $\sum_{a\in\mathcal{A}_*} \mathbb{E}[n_T(a)] \ge \frac{|\mathcal{A}_*|}{K} T$.

Proposition 2.2. *Denote* $B_a(t) = \mathbb{E}[n_t^q(a)]$; *also, let* $B_a^{-1}(x) = \sup\{t \in \mathbb{N} : B_a(t) \le x\}$ *for* $x \ge B(1)$ *and otherwise* $B_a^{-1}(x) = 0$. *Then, for any instance, better-than-uniform strategy and* $T \ge \max_{a\notin\mathcal{A}_*} B_a^{-1}\Big(\frac{1}{4K\Delta_a^2}\Big)$, *we have* $\mathrm{Reg}(T) \ge \sum_{a\notin\mathcal{A}_*} \frac{\Delta_a}{2K} B_a^{-1}\Big(\frac{1}{4K\Delta_a^2}\Big)$.

We remark that the same result naturally holds if we only have bounds on the number of queries ($\mathbb{E}[n_t^q(a)] \le B_a(t)$ for some positive nondecreasing $B_a(t)$), e.g., per-arm querying budget. Importantly, when arms are queried sub-polynomially ($B_a(t) \approx \ln T$ but do not depend on Δ_a), this might lead to an exponential lower bound in the gaps.

Algorithm 1 Budget-Feedback Aware Lower Upper Confidence Bound (BuFALU)

Define: $UCB_t(a) = \hat{\mu}_{t-1}(a) + \sqrt{\frac{3\ln t}{2n^q_{t-1}(a)}}$, $LCB_t(a) = \hat{\mu}_{t-1}(a) - \sqrt{\frac{3\ln t}{2n^q_{t-1}(a)}}$ ('Hoeffding')
Play each arm once and ask for feedback ($q_t = 1$); observe R_t and update $n^q_t(a) = 1, \hat{\mu}_t(a_t)$
for $t = K+1, ..., T$ **do**
    Observe $\epsilon(t) \geq 0$
    Set $l_t \in \arg\max_a LCB_t(a)$, $u_t = \in \arg\max_{a \neq l_t} UCB_t(a)$ and $c_t \in \arg\max_{a \in \{u_t, l_t\}} CI_t(a)$
    if $UCB_t(u_t) \leq LCB_t(l_t)$ or $UCB_t(c_t) - LCB_t(l_t) \leq \epsilon(t)$ **then**
        Play $a_t = l_t$ and do not ask for feedback ($q_t = 0$)
    else
        Play $a_t = c_t$ and ask for feedback ($q_t = 1$) ; observe R_t and update $n^q_t(a_t), \hat{\mu}_t(a_t)$
    end if
end for

3 Upper Bounds

For simplicity, we focus on problems with rewards bounded in $[0, 1]$, but our results hold any confidence intervals that follow some mild assumptions (including Bernstein bounds).

To be more feedback-conscious, we adopt a confidence-based approach. Namely, we assume that each arm a is equipped with a confidence interval of width $CI_t(a) = UCB_t(a) - LCB_t(a) \geq 0$, such that with a sufficiently high probability, $\mu_a \in [LCB_t(a), UCB_t(a)]$. In general, by controlling the widths of confidence intervals, we can identify good actions and bound their suboptimality. However, narrowing these intervals requires reward samples, which are limited at our setting.

Our approach carefully shrinks the confidence intervals, with the goal to separate a unique optimal arm (whenever such exists), while controlling the number of reward queries. We divide our algorithm into two steps – action-selection and confidence-control. For action-selection, inspired by the lower bounds, we aim to *separate* the confidence interval of a unique optimal arm a^* from all other arms. If we manage to do so, we can safely declare that $a^* = \arg\max_a LCB_t(a) \triangleq l_t$ and play l_t without reward querying. Letting $u_t \in \arg\max_{a \neq l_t} UCB_t(a)$, this case happens if $UCB_t(u_t) \leq LCB_t(l_t)$. When this condition does not hold, we separate u_t from l_t by actively shrinking their confidence by playing (and querying) the arm with the wider confidence interval, $c_t \in \arg\max_{a \in \{u_t, l_t\}} CI_t(a)$.

Yet, this separation might be costly in queries and is impossible if there are multiple optimal arms. To avoid this, we regulate the number of queries by controlling the widths of the confidence intervals. Let $\epsilon(t)$ be a variable that controls the widths of the confidence intervals. Then, we let our algorithm to query for reward only if $UCB_t(c_t) - LCB_t(l_t) > \epsilon(t)$. We allow $\epsilon(t)$ to be externally controlled (be chosen adversarially), to allow external effects on the querying rule (e.g., querying budget), but it can also be chosen the algorithm designer. Lastly, if $UCB_t(c_t) - LCB_t(l_t) \leq \epsilon(t)$, we revert to playing the 'default action' l_t without reward querying. We can show that doing so is safe, in a sense that $\Delta_{l_t} \leq \epsilon(t)$. Combining both playing and querying schemes leads to the BuFALU algorithm, depicted in Algorithm 1.

Before stating the regret and querying guarantees of BuFALU, we present two fundamental quantities on which our bounds will depend: $L_\epsilon(T, \Delta) = \sum_{t=1}^{T} \mathbb{1}\{\epsilon(t) \geq \Delta\}$ and $\bar{N}(T, \Delta) = \max_{t \in [T]} \frac{6\ln t}{\max\{\Delta^2, \epsilon^2(t)\}}$. The first quantity $L_\epsilon(T, \Delta)$ counts the number of rounds that $\epsilon(t)$ exceeds a fixed confidence level Δ until time T. A notable case is when $\epsilon(t)$ is nonincreasing, and then $L_\epsilon(T, \Delta)$ can be conveniently bounded by $L_\epsilon(T, \Delta) \leq \epsilon^{-1}(\Delta) \triangleq \sup\{t \geq 1 : \epsilon(t) \geq \Delta\}$. The second quantity $\bar{N}(T, \Delta)$ represents the maximal number of queries required to shrink the confidence intervals $\max\{\epsilon(t), \Delta\}$. Importantly, if $\epsilon(t) = \sqrt{6K \ln t / B(t)}$ for a nondecreasing budget $B(t)$, see that $\bar{N}(T, \Delta) = \frac{6\ln T}{\max\{\Delta^2, \epsilon^2(T)\}}$. Then, we get that $\bar{N}(T, 0) \leq B(T)/K$, so this term can ensure that the budget constraints are never violated.

We now state problem-dependant and problems independent performance bounds for BuFALU. The bounds are stated for oblivious $\epsilon(t)$, but also hold for an adaptive one (by taking an expectation on all bounds).

Theorem 3.1. *Assume that the rewards are bounded in $[0, 1]$. Also, let $T \geq 1$ and assume that $\{\epsilon(t)\}_{t \in [T]}$ is some nonnegative sequence. Then, when running Algorithm 1, for all $a \in [K]$, it holds that $n^q_T(a) \leq \bar{N}(T, 0) + 1$. Moreover, the following hold:*

1. *If there are multiple optimal arms ($|\mathcal{A}_*| > 1$), then*

$$\mathrm{Reg}(T) \leq \sum_{a \notin \mathcal{A}_*} \Delta_a \big(\bar{N}(T, \Delta_a) + L_\epsilon(T, \Delta_a)\big) + 3K\Delta_{\max}, \qquad \mathbb{E}[B^q(T)] \leq \sum_{a \notin \mathcal{A}_*} \bar{N}(T, \Delta_a) + |\mathcal{A}_*|\bar{N}(T, 0) + 3K.$$

2. *If the optimal arm a^* is unique, then*

$$\mathrm{Reg}(T) \leq \sum_{a \neq a^*} \Delta_a \left(\bar{N}\left(T, \frac{\Delta_a}{2}\right) + L_\epsilon(T, \Delta_a)\right) + 3K\Delta_{\max}, \qquad \mathbb{E}[B^q(T)] \leq \sum_{a \neq a^*} \bar{N}\left(T, \frac{\Delta_a}{2}\right) + \bar{N}\left(T, \frac{\Delta_{\min}}{2}\right) + 3K.$$

Proposition 3.2. *For any sequence $\epsilon(t) \geq 0$ and any $T \geq 1$, it holds that* $\mathrm{Reg}(T) \leq 4\sqrt{6KT\ln T} + \sum_{t=1}^{T} \epsilon(t) + 3K\Delta_{\max}$.

3.1 Discussion and Comparisons

Given a positive nondecreasing (possibly adversarial) querying budget $B(t)$, one can choose $\epsilon(t) = \sqrt{6K \ln t / B(t)}$. Then, Theorem 3.1 provides strict (almost sure) querying guarantee of $B(T) + K$ queries, while Proposition 3.2 achieves a regret bound of $\mathrm{Reg}(T) = \mathcal{O}\left(\sqrt{KT\ln T} + \sum_{t=1}^{T} \sqrt{\frac{K \ln T}{B(T)}}\right)$. These are the same guarantees achieved by CBM-UCB [Efroni et al., 2021]. On the other hand, BuFALU also enjoys problem dependent regret and querying guarantees. In particular, when the optimal arm is unique, the number of queries is logarithmic – usually far less than the allocated budget.

Next, we compare Theorem 3.1 to the lower bounds of Section 2, and for simplicity, assume that $\epsilon(t)$ is nonincreasing. First, the per-arm query bound of $B_a(t) = \mathcal{O}\big(\ln t / \epsilon^2(t)\big)$ implies that $\epsilon^{-1}(\Delta_a) \approx B_a^{-1}\big(\ln T / \Delta_a^2\big)$. Then, by bounding $L_\epsilon(t, \Delta) \leq \epsilon^{-1}(\Delta)$, the regret term of $\Delta_a L_\epsilon(T, \Delta_a)$ corresponds with the lower bound of Proposition 2.2. Moreover, for any $\epsilon(t) \searrow 0$, the regret is logarithmic, and all suboptimal arms are logarithmically sampled. This matches the asymptotic lower bound up to absolute constants. However, if there are multiple optimal arms, they will be queried $\mathcal{O}\left(\frac{\ln T}{\epsilon^2(T)}\right)$ times each – super-logarithmically. Also, notice that the suboptimal queries depend on $\bar{N}(T, \Delta_a) \leq \frac{6 \ln T}{\Delta_a^2}$ when there are multiple optimal arms, and in contrast, when the optimal arm is unique, the bounds depend on $\bar{N}(T, \Delta_a/2) \approx 4\bar{N}(T, \Delta_a)$. This factor is similar to the one in Theorem 2.1, part 2, and is related to the separation of the optimal arm from suboptimal ones.

3.2 Numerical Illustration

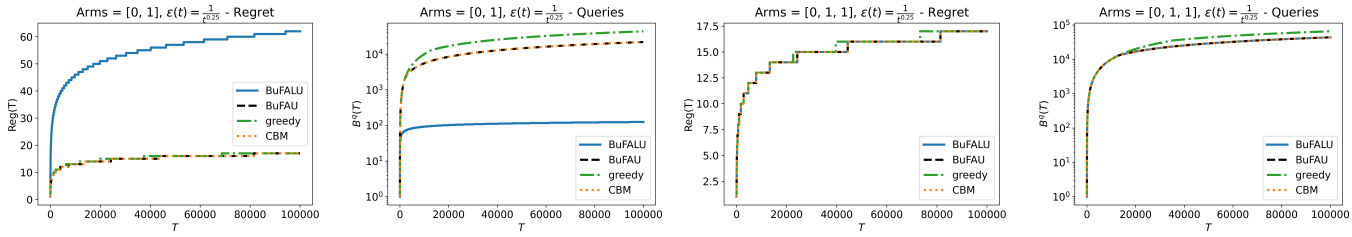


Figure 1: Regret and queries on two deterministic instances. In the two left plots there is a unique optimal arm, while in two right ones there are two. All evaluations used $\epsilon(t) = t^{-1/4}$ and tested one seed (deterministic problems).

We illustrate the behavior of BuFALU in the presence of a unique or multiple optimal arms on two deterministic MAB instances (see Figure 1). In the first instance, the optimal arm is unique (with $\mu^* = 1$) and there exists a single suboptimal arm (with $\mu_1 = 0$). The second instance is the same, except for an additional optimal arm. We compare BuFALU to a few natural baselines. The first, called BaFAU, uses the same querying rule as BuFALU but sets $c_t \in \arg\max_a UCB_t(a)$. The second is CBM-UCB [Efroni et al., 2021], whose problem-independent bounds are similar to BuFALU. The final baseline is a greedy algorithm that receives a total budget as guaranteed by Theorem 3.1. If the budget is not exhausted, it plays (and queries) the maximal UCB; otherwise, it plays (without querying) the arm with the maximal empirical mean. Notice that Theorem 3.1 guarantees that BuFALU cannot query any arm more than $\sim 20,000$ times. We remark that evaluations in stochastic 5-armed problems yield similar insights, but are not presented due to space limitations.

One immediate conclusion is that in these simple instances, all baseline algorithms behave roughly the same, where the only difference is that the greedy baseline completely exhausts its querying budget while all other baselines only exhaust it for optimal arms. Moreover, the simulated behavior of BuFALU validates the characterization of Theorem 3.1: when there are multiple optimal arms, BuFALU uses all available budget to query them and achieves the same regret and querying performance as the baselines. In contrast, when the optimal arm is unique, BuFALU only sparingly asks for feedback ($\sim 1/150$, compared to the baselines), but its regret is larger by a factor of 4.

References

Yonathan Efroni, Nadav Merlis, Aadirupa Saha, and Shie Mannor. Confidence-budget matching for sequential budgeted learning. *arXiv preprint arXiv:2102.03400*, 2021.

Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer, and Peter Stone. Pac subset selection in stochastic multi-armed bandits. In *ICML*, volume 12, pages 655–662, 2012.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6 (1):4–22, 1985.

Accounting for the Sequential Nature of States to Learn Features for Reinforcement Learning

Nathan Michlo, Devon Jarvis, Richard Klein, Steven James
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
{nathan.michlo1, devon.jarvis1}@students.wits.ac.za
{richard.klein,steven.james}@wits.ac.za

Abstract

In this work, we investigate the properties of data that cause popular representation learning approaches to fail. In particular, we find that in environments where states do not significantly overlap, variational autoencoders (VAEs) fail to learn useful features. We demonstrate this failure in a simple gridworld domain, and then provide a solution in the form of metric learning. However, metric learning requires supervision in the form of a distance function, which is absent in reinforcement learning. To overcome this, we leverage the sequential nature of states in a replay buffer to approximate a distance metric and provide a weak supervision signal, under the assumption that temporally close states are also semantically similar. We modify a VAE with triplet loss and demonstrate that this approach is able to learn useful features for downstream tasks, without additional supervision, in environments where standard VAEs fail.

Keywords: representation learning, unsupervised learning, metric learning, reinforcement learning

1 Introduction

A fundamental challenge in machine learning is to discover useful representations from high-dimensional data, which can then be used to solve subsequent tasks effectively. Recently, deep learning approaches have showcased the ability of neural networks to extract meaningful features from high-dimensional inputs for tasks in both the supervised and reinforcement learning (RL) setting [1, 2]. While these learnt representations have obviated the need for manual feature selection or preprocessing pipelines, they are often not semantically meaningful, which can negatively impact downstream task performance [3]. Prior work has therefore argued that it is desirable to learn a representation that is *disentangled*. While there is no consensus on what constitutes a disentangled representation, it is generally agreed that such a representation should be factorised so that each latent variable corresponds to a single explanatory variable responsible for generating the data [4]. For example, a single image from a video game may be represented by latent variables governing the x and y positions of the player, enemies and collectable items.

In deep RL, there are two general approaches to feature learning. The first is to implicitly discover features by simply training an agent to solve a given task, relying on a neural network function approximator and stochastic gradient descent to uncover useful features. An alternative approach is to explicitly learn features through some auxiliary objective. One such method is to use variational autoencoders (VAEs) [5], which are trained on collected data to learn a lower-dimensional representation capable of reconstructing the given input. VAEs have been shown to produce disentangled representations when trained on synthetically generated data, although there is still no explicit reason why these representations should align with generative factors in the data.

In this work, we investigate the correspondence between the VAE reconstruction loss and learnt distances in the latent space. We demonstrate the ineffectiveness of state-of-the-art models by designing a simple visual gridworld domain over which VAE-based frameworks fail to learn useful features for a downstream RL task. We overcome this limitation by utilising the sequential nature of an RL replay buffer along with metric learning to guide representations learnt by VAEs. Furthermore, we design a disentangled metric learning approach which further improves learnt representations for downstream tasks.

2 Background

We model an agent acting in an environment by a Markov decision processes (MDP) $\mathcal{M} = \langle S, A, T, R, \gamma \rangle$ where (i) S is the state space; (ii) A is the set of actions; (iii) $T(s, a, s')$ describes the transition dynamics, specifying the probability of arriving in state s' after action a is executed from s; (iv) $R(s, a)$ specifies the reward for executing action a in state s; and (v) $\gamma \in [0, 1)$ is a discount factor. The agent's aim is to compute a policy π that maps from states to actions and optimally solves the task. Given a policy π, RL algorithms often estimate a value function $v^\pi(s) = \mathbb{E}_\pi\big[\sum_{i=0}^{\infty} \gamma^i R(s_i, a_i) | s_0 = s\big]$, representing the expected return obtained following π from state s. The *optimal* policy π^* is the policy that obtains the greatest expected return at each state: $v^{\pi^*}(s) = v^*(s) = \max_\pi v^\pi(s)$ for all $s \in S$. A related quantity is the action-value function, $q^\pi(s, a)$, which defines the expected return obtained by executing a from s, and thereafter following π. Similarly, the optimal action-value function is given by $q^*(s, a) = \max_\pi q^\pi(s, a)$ for all states s and actions a [6].

For high-dimensional or continuous state spaces, representing the value function precisely is intractable. Instead, the value function is normally approximated through a parameterised function $v^\pi(s) \approx \hat{v}(s; \mathbf{w})$, where $\mathbf{w}$ is some parameter vector to be learned; in deep RL, these parameters are the weights of a neural network that can be trained to approximate the value function. Since these neural networks require significant amounts of data to train, one way to improve sample efficiency is to use a *replay buffer*, where old experience data is stored in memory and reused to update the network as it continues to interact with the environment.

2.1 Representation Learning

Assume a dataset $\mathcal{X} = \{\boldsymbol{x}^{(0)}, ..., \boldsymbol{x}^{(n)}\}$ is a set of independent and identically distributed (i.i.d)[1] observations $\boldsymbol{x} \in \mathbb{R}^{\mathrm{N}}$, generated by some random process involving an unobserved random variable $\boldsymbol{z} \in \mathbb{R}^{\mathrm{D}}$ of lower dimensionality $\mathrm{D} \ll \mathrm{N}$. Additionally, the true *prior distribution* $\boldsymbol{z} \sim p_*(\boldsymbol{z})$ and true *conditional distribution* $\boldsymbol{x} \sim p_*(\boldsymbol{x}|\boldsymbol{z})$ are unknown. Variational autoencoders (VAEs) aim to learn this generative process. Unlike autoencoders (AEs), which consist of an encoder $f_{\boldsymbol{\phi}}(\boldsymbol{x}) = \boldsymbol{z}$ and decoder $g_{\boldsymbol{\theta}}(\boldsymbol{z}) = \hat{\boldsymbol{x}}$ with weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, VAEs instead construct a probabilistic encoder by using the output from the encoder or inference model to parameterise approximate posterior distributions $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. The approximate posterior is then sampled from during training to obtain representations $\boldsymbol{z}$, which are then decoded using the generative model to obtain reconstructions $\hat{\boldsymbol{x}} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$.

A *factorised Gaussian encoder* [5] is commonly used to model the posterior using a multivariate Gaussian distribution with diagonal covariance $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \ \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}))$, with the prior given by the multivariate normal distribution

[1] In RL, a replay buffer is not i.i.d, as data is generated by a sequential decision process. This can be mitigated via random sampling.

$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0},\ \mathbf{I})$, with a mean of $\mathbf{0}$ and diagonal covariance $\mathbf{I}$. To enable backpropagation, the reparameterisation trick in Equation (1) is used to sample from the posterior distribution during training by offsetting the distribution means by scaled noise values.

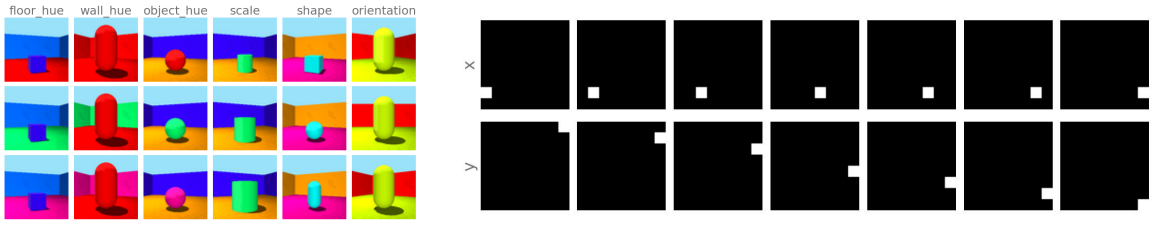
$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}); \quad \boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \odot \boldsymbol{\epsilon} \tag{1}$$

VAEs maximise the evidence lower bound (ELBO) by minimising the loss given by Equation (2). VAE-based approaches often make slight modifications to this loss but generally the terms can still be grouped into regularisation and reconstruction components. The regularisation term constrains the representations learnt by the encoder, while the reconstruction term improves the outputs of the decoder so that they better match the inputs to the encoder. These terms usually conflict in practice, strong regularisation leads to worse reconstructions but often better disentanglement [4, 7].

$$\mathcal{L}_{\text{rec}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]; \quad \mathcal{L}_{\text{reg}}(\boldsymbol{x}) = -D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})\right); \quad \mathcal{L}_{\text{VAE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \mathcal{L}_{\text{rec}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \mathcal{L}_{\text{reg}}(\boldsymbol{x}) \tag{2}$$

3 A Motivating Gridworld Environment

Before investigating representation learning in RL, we first consider the simpler unsupervised learning setting, in which VAEs and their variants are primarily developed through benchmarks over synthetic data. The 3D Shapes dataset [8] in Figure 1a contains observations of shapes fixed in the centre of the image with progressively changing attributes or factors such as size and colour. If, as humans, we are given unordered observations from a traversal along the size factor of 3D Shapes, it would be easy to order these observations using a perceived increase or decrease in the size of the shape. We might even say that the shapes in the images overlap by different amounts. We may consider shapes that are closer in size to possess more overlap, and therefore also consider them to be closer together in terms of distance. In contrast, consider a simple gridworld environment (Figure 1b) where an agent moves to adjacent cells in any of the four cardinal directions. The agent is a square of size 8×8 pixels and each action translates it 8 pixels in a given direction. Note that in this case, there is no overlap between the agent in any two images because of the distance it moves.



(a) 3D Shapes [8]

(b) Simple gridworld domain with pixel-based states

Figure 1: (a) A common synthetic dataset—images are generated by smoothly varying ground-truth factors such as floor hue and object shape. (b) Gridworld environment whose underlying factors are the agent's xy-position.

It is therefore natural to ask whether a lack of overlap is problematic. To answer this question, let $\boldsymbol{x}$ be an observation and $\boldsymbol{y}$ be the underlying factors that generated it. We define the *ground-truth distance* between pairs of observations $\{\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}\}$ as the Manhattan distance between the observations' underlying factors: $\mathrm{d}_{\text{gt}}(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \|\boldsymbol{y}^{(a)} - \boldsymbol{y}^{(b)}\|_1$.[2] Similarly, we define the *perceived distance* as the difference between the two observations in pixel space as measured by the model's reconstruction loss function (usually MSE): $\mathrm{d}_{\text{vis}}(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \mathcal{L}_{\text{rec}}(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)})$.[3]

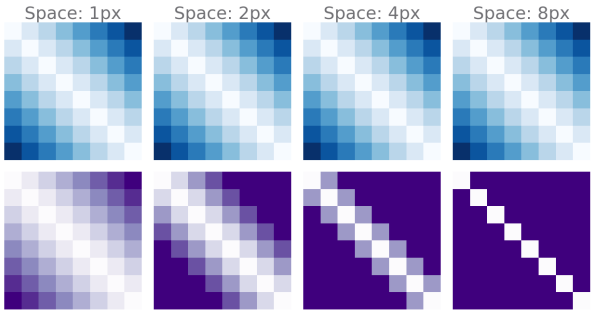
To investigate the above question, we modify the environment so that the agent moves in smaller increments. For each step size s, we generate all possible image states and store them in a buffer.[4] As s decreases, the probability that the agent position overlaps in any two randomly sampled images increases. We next visualise ground-truth and perceived distances for each step size setting, with the results given by Figure 2a. The results show a clear relationship between the two distance measures—at very small step sizes (with high probability of overlap) there is almost perfect correspondence.

We further probe this relationship by using the data buffers as an unsupervised dataset. We make the problem harder by converting it to a multiagent environment which increases the number of states—three objects are now each described by their xy-positions. We train a β-VAE [7] and the state-of-the-art weakly-supervised Ada-GVAE [9]. The β-VAE scales the VAE regularisation term with a coefficient $\beta > 0$, while the Ada-GVAE encourages axis alignment and shared latent variables between pairs of observations. This is achieved by averaging together latent distributions between observation pairs that are estimated to remain unchanged when the KL divergence is below some threshold. To evaluate if the

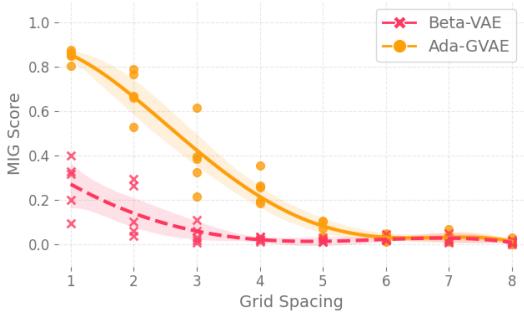
[2] The notation $\boldsymbol{a}^{(i)}$ is alternative notation for either a named variable or the i^{th} element of the ordered set $\mathcal{A}$.

[3] We use the term increased *perceived overlap* as a synonym for lower *perceived distance* between neighbouring states.

[4] We modify the size of the environment to ensure the size of the buffer remains constant regardless of the step size.



(a) Top Row: Manhattan ground-truth distance matrices over factor traversals. Bottom Row: Pixel-wise perceived distance matrices between observations over factor traversals. Left to Right: Step size increases from 1px to 8px. Smaller values result in more overlap in the data space, which leads to higher a probability of being able to find an ordering along a factor traversal.



(b) Regression plot of increasing domain spacing versus MIG score (higher is better). As the β-VAE (dashed line) and Ada-GVAE (solid line) are trained with decreasing (left to right) levels of overlap, their disentanglement performance worsens.

Figure 2: Varying spacing of valid locations in gridworld domain affects overlap as perceived by an VAE.

representations are disentangled, we use the Mutual Information Gap (MIG) [10] The results in Figure 2b indicate that as the spacing decreases and overlap is introduced, the disentanglement performance improves as perceived pixel-wise distances better correspond to latent and ground-truth distances.

4 Triplet Loss

VAEs, therefore, appear to learn latent distances over the data that correspond to their reconstruction loss. However, if these distances are not useful, we may be able to instead guide the learning process by introducing *metric* or *similarity* learning. A common approach is triplet loss, which makes use of three observations $\mathcal{X}_{\text{triple}} = (\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(p)}, \boldsymbol{x}^{(n)})$. These are sampled using supervision such that the *anchor* $\boldsymbol{x}^{(a)}$ is considered closer to *positive* $\boldsymbol{x}^{(p)}$ than it is to the *negative* $\boldsymbol{x}^{(n)}$. The intuition is that corresponding representations $\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(p)}, \boldsymbol{z}^{(n)}$ output by a network should satisfy the original anchor-positive and anchor-negative distance constraints: $d(\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(p)}) < d(\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(n)})$. We construct the β-TVAE as three parallel versions of the same β-VAE augmented by the soft-margin formulation of triplet loss, where $\alpha > 0$ is the scaling factor:

$$\mathcal{L}_{\beta\text{TVAE}} = \underbrace{\alpha \ln\left(1 + \exp\left(\mathrm{d}(\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(p)}) - \mathrm{d}(\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(n)})\right)\right)}_{\text{Triplet loss}} + \frac{1}{3} \sum_{\boldsymbol{x} \in \mathcal{X}_{\text{triple}}} \mathcal{L}_{\beta\text{VAE}}(\boldsymbol{x}) \tag{3}$$

4.1 Adaptive Triplet Loss

Standard triplet loss provides no direct pressure to learn factored representations; for example the x and y factors may be encoded with some arbitrary rotation in the latent space. To fix this, we adapt the Ada-GVAE[9] to construct Ada-Triplet and the Ada-TVAE. These adaptive methods encourage partially shared representations such that differences between observations are encoded in subsets of latent units. Shared latent units are estimated as those whose distances $\delta_i = |z_i^{(a)} - z_i^{(n)}|$ are less than half way between the maximum and minimum: $\delta_i < \frac{1}{2}(\min_i \delta_i + \max_i \delta_i)$. To encourage factored latents, we elementwise multiply $\odot$ arguments of the anchor-negative triplet term by the weight vector $\boldsymbol{w} = (\omega_1, ..., \omega_{\mathrm{D}})$. Shared latents are weighted less $\omega_i \in (0, 1)$ and non-shared units $\omega_i = 1$ remain unchanged:

$$\mathcal{L}_{\text{Ada}-\text{Triplet}} = \ln\left(1 + \exp\left(\mathrm{d}(\boldsymbol{z}^{(a)}, \boldsymbol{z}^{(p)}) - \mathrm{d}(\boldsymbol{\omega} \odot \boldsymbol{z}^{(a)}, \boldsymbol{\omega} \odot \boldsymbol{z}^{(n)})\right)\right) \tag{4}$$

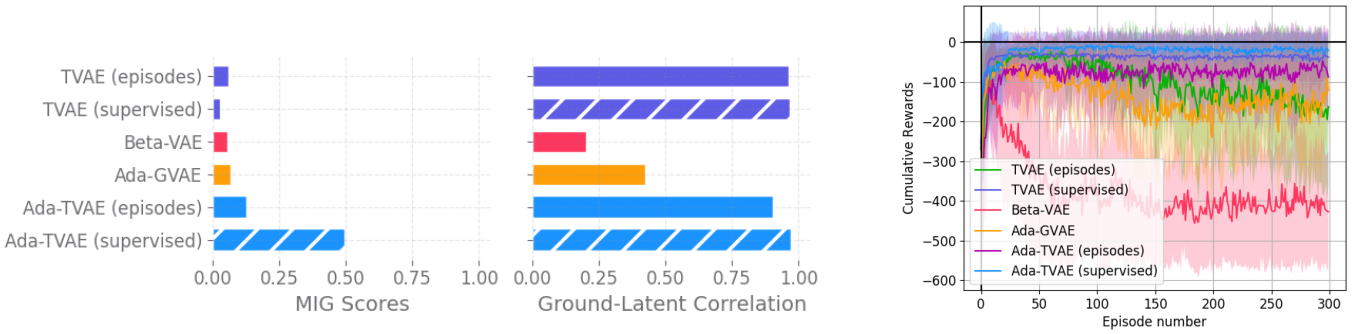
4.2 Using the Replay Buffer for Metric Learning and Feature Extraction

Typically, triplet loss is used in the supervised learning case, where the anchor, positive and negative samples are known. However, no such signal exists in the RL setting. To overcome this, we leverage the sequential nature of the problem and assume that *states closer together in time are also on average closer in terms of their ground-truth factors.*

Given a replay buffer $\mathcal{B} = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots\}$, we use this assumption to sample some anchor state s_a, a positive s_p and a negative s_n such that $n > p > a$. To test the effectiveness of this assumption, we execute a uniformly random policy to populate a replay buffer. We train a β-VAE and Ada-GVAE as baselines, which we compare against the β-TVAE and Ada-TVAE from the previous sections. As a sanity check we also train a fully supervised β-TVAE and Ada-TVAE on

the underlying ground-truth factors of the environment (the agent's xy-position) so that the anchor-positive ℓ_1 distance is always less than the anchor-negative. The results in Figure 3a demonstrate that triplet-based models significantly outperform the baselines by learning better features.

Finally, we test the effect of using the encoders of these learnt models as feature extractors in an RL setting. We freeze the weights of the previously trained encoders, and apply deep Q-learning [2] over these features to solve the task of reaching the top left corner from the bottom right (rewards of -1 on all timesteps). Results in Figure 3b demonstrate that features from the adaptive triplet loss model are the most useful for the RL task; however, this is well approximated by sampling triplets using only the ordering of states from the replay buffer. Both triplet methods outperform the baselines; however, adaptive triplet encourages factored representations which are better features for the downstream RL task.



(a) (b)

Figure 3: (a) Different VAE frameworks trained to extract features. The plot shows the average rank correlation between the ground-truth distances and corresponding distances between these learnt features. Striped bars are models trained using ground-truth distances, while solid bars use our approach of sampling sequentially from the replay buffer. (b) Downstream RL training rewards using previously learnt features. Mean and standard deviation are given over 30 runs.

5 Conclusion

We identified a shortcoming of VAE-based approaches in domains where there is little or no variation in perceived distances or overlap. This may be inherent in the environment, or arise inadvertently due to design decisions. For example, if frame skipping [2] or high-level skills are used, there may be no perceived overlap between successive states in the buffer. We overcome this problem by incorporating the inherent temporal information present in an episode through the simple use of metric learning. Furthermore, we improve the performance by enabling representation disentanglement by designing an adaptively weighted version of triplet loss. Leveraging the sequential nature of the replay buffer to perform metric learning may open up interesting new avenues for future feature learning approaches in RL.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, and O. Bachem, "Challenging common assumptions in the unsupervised learning of disentangled representations," in *International Conference on Machine Learning*, 2019, pp. 4114–4124.

[4] C. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in β-VAE," in *Workshop on Learning Disentangled Representations at the 31st Conference on Neural Information Processing Systems*,, 2017.

[5] D. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations*, 2014.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2016.

[8] C. Burgess and H. Kim, "3D shapes dataset," https://github.com/deepmind/3dshapes-dataset/, 2018.

[9] F. Locatello, B. Poole, G. Rätsch, B. Schölkopf, O. Bachem, and M. Tschannen, "Weakly-supervised disentanglement without compromises," in *International Conference on Machine Learning*, 2020, pp. 6348–6359.

[10] R. Chen, X. Li, R. Grosse, and D. Duvenaud, "Isolating sources of disentanglement in variational autoencoders," in *Advances in Neural Information Processing Systems*, 2018, pp. 2615–2625. 423

Quantifying the latent-cause inference process and its relationship with mental health symptoms

Dan-Mircea Mirea
Department of Psychology
Princeton University
Princeton, NJ 08540
dmirea@princeton.edu

Yeon Soon Shin
Department of Psychology
Yale University
New Haven, CT 06511
yeonsooncshin@gmail.com

Sofiya Yusina
Princeton Neuroscience Institute
Princeton University
Princeton, NJ 08540
syusina@princeton.edu

Sarah DuBrow
Department of Psychology
University of Oregon
Eugene, OR 97403
sdubrow@uoregon.edu

Yael Niv
Princeton Neuroscience Institute
Princeton University
Princeton, NJ 08540
yael@princeton.edu

Abstract

Research suggests that humans learn by grouping experiences into clusters or latent causes, which help organize context-dependent state representations in reinforcement learning. Latent-cause inference supports optimal generalization – learning can be applied to all situations that are deemed to be similarly generated. In contrast, as learning state representations is so fundamental to adaptive behavior, suboptimal latent-cause inference could underlie psychopathological cognitive deficits, such as overgeneralization of negative events in depression or incoherent world models in schizophrenia. To test this hypothesis, we quantified individual differences in latent-cause inference using a novel task and a Bayesian inference model, and correlated these with self-reported psychiatric symptoms. N = 565 online participants assigned abstract visual stimuli to clusters. We fitted the latent-cause inference model hierarchically to task behavior to estimate individual-level parameters: the tendency to start a new latent cause, the temporal decay of existing causes, and priors for the size or variability of causes. Participants also answered psychiatric symptom questions as part of a larger sample (N = 1234), and we used exploratory factor analysis to derive transdiagnostic factors (i.e., clusters of symptoms that co-occur) and factor scores for each participant. Higher scores on the 'Schizotypy-disinhibition-mania' factor were correlated with an increased tendency to create new clusters, as well as a tendency to create larger, overlapping clusters. This behavior was also correlated with the 'Obsessive-compulsivity' and 'Positive affectivity' factor, albeit less strongly. These results establish our task as a means to quantify latent-cause inference and relate this process to state-space learning, and suggest that a tendency to group experiences into a large number of overlapping clusters might contribute to a broad range of clinical conditions, from schizophrenia to ADHD.

Keywords: computational psychiatry; generalization; latent-cause inference; state representation

Acknowledgements

This work is dedicated to Dr. Sarah DuBrow, who passed away in February 2022 and without whom none of it would have happened. Sarah, thank you for being the most wonderful scientist and human being - we miss you dearly!

This research was supported by NIMH grant R21MH120798.

1 Introduction

Forming internal representations of the outside world, such as state representations in reinforcement learning, is fundamental to adaptive behavior. One of the ways in which we form these representations and carve up the world for effective learning is by incrementally grouping experiences into clusters. Because the world is full of statistical regularities, it is rational to assume a generative model in which experiences come from unobservable (latent) causes, with similar experiences being generated by the same cause. Clustering experiences then becomes inferring their hypothesized latent causes, or latent-cause inference.

Normatively, latent-cause inference can be modeled as Bayesian inference with an infinite capacity prior over clusters and a cluster-specific likelihood. This allows for an unbounded number of clusters to be formed as new events are experienced, leading to a dynamically growing hypothesis space for learning. While initially proposed in the context of categorization [1], recent work has shown that this type of inference is also employed by both humans and animals in various reinforcement-learning settings, from Pavlovian [2] and instrumental conditioning [3] to learning task structures [4] and social evaluation [5]. In all these examples, latent causes do not strictly refer to cause-effect relationships, but can be interpreted as any underlying grouping that reflects the statistical structure of the world (including categories or contexts). In particular, in the case of reinforcement learning, both partially-observable states and entire context-dependent Markov decision processes can be interpreted and modeled as latent causes, and latent-cause inference is likely involved at both levels of analysis [6].

The major adaptive benefit of latent-cause inference is that it supports optimal generalization of behavioral policies across different situations that are similar in some way (such as having a similar reward function - e.g., crossing different streets, where the goal is to cross safely, and a learned policy can be easily generalized). Given the importance of latent-cause inference, its failure could underlie different forms of psychopathology. Overgeneralization has been observed both clinically and in the lab in many psychiatric disorders, including generalized anxiety disorder [7], depression [8] and post-traumatic stress disorder (PTSD) [9]. This phenomenon can be explained by an increased tendency to assign new experiences to previously inferred latent causes. Conversely, schizophrenia and schizotypy (schizophrenia-like traits present in the general population) are associated with cognitive disorganization and an incoherent perception of reality, behaviorally manifesting as e.g. both under- and overgeneralization in categorization tasks [10, 11]. Such clinical symptoms and task behavior could be explained by having a tendency to infer a large number of overlapping causes, leading to an inconsistent representation of reality.

To explore the links between suboptimal latent-cause inference and psychopathology at the level of individual participants, we developed a task through which we could quantify individual differences in latent-cause inference using parameters of a Bayesian model estimated from participants' behavior. A large number of online participants performed this task alongside completing several psychiatric questionnaires. We then correlated factors summarizing self-reported psychopathology with individual-level parameters of latent-cause inference.

2 Methods

Psychiatric symptom questions: Online participants (N = 1234) answered a set of questions assessing psychiatric trait symptoms of depression and anxiety, obsessive-compulsive disorder (OCD), PTSD, social anxiety and bipolar disorder (adapted from the Inventory of Depression and Anxiety Symptoms-II [12], Depressive Attributions Questionnaire [13], Positive Overgeneralization Scale [14] and Intolerance of Uncertainty Scale [15]). They also answered questions assessing detachment (low positive affect and social withdrawal), schizotypy and disinhibition (risk-taking and impulsivity), taken from the Personality Inventory for DSM-5 [16]. To address the issue of multicollinearity when using these measures as regressors for task behavior and parameters, we performed exploratory factor analysis using maximum-likelihood estimation with varimax rotation.

Task: A subset of participants (n = 565) also completed a task in which they assigned abstract visual stimuli to either old or new clusters (Figure 1). The stimuli, introduced in the cover story as 'microbes,' had spikes coming out of a core. These spikes varied along two dimensions: the number of spikes (dimension 1) and the length of the spikes (dimension 2). Participants were asked to classify the microbes into 'strains' (clusters) based on their perceptual similarity. They were told that the stimuli were photos of microbes taken at consecutive time points and that, at any given time, one microbe strain is dominant; however, microbes mutate sometimes to generate a new strain, which quickly starts to dominate but does not take over completely, so they could still sometimes see exemplars of old strains. The task consisted of 6 blocks, each block involving 4 ground-truth clusters. However, after a short practice session with feedback on their classification, participants were not given any further feedback and could classify stimuli into as many clusters per block as there were trials. After the latent-cause inference task, participants also completed the symmetry span task measuring spatial working memory [17].

Model: We modeled task behavior using a Bayesian model of latent-cause inference with a prior over clusters and a likelihood for each cluster. The prior was a time-sensitive Dirichlet process mixture (tDPM) [18], with the concentration

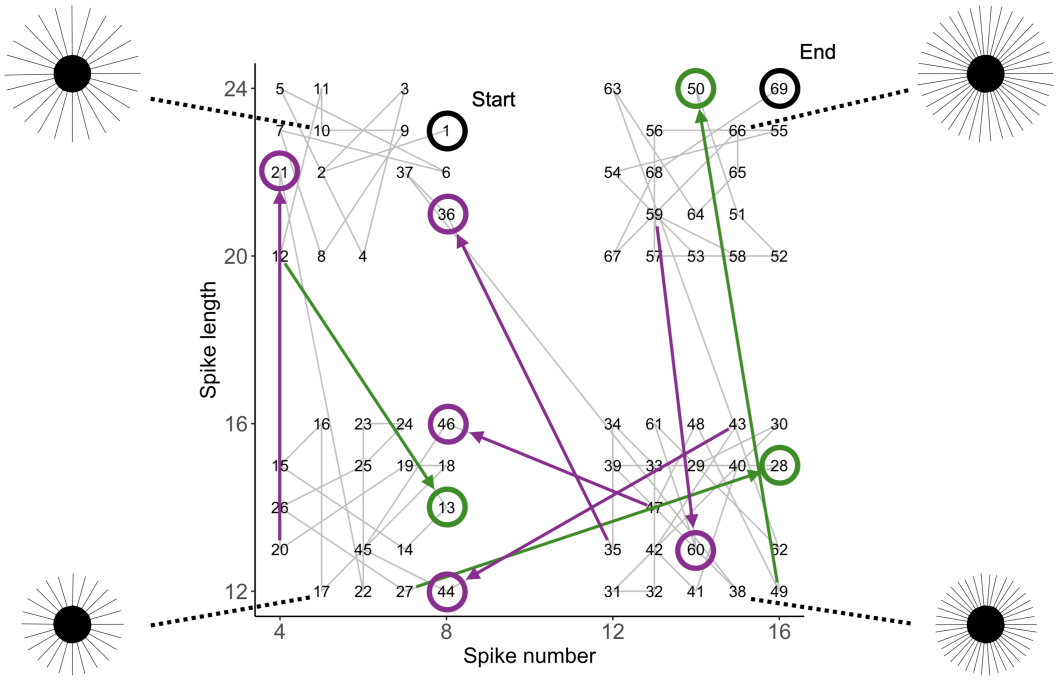


Figure 1: Schematic of the latent-cause inference task. Numbers denote the sequence of 'microbes' for the first block, and the axes denote the spike number and length of each microbe in the sequence. Example microbes are shown linked with dotted arrows to their corresponding number in the sequence. Microbes on most consecutive trials came from the same ground-truth cluster, but there would occasionally be a jump to a new cluster (green arrows and circles), or a jump back to an old cluster (purples arrows and circles).

parameter α (representing the tendency to form a new cluster, i.e. the 'new-cluster parameter') as a free individual-level parameter:

$$p(z_t = k) = \begin{cases} \frac{w_k}{\sum_{k'=1}^{K} w_{k'} + \alpha}, & k \leq K. \\ \\ \frac{\alpha}{\sum_{k'=1}^{K} w_{k'} + \alpha}, & k = K + 1. \end{cases} \tag{1}$$

where z_t is the cause to be inferred on trial t, K is the number of existing clusters, and w_k is the weight of cluster $k <= K$. This weight function is an exponential decay of the number of observations (with a free decay rate parameter λ), such that unused clusters become less and less weighted in the prior:

$$w_k = \sum_{\{i|t_i<t, z_t=k\}} exp(-\lambda(t - t_i)) \tag{2}$$

The likelihood for each cluster was computed as a product of the likelihoods for each dimension, as these were assumed to be independent. These likelihood functions were Gaussian-shaped with means equal to the average of previous observations the participant had assigned to the cluster, and variances (representing how variable/wide a cluster is assumed to be, i.e. the 'cluster size' parameters) as free parameters:

$$p(y_t|z_t = k) = exp(\frac{-(y_t - \mu_k)^2}{2\sigma^2}) \tag{3}$$

where y_t is the stimulus value on trial t and μ_k is the average of previous stimulus values for cluster k.

We estimated the four free parameters of this latent-cause inference model from participants' behavior using hierarchical Bayesian fitting. We also performed parameter recovery by simulating artificial behavior using parameter values from

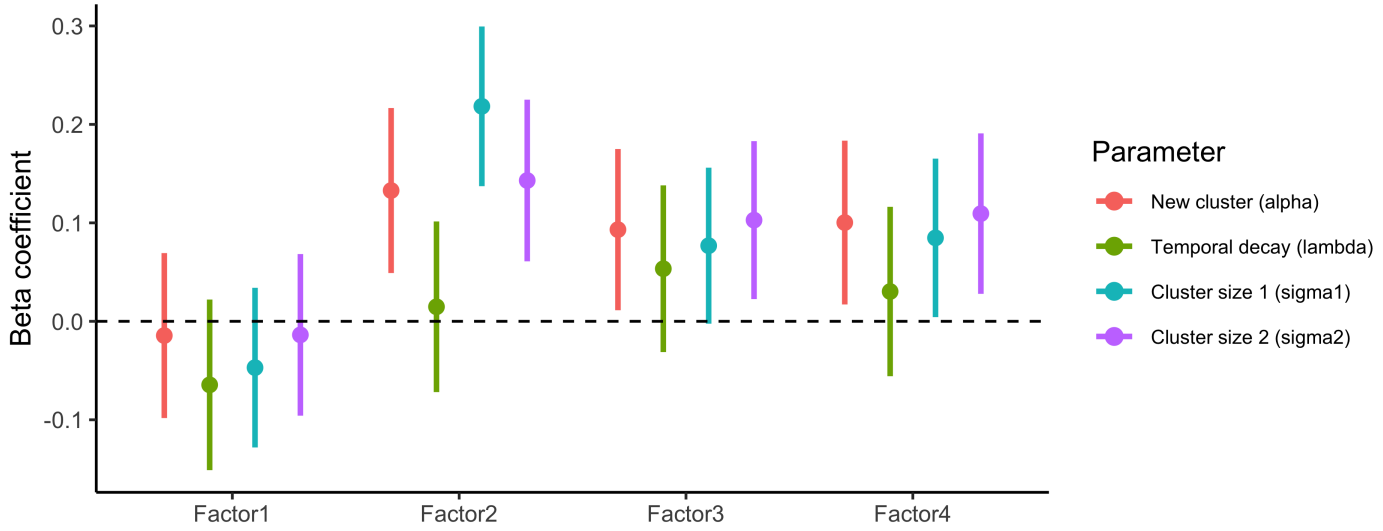


Figure 2: Associations between model parameters and transdiagnostic factors. Dots depict regression coefficient estimates for each factor from multiple regressions with a model parameter as dependent variable and with the four factors, gender, age and working memory as predictors. Whiskers represent 95% confidence intervals.

a subset of participants, re-estimating those parameters from the simulated behavior, and correlating the actual and recovered values.

Regression analyses: For each parameter of latent cause inference, we built linear regression models predicting the parameter values from the scores on the four psychiatric factors, while controlling for age, gender and working memory.

3 Results

All four model parameters were highly recoverable (Spearman's $\rho > 0.9$ between all actual and recovered parameters). As expected, the cluster-size parameters for the two dimensions correlated with each other (ρ = 0.65), but also with the new-cluster parameter (ρ = 0.65 and ρ = 0.72, respectively). Additional analyses suggested that the correlations between the new-cluster and cluster-size parameters were partially explained by lower working memory.

The exploratory factor analysis identified 4 factors. We named these factors based on the questions they loaded on: 'Anxious-depression', 'Schizotypy-disinhibition-mania', 'Obsessive-compulsivity', and 'Positive affectivity'. Of these, all but the 'Anxious-depression' factors significantly predicted the new-cluster (α) and cluster-size parameters (σ_1 and σ_2 , Figure 2). The 'Schizotypy-disinhibition-mania' factor had the largest effect sizes. The associations with the 'Obsessive-compulsivity' factor were mainly driven by the Checking subscale of OCD symptoms, which was the only subscale loading onto this factor that predicted in separate regressions the new-cluster (β = 0.17; SE = 0.05; $p < 0.001$) and cluster-size parameters (for dimension 1: β = 0.14; SE = 0.04; p = 0.002; for dimension 2: β = 0.16; SE = 0.04; $p < 0.001$).

4 Discussion

Our results establish our task as a means to quantify the latent-cause inference process in humans, and show that this process relates to mental health symptoms. In particular, our findings suggest that an increased tendency to infer new latent causes, as well as a tendency to infer large and variable causes, are associated with several transdiagnostic forms of psychopathology. We found a primary association with a transdiagnostic factor related to schizotypy, disinhibition (impulsivity, risk-taking) and mania. This suggests that suboptimal latent-cause inference might be related to deficits in executive function, which are a unifying characteristic of conditions related to our transdiagnostic factor, such as schizophrenia [19], ADHD [20] and bipolar disorder [21]. Interestingly, we found no association between working memory and psychopathology, suggesting the observed link is due to other aspects of executive function such as planning.

We also found weaker associations with checking compulsivity and a transdiagnostic factor related to heightened positive affect (euphoria) and positive overgeneralization (the tendency to overgeneralize from positive experiences to broader aspects of life). These might be related to a perception of the world as constantly prone to change (which might encourage repetitive checking to ensure that a feared negative outcome has not occurred), or the perception of new experiences as more novel than they actually are (as increased novelty has been linked to more positive affect [22]).

Overall, our results suggest that a tendency to group one's experiences into a large number of overlapping clusters might contribute to a broad range of clinical symptoms and conditions, from schizophrenia to ADHD. This highlights the benefit of transdiagnostic approaches and at the same time warrants further investigation of latent-cause inference deficits in specific disease populations. Lastly, our results underscore the role of fundamental processes related to reinforcement learning, such as learning a state representation using clustering/latent-cause inference [6], in maintaining mental health.

References

1. Anderson, J. R. The adaptive nature of human categorization. *Psychological Review* **98,** 409–429. ISSN: 1939-1471 (1991).
2. Gershman, S. J., Blei, D. M. & Niv, Y. Context, learning, and extinction. eng. *Psychological Review* **117,** 197–209. ISSN: 1939-1471 (Jan. 2010).
3. Collins, A. G. E. & Frank, M. J. Neural signature of hierarchically structured expectations predicts clustering and transfer of rule sets in reinforcement learning. eng. *Cognition* **152,** 160–169. ISSN: 1873-7838 (July 2016).
4. Franklin, N. T. & Frank, M. J. Generalizing to generalize: Humans flexibly switch between compositional and conjunctive structures during reinforcement learning. en. *PLOS Computational Biology* **16,** e1007720. ISSN: 1553-7358 (Apr. 2020).
5. Shin, Y. S. & Niv, Y. Biased evaluations emerge from inferring hidden causes. en. *Nature Human Behaviour* **5,** 1180–1189. ISSN: 2397-3374 (Sept. 2021).
6. Niv, Y. Learning task-state representations. en. *Nature Neuroscience* **22.** Number: 10 Publisher: Nature Publishing Group, 1544–1553. ISSN: 1546-1726 (Oct. 2019).
7. Laufer, O., Israeli, D. & Paz, R. Behavioral and Neural Mechanisms of Overgeneralization in Anxiety. eng. *Current biology: CB* **26,** 713–722. ISSN: 1879-0445 (Mar. 2016).
8. Huys, Q. J. M. & Dayan, P. A Bayesian formulation of behavioral control. eng. *Cognition* **113,** 314–328. ISSN: 1873-7838 (Dec. 2009).
9. Norbury, A. *et al.* Latent cause inference during extinction learning in trauma-exposed individuals with and without PTSD. *Psychological Medicine,* 1–12. ISSN: 1469-8978 (2021).
10. Doughty, O. J., Lawrence, V. A., Al-Mousawi, A., Ashaye, K. & Done, D. J. Overinclusive thought and loosening of associations are not unique to schizophrenia and are produced in Alzheimer's dementia. *Cognitive Neuropsychiatry* **14,** 149–164. ISSN: 1464-0619 (2009).
11. Morgan, C. J. A., Bedford, N. J., O'Regan, A. & Rossell, S. L. Is Semantic Processing Impaired in Individuals With High Schizotypy? en-US. *The Journal of Nervous and Mental Disease* **197,** 232–238. ISSN: 0022-3018 (Apr. 2009).
12. Watson, D. *et al.* Development and Validation of New Anxiety and Bipolar Symptom Scales for an Expanded Version of the IDAS (the IDAS-II). *Assessment* **19.** Publisher: SAGE Publications Inc, 399–420. ISSN: 1073-1911 (Dec. 2012).
13. Kleim, B., Gonzalo, D. & Ehlers, A. The Depressive Attributions Questionnaire (DAQ): Development of a Short Self-Report Measure of Depressogenic Attributions. *Journal of Psychopathology and Behavioral Assessment* **33,** 375–385. ISSN: 0882-2689 (2011).
14. Eisner, L. R., Johnson, S. L. & Carver, C. S. Cognitive responses to failure and success relate uniquely to bipolar depression versus mania. *Journal of Abnormal Psychology* **117.** Place: US Publisher: American Psychological Association, 154–163. ISSN: 1939-1846 (2008).
15. Carleton, R. N., Norton, M. A. P. J. & Asmundson, G. J. G. Fearing the unknown: A short version of the Intolerance of Uncertainty Scale. en. *Journal of Anxiety Disorders* **21,** 105–117. ISSN: 0887-6185 (Jan. 2007).
16. Anderson, J. L., Sellbom, M. & Salekin, R. T. Utility of the Personality Inventory for DSM-5-Brief Form (PID-5-BF) in the Measurement of Maladaptive Personality and Psychopathology. eng. *Assessment* **25,** 596–607. ISSN: 1552-3489 (July 2018).
17. Kane, M. J. *et al.* The generality of working memory capacity: a latent-variable approach to verbal and visuospatial memory span and reasoning. eng. *Journal of Experimental Psychology. General* **133,** 189–217. ISSN: 0096-3445 (June 2004).
18. Zhu, X., Ghahramani, Z. & Lafferty, J. Time-Sensitive Dirichlet Process Mixture Models. en (2005).
19. Orellana, G. & Slachevsky, A. Executive Functioning in Schizophrenia. *Frontiers in Psychiatry* **4,** 35. ISSN: 1664-0640 (June 2013).
20. Brown, T. ADD/ADHD and Impaired Executive Function in Clinical Practice. *Current psychiatry reports* **10,** 407–11 (Nov. 2008).
21. Dixon, T., Kravariti, E., Frith, C., Murray, R. M. & McGuire, P. K. Effect of symptoms on executive function in bipolar illness. eng. *Psychological Medicine* **34,** 811–821. ISSN: 0033-2917 (July 2004).
22. Heller, A. S. *et al.* Association between real-world experiential diversity and positive affect relates to hippocampal–striatal functional connectivity. en. *Nature Neuroscience* **23.** Number: 7 Publisher: Nature Publishing Group, 800–804. ISSN: 1546-1726 (July 2020).

Meta-Analysis of Reward Prediction Error Signals in Substance Users

Jessica A. Mollick
Department of Psychiatry
Yale School of Medicine
New Haven, CT 06510
Jessica.Mollick@yale.edu

Stephanie Malta
Medical Sciences
Boston University
Boston, MA 02118
malta@bu.edu

Philip R. Corlett
Department of Psychiatry
Yale School of Medicine
New Haven, CT 06510
philip.corlett@yale.edu

Hedy Kober
Department of Psychiatry
Yale School of Medicine
New Haven, CT 06510
hedy.kober@yale.edu

Abstract

Addiction, or substance use disorders (SUDs) are the most prevalent of all psychiatric conditions, with dire costs to individuals and society. Computational models of SUDs have proposed that drug use influences valuation of drug and non-drug stimuli, including via changes in prediction error (PE) driven learning.

However, there are inconsistent neuroimaging findings examining PE learning in substance use: Some studies have observed changes in PE related neural activity in drug-using populations compared to healthy adults, while others have not. To assess the evidence for changes in PE signaling in substance use, we conducted a coordinate-based meta-analysis of BOLD activity to reward PEs in individuals with SUDs and substance use problems [Nstudies=9, Nparticipants=250] using multi-level kernel density analysis (MKDA). PEs assessed in included studies tended to be model-free, instrumental PEs, incorporating unexpected rewards or punishments. We selected a similar, representative database of PE coordinates from healthy adults matching these features (Nstudies = 141, Nparticipants 3630; Corlett*, Mollick*, and Kober, 2022).

We found consistent activity to reward PEs in striatal regions in substance users, primarily ventral striatum, and cortically in the right inferior frontal gyrus, and in the supramarginal gyrus (including parietal lobe). We compared brain areas consistently encoding PE in the substance users with those in healthy adults, finding that PEs were more consistently represented in the striatum for substance-using participants.

These results have important ramifications for theories of PE and addiction. More consistent PE signals in substance use may be linked to the pharmacological effects of drugs of abuse on the dopamine system. Further, changes in PE signals contribute to value signals for non-drug rewards, which may be particularly important for addiction recovery, involving choices of alternative actions over choices to use drugs.

Keywords: prediction error, addiction, neuroimaging, reward

Acknowledgements

Funding: Supported by R01DA043690 (to HK) and R21MH116258 and R01MH12887 (to PC).

1 Introduction

Substance use disorders (SUDs) are extremely prevalent (with around 40.3 million people experiencing a substance use disorder in the US in the past year [1], and incorporate many symptoms including risky and compulsive use, impaired control, physiological alterations, and craving [2]. Importantly, individuals with SUDs make continued choices to take drugs, despite negative consequences that impact many aspects of their lives, including work, social relationships, and emotional well-being. Understanding how learning mechanisms contribute to continued drug-taking will increase our understanding and the design of effective treatments.

We can consider how learning contributes to drug-taking choices with a reinforcement or temporal-differences learning framework, which involves updating the value of actions with reward prediction errors (PEs: δ) see Equation 1. Reward prediction errors are calculated based on the difference between expected values of an action or cue (V) and the obtained reward outcome (r) [3]). Neurobiologically, PEs have been linked to the phasic firing of dopamine neurons [4], and dopamine release, measured using fast-scan cylic voltammetry in animal studies [5]. Notably, human fMRI BOLD signals in the ventral and dorsal striatum have also been correlated with prediction error signals [6, 5, 7].

$$V < -V + \delta(r - V). \tag{1}$$

Drugs of abuse increase dopamine in the nucleus accumbens as measured by voltammetry [8] and cause changes in synaptic plasticity in the midbrain and ventral striatum that affect PEs [9]. For example, [10] demonstrated that cocaine use impaired the ability of dopamine neurons to suppress firing during the omission of an expected reward. Drug use is also important for theories of reward PE generally, as initial drug use may lead to positive reinforcement, but continued drug use leads to negative valence emotional states that may enhance the value of positive rewards to offset the aversive state [11]. Thus, how drug use affects PE may also help us understand more basic interactions between positive and negative valence and their effects on the dopamine system.

The effects of drugs on PE signaling have also been linked with computational models of the dopamine system. For example, Redish [12] proposed that the pharmacological effects of drug use on the dopamine system causes an enhancement of PE signals that cannot be compensated by the value function, causing drug-related actions to continue to accrue value. However, it is unclear to what extent the drug-induced PE enhancement influences learning for non-drug rewards. We would predict that non-drug related actions have relatively smaller value increases due to relatively reduced effects of natural rewards on the dopamine system (a similar finding was modeled by Redish), but the effects of drug use on the dopamine system may also enhance PEs more generally, even for non-drug rewards.

Based on this theoretical background, it is important to characterize the neurobiological effects of drug use on PE and value signals and how these signals are influenced by substance use. Studies have directly compared value signals for drug and non-drug rewards, generally finding a blunting of value signals to non-drug rewards relative to drug rewards [13, 14]. However, recent studies on the neural representations of PEs in substance users and SUDs have found conflicting results. While Tanabe et al. [15] observed reduced neural PE tracking in individuals with stimulant SUD, Park et al. [16] did not observe differences in striatal PEs in individuals with alcohol SUD compared to healthy controls. A recent small meta-analysis of reward PEs and value signals compared substance users with control participants, finding blunted PE/value signals in substance users in bilateral putamen, insula, and medial frontal gyrus [17].

Here, we focus on understanding PE brain signals for non-drug rewards, which guide value updating and action selection. Understanding how difficulties with substance use influence neural PE representations is critically important for understanding the learning mechanisms behind reinforcement of actions to use drugs and to abstain from drugs. Importantly, we can also relate PE signals measured by fMRI to the function of the dopamine system as captured by preclinical data, which inspired the computational models described above. We do so by interpreting our results in light of combined pharmacological and fMRI studies [18], or combined optogenetic and fMRI studies [19]. Understanding striatal biology, which contains medium spiny neurons (MSNs) with D1 or D2 receptors, is important for interpreting these effects. Positive BOLD signals in the nucleus accumbens occur when dopamine neurons are optogenetically stimulated and accumulating data supports a model suggesting that dopamine release in the accumbens activates post-synaptic D1 receptors, which changes postsynaptic membrane potential, leading to BOLD increases [18]. Both results are important for interpreting BOLD signals to reward PEs in substance users, as SUDs have been linked to changes in post-synaptic D2 receptors as measured with PET [20].

Overall, our goals involved examining how drug use influences the regions encoding reward PEs. To examine this, we conducted a fMRI meta-analysis of reward PEs in individuals with SUDs, substance use problems and regular substance users. Importantly, we focus selectively on PEs for non-drug rewards because this has been the focus of the literature and thus of the included studies. We compared consistency of brain activity in responses to prediction error in studies with healthy control participants and regular substance use on similar types of prediction error tasks.

2 Methods

Search strategy: We conducted the search and selection of articles for this meta-analysis in accordance with Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines and analyzed using Multi-level Kernel Density Analysis (MKDA; [21, 6]). Overall, 56,667 abstracts were identified. After removing duplicates, 94 full-text articles were assessed for inclusion; of these, we included 9 fMRI studies in the meta-analysis, contributing 20 independent contrasts and representing 250 participants.

Inclusion: We included fMRI studies that involved human participants between the ages of 18-65 who used drugs regularly (alcohol, nicotine, cocaine, marijuana, or opiates), experienced substance use problems, or had an SUD. The included studies reported xyz activation coordinates (standard MNI or Talairach) corresponding to prediction errors, like unexpected reward or punishment, or unexpected sensory or motor outcomes that violated beliefs. Studies only comparing rewards to punishments were excluded. Each study was assessed by two researchers, and any disagreements were discussed until both were in agreement.

Data extraction: We extracted the following information from each included contrast: N (number of participants), xyz coordinates for each activation foci, whether the foci came from an ROI or whole-brain analysis, and information related to drug use and the participant group, including the type of substance, and the severity of drug use/SUD. We also extracted information about the type of PE, following methods based on a meta-analysis on PEs in healthy adults [6], including whether the PE resulted from a (1) primary or secondary reward, (2) was positive or negative (3) instrumental or Pavlovian conditioning, and (4) outcome valence (an appetitively reward or an aversive punishment, and (5) whether the PE involved an unexpected outcome, or an atypical PE capturing a violation of beliefs.

To quantify the type of included PEs, we examined the distribution of characteristics of PE in included contrasts, as shown in Figure 1. All included contrasts focused on instrumental, model-free PE. 85% of contrasts focused on secondary rewards and 15% focused on primary rewards, and incorporated either appetitive valence (25%), aversive (30%), or both (45%). The majority of included contrasts focused on signed PE (95%), and included positive (50%) and negative PE (15%), and PEs in both positive and negative directions (35%). 70% of included contrasts had participants with a substance use disorder diagnosis, while 20% included participants with substance use problems or used substances regularly, and 10% included participants in both categories. One contrast included a fictive PE signal and one included a social outcome.

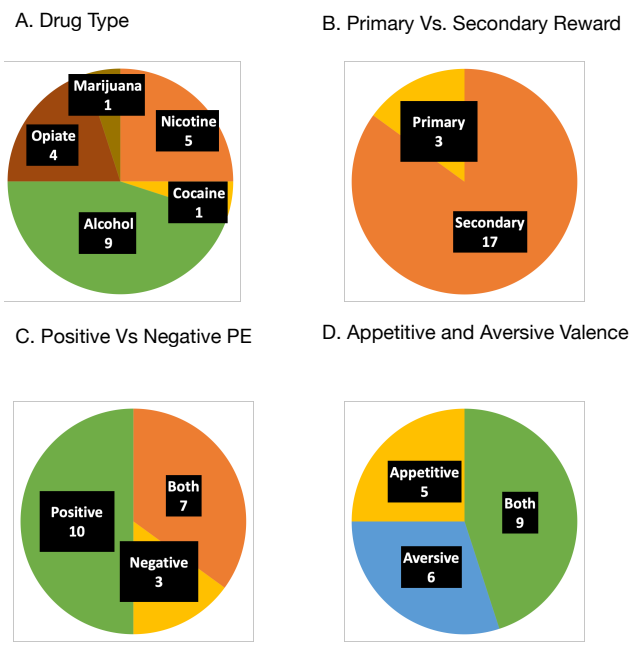


Figure 1: *Characterizing the proportion of drug studies by: A. Drug type B. Primary or secondary reinforcers C. positive or negative PE D. appetitive or aversive valence*

Based on these characteristics, we extracted a matched set of contrasts from our large published database of prediction error coordinates from healthy participants [6]. More specifically, we selected studies that were instrumental, model free prediction errors, and excluded cognitive and perceptual PEs (but included social and fictive PEs). This selected database incorporated 141 studies and 226 contrasts.

Analysis Using MKDA, we incorporated coordinates from each included contrast and convolved them with a 10mm spherical kernel to create a contrast indicator map containing voxels within 10mm of the peak for that contrast. We then generated a weighted average of CIMs to obtain a density map, weighted by the square root of the sample size for each contrast (which weights larger studies more heavily). This creates an interpretable meta-analytic statistic (P) at each voxel, representing the weighted proportion of contrasts that activate within 10 mm of each voxel. The images were thresholded at FWE (family wise-error) $p < .05$ (voxelwise $p < .005$). Cluster level thresholds were determined using Monte-Carlo simulation. Results represent meta-analyses across all included studies, and meta-contrasts that compare the consistency of activation across two subsets of coordinates by comparing the proportion of contrasts activating within 10mm of that voxel in each subset.

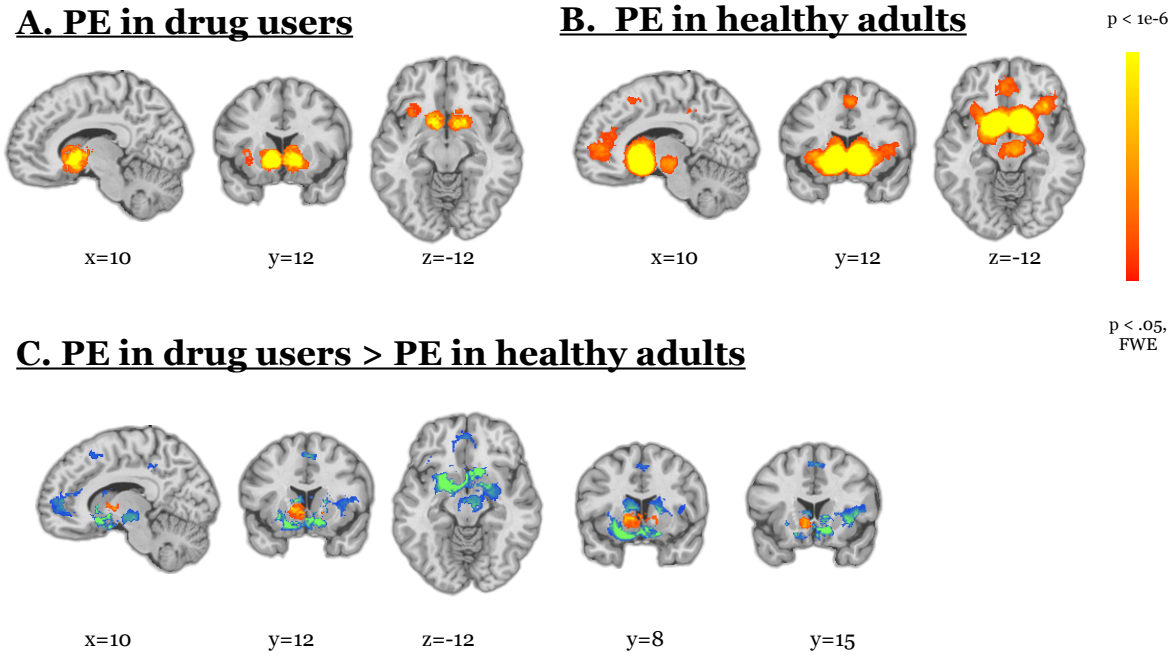


Figure 2: *Comparison of PEs in Drug-using and healthy individuals A. PEs in drug-using and SUD participants. B. PEs in healthy individuals. C. Contrast comparing PE in drug-using/SUD participants and healthy individuals*

3 Results

Among drug users (Fig 2A), we observed consistent PE signals in the left and right ventral striatum, extending into the pallidum, as well as dorsal striatum (including both caudate and putamen). We also observed cortical signals to PE in the right inferior frontal gyrus and insula, and in the inferior parietal lobe (including the right supramarginal gyrus).

In our matched-subset database of PEs from healthy adults – and consistent with the published results in Corlett* et al. [6] – we observed prediction error signals both cortically and subcortically, including midbrain, bilateral, ventral and dorsal striatum, thalamus, orbitofrontal cortex, medial PFC, anterior cingulate, and bilateral inferior parietal lobule (Fig 2B).

A direct contrast between these two meta-analytic databases (Fig 2C) revealed a peak in the left ventral striatum, extending into the pallidum, that showed more consistent prediction error signaling in substance users compared to healthy adults. Right mid- to dorsal striatum also showed more consistent PE encoding in substance users compared to healthy adults. Control participants showed more consistent PE-related activations in caudate, putamen, subgenual and posterior cingulate, medial PFC, parietal cortex, and midbrain.

4 Discussion

Using meta analysis, we report consistent PE-related activations in substance users in both cortical and subcortical regions. As expected, substance users exhibited consistent PE-related activations in both dorsal and ventral striatum, insula/inferior frontal gyrus, and in the inferior parietal lobe. We further showed that, compared to a matched meta-analytic set of PE studies in healthy adults, substance users exhibited more consistent PE-related activations in left ventral striatum, while control participants had more consistent PE in dorsal striatum, medial PFC, midbrain and parietal regions. However, interpreting the brain regions consistently encoding PE signals in substance users, and comparing consistency of brain regions encoding PE across substance users and healthy adults, is complex, particularly when combining effects across different drug types, as we did in this meta-analysis. Importantly, as we examined PE in participants who were already using drugs, we cannot tell whether the changes in PE observed are due to differences in brain regions encoding PE before use, or the consequence of the effects of drugs on these signals.

The dorsal and ventral striatum have been implicated in coding of PE signals across numerous studies in healthy individuals [6, 7]. We found consistent PE signals in the dorsal and ventral striatum in individuals using drugs, which was expected based on prior literature. This could be interpreted as an increase in the signal-to-noise ratio of the PE signal itself – which could suggest an enhanced signal to reward PE or less noise, i.e. enhanced fidelity of PE signals compared

to other signals. It is possible that drug users hyperfocus on drug rewards, increasing the gain of drug-related signals, and have reduced or blunted signals to non-drug rewards. However, our data does not support a blunting of non-drug related PE signals, in fact suggesting that such signals are consistently represented in individuals who use drugs. One possibility is that individuals with SUDs compute and represent the PE signal adequately but don't use it well to update behavior choice. Another important thing to consider when interpreting our results is that money is a secondary reinforcer which can be used to obtain drugs, so drug use may also enhance attention to money due to the ability to use money to obtain drugs.

The observed results have important implications for models of PE and addiction. We observed that PE signals were more consistently represented in striatum in participants who used substances regularly compared to PEs in healthy participants. Importantly, we compared PEs in healthy adult participants during similar types of tasks as the prediction error tasks used in participants who used substances. This is significant given that findings are mixed on how substance use affects striatal PE, with some studies observing reduced PE tracking in individuals with SUDs in striatum [15] while others observed intact striatal PE signals in individuals with SUDs [16]. The results are also significant from a computational modeling perspective, particularly the idea that ventral striatum may act a a "critic", providing prediction error signals that influence action selection by influencing updating of action values in dorsal striatum [22].

The observed increase in PE signals in substance-using participants may be consistent with the Redish [12] model which proposes that the effects of drug reward on the dopamine system leads to an increase in PE signals. However, importantly, Redish [12] proposes that this increase in dopamine causes drug-related cues and actions to grow without bound, while non-drug rewards and cues grow until they reach a certain asymptote. Prior work showed reduced differentiation in value signals between drug and non-drug rewards in dependent smokers [14] and enhanced PE signals for unexpected alcohol in alcohol-dependent individuals [23]. Future work could examine drug-related values and PE as data accumulates.

One hypothesis is that individuals who use drugs may demonstrate changes in model-free temporal difference learning (or reward learning [24]). Differences in model-free updating following a rewarded choice have been linked to greater escalation of methamphetamine administration [25].

We plan to compare the brain regions associated with different types of PEs in substance using participants compared to healthy adults. We have observed differences in brain areas encoding appetitive and aversive PE, and future work could examine how drug use changes these signals. Our work may also inspire future studies of PE in addictive disorders. We observed a lack of studies considering Pavlovian PE and few studies with primary rewards.

PEs are important for updating the values of cues and actions, and one aspect of addiction is continued choice of drug-related actions over alternative actions. PE signals to non-drug rewards may be critically important for developing addiction treatments and understanding recovery, as PE signals update the value of non-drug related actions. One effective treatment of addiction, contingency management, pays participants for abstaining from drugs, thus enhancing the value of abstinence [26]. A general enhancement in PE signals in substance use is important because it suggests that treatments incentivizing abstinence may be particularly effective. Further, understanding how repeated reinforcements (such as drugs) affect localization of PE computations will influence our understanding of how these computations influence behavior selection. For example, more consistent representation of PE in striatum after continued drug use may enhance attention toward reward values when selecting behaviors, and reduce updating of goals with PE signals that may be represented prefrontally. More broadly, examining how habits such as drug use influence PE will enhance our understanding of brain systems involved in goal-directed behaviors and can inform design of biologically plausible algorithms.

References

[1] SAMHSA. Substance abuse and mental health services administration: Key substance use and mental health indicators in the united states: Results from the 2020 national survey on drug use and health. *(HHS Publication No. PEP21-07-01-003, NSDUH Series H-56)*, 2021. URL `Retrieved from https://www.samhsa.gov/data/`.

[2] Shosuke Suzuki and Hedy Kober. *Substance-Related and Addictive Disorders*, volume 1, pages 481–506. American Psychological Association, Washington, DC, 2018.

[3] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[4] Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science*, 275 (5306):1593–1599, 1997. ISSN 00368075 10959203. doi: 10.1126/science.275.5306.1593.

[5] Andrew S Hart, Robb B Rutledge, Paul W Glimcher, and Paul EM Phillips. Phasic dopamine release in the rat nucleus accumbens symmetrically encodes a reward prediction error term. *Journal of Neuroscience*, 34(3):698–704, 2014. ISSN 0270-6474.

[6] Philip R. Corlett*, Jessica A. Mollick*, and Hedy Kober. Meta-analysis of human prediction error for incentives, perception, cognition, and action. *Neuropsychopharmacology*, 2022. ISSN 1740-634X. doi: 10.1038/s41386-021-01264-3. URL https://doi.org/10.1038/s41386-021-01264-3.

[7] Robb B Rutledge, Mark Dean, Andrew Caplin, and Paul W Glimcher. Testing the reward prediction error hypothesis with an axiomatic model. *Journal of Neuroscience*, 30(40):13525–13536, 2010. ISSN 0270-6474.

[8] Gaetano D. Di Chiara and Assunta Imperato. Drugs abused by humans preferentially increase synaptic dopamine concentrations in the mesolimbic system of freely moving rats. *Proceedings of the National Academy of Sciences*, 85(14): 5274–5278, 1988.

[9] Christian Lüscher. The emergence of a circuit model for addiction. *Annual review of neuroscience*, 39:257–276, 2016. ISSN 0147-006X.

[10] Yuji K Takahashi, Thomas A Stalnaker, Yasmin Marrero-Garcia, Ray M Rada, and Geoffrey Schoenbaum. Expectancy-related changes in dopaminergic error signals are impaired by cocaine self-administration. *Neuron*, 101(2):294–306. e3, 2019. ISSN 0896-6273.

[11] George F Koob and Michel Le Moal. Drug addiction, dysregulation of reward, and allostasis. *Neuropsychopharmacology*, 24(2):97–129, 2001.

[12] A David Redish. Addiction as a computational process gone awry. *Science*, 306(5703):1944–1947, 2004. ISSN 0036-8075.

[13] Rita Z. Goldstein and Nora D. Volkow. Dysfunction of the prefrontal cortex in addiction: neuroimaging findings and clinical implications. *Nature Reviews Neuroscience*, 12(11):652–669, 2011. ISSN 1471-0048. URL http://www.ncbi.nlm.nih.gov/pubmed/22011681.

[14] Mira Bühler, Sabine Vollstädt-Klein, Andrea Kobiella, Henning Budde, Laurence J Reed, Dieter F Braus, Christian Büchel, and Michael N Smolka. Nicotine dependence is characterized by disordered reward processing in a network driving motivation. *Biological psychiatry*, 67(8):745–752, 2010. ISSN 0006-3223.

[15] Jody Tanabe, Jeremy Reynolds, Theodore Krmpotich, Eric Claus, Laetitia L Thompson, Yiping P Du, and Marie T Banich. Reduced neural tracking of prediction error in substance-dependent individuals. *American Journal of Psychiatry*, 170(11):1356–1363, 2013. ISSN 0002-953X.

[16] Soyoung Q Park, Thorsten Kahnt, Anne Beck, Michael X Cohen, Raymond J Dolan, Jana Wrase, and Andreas Heinz. Prefrontal cortex fails to learn from reward prediction errors in alcohol dependence. *Journal of Neuroscience*, 30(22): 7749–7753, 2010. ISSN 0270-6474.

[17] Serenella Tolomeo, Zachary A Yaple, and Rongjun Yu. Neural representation of prediction error signals in substance users. *Addiction Biology*, 26(3):e12976, 2021. ISSN 1355-6215.

[18] Brian Knutson and Sasha EB Gibbs. Linking nucleus accumbens dopamine and blood oxygenation. *Psychopharmacology*, 191(3):813–822, 2007. ISSN 1432-2072.

[19] Emily A Ferenczi, Kelly A Zalocusky, Conor Liston, Logan Grosenick, Melissa R Warden, Debha Amatya, Kiefer Katovich, Hershel Mehta, Brian Patenaude, and Charu Ramakrishnan. Prefrontal cortical regulation of brainwide circuit dynamics and reward-related behavior. *Science*, 351(6268), 2016. ISSN 0036-8075.

[20] Nora D Volkow, Joanna S Fowler, Gene-Jack Wang, James M Swanson, and Frank Telang. Dopamine in drug abuse and addiction: results of imaging studies and treatment implications. *Archives of neurology*, 64(11):1575–1579, 2007.

[21] Hedy Kober and Tor D. Wager. Meta-analysis of neuroimaging data. *Wiley Interdisciplinary Reviews: Cognitive Science*, 1(2):293–300, 2010. ISSN 1939-5086.

[22] Yuji Takahashi, Geoffrey Schoenbaum, and Yael Niv. Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in neuroscience*, 2: 14, 2008.

[23] Anita Cservenka, Kelly E Courtney, Dara G Ghahremani, Kent E Hutchison, and Lara A Ray. Development, initial testing and challenges of an ecologically valid reward prediction error fmri task for alcoholism. *Alcohol and Alcoholism*, 52(5):617–624, 2017. ISSN 0735-0414.

[24] Quentin JM Huys, Lorenz Deserno, Klaus Obermayer, Florian Schlagenhauf, and Andreas Heinz. Model-free temporal-difference learning and dopamine in alcohol dependence: examining concepts from theory and animals in human imaging. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, 1(5):401–410, 2016. ISSN 2451-9022.

[25] Stephanie M Groman, Bart Massi, Samuel R Mathias, Daeyeol Lee, and Jane R Taylor. Model-free and model-based influences in addiction-related behaviors. *Biological psychiatry*, 85(11):936–945, 2019.

[26] Stephen T Higgins, Sarah H Heil, and Jennifer Plebani Lussier. Clinical implications of reinforcement as a determinant of substance use disorders. *Annu. Rev. Psychol.*, 55:431–461, 2004. ISSN 0066-4308.

Solving infinite-horizon POMDPs with memoryless stochastic policies in state-action space

Johannes Müller
Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany
jmueller@mis.mpg.de

Guido Montúfar
Department of Mathematics and Department of Statistics, UCLA, CA, USA
Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany
montufar@math.ucla.edu

Abstract

Reward optimization in fully observable Markov decision processes is equivalent to a linear program over the polytope of state-action frequencies. Taking a similar perspective in the case of partially observable Markov decision processes with memoryless stochastic policies, the problem was recently formulated as the optimization of a linear objective subject to polynomial constraints. Based on this we present an approach for Reward Optimization in State-Action space (ROSA). We test this approach experimentally in maze navigation tasks. We find that ROSA is computationally efficient and can yield stability improvements over other existing methods.

Keywords: POMDP, policy gradient, state-action frequencies, constrained optimization.

Acknowledgements

The authors acknowledge support from the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement no 757983). JM also acknowledges support from the International Max Planck Research School for Mathematics in the Sciences and the Evangelisches Studienwerk Villigst e.V..

1 Introduction

Partially Observable Markov Decision Processes (POMDPs) offer a popular model for sequential decision making with state uncertainty. Here, actions are selected based on partial observations of the system's state with the objective to maximize a cumulative discounted reward. We focus on infinite horizon problems and memoryless stochastic policies which offer an alternative to difficult-to-optimize policies based on belief states and policies with memory. Techniques such as policy iteration and value iteration usually require a belief state or a finite-state controller representation of the policy, and the standard approaches based on policy gradients [SMS+99, AYA18] can suffer from ill-conditioning when future rewards are not sufficiently discounted. Recently, a polynomial programming formulation of POMDPs was derived in [MM22], generalizing the linear program associated to MDPs. In this work we provide a practical implementation of this approach and demonstrate experimentally, in navigation problems of different sizes, that it offers a competitive alternative improving computational cost and numerical stability for a range of discount factors.

Policy gradients A very popular approach in Reinforcement Learning are policy gradients methods. In fully observable problems, the iteration complexity of policy gradient methods behaves like $O((1-\gamma)^{-\kappa})$, where $\gamma \in (0,1)$ is the discount factor and $\kappa \in \mathbb{N}$ depends on the specific method [CCC+21]. This is reminiscent of the Lipschitz constant of the reward function (as a function of the policy), which behaves like $O((1-\gamma)^{-1})$ [PRB15]; see Figure 1. This leads to increasingly ill-conditioned problems as $\gamma \to 1$ and can cause undesired oscillations during optimization [Wag11]. However, choosing a discount factor close to 1 is desirable as one often wishes to optimize the mean reward rather than a discounted reward. This is also required to prevent vanishing policy gradients in sparse reward MDPs, where, denoting n_S the number of states, gradients can of order $O(2^{-n_S/2})$ if $\gamma \leq n_S/(n_S+1)$ [AKLM21]. In principle the ill-conditioning problem can be addressed by introducing an appropriate metric, as in natural policy gradients or trust region policy optimization, which can be costly, however.

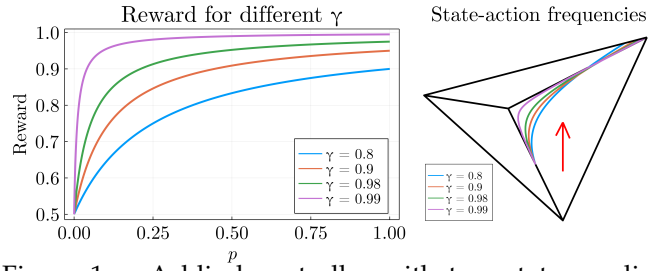


Figure 1: A blind controller with two states; policies can be parameterized by $p \in [0,1]$; for increasing γ the Lipschitz constant of the reward increases (left); the corresponding feasible state-action frequencies, probability simplex $\Delta_{\mathcal{S}\times\mathcal{A}}$, and instantaneous reward vector (right).

Optimization in state-action space An alternative to optimizing over the policy parameters is to optimize the reward over all feasible state-action frequencies of the POMDP. State-action frequencies are weighted averages of the time spent by the Markov process at different state-action pairs. The reward of a policy depends linearly on its state-action frequency and the optimization in state-action space maximizes the time spend in favorable states. The policy corresponding to a state-action frequency can be recovered by conditioning over states. In MDPs, the state-action frequencies form a polytope and hence the problem becomes a linear program [Der70, Kal94]. This yields a strongly polynomial algorithmic approach, i.e., does not degrade for $\gamma \to 1$ [PY15]. In the case of POMDPs, additional polynomial constraints describe the set of all feasible state-action frequencies of POMDPs, which were recently described by [MM22]. This yields a polynomial program of POMDPs generalizing the linear programs of MDPs. In this work we investigate the practical viability of this approach to optimize the reward in POMDPs. We consider navigation tasks in random mazes of different sizes, for which we develop a tool to generate the constraints and solve the constrained optimization problem using interior point methods. Our experiments show that the proposed method can yield significant computational savings compared to several baselines, while also remaining numerically stable across values of γ where other methods fail.

2 Notation and setup

We denote the simplex of probability distributions on a finite set $\mathcal{X}$ by $\Delta_{\mathcal{X}}$ and the set of Markov kernels from a finite set $\mathcal{X}$ to another finite set $\mathcal{Y}$ by $\Delta_{\mathcal{Y}}^{\mathcal{X}}$. A *partially observable Markov decision process* or shortly *POMDP* is a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \alpha, \beta, r)$. We assume that $\mathcal{S}, \mathcal{O}$ and $\mathcal{A}$ are finite sets which we call the *state*, the *observation* and the *action space* respectively. We fix a Markov kernel $\alpha \in \Delta_{\mathcal{S}}^{\mathcal{S}\times\mathcal{A}}$ which we call the *transition mechanism* and a kernel $\beta \in \Delta_{\mathcal{O}}^{\mathcal{S}}$ which we call the *observation mechanism*. Further, we consider an *instantaneous reward vector* $r \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$. As *policies* we consider elements $\pi \in \Delta_{\mathcal{A}}^{\mathcal{O}}$, which are referred to as *memoryless stochastic policies*. Every policy defines a transition kernel $P_\pi \in \Delta_{\mathcal{S}\times\mathcal{A}}^{\mathcal{S}\times\mathcal{A}}$ by $P_\pi(s',a'|s,a) := \alpha(s'|s,a)\sum_o \pi(a'|o)\beta(o|s')$. For any initial state distribution $\mu \in \Delta_{\mathcal{S}}$, a policy $\pi \in \Delta_{\mathcal{A}}^{\mathcal{O}}$ defines a Markov process on $\mathcal{S}\times\mathcal{A}$ with transition kernel P_π which we denote by $\mathbb{P}^{\pi,\mu}$. For a *discount rate* $\gamma \in (0,1)$ we define the *infinite horizon expected discounted reward*

$$R(\pi) := \mathbb{E}_{\mathbb{P}^{\pi,\mu}}\left[(1-\gamma)\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right]. \tag{1}$$

We consider the *reward maximization problem*, i.e., the problem of maximizing $R(\pi)$ subject to $\pi \in \Delta_{\mathcal{A}}^{\mathcal{O}}$. The discount factor γ is most commonly introduced for mathematical convenience. For large state spaces and sparse rewards or more generally for POMDPs where the optimal policy should not depend on an unknown initial state distribution, it may be more desirable to consider the expected reward per time step, which corresponds to the limit $\gamma \to 1$.

3 Feasible state-action frequencies

It is well known that $R(\pi) = \langle r, \eta^\pi \rangle_{\mathcal{S}\times\mathcal{A}}$, where $\eta^\pi(s,a) = (1-\gamma)\sum_{t\geq 0} \gamma^t \mathbb{P}^{\pi,\mu}(s_t = s, a_t = a)$ is the *state-action frequency* of π. We denote the set of feasible state-action frequencies in the fully observable and in the partially observable case by $\mathcal{N}$ and $\mathcal{N}_\beta$, respectively. Hence, instead of solving the reward maximization problem over π, one can also solve the *reward maximization problem in state-action space*, which is given by

$$\text{maximize } \langle r, \eta \rangle \quad \text{subject to } \eta \in \mathcal{N}_\beta \subseteq \Delta_{\mathcal{S}\times\mathcal{A}}. \tag{2}$$

To do this in practice, one requires a suitable characterization of the set of feasible state-action frequencies $\mathcal{N}_\beta$. To describe this set via polynomial conditions, we introduce the *effective policy poltyope*[1] $\Delta_{\mathcal{A}}^{\mathcal{S},\beta} \coloneqq \{\pi \circ \beta : \pi \in \Delta_{\mathcal{A}}^{\mathcal{O}}\}$, which is the set of state policies that can be realized by selecting the actions based on observations made according to β. Note that the set of effective policies $\Delta_{\mathcal{A}}^{\mathcal{S},\beta}$ is a polytope since it is the image of the polytope $\Delta_{\mathcal{A}}^{\mathcal{O}}$ under the linear map $\pi \mapsto \pi \circ \beta$.

The state-action frequencies of an MDP As mentioned before, the state-action frequencies of an MDP form a polytope $\mathcal{N} = \{\eta \in \mathbb{R}_{\geq 0} : \ell_s(\eta) = 0 \text{ for } s \in \mathcal{S}\} \subseteq \Delta_{\mathcal{S}\times\mathcal{A}}$, where $\ell_s(\eta) = \langle \delta_s \otimes \mathbb{1}_{\mathcal{A}} - \gamma\alpha(s|\cdot,\cdot), \eta\rangle_{\mathcal{S}\times\mathcal{A}} - (1-\gamma)\mu_s$; see [Der70]. Hence for MDPs the reward maximization problem (2) becomes a linear program in state-action frequency space. This is known as the dual linear programming formulation of MDPs [Kal94].

The state-action frequencies of a POMDP The effective policy corresponding to a state-action frequency can be computed by conditioning. In order for the conditioning to be well defined, we require the following assumption, which holds for instance if the initial distribution μ has full support.

Assumption 1. For any state-action frequency $\eta \in \mathcal{N}$ and any state $s \in \mathcal{S}$ it holds that $\sum_a \eta_{sa} > 0$. In the mean reward case we further assume that for every policy $\pi \in \Delta_{\mathcal{A}}^{\mathcal{O}}$ there exists a unique stationary distribution of P_π. This is a standard assumption in MDP linear programming [Kal94] and necessary for the convergence of PG methods [MXSS20].

The correspondence of state-action frequencies $\eta \in \mathcal{N}$ and state policies $\tau \in \Delta_{\mathcal{A}}^{\mathcal{S}}$ via conditioning provides a correspondence of polynomial inequalities in the two sets $\Delta_{\mathcal{A}}^{\mathcal{S}}$ and $\mathcal{N}$. More precisely, setting $S \coloneqq \{s \in \mathcal{S} : b_{sa} \neq 0 \text{ for some } a \in \mathcal{A}\}$ it holds that

$$\sum_{s,a} b_{sa}\tau_{sa} \geq 0 \quad \text{if and only if} \quad \sum_{s\in S}\sum_a b_{sa}\eta_{sa} \prod_{s'\in S\setminus\{s\}} \sum_{a'} \eta_{s'a'} \geq 0. \tag{3}$$

Therefore, the set of feasible state-action frequencies $\mathcal{N}_\beta$ can be described by finitely many polynomial (in)equalities corresponding to the linear (in)equalities describing $\Delta_{\mathcal{A}}^{\mathcal{S},\beta}$ in $\Delta_{\mathcal{A}}^{\mathcal{S}}$. In particular, this shows that solving the reward optimization problem in infinite-horizon POMDPs with memoryless stochastic policies is equivalent to a polynomially constrained optimization problem with linear objective, which generalizes the lienar program associated to MDPs. This formulation was obtained in [MM22] and was used to establish upper bounds on the number of critical points of the reward optimization problem. In this work, we focus the solution of POMDPs via this polynomial program.

4 Reward optimization in state-action frequency space (ROSA)

We formulate our approach for **R**eward **O**ptimization in **S**tate-**A**ction space (ROSA) in Algorithm 1. The two non-trivial steps in the algorithm are the computation of the defining linear inequalities of the polytope $\Delta_{\mathcal{A}}^{\mathcal{S},\beta}$ in line 4 and the solution of the constrained optimization problem in line 6.

Computing the polynomial constraints The linear inequalities defining $\Delta_{\mathcal{A}}^{\mathcal{S},\beta}$ and therefore the polynomial constraints of $\mathcal{N}_\beta$ can be computed in closed form if β has linear independent columns [MM22, Thm. 12]. If this is not the case, they can be computed algorithmically using Fourier-Motzkin elimination, block elimination, vertex approaches, or equality set projection [JKM04]. Let us discuss the special case of deterministic observations. We associated β with a mapping $\mathcal{S} \to \mathcal{O}$, which partitions the state space into sets $S_o \coloneqq \{s \in \mathcal{S} : \beta(o|s) = 1\} \subseteq \mathcal{S}$. If we fix an arbitrary action $a_0 \in \mathcal{A}$ and

[1] Here, $\pi \circ \beta$ denotes the composition of the Markov kernels given by $(\pi \circ \beta)(a|s) \coloneqq \sum_{o\in\mathcal{O}} \pi(a|o)\beta(o|s)$.

Algorithm 1 Reward **O**ptimization in **S**tate-**A**ction space (ROSA)

Require: $\alpha \in \Delta_{\mathcal{S}}^{\mathcal{S}\times\mathcal{A}}$, $\beta \in \Delta_{\mathcal{O}}^{\mathcal{S}}$, $\gamma \in (0,1)$, $\mu \in \Delta_{\mathcal{S}}$

for all $s \in \mathcal{S}$ **do**
  $\ell_s(\eta) \leftarrow \langle \delta_s \otimes \mathbb{1}_{\mathcal{A}} - \gamma\alpha(s|\cdot,\cdot), \eta\rangle_{\mathcal{S}\times\mathcal{A}} - (1-\gamma)\mu_s$ ▷ Define the linear equalities
end for
Compute the defining linear inequalities of $\Delta_{\mathcal{A}}^{\mathcal{S},\beta}$ ▷ In closed form or algorithmically
Compute the defining polynomial inequalities $p_i(\eta) \geq 0$ of $\mathcal{N}_\beta$ ▷ Can be done according to (3)
$\eta^* \leftarrow \arg\max\langle r, \eta\rangle$ sbj to $\eta \geq 0$, $\ell_s(\eta) = 0$, $p_i(\eta) \geq 0$ ▷ Solve the constrained maximization problem
$R^* \leftarrow \langle r, \eta^*\rangle$ ▷ Evaluate the optimal value
$\tau^* \leftarrow \eta^*(\cdot|\cdot) \in \Delta_{\mathcal{A}}^{\mathcal{S}}$ ▷ Compute an optimal state policy
$\pi^* \leftarrow$ solution of $\beta\pi = \tau^*$ ▷ Compute an optimal observation policy
return η^*, R^*, π^* ▷ maximizer, optimal value, optimal policy

arbitrary states $s_o \in S_o, o \in \mathcal{O}$, the polynomial equations cutting out the set $\mathcal{N}_\beta$ of feasible state-action frequencies from the set $\mathcal{N}$ of all state-action frequencies are given by

$$p_{sa}^o(\eta) \coloneqq \eta_{s_o a}\sum_{a'}\eta_{sa'} - \eta_{sa}\sum_{a'}\eta_{s_o a'} = \sum_{a'\neq a}(\eta_{s_o a}\eta_{sa'} - \eta_{s_o a'}\eta_{sa}) = 0, \tag{4}$$

for all actions $a \in \mathcal{A}\setminus\{a_0\}$, states $s \in S_o \setminus \{s_o\}$ and observations $o \in \mathcal{O}$. Hence, for deterministic observations the reward maximization problem in state-action space takes the form

$$\text{maximize } \langle r, \eta\rangle \quad \text{subject to } \begin{cases} \ell_s(\eta) = 0 & \text{for } s \in \mathcal{S} \\ p_{sa}^o(\eta) = 0 & \text{for } o \in \mathcal{O}, a \in \mathcal{A}\setminus\{\tilde{a}\}, s \in S_o \setminus \{s_o\} \\ \eta_{sa} \geq 0 & \text{for } s \in \mathcal{S}, a \in \mathcal{A}. \end{cases} \tag{5}$$

This is a problem in $|\mathcal{S}||\mathcal{A}|$ variables with $|\mathcal{S}|$ linear and $\sum_o(|S_o|-1)(|\mathcal{A}|-1) = (|\mathcal{S}|-|\mathcal{O}|)(|\mathcal{A}|-1)$ quadratic equality constraints and $|\mathcal{S}||\mathcal{A}|$ inequality constraints (of which only $|\mathcal{O}||\mathcal{A}|$ are non redundant).

Implementation and solution of the optimization problem We provide a `Julia` [BEKS17] implementation of ROSA for deterministic observations. In general, problem (5) can be solved with any constrainted optimization solver. Our implementation is built on `Ipopt`, an interior point line search method [WB06]. We call `Ipopt` via the modeling language `JuMP` in which the constraints are easy to implement [DHL17]. The implementation is available under `https://github.com/muellerjohannes/POMDPs-ROSA`.

5 Experiments

To demonstrate the performance of ROSA we test it on navigation problems in mazes. For this, we generate connected mazes using a random depth first search [maz]. Then we randomly select a state as the goal state at which a reward of $|\mathcal{S}|$ is picked up and from which the agent transitions to a uniform state. For all other states four actions move the agent right, left, up or down. The agent can only observe the 8 neighboring cells and starts at a uniform position.

We compare against two other optimization approaches. First, we consider tabular softmax policies and directly optimize the parameters for the exact discounted reward. Instead of a vanilla policy gradient ascent, we use L-BFGS, which is a first order method that estimates second order information. In comparison to a naive policy gradient, we observed L-BFGS to converge faster. We refer to this approach as direct policy optimization (DPO). As a second baseline we consider the reformulation of the reward maximization problem as a quadratically constrained linear program [ABZ06]

$$\text{maximize } \langle \mu, v\rangle \quad \text{subject to } \pi \in \Delta_{\mathcal{A}}^{\mathcal{O}} \text{ and } v = \gamma p_\pi v + (1-\gamma)r_\pi, \tag{6}$$

where $p_\pi(s'|s) \coloneqq \sum_{o,a}\pi(a|o)\beta(o|s)\alpha(s'|s,a)$ and $r_\pi(s) \coloneqq \sum_{a,o} r(s,a)\pi(a|o)\beta(o|s)$. Note that here the constraint is on the value function. We use `Ipopt` to solve (6). We call this approach *Bellman constrained programming* (BCP).

In order to compare the running times of the three approaches, we generate square mazes of side length $2n-1$ and $2n^2-1$ states, for $n = 2, \ldots, 10$. We solve the POMDPs for a discount factor of $\gamma = 0.9999$ using ROSA, BCP and DPO for 10^2 different mazes of each size[2] and report the mean solution times and achieved rewards as well as their 16% and 84% quantiles in Figure 2. We observe that all three methods achieve comparable rewards. However, DPO becomes inefficient even for problems of moderate size and the running time of BCP grows significantly faster compared to ROSA.

[2]For DPO we solved only 20 mazes of each size due to the long solution time.

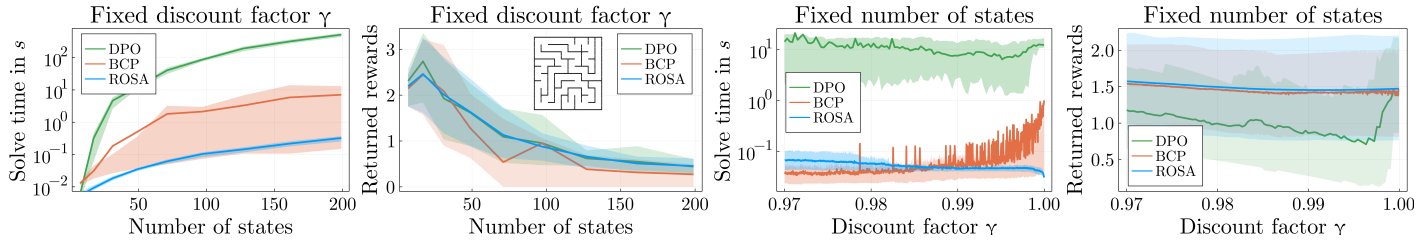


Figure 2: Shown is the solution time and cumulative reward obtained by different methods solving navigation tasks depending on the number of states or the discount factor. Inset shows one of the mazes with 199 states. ROSA reaches higher reward in less time, with stability improvements and time savings becoming more pronounced for larger γ.

To evaluate the performance of ROSA for $\gamma \to 1$ we solve 10^2 mazes[3] with side length 9 and 49 states for increasing discount factors. We report the average solution times and achieved reward in Figure 2. In the comparison of the rewards, examples where BCP did not converge are excluded. In these experiments we see that BCP becomes unstable, whereas the solution time of ROSA appears to be very robust and even decrease for $\gamma \to 1$. In fact, in the solution of (6) `Ipopt` fails to converge to local optimality for about 15% of all problems with discount factor at least 0.9999.

References

[ABZ06] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein, *Solving POMDPs using quadratically constrained linear programs*, Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, 2006, pp. 341–343.

[AKLM21] Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan, *On the theory of policy gradient methods: Optimality, approximation, and distribution shift*, Journal of Machine Learning Research **22** (2021), no. 98, 1–76.

[AYA18] Kamyar Azizzadenesheli, Yisong Yue, and Animashree Anandkumar, *Policy gradient in partially observable environments: Approximation and convergence*, arXiv preprint arXiv:1810.07900 (2018).

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah, *Julia: A fresh approach to numerical computing*, SIAM review **59** (2017), no. 1, 65–98.

[CCC+21] Shicong Cen, Chen Cheng, Yuxin Chen, Yuting Wei, and Yuejie Chi, *Fast global convergence of natural policy gradient methods with entropy regularization*, Operations Research (2021).

[Der70] C. Derman, *Finite state Markovian decision processes*, Tech. report, 1970.

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin, *JuMP: A modeling language for mathematical optimization*, SIAM review **59** (2017), no. 2, 295–320.

[JKM04] Colin Jones, E. C. Kerrigan, and Jan Maciejowski, *Equality set projection: A new algorithm for the projection of polytopes in halfspace representation*, 45.

[Kal94] L.C.M. Kallenberg, *Survey of linear programming for standard and nonstandard Markovian control problems. Part I: Theory*, Zeitschrift für Operations Research **40** (1994), no. 1, 1–42.

[maz] *Maze generation*, `https://rosettacode.org/wiki/Maze_generation`, Accessed: 2022-01-10.

[MM22] Johannes Müller and Guido Montúfar, *The Geometry of Memoryless Stochastic Policy Optimization in Infinite-Horizon POMDPs*, International Conference on Learning Representations, 2022.

[MXSS20] Jincheng Mei, Chenjun Xiao, Csaba Szepesvari, and Dale Schuurmans, *On the global convergence rates of softmax policy gradient methods*, International Conference on Machine Learning, PMLR, 2020, pp. 6820–6829.

[PRB15] Matteo Pirotta, Marcello Restelli, and Luca Bascetta, *Policy gradient in Lipschitz Markov decision processes*, Machine Learning **100** (2015), no. 2, 255–283.

[PY15] Ian Post and Yinyu Ye, *The simplex method is strongly polynomial for deterministic Markov decision processes*, Mathematics of Operations Research **40** (2015), no. 4, 859–868.

[SMS+99] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al., *Policy gradient methods for reinforcement learning with function approximation*, NIPs, vol. 99, Citeseer, 1999, pp. 1057–1063.

[Wag11] Paul Wagner, *A reinterpretation of the policy oscillation phenomenon in approximate policy iteration*, Advances in Neural Information Processing Systems **24** (2011).

[WB06] Andreas Wächter and Lorenz T Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical programming **106** (2006), no. 1, 25–57.

[3]For DPO we consider only 30 mazes and 10^2 values of γ. 439

On Directing Behavior to Learn Collections of Subtasks

Prabhat Nagarajan, Chunlok Lo, Daniela Teodorescu
Department of Computing Science
University of Alberta
{nagarajan,chunlok,dteodore} @ualberta.ca

Adam White, Martha White
Department of Computing Science
University of Alberta
{amw8,whitem}@ualberta.ca

Abstract

In this paper, we explore the problem of learning a collection of subtasks in parallel from a single stream of experience. Our motivating hypothesis is that successful goal-achieving continual learning agents in complex environments are ones that can perform tasks and predictions within their environment. We formalize the problem of effectively gathering this experience as a large Markov decision process. We then provide an alternative problem setting that simplifies this complex problem into a tractable non-stationary reinforcement learning problem. In this setting, we have a behavior learner which learns to act in the environment and acquire experiences. Then, subtask learners update their parameters off-policy in parallel using these experiences generated by the behavior learner. The behavior learner's goal is to gather experiences so as to optimize subtask learning. We then investigate the role of replay on the behavior learner, and propose a simple new method, *non-stationary replay*, as a potential improvement on standard replay in this non-stationary setting.

Keywords: reinforcement learning; continual learning; subtasks; general value functions;

Acknowledgements

We would like to thank the Alberta Machine Intelligence Institute and Canada CIFAR AI Chairs Program for the funding for this research.

1 Introduction

For complex tasks, reward alone need not necessarily be the only form of feedback to drive learning of reward-maximizing behavior. The primary goal of the agent is to maximize cumulative rewards, and the reward function provides evaluative feedback of the agent's performance. However, reward is one of many learning signals, and the environment is often rich with feedback and learning signals about the world. The hypothesis is that effective reinforcement learning (RL) agents should have a general understanding of the world, and be able to perform tasks and predictions about its environment which will ultimately help the agent achieve the primary task of maximizing reward.

One such hypothesis on how to represent knowledge of the environment is through solutions to RL subtasks [Sutton et al., 2011], such as prediction or control general value functions. However, an open question is how an agent can efficiently gather useful data to learn a large set of subtasks in parallel. Ideally, an agent is able to select actions in the world that leads it to gather important data needed for its subtasks learners, while not neglecting learning of other subtasks, such that the entire collection of subtask learners are learned in the most efficient way possible in an off-policy manner from a single stream of experience.

In this paper, we answer two questions:

1. How can we formulate the problem of learning a collection of subtasks from a single stream of experience in a continual learning setting?
2. How can agent direct its experience to be more efficient at learning a collection of subtasks?

For the former, we formalize this problem as an RL problem with the rewards serving as a measure of learning progress on the subtasks. Towards the latter, we perform a preliminary investigation of the use of replay in this problem setting so the agent can rapidly adapt to the changing non-stationary learning signal that the behavior receives, and introduce non-stationary replay to tackle the problems that standard replay faces with non-stationary rewards.

2 Problem Formulation

2.1 Contextualizing the Problem Setting

In our problem setting, which we consider from McLeod et al. [2021], an agent is *continually* interacting and learning in a complex environment to accumulate knowledge about the world to facilitate reward maximization. This knowledge can be in many forms, such as transition dynamics models, options, skills, and general value functions. Learning these *subtasks* helps the agent progressively improve on its primary task, which is to maximize the (external) reward signal in the environment. The agent needs to discover useful subtasks and efficiently learn these subtasks to help it efficiently improve on the primary task. This larger continual learning problem setting has many components: subtask discovery, directed exploration to learn about both the subtasks and the primary task, and sample efficient update algorithms to learn the subtasks and the primary task.

To make progress towards addressing this larger problem setting, it is fruitful to study each component in isolation to develop a better understanding of these sub-problems before building a unified system that addresses all of these challenges. In this paper, we focus on the exploration aspect of learning a collection of subtasks, and leave the questions of subtask discovery and how to integrate subtask learning with primary task learning for future work.

2.2 Formalization

We assume the agent interacts in a Markov decision process $\mathcal{M}$ defined by a set of states $\mathcal{S}$, set of actions $\mathcal{A}$, and transition probability function $P(s'|s,a)$, where the set of states and actions can be continuous. We omit the reward function as we are not considering the primary task.

We formulate the *continual subtask learning problem* as an RL problem. The agent has a set of subtasks $\{T_j\}_{j=1}^N$, with each with T_j consisting of an objective function $J_j : \Omega_j \to \mathbb{R}$ that is dependent on a set of weights $\theta^{(j)} \in \Omega_j$. For each subtask T_j, the agent has an associated subtask learner l_j which attempts to update $\theta^{(j)}$ such that J_j is minimized. Each subtask learner may also contain a set of internal parameters $\phi^{(j)} \in \Phi_j$ that is updated and affects the learner's updates to $\theta^{(j)}$, which may be replay buffers, eligibility traces, per-weight step-sizes, or more. Subtasks can be quite general, including prediction tasks, control tasks, or even supervised learning tasks, as long as they are tasks learnable from observations in the MDP. In the continual subtask learning problem, the agent's objective is to minimize the objective functions across subtasks $\sum_{j=1}^N J_j(\theta_t^j)$. We call an algorithm that learns to acquire experiences to minimize this objective a behavior learner. This is in contrast to a subtask learner which learns subtasks from off-policy experiences generated by the behavior learner. That is, a behavior learner aims to visit states that benefit the subtask learner's learning. Both the

behavior learner and subtask learners can be considered to be part of the agent, but the behavior learner's parameters more directly impact the states that are visited.

The behavior learner's task of learning this collection of subtasks can be formalized in an alternative, augmented MDP. This augmented MDP $\mathcal{M}'$ is formalized as a tuple $(\mathcal{S}', \mathcal{A}, P', R')$ where the state consists of both the environment state and the state of all its subtask learners. That is, $\mathcal{S}' = \mathcal{S} \times \prod_{j=1}^{N} \Omega_j \times \prod_{j=1}^{N} \Phi_j$, where $(s_t, \theta_t^{(1)}, \phi_t^{(1)}, ..., \theta_t^{(N)}, \phi_t^{(N)}) = \tilde{s}_t \in \mathcal{S}'$. In this formulation, P' is a combination of the environment transition dynamics P and all the learning updates l_j. At each timestep, each subtask learner updates its weights and internal parameters using the current environment transition: $l_j(s_t, a_t, s_{t+1}, \theta_t^{(j)}, \phi_t^{(j)}) = \theta_{t+1}^{(j)}, \phi_{t+1}^{(j)}$. The reward $R'(\tilde{s}_t, a, \tilde{s}'_{t+1}) = \sum_{j=1}^{N} \Delta_{t+1}^{j}$, where $\Delta_{t+1}^{j} = J_j(\theta_t^{(j)}) - J_j(\theta_{t+1}^{(j)})$, representing learning progress after a timestep.

2.3 Simplified Problem Setting

The above problem setting is Markov, and there is some optimal policy that will choose actions based on the parameters and the environment state so as to best minimize the objective functions of the subtasks. Unfortunately, finding such an optimal policy is difficult for several reasons. The parameter and action space of this augmented MDP is prohibitively large, and grows as the number of subtasks increases. Moreover, the reward function which represents progress towards minimizing the objectives, e.g., computing changes in the objective function, can be quite expensive to compute. For example, when learning the value function for a policy, we can use stochastic algorithms that minimize the (projected) Bellman error over many iterations, even though it is expensive to actually compute this full objective. As such, we simplify this complex augmented MDP into a non-stationary, or partially observable setting to make this complex problem tractable while still performing well.

First, the objective is difficult to measure directly at each step, as $J_j(\theta_t^{(j)})$ may be expensive to compute. Since the objective $J_j(\theta_t^{(j)})$ is difficult to measure directly at each timestep, we use a proxy reward signal that encourages the behavior learner to achieve this objective. A simple choice for this *intrinsic reward* is the cumulative change in the weights for the subtasks: after taking action A_t from S_t and transitioning to S_{t+1}, the agent receives intrinsic reward

$$R_{t+1} = \sum_{\mathrm{J}=1}^{N} \|\theta(j)_{t+1} - \theta(j)_t\|_1,$$

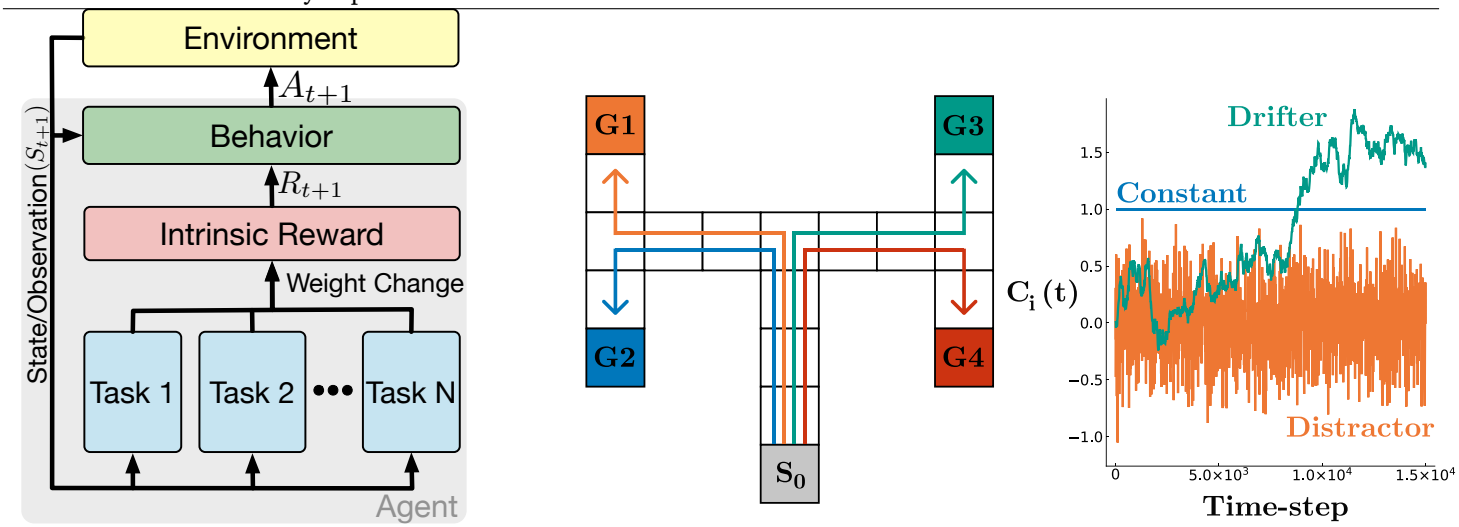
where we assume that $\theta_{t+1}^{(j)}$ is computed from $\theta_t^{(j)}$ using the data generated from transition (S_t, A_t, S_{t+1}). This intrinsic reward reflects that, if the weights changed, then the data generated by taking that action induced learning for that subtask. In principle, we can select some other estimator of learning progress, but our choice is based off of empirical results [Linke et al., 2020] that suggest that weight change is a good measure of learning progress to effectively drive adaptation of the agent's behavior. To further simplify the setting, we additionally assume that the subtask learners are online learners. That is, we require that the subtask learners learn and produce a weight change at each timestep. Subtask learners that update infrequently, e.g., after every 100 timesteps, can make credit assignment extremely difficult for the behavior learner because it must attribute the intrinsic rewards to specific states across a longer time interval. Note that this simplified problem setting is inherently non-stationary because the intrinsic reward at each time step changes depending on the state of the subtask learners, which is not part of the state.

Figure 1a depicts our overall system. Our subtask learners produce weight changes, which are aggregated into an intrinsic reward for our agent. The agent then interacts with the environment to acquire new experiences which are used to train the subtask learners.

3 Non-stationary Replay (NS-Replay)

Experience replay [Lin, 1992] has long been used to increase the sample efficiency of agents by reusing previously experienced transitions. However, one issue with experience replay in a non-stationary reward setting is that the rewards associated with old transitions are stale due to the non-stationary nature of the environment. This causes the replay sample to inaccurately reflect the current environment, negatively affecting learning especially if the buffer is large and the samples are very outdated.

Non-stationary replay (NS-Replay) consists of the experience replay buffer and a separately learned reward model. This reward model is repeatedly updated online based on observed rewards. When a sample is drawn from the buffer, the experienced reward is replaced with the reward model's predicted reward. The goal is that the reward model's prediction will be more reflective of the true reward that would be given by the environment if the agent were to experience that transition online. In this paper, our reward model is a simple reward memorizer for tabular setting, where the predicted reward for a transition is the last reward experienced for that transition.



(a) The Continual Subtask Learning problem.

(b) Tmaze environment.

Figure 1: **(a)** shows the architectural overview of the RL system **(b)** The left plot shows an overview of the TMaze, and the right plot shows an example of the cumulant schedule used for each GVF at the goals. In our experiment, GVF questions for goal 2 and 4 have constant cumulants of 10, while the GVF question for goal 1 has a distractor cumulant $N(1, 5)$, and the GVF question for goal 3 has a drifting cumulant with a random walk behavior with random step drawn $N(0, 0.0317)$.

4 Experiments

We ran various behavior learners in the TMaze depicted in Figure 1b. The agent's four subtasks are general value function prediction (GVF) problems, where the policy is the shortest-path policy to each goal state. The pseudo-termination functions for the GVFs terminate at the respective goal states. The cumulants for the GVFs are 0 at all states other than goal states. At goal states G2 and G4, the cumulants are constant values of 10. At G1, the cumulant is drawn from a normal random distribution. At G3, the cumulant is constant but nonstationary, changing according to a random walk at each timestep.

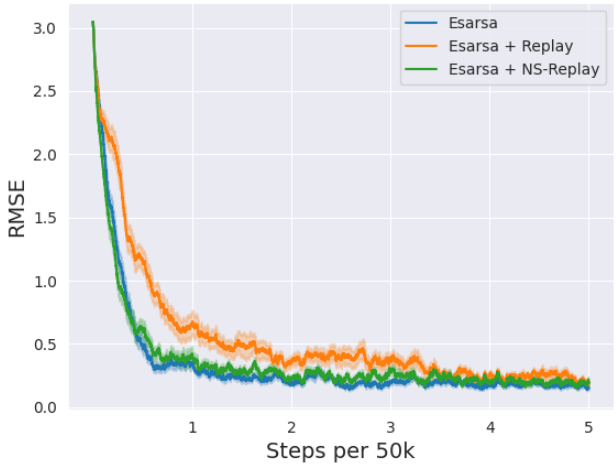
Our behavior agents are Expected Sarsa, Expected Sarsa + Replay, and Expected Sarsa + NS-Replay. For all experiments, our subtask learners uses TB(λ) with the Auto optimizer [McLeod et al., 2021] to learn the GVF subtasks. We set the replay buffer capacity to 10,000, and the number of replay steps per timestep to 16. Our results over 30 runs are depicted in Figure 2, with a 95% confidence interval.

We find that while the final predictions are similar between all three methods, Expected Sarsa and Expected Sarsa with NS-Replay outperform Expected Sarsa with Replay during the first 60k steps, with close performance between Expected Sarsa and Expected Sarsa with NS-Replay. We hypothesize the NS-Replay may outperform Expected Sarsa with a reward model that is more sophisticated than replaying the last seen reward at that state.

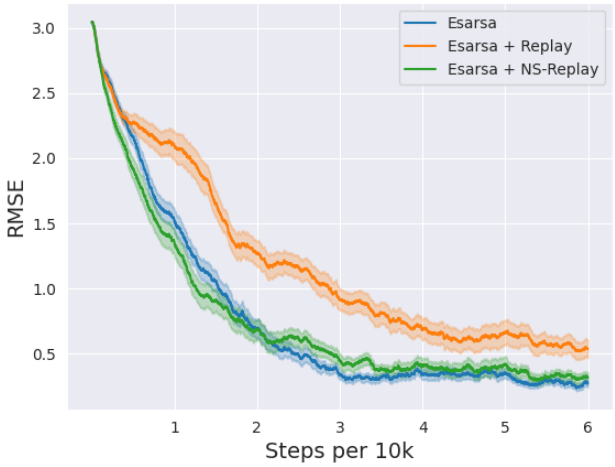
5 Related Work

This work primarily extends from the work and formulation of McLeod et al. [2021]. Linke et al. [2020] studies our research problem in a bandit setting, and explores the effectiveness of different types of intrinsic rewards. Our work also has connections to curiosity-based exploration Pathak et al. [2017], which explores skill learning in the absence of an extrinsic reward. However, our problem setting is related to areas such as unsupervised RL or skill learning, curiosity-based exploration, the most similar setting was to our knowledge introduced by Sutton et al. [2011]. Our work is also related to goal-conditioned RL Schaul et al. [2015], though in goal-conditioned RL the state is usually parameterized by some goal that the agent should be achieving, whereas we are learning about multiple tasks in parallel and do not have goal parameterizations.

Riedmiller et al. [2018] introduces Scheduled Auxiliary Control which learns tasks and learns to schedule and execute these tasks. Veeriah et al. [2019] focuses on GVF discovery and on on-policy learning of GVFs, as opposed to our focus on off-policy parallel learning.



(a) Comparison between behavior learning algorithms.



(b) First 60k timesteps

Figure 2: Results from experiments in the TMaze with different behavior learning algorithms. **(a)** shows the average RMSE of each prediction learner at each timestep for Expected Sarsa, Expected Sarsa with Experience Replay, and Expected Sarsa with NS-Replay. Note that while the final performance between all 3 algorithms are similar, the initial performance of Expected Sarsa + NS-Replay and Expected Sarsa outperforms Expected Sarsa + Replay over the first 60k steps as shown in **(b)**.

References

Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

Cam Linke, Nadia M Ady, Martha White, Thomas Degris, and Adam White. Adapting behavior via intrinsic reward: a survey and empirical study. *Journal of Artificial Intelligence Research*, 69:1287–1332, 2020.

Matthew McLeod, Chunlok Lo, Matthew Schlegel, Andrew Jacobsen, Raksha Kumaraswamy, Martha White, and Adam White. Continual auxiliary task learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

World Value Functions: Knowledge Representation for Multitask Reinforcement Learning

Geraud Nangue Tasse, Steven James, Benjamin Rosman
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
geraud.nanguetasse1@students.wits.ac.za, {steven.james,benjamin.rosman1}@wits.ac.za

Abstract

An open problem in artificial intelligence is how to learn and represent knowledge that is sufficient for a general agent that needs to solve multiple tasks in a given world. In this work we propose world value functions (WVFs), which are a type of general value function with mastery of the world—they represent not only how to solve a given task, but also how to solve any other goal-reaching task. To achieve this, we equip the agent with an internal goal space defined as all the world states where it experiences a terminal transition—a task outcome. The agent can then modify task rewards to define its own reward function, which provably drives it to learn how to achieve all achievable internal goals, and the value of doing so in the current task. We demonstrate a number of benefits of WVFs. When the agent's internal goal space is the entire state space, we demonstrate that the transition function can be inferred from the learned WVF, which allows the agent to plan using learned value functions. Additionally, we show that for tasks in the same world, a pretrained agent that has learned any WVF can then infer the policy and value function for any new task directly from its rewards. Finally, an important property for long-lived agents is the ability to reuse existing knowledge to solve new tasks. Using WVFs as the knowledge representation for learned tasks, we show that an agent is able to solve their logical combination zero-shot, resulting in a combinatorially increasing number of skills throughout their lifetime.

Keywords: reinforcement learning, multitask, transfer, logic, composition

1 Introduction

The ultimate goal of artificial intelligence is to create general agents capable of solving multiple tasks in the world in the same way that humans are able to. To achieve this, we need a general decision-making framework for agents that interact with a world to solve tasks, and a sufficiently general knowledge representation for what those agents learn. Reinforcement learning (RL) [2] is one such framework that has made several major breakthroughs in recent years, ranging from robotics to board games, but these agents typically are narrowly designed to solve only a single task.

In RL, tasks are specified through a reward function from which the agent receives feedback. Most commonly, an agent represents its knowledge in the form of a value function, which represents the sum of future rewards it expects to receive. However, since the value function is directly tied to one single reward function (and hence task), it is definitionally insufficient for constructing agents capable of solving a wide range of tasks. We seek to overcome this limitation by proposing *world value functions* (WVFs), a knowledge representation that encodes how to solve not just the current task, but any other task, in the world. In the literature, agents with such abilities are said to possess *mastery* [5]. Importantly, WVFs are a form of general value function [3] that can be learned from a single stream of experience like standard value functions—no additional information need be provided to the agent.

In this work, we define the WVF and demonstrate how it can be learned using standard RL algorithms. We then demonstrate several advantages of learning such a representation. In particular, we show that (a) WVFs implicitly encode the dynamics of the world and can be used for model-based RL; (b) having learned a WVF, any new task that an agent encounters can be solved by estimating its reward function, reducing the problem to a supervised learning one; and (c) WVFs can be *composed* with Boolean operators to produce novel and human-interpretable behaviours.

2 World Value Functions

We model the agent's interaction with the world as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, R)$, where (i) $\mathcal{S}$ is the state space, (ii) $\mathcal{A}$ is the action space, (iii) $P(s, a, s')$ are the transition dynamics of the world, and (iv) R is a reward function bounded by $[R_{\text{MIN}}, R_{\text{MAX}}]$, representing the task the agent needs to solve. Note that in this work, we focus on environments with *deterministic* dynamics, but put no restrictions on their complexity.

The agent's aim is to compute a *policy* π from $\mathcal{S}$ to $\mathcal{A}$ that optimally solves a given task. This is often achieved through a value function that represents the expected return obtained under π starting from state s: $V^\pi(s) = \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} r(s_t, a_t, s_{t+1})\right]$. Similarly, a Q-value function $Q^\pi(s, a)$ represents the expected return obtained by executing a from s, and thereafter following π. The optimal Q-value function is given by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ for all states s and actions a, and the optimal policy follows by acting greedily with respect to Q^* at each state.

2.1 A New Form of General Value Function

We now introduce *world value functions* (WVFs), which provably encode how to reach all achievable goals in a world with arbitrary task rewards. We first define the internal goal space $\mathcal{G}$ of the agent as all states where it experiences a terminal transition. Importantly, the goal the agent wishes to achieve is not specified by the world, but rather chosen by itself. In order to simultaneously learn how to achieve the overall task goal and the agent's internal goals, the agent can define its own reward function $\bar{R}$, which extends the task rewards R to simply penalise itself for achieving goals it did not intend to:

$$\bar{R}(s, g, a, s') \coloneqq \begin{cases} \bar{R}_{\text{MIN}} & \text{if } g \neq s \text{ and } s' \text{ is absorbing} \\ R(s, a, s') & \text{otherwise,} \end{cases}$$

Algorithm 1: Q-learning for WVFs

Initialise: WVF $\bar{Q}$, goal buffer $\mathcal{G}$
foreach *episode* **do**
  Observe an initial state $s \in \mathcal{S}$ and sample a goal $g \in \mathcal{G}$
  while *episode is not done* **do**
    $a \leftarrow \begin{cases} \arg\max_{a \in \mathcal{A}} \bar{Q}(s, g, a) & \text{probability } 1 - \varepsilon \\ \text{a random action} & \text{probability } \varepsilon \end{cases}$
    Take action a, observe reward r and next state s'
    if s' *is absorbing* **then** $\mathcal{G} \leftarrow \mathcal{G} \cup \{s\}$
    for $g' \in \mathcal{G}$ **do**
      $\bar{r} \leftarrow \bar{R}_{\text{MIN}}$ **if** $g' \neq s$ and $s \in \mathcal{G}$ **else** r
      $\bar{Q}(s, g', a) \xleftarrow{\alpha} \left[\bar{r} + \max_{a'} \bar{Q}(s', g', a')\right] - \bar{Q}(s, g', a)$
    $s \leftarrow s'$

where $\bar{R}_{\text{MIN}}$ is a large negative penalty that can be derived from the bounds of the reward function [1]. We can think of the penalty $\bar{R}_{\text{MIN}}$ as adding one bit of information to the agent's rewards, which we will later prove is sufficient for the agent to learn how to achieve its internal goals and the value of achieving them in the current task. The overall goal of the agent now is to compute a *world policy* $\bar{\pi} : \mathcal{S} \times \mathcal{G} \to Pr(\mathcal{A})$ that optimally reaches its internal goal states. Given a world policy $\bar{\pi}$, the corresponding WVF is defined as follows:

$$\bar{Q}^{\bar{\pi}}(s, g, a) \coloneqq \mathbb{E}^{\bar{\pi}}_{s'}\left[\bar{R}(s, g, a, s') + \bar{V}^{\bar{\pi}}(s', g)\right], \text{ where } \bar{V}^{\bar{\pi}}(s, g) \coloneqq \mathbb{E}^{\bar{\pi}}\left[\sum_{t=0}^{\infty} \bar{R}(s_t, g, a_t, s_{t+1})\right].$$

Since the WVF satisfies the Bellman equations, $\bar{Q}^*(s,g,a)$ can be learned using any suitable RL algorithm like Q-learning (Algorithm 1). To show that a WVF does still encode knowledge of how to solve the task in which it was learned, we prove that the regular task reward function and value function can be recovered by simply maximising over goals (Theorem 1). The task policy can then be obtained as: $\pi^*(s) \in \arg\max_{a\in\mathcal{A}} \left(\max_{g\in\mathcal{G}} \bar{Q}^*(s,g,a)\right)$.

Theorem 1. *Let $M = (\mathcal{S}, \mathcal{A}, P, R)$ be a task with optimal Q-value function Q^* and optimal world Q-value function $\bar{Q}^*$. Then for all (s,a) in $\mathcal{S}\times\mathcal{A}$, we have (i) $R(s,a) = \max_{g\in\mathcal{G}} \bar{R}(s,g,a)$, and (ii) $Q^*(s,a) = \max_{g\in\mathcal{G}} \bar{Q}^*(s,g,a)$.*

Having established WVFs as a type of task-specific GVF, we prove in Theorem 2 that they do indeed have mastery—that is, they learn how to reach all reachable goal states in the world. We formally define mastery as follows:

Definition 1. *Let $\bar{Q}^*$ be the optimal world Q-value function for a task M. Then $\bar{Q}^*$ has mastery if for all $g \in \mathcal{G}$ reachable from $s \in \mathcal{S}\setminus\mathcal{G}$, there exists an optimal world policy*

$$\bar{\pi}^*(s,g) \in \arg\max_{a\in\mathcal{A}} \bar{Q}^*(s,g,a) \text{ such that } \bar{\pi}^* \in \arg\max_{\bar{\pi}} P_s^{\bar{\pi}}(s_T = g),$$

where $P_s^{\bar{\pi}}(s_T = g)$ is the probability of reaching g from s under a policy $\bar{\pi}$.

Theorem 2. *Let $\bar{Q}^*$ be the optimal world Q-value function for a task M. Then $\bar{Q}^*$ has mastery.*

Finally, if the agent's goal space is the state space ($\mathcal{G} = \mathcal{S}$), then we can estimate the transition probabilities for each $s,a \in \mathcal{S}\times\mathcal{A}$ using only the reward function and optimal WVF. That is, $P(s,a,s')$ for all $s' \in \mathcal{S}$ can be obtained by simply solving for them in the system of Bellman optimality equations given by each goal $g \in \mathcal{S}$: $\bar{Q}^*(s,g,a) = \sum_{s'\in\mathcal{S}} p(s,a,s')\left[\bar{R}(s,g,a,s') + \bar{V}^*(s',g)\right]$. This shows that optimal WVFs implicitly encode the dynamics of the world. In practice, if the learned WVF is sufficiently optimal in a neighbourhood $\mathcal{N}(s)$, then we only need to consider $s', g \in \mathcal{N}(s)\times\mathcal{N}(s)$ if the transition probabilities are known to be non-zero only in $\mathcal{N}(s)$, common in most domains.

Experiment 2.1. *Consider a 4-rooms gridworld where an agent may be required to visit various goal positions. The agent can move in any of the four cardinal directions at each timestep (with reward -0.1), and also has a "done" action that it chooses to terminate at any position (with reward 2 if it is a task goal). We train an agent on the task where it must learn to navigate to the middle of the top-left or bottom-right rooms. Figure 1 (left) shows the learned WVF. The figure is generated by plotting the value functions for every goal position (since $\mathcal{G} = \mathcal{S}$), and displaying it at their respective xy position. Note how the values with respect the "top-left" and "bottom-right" goals are high (red), reflecting the high rewards the agent received for reaching those goals when it was trying to reach them. Figure 1 (middle) shows a close up view of the learned WVF around the "top-left" goal. We can observe from the value gradient of the plots that the WVF does indeed learn how to reach all positions in the gridworld. We can then maximise over goals to obtain the regular value function and policy (Figure 1 (right)).*

Finally, we demonstrate that the transition probabilities can be inferred from the learned WVF. Figures 2 (left) and (middle) respectively show the transitions inferred by solving the Bellman equations with $s', g \in \mathcal{S}\times\mathcal{S}$ and $s', g \in \mathcal{N}(s)\times\mathcal{N}(s)$. For each, we inferred the next state probabilities for taking each cardinal action at the center of each room, and placed the corresponding arrow in the state with highest probability. The red arrows in Figure 2 (left) correspond to incorrectly inferred next states, which is a consequence of the learned WVF not being near optimal at all states for all goals. Figure 2 (middle) shows that in practice, if the WVF is not near-optimal, we can still infer dynamics by using $s', g \in \mathcal{N}(s)\times\mathcal{N}(s)$. Figure 2 (right) shows sample trajectories for following the optimal policy using the inferred transition probabilities. The gray-scale color of each arrow corresponds to the normalised value prediction for that state.

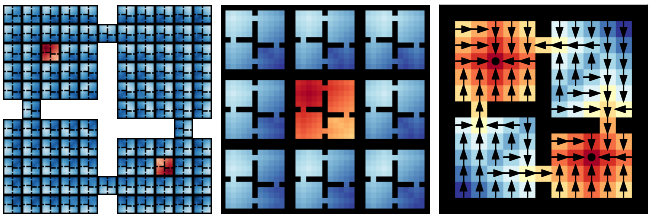


Figure 1: Learned WVF (left), close up of WVF for "top-left" goal (middle), and inferred values and policy (right).

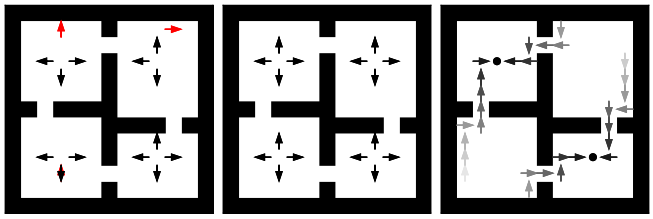


Figure 2: Inferred transitions (left-middle) and imagined rollouts using the learned WVF (right).

3 Benefits of WVFs across tasks in the same world

We now highlight some benefits of WVFs that are a natural result of assuming tasks come from the same world—that is we assume that all tasks share the same state space, action space and dynamics, but differ in their reward functions.

More specifically, we define the world as a background MDP $M_0 = \langle \mathcal{S}_0, \mathcal{A}_0, P_0, R_0 \rangle$ with its own state space, action space, transition dynamics and background reward function. Any individual task M is specified by a task-specific reward function $R^\tau_M(s,a)$ that is non-zero only for transitions entering terminal states. The reward function for the resulting MDP is then simply $R_M(s,a,s') \coloneqq R_0(s,a,s') + R^\tau_M(s,a)$. We denote the set of all these tasks as $\mathcal{M}$ and the set of their corresponding optimal WVFs as $\bar{\mathcal{Q}}^*$.

Interestingly, this definition means that if tasks come from the same world, then their WVFs share the same world policy—that is, the agent has the same notion of goals and how to reach them no matter the task.

Theorem 3. *Let $\bar{\mathcal{Q}}^*$ be the set of optimal world $\bar{Q}$-value functions with mastery of tasks in $\mathcal{M}$. Then for all $s \neq g \in \mathcal{S} \times \mathcal{G}$,*

$$\bar{\pi}^*(s,g) \in \arg\max_{a \in \mathcal{A}} \bar{Q}^*_{M_1}(s,g,a) \iff \bar{\pi}^*(s,g) \in \arg\max_{a \in \mathcal{A}} \bar{Q}^*_{M_2}(s,g,a) \; \forall M_1, M_2 \in \mathcal{M}.$$

Similarly, if we require that the world policies need to be the same across tasks, then we have that the tasks must come from the same world as defined above. In a sense, this gives us a notion of task invariance (the world does not change with changes in task) implying policy conservation (the world policy is constant across tasks), and vice-versa.

3.1 Zero-shot values and policies from rewards

Since each task $M \in \mathcal{M}$ share the same dynamics (and consequently the same world policy), their corresponding WVFs can be written as $\bar{Q}^*_M(s,g,a) = G^*_{s,g,a} + \bar{R}^\tau_M(s',a')$ for some $s', a' \in \mathcal{S} \times \mathcal{A}$, where $G^*_{s,g,a}$ is a constant across tasks that represents the sum of rewards starting from s and taking action a up until g, but not including the terminal reward. Using this fact, we can show that the optimal value function and policy for any task can be obtained zero-shot from an arbitrary WVF given the task-specific rewards:

Theorem 4. *Let R^τ_M be the given task-specific reward function for a task $M \in \mathcal{M}$, and let $\bar{Q}^* \in \bar{\mathcal{Q}}^*$ be an arbitrary WVF. Define*

$$\tilde{\bar{V}}_M(s,g) \coloneqq \max_{a \in \mathcal{A}} \bar{Q}^*(s,g,a) + \left(\max_{a \in \mathcal{A}} R^\tau_M(g,a) - \max_{a \in \mathcal{A}} \bar{Q}^*(g,g,a) \right), \text{ the estimated WVF of } M.$$

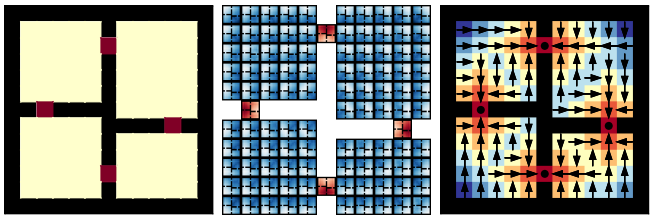
Then,

(i) for all $g \in \mathcal{G}$ reachable from $s \in \mathcal{S}$, $\bar{V}^*_M(s,g) = \tilde{\bar{V}}_M(s,g)$.

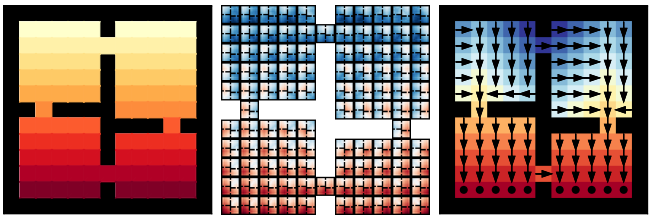
(ii) $V^*_M(s) = \max_{g \in \mathcal{G}} \tilde{\bar{V}}(s,g)$, and $\pi^*_M(s) \in \arg\max_{a \in \mathcal{A}} \bar{Q}^*(s, \arg\max_{g \in \mathcal{G}} \tilde{\bar{V}}_M(s,g), a)$.

This has several important implications for transfer learning. For example, one can learn an arbitrary WVF with unsupervised pretraining, then solve any new task by only learning the reward function (from experience or demonstrations).

Experiment 3.1. *Consider the 4-rooms domain introduced in Experiment 2.1 where the agent has learned the WVF for navigating to the top-left or bottom-right rooms. Figure 3 shows the estimated WVFs, values and policy for each task, computed as per Theorem 4 using the given task rewards and the learned WVF for the bottom-left task. Notice how the WVFs look similar to the rewards. This highlights the structural similarity between the task space and value function space which enables the zero-shot inference of task policies from task rewards alone.*



(a) Navigating to the hallways.



(b) Navigating to the bottom of the grid.

Figure 3: Inferring values and policies zero-shot from goal rewards. From left to right on each figure: The given task specific rewards, the inferred WVF using Theorem 4, and the inferred values and policy from maximising over goals.

3.2 Zero-shot logical composition

We recently proposed a framework for agents to apply logical operations—conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$)—over the space of tasks and WVFs [1]. We first define these logical operations over tasks and WVFs as follows (we

omit the value functions' parameters for readability):

$$R_{M_1 \vee M_2} \coloneqq \max\{R_{M_1}, R_{M_2}\}; \quad R_{M_1 \wedge M_2} \coloneqq \min\{R_{M_1}, R_{M_2}\}; \quad R_{\neg M_1} \coloneqq (R_{\mathcal{M}_{SUP}} + R_{\mathcal{M}_{INF}}) - R_{M_1},$$
$$\bar{Q}^*_{M_1} \vee \bar{Q}^*_{M_2} \coloneqq \max\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\}; \quad \bar{Q}^*_{M_1} \wedge \bar{Q}^*_{M_2} \coloneqq \min\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\}; \quad \neg \bar{Q}^*_{M_1} \coloneqq \left(\bar{Q}^*_{SUP} + \bar{Q}^*_{INF}\right) - \bar{Q}^*_{M_1},$$

where $\bar{Q}^*_{SUP}$ is the goal-oriented value function for the maximum task $R_{\mathcal{M}_{SUP}}$ where $r = R_{\text{MAX}}$ for all $\mathcal{G}$. Similarly $\bar{Q}^*_{INF}$ is goal-oriented value function for the minimum task $R_{\mathcal{M}_{INF}}$ where $r = R_{\text{MIN}}$ for all $\mathcal{G}$.

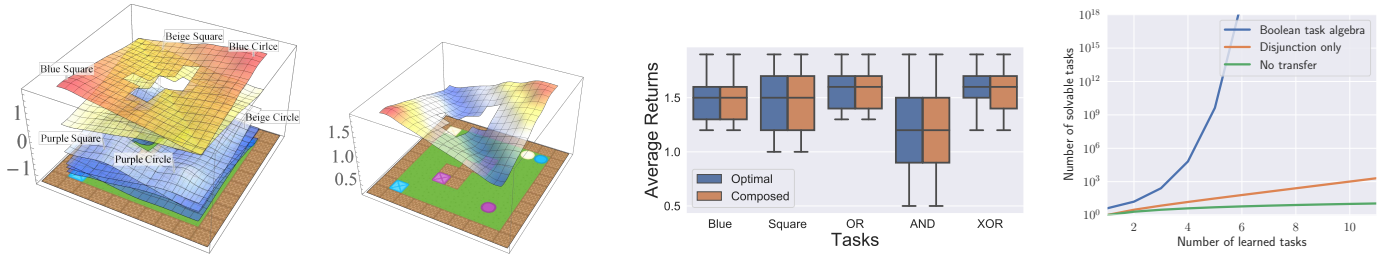
We then showed that the logical composition of WVFs produces exactly the WVF for the corresponding task composition (Theorem 5 (i)). We further showed that when the task specific rewards are binary, $\mathcal{M}$ and $\bar{\mathcal{Q}}^*$ from homomorphic Boolean algebras (Theorem 5 (ii)). This formally defines classical logic over the space of tasks and value functions.

Theorem 5. *Let $\bar{\mathcal{Q}}^*$ be the set of optimal world $\bar{Q}$-value functions for tasks in $\mathcal{M}$. Then,*

(i) for all tasks $M_1, M_2 \in \mathcal{M}$, we have $\bar{Q}^*_{\neg M_1} = \neg \bar{Q}^*_{M_1}$, $\bar{Q}^*_{M_1 \vee M_2} = \bar{Q}^*_{M_1} \vee \bar{Q}^*_{M_2}$, and $\bar{Q}^*_{M_1 \wedge M_2} = \bar{Q}^*_{M_1} \wedge \bar{Q}^*_{M_2}$.

(ii) If the task specific rewards are binary, then $(\mathcal{M}, \vee, \wedge, \neg)$ and $(\bar{\mathcal{Q}}^*, \vee, \wedge, \neg)$ are homomorphic Boolean algebras.

Experiment 3.2. *Consider video game world where an agent must navigate a 2D world and collect objects of different shapes and colours [4]. The state space is an 84×84 RGB image, and the agent is able to move in any of the four cardinal directions. The agent also possesses a `pick-up` action, which allows it to collect an object when standing on top of it. The positions of the agent and objects are randomised at the start of each episode. We first use deep Q-learning (similarly to Algorithm 1) to learn two base tasks: collecting blue objects and collecting squares. Figure 4a shows the values with respect to each goal of the learned WVF for collecting blue objects, and maximising over goals gives us the regular value function (Figure 4b). We can then compose them to solve their disjunction (OR), conjunction (AND) and exlusive-or (XOR), as shown in Figure 4c. In general, after learning n base tasks we can solve all their 2^{2^n} logical compositions, giving agents a super-exponential explosion of skills (Figure 4d).*



(a) Learned WVF for collecting blue objects.

(b) Inferred regular value function for collecting *Blue* objects.

(c) Average returns over 1000 episodes of optimal and composed WVFs.

(d) Super-exponential number of solvable tasks from learned ones.

Figure 4: Experiments in a 2D domain where the agent needs to collect objects of various shapes and colors.

4 Conclusion

For an agent to learn how to solve multiple tasks in a given world from a single stream of experience, it should not only encode knowledge about how to optimally solve the current task, but also how achieve agent-centric goals in the world. We introduced WVFs as a principled knowledge representation that captures this idea. We then showed that WVFs learned on any task have mastery of the world—they encode how to solve all possible goal-reaching tasks. This results in agents that can infer the dynamics of the environment (which can be used for model based approaches like planning), enables them to solve new tasks zero-shot given just their terminal rewards, and finally enables zero-shot logical composition of learned skills (an important ability for the combinatorial explosion of skills in long-lived agents).

References

[1] G. Nangue Tasse, S. James, and B. Rosman. A Boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[2] R. Sutton, A. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[3] R. Sutton, J. Modayil, M. Delp, T. Degris, P. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

[4] B. van Niekerk, S. James, A. Earle, and B. Rosman. Composing value functions in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 6401–6409. PMLR, 09–15 Jun 2019.

[5] V. Veeriah, J. Oh, and S. Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Hierarchical Reinforcement Learning under Mixed Observability

Hai Nguyen*
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
nguyen.hai1@northeastern.edu

Zhihan Yang*
Calerton College
Northfield, MN 55057
yangz2@carleton.edu

Andrea Baisero
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
baisero.a@northeastern.edu

Xiao Ma
National University of Singapore
Singapore
xiao-ma@comp.nus.edu.sg

Robert Platt
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
r.platt@northeastern.edu

Christopher Amato
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
c.amato@northeastern.edu

Abstract

The framework of mixed observable Markov decision processes (MOMDP) models many robotic domains in which some state variables are fully observable while others are not. In this work, we identify a significant subclass of MOMDPs defined by how actions influence the fully observable components of the state and how those, in turn, influence the partially observable components and the rewards. This unique property allows for a two-level hierarchical approach we call HIerarchical Reinforcement Learning under Mixed Observability (HILMO), which restricts partial observability to the top level while the bottom level remains fully observable, enabling higher learning efficiency. The top level produces desired goals to be reached by the bottom level until the task is solved. We further develop theoretical guarantees to show that our approach can achieve optimal and quasi-optimal behavior under mild assumptions. Empirical results on long-horizon continuous control tasks demonstrate the efficacy and efficiency of our approach in terms of improved success rate, sample efficiency, and wall-clock training time. We also deploy policies learned in simulation on a real robot. For a full version of this work, please refer to https://arxiv.org/pdf/2204.00898.pdf.

Keywords: Robot Learning, Hierarchical, Mixed Observability

1 Introduction

Many robotic domains feature a state space that factorizes into high and low observability subspaces, in which actions primarily influence the high observability components of the state. For example, robot navigation with unknown dynamics and noisy sensors to an unknown dynamic target (Fig. 1a), or robot manipulation to reach an unknown target pose, *e.g.*, find an object in cluttered and occluded environments (Fig. 1b), grasp under uncertainty, or collaborate with humans with partially observed human factors (*e.g.*, trust, preferences, or goals). In these examples, state variables such as a robot's position or an arm's pose can be much more certain (*e.g.*, using GPS signals and sensors) than partially observable variables relative to the task and are often fully observable. Moreover, actions directly influence the fully observable state components and indirectly affect partially observable ones. For example, in Fig. 1, movement actions can lead the robot to different positions that might contain useful information to reach the destination; sequences of poses help the robot arm open boxes to examine which contains the object.

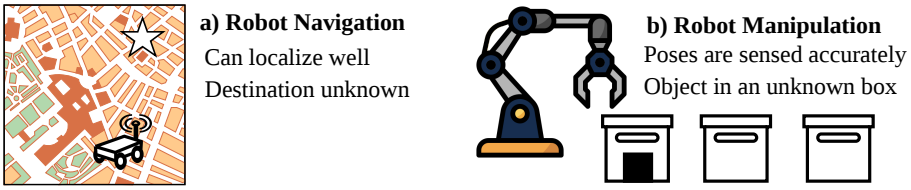


Figure 1: Examples of partially observable domains of our interests.

Our contributions are as follows. First, we formulate such tasks using the framework of mixed observability Markov decision process (MOMDP) [1] and define motion-based MOMDPs (MOB-MOMDPs), a subclass of MOMDPs which makes mild assumptions concerning dynamics and tasks. Second, we introduce a hierarchical solution to solve MOB-MOMDPs, consisting of a high-level policy with partial observability and a low-level policy with full observability. In our agent, the top policy uses high-level observations to compute the bottom policy's goals, which are desired points in the fully observable space. When the bottom policy achieves a given goal or times out, emitted observations are used to produce the next high-level observation for selecting the next goal, and so on. Our approach strikes a great analogy to how different parts of the human body function: while at a high level, the brain is equipped with memory, low-level sensorimotor components, e.g., limbs, are more feed-forward. Our approach can potentially offer efficient learning by breaking a long-horizon task into easier-to-learn subtasks, which enjoy full observability. Moreover, a hierarchical approach would explore more efficiently when rewards are sparse, thanks to high-level actions. In both theory and empirical experimentation, we show that our proposed hierarchical approach can achieve optimal or quasi-optimal behavior in MOB-MOMDPs with sufficiently low stochasticity constraints. Due to the space limit, our theoretical proof and real-world robot experiments can be seen at `https://arxiv.org/pdf/2204.00898.pdf`.

2 Background

A **POMDP** [2] is specified by a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, where ($\mathcal{S}$, $\mathcal{A}$, T, R), are respetively the state space, action space, transition function, and the reward function. Instead of directly observing the state s, the agent only observes $o \in \Omega$ after taking an action a and reaching state s' governed by the observation function $O(s', a, o) = p(o \mid s', a)$. The goal is to find a policy π that maximizes the expected discounted return defined as $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$ with some discount factor $\gamma \in [0, 1)$. To take an optimal action at timestep t, an agent often must condition its policy on the entire action-observation history $h_t = (o_{\leq t}, a_{<t}) \in \mathcal{H}_t$ that it has seen so far. However, the size of the history $\mathcal{H}_t$ grows exponentially with t. Therefore, a recurrent neural network (RNN) is often used to summarize h_t with its fixed-sized hidden state.

A **MOMDP** [1] is a POMDP in which a state can be decomposed as $s = (x, y)$, where x is fully observable and y is partially observable. Since x is fully observable, an observation o can be decomposed as $o = (x, z) \in \Omega$ where z is the remaining component of o. The observation function $O(s', a, o) = p(o \mid s', a) = p(x, z \mid x', y', a) = \mathbb{1}[x = x']p(z \mid x', y', a)$, specifies which observation the agent gets after it took action a and reached state $s' = (x', y')$ with $\mathbb{1}$ denoting the indicator function. The transition function $T(s, a, s') = p(s' \mid s, a) = p(x', y' \mid x, y, a)$ for $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$ ($\mathcal{X}$ and $\mathcal{Y}$ are respectively the spaces of all possible values of x and y) specifies the probabilities of reaching a state (x', y') after taking an action a in state (x, y). $T(s, a, s')$ can be decomposed as $T(s, a, s') = p(s' \mid s, a) = p(x', y' \mid x, y, a) = p(x' \mid x, y, a)p(y' \mid x, y, a, x')$. Let $T^{\mathcal{X}}(x, y, a, x') = p(x' \mid x, y, a)$ and $T^{\mathcal{Y}}(x, y, a, x', y') = p(y' \mid x, y, a, x')$, then the tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, T^{\mathcal{X}}, T^{\mathcal{Y}}, R, \Omega, O, \gamma)$ formally defines a MOMDP. 451

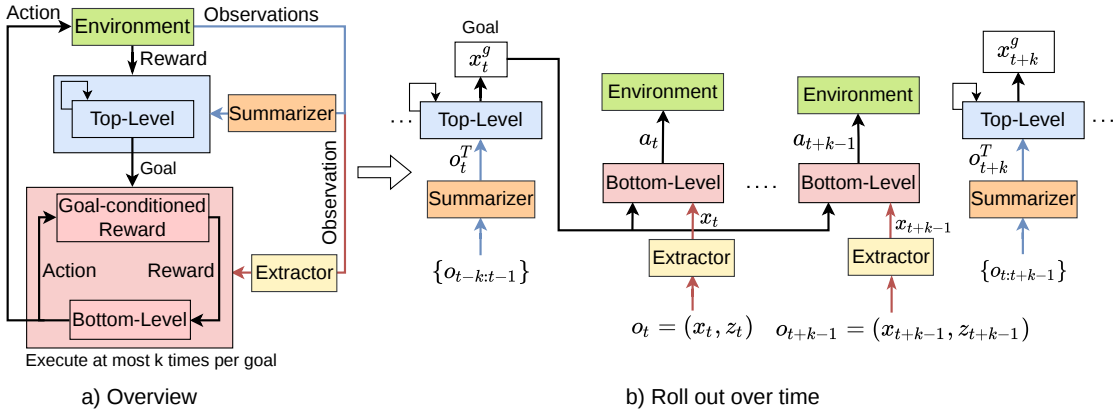


Figure 2: Our proposed agent acts through a two-level hierarchy. A *memory-based* top-level policy looks at a summary of several past observations to select a desired state (goal) x_t^g for the bottom-level policy. A *memoryless* bottom policy then looks at x_t (extracted from an observation $o_t = (x_t, z_t)$) and the the goal state x_t^g to produce at most k primitive actions to achieve x_t^g, which emits a new sequence of observations. Next, these observations will be fed to a summarizer to create a new high-level observation for the top policy to select a new goal.

3 Motion-Based MOMDPs

We define motion-based MOMDPs (MOB-MOMDPs) as MOMDPs which satisfy the following assumptions,

$$p(s' \mid s, a) = p(x' \mid x, a)p(y' \mid x, y, x'), \quad p(o \mid s', a) = p(o \mid s'), \quad R(s, a) = R(s). \tag{1}$$

In other words, a) the fully observable component x of the state satisfies the Markov property without depending on the partially observable component y of the state, b) both the partially observable component y of the state and the observed component z are conditionally independent on the action a (when conditioned on the fully observable component x of the state), and c) the task is encoded by a reward function which exclusively depends on the reached states. We further refer to MOB-MOMDPs, which have deterministic (stochastic) $T^{\mathcal{X}}$ as *deterministic* (*stochastic*) MOB-MOMDPs; note that such terminology does not refer to the stochasticity of $T^{\mathcal{Y}}$, which remains unconstrained.

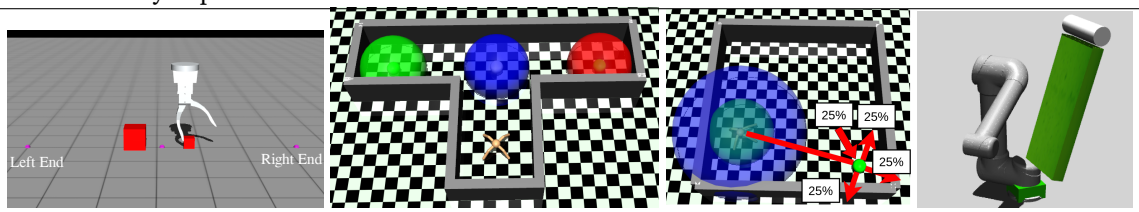
In MOB-MOMDPs, actions only have a direct influence on the resulting x trajectories, while their influence on the y, z, and reward trajectories is indirect through x. MOB-MOMDPs include (but are not limited to) navigation tasks where x represents the fully observable pose of the agent in the environment, while y represents other partially observable information about the environment and task. In such navigation MOB-MOMDPs, actions relate to the *motion* of the agent, and it is exclusively through such motion that the agent is able to interact with the environment, gather information, and complete the task. Although not all MOB-MOMDPs intrinsically represent navigation tasks, we will use the imagery of navigation tasks as a useful analogy to simplify the way we discuss and analyze MOB-MOMDPs and the respective learning algorithms. Therefore, we reinterpret general MOB-MOMDPs as navigation tasks where x figuratively represents the agent's pose, a is the movement to change its pose, and y as any other partially observable components.

Using such analogy, and because actions exclusively influence the environment through the resulting agent pose, it is possible to abstract "motion"-based control (*i.e.*, based on the actions which move the agent) as "pose"-based control (*i.e.*, based on the poses which the agent should reach in order to gain information or complete the task). "Pose"-based control is executed not by choosing how the agent should move (action a), but rather where it should move to (pose x'). Such abstraction is the inspiration for a hierarchical reinforcement learning method specifically for MOB-MOMDPs.

4 Hierarchical Reinforcement Learning under Mixed Observability

Method. As shown in Fig. 2, our method - HIerarchical reinforcement Learning under Mixed Observability (HILMO) makes decisions through a two-level hierarchy. The top-level policy is a recurrent module (symbolized by an arrow pointing to itself) which takes in a top-level observation o_t^T (a summary of several past primitive observations $o \in \Omega$) to produce a goal $x_t^g \in \mathcal{X}$ being the desired value for x_t. Then the memoryless bottom-level policy selects an action a_t using x_t (extracted from $o_t = (x_t, z_t)$) and x_t^g. Notice this also covers the case when we want to set the goal only for a subspace of $\mathcal{X}$. For instance, although x of the mobile robot in Fig. 1 might include both positions and velocities, we might just want it to reach a certain location regardless of its velocity successfully.

The goal x_t^g remains unchanged for the next few timesteps until x_t^g is achieved or k bottom-level actions have been performed (for clean notations, from now, we assume that the bottom episode will *always* last k timesteps). When the



a) Two-Boxes b) Ant-Heaven-Hell c) Ant-Tag d) Door-Push

Figure 3: Four continuous control domains to perform experiments.

bottom level finishes acting, k observations $o_{t:t+k-1}$ (*i.e.*, $o_t, \ldots, o_{t+k-1}$) are emitted. These observations will be fed to a summarizer to create the next top-level observation o^T_{t+k} for the top policy to choose the next goal. In this hierarchy, the top level acts at a higher temporal resolution than the bottom level.

Bottom-Level. A bottom-level goal-conditioned MDP $\mathcal{M}^{\mathcal{X}}$ is specified by $(\mathcal{X}, \mathcal{A}, \mathcal{G} = \mathcal{X}, T^{\mathcal{X}}, R^{\mathcal{X}}, \gamma)$. The reward function $R^{\mathcal{X}}$ is defined for each goal $x^g \in \mathcal{X}$ as $R^{\mathcal{X}}(x', x^g) = -\mathbb{1}[d(x', x^g) \geq \epsilon]$, where d is some given distance metric in $\mathcal{X}$ and ϵ is a small reaching threshold. The transition function $T^{\mathcal{X}}(x, a, x') = p(x' \mid x, a)$ specifies the probabilities of reaching x' after taking action a in x. A goal-conditioned policy $\pi^{\mathcal{X}}(a \mid x, x^g)$ that solves $\mathcal{M}^{\mathcal{X}}$ will maximize the discounted cumulative reward $\sum_{t'=t}^{t+k-1} \gamma^{t'-t} R^{\mathcal{X}}(x_{t'}, x^g)$, where x_t is the starting state. The input x of $\pi^{\mathcal{X}}$ is from an extractor that extracts x from $o = (x, z)$.

Top-Level. A top-level POMDP $\mathcal{P}^T$ is specified by $(\mathcal{S}, \mathcal{A}^T = \mathcal{X}, T^T, R^T, \Omega^T, O^T, \gamma)$. In particular, its action space $\mathcal{X}$ is the goal space in $\mathcal{M}^{\mathcal{X}}$, therefore a *top-level* policy $\pi^T(\cdot \mid o^T)$ that solves $\mathcal{P}^T$ will output a desired state x^g to be reached by $\pi^{\mathcal{X}}$. The top-level transition function can be specified as a multi-step policy taking in the goal a^T from state s, *i.e.*, $T^T(s, a^T, s') = \sum_{m=1}^{k} p(s', m \mid s, a^T)$, where $p(s', m \mid s, a^T)$ is the probability that the bottom policy $\pi^{\mathcal{X}}$ terminates at state s' after exactly m primitive actions when it acts to achieve the goal a^T starting from state s. Unlike $\pi^{\mathcal{X}}$, the objective of π^T is to optimize the discounted cumulative reward $\sum_{t=0;t+=k}^{\infty} \gamma^{t/k} R^T(s_t, a^T_t)$ where each $R^T(s_t, a^T_t)$ is the expected accumulated environment rewards when π^X acts for k timesteps to achieve the goal a^T_t starting at state s_t. The top-level observation function in $\mathcal{P}^T$ is constructed as

$$O^T(s', a^T, o^T) = p(o^T \mid s', a^T) = \begin{cases} 1 \text{ for } o^T = \mathcal{F}(\{o\}_1^k), \\ 0 \text{ otherwise,} \end{cases} \tag{2}$$

where $\mathcal{F}$ is some summarizing operator that the summarizer applies to k observations $o \in \Omega$ emitted when $\pi^{\mathcal{X}}$ was acting to achieve the goal $a^T = x^g$ and ended up at state s'. In this perspective, a high-level observation o^T can be considered as an implication of temporally extended perception.

5 Experiments

5.1 Domains

Two-Boxes. A finger moves on a 1D track to perform a dimension check of two boxes (Fig. 3a). Since the finger is kept compliant, it will be deflected from the vertical axis when it glides over a box. The agent observes the finger's position and angle but not the positions of the two boxes. When the two boxes have the same size, the agent must go to the right end to get a non-zero reward and otherwise to the left end. The agent receives a penalty if reaching wrong ends.

Ant-Heaven-Hell. An ant moving in a 2D T-shaped world will receive a non-zero reward by reaching a green area (heaven) that can be on the left or the right corner (Fig. 3b) of a junction. It receives a penalty when entering a red area (hell). When it enters the blue ball, it can observe heaven's side (left/right/null). Here the observation includes the joints' angles & velocities of the four legs and the side indicator.

Ant-Tag. The same ant now has to search and "tag" a moving opponent by being sufficiently close to it (having the opponent inside the green area centered at the ant in Fig. 3c) to get a non-zero reward. Both start randomly but not too close to each other. The opponent follows a fixed stochastic policy, moving a constant distance away from the ant 75% of the time or staying otherwise. An observation includes the joints' angles & velocities of four legs and the 2D coordinate of the opponent, containing the opponent's position only when it is inside the visibility (blue) area centered at the ant.

Door-Push. A 3-DoF gripper in 3D must successfully push a door to receive a non-zero reward (Fig. 3d). The door, however, can only be pushed in one direction (front-to-back or vice versa), and the correct push direction is unknown to the agent. Here the agent can observe the joints' angles and velocities and the door's angle.

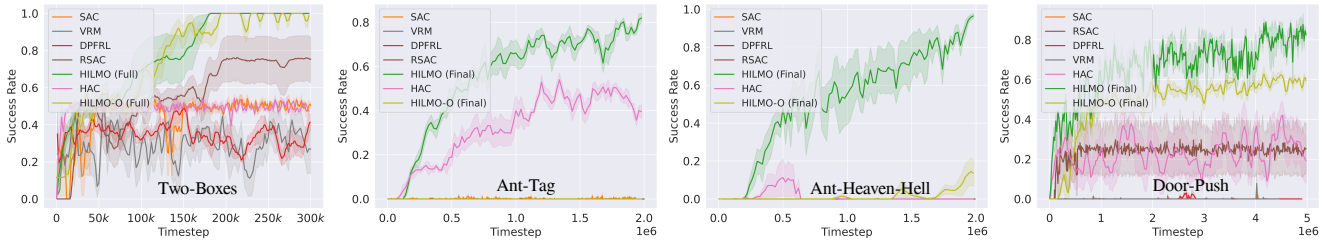


Figure 4: Success rate means and standard deviations (4 seeds). *Final* and *Full* are different choices for $\mathcal{F}$ in Eq. 2.

Table 1: Wall-clock training time in *hours* for selected agents in Fig. 4.

Domain	RSAC	HILMO-O	HILMO
`Two-Boxes`	3.31 ± 0.2	$\mathbf{0.96 \pm 0.2}$	1.04 ± 0.3
`Ant-Heaven-Hell`	71.02 ± 0.3	11.31 ± 0.2	$\mathbf{4.69 \pm 0.3}$
`Ant-Tag`	68.25 ± 0.4	12.51 ± 0.3	$\mathbf{6.41 \pm 0.3}$
`Door-Push`	112.7 ± 0.4	26.10 ± 0.2	$\mathbf{23.2 \pm 0.2}$

5.2 Agents

We consider the following hierarchical **(H)** and flat **(F)** agents:

- Two versions of our proposed two-level agent: **(H) HILMO** implemented using the HAC [3] framework and **(H) HILMO-O** implemented using the HIRO [4] framework.
- **(H)** A two-level **HAC** [3] agent.
- **(F)** Soft Actor-Critic (**SAC**) [5] and its recurrent version **(F)** Recurrent Soft Actor-Critic (**RSAC**) [6].
- **(F) DPFRL** [7] is one of state-of-the-art model-free POMDP methods.
- **(F) VRM** [8] is one of state-of-the-art model-based agents.

5.3 Results

Success Rates. Fig. 4 shows that HILMO consistently outperforms all flat baselines and is on par or better than HILMO-O. Moreover, HILMO is the only agent that can learn well in `Ant-Heaven-Hell` and `Ant-Tag`. With no memory, SAC could not solve any domains. A memory-less HAC can perform quite well in `Ant-Tag` due to a well-trained bottom policy that sometimes can corner and tag the opponent. RSAC can only succeed in `Two-Boxes` for some seeds, while struggling in the other tasks with longer episodes. Both DPFRL and VRM perform poorly across all domains.

Wall-clock Training Time. Table 1 shows that our agents (HILMO and HILMO-O) take significantly less time to train than RSAC in all domains. The acceleration can be attributed to shorter top-level episodes for these agents because each top-level observation summarizes several primitive observations. Moreover, the bottom policy trained in parallel can learn significantly faster due to full observability.

References

[1] Ong et al. Pomdps for robotic tasks with mixed observability. In *RSS*, 2009.

[2] Karl J Astrom. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 1965.

[3] Levy et al. Learning multi-level hierarchies with hindsight. In *ICLR*, 2019.

[4] Nachum et al. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[5] Haarnoja et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[6] Yang et al. Recurrent off-policy baselines for memory-based continuous control. *Deep RL Workshop NeurIPS*, 2021.

[7] Ma et al. Discriminative particle filter reinforcement learning for complex partial observations. In *ICLR*, 2020.

[8] Han et al. Variational recurrent models for solving partially observable control tasks. In *ICLR*, 2020.

Uncertainty alters the balance between incremental learning and episodic memory

Jonathan Nicholas
Department of Psychology
Mortimer B. Zuckerman Mind, Brain, Behavior Institute
Columbia University
New York, NY 10027
jdn2133@columbia.edu

Nathaniel Daw
Princeton Neuroscience Institute
Department of Psychology
Princeton University
Princeton, NJ 08540
ndaw@princeton.edu

Daphna Shohamy
Department of Psychology
Mortimer B. Zuckerman Mind, Brain, Behavior Institute
Columbia University
New York, NY 10027
ds2619@columbia.edu

Abstract

Memory is essential for adaptive decision making as it enables choices to be guided by past experience. Value-based decisions can be guided by past experience in at least two different ways. One approach akin to habit-learning consists of consulting the average value for a candidate choice, built up incrementally over many past experiences. A second approach consists of retrieving value from a single past experience in episodic memory. While there have been major advances in understanding how average values are acquired and used, less is known about the circumstances under which episodic memory is recruited and about how these two approaches interact. Here we focus on the role of uncertainty in modulating the use of episodic memory for decision making. Healthy adults completed a value-based decision making task in which the uncertainty around incrementally constructed value varied over time and between conditions, allowing us to assess how this uncertainty modulates the contribution of episodic memory to choice. As expected, we found that participants relied more on episodic memory when there was more reward-related uncertainty. These results help to clarify the impacts of uncertainty on episodic memory and suggest that rational principles of cost and benefit determine how and when different forms of memory are used for decision making.

Keywords: incremental learning, episodic memory, uncertainty, volatility

Acknowledgements

The authors thank Raphael Gerraty, Camilla van Geen, and members of the Shohamy Lab for insightful discussion and conversations. J.N. was supported by funding from the NSF Graduate Research Fellowship (1644869). D.S. was supported by funding from the NSF (1822619), NIMH/NIH (MH121093), and the Templeton Foundation (60844).

1 Introduction

Adaptive decision making depends on using memories of the past to inform the present. This process has been extensively studied using models of reinforcement learning,[1] which capture how feedback is used to gradually improve decisions. Learning in this way is useful for decision making due to its simple and time-efficient ability to update value. However, recent findings indicate that episodic memory also contributes to learned value and decision making through the retrieval of individual memories.[2,3] Although theoretical work has suggested a role for episodic memory when experience is limited,[4,5] the use of episodes has proven surprisingly pervasive, detected even for decisions that can be resolved normatively using incremental learning alone.[2] The persistence of episodic memory as a substrate for decision making is thus somewhat puzzling and raises important questions about the circumstances under which it is recruited for decisions. Here we sought to provide answers by investigating the impact of uncertainty on decisions based on either episodic memory or incremental learning. Our main hypothesis was that episodic memory is well-suited to contribute to decision making when incremental value estimates are uncertain.

Previous computational work has suggested that uncertainty plays an important role in arbitrating between candidate systems for decision making.[6,7] However, this work does not consider episodic memory, which has the potential to provide a lossless representation of experience that can be particularly helpful for decision making.[4] For example, artificial agents endowed with episodic memory are able to learn more quickly and effectively on problems that challenge traditional reinforcement learning algorithms.[5,8] The computations required by episodic memory can, however, be time-inefficient when compared to retrieving simple summaries of the type provided by incremental learning. Episodic memory is thus somewhat costly. While past computational work has focused on a specific role for episodic memory when deciding between relatively novel options,[4,5] or at different time scales,[4,8] the common thread underlying these proposals is that episodic memory is most useful for behavior when an alternative learning system is uncertain. We therefore chose to investigate this idea directly by altering uncertainty as tracked by arguably the simplest of these systems, incremental learning, and testing how increased uncertainty impacts reliance on episodic memory.

To quantify how uncertainty modulates the use of episodic memory for decisions, we developed a procedure that allowed us to i) independently measure the contribution of episodic memory to choice and ii) manipulate the impact of uncertainty on episodic memory-based choices. A large sample of healthy adults (n=374) recruited from Amazon Mechanical Turk completed two tasks. The main task of interest was the *deck learning and card memory task*, which combined incremental learning and episodic memory (**Figure 1A**). On each trial, participants chose between an orange and a blue card and received feedback following their choice. The cards appeared throughout the task, but their relative value changed over time such that one deck was always worth more than the other (**Figure 1B**). In addition to the color of the card, each card also displayed an object. Critically, objects appeared on a card at most twice throughout the task, such that a chosen object could re-appear after 9-30 trials. Thus, participants could make decisions based on incremental learning of the average value of the orange vs. blue decks, or based on episodic memory for the value of an object which they only saw once before. Finally, participants made these choices across two environments: a high volatility and a low volatility environment. The environments differed in how often reversals in deck value occurred, on the assumption that the larger and more frequent prediction errors which occur following a reversal signal a decrease in the accuracy of incrementally learned value, thereby increasing uncertainty. Prior to the main task, participants also completed a simple *deck learning* task to quantify the effects of uncertainty without trial-unique episodic memories.

2 Results

2.1 Episodic memory is used more under conditions of greater uncertainty

To determine whether both forms of memory were used in the combined deck learning and card memory task, we first examined the extent to which participants were sensitive to the value of previously seen (old) objects and reversals in deck value. Despite an overall bias toward choosing an old object regardless of its value (β_0 = 0.465, 95% CI = [0.424, 0.505]), high-valued old objects were substantially more likely to be chosen than low-valued old objects ($\beta_{oldvalue}$ = -0.415, 95% CI = [-0.561, -0.268]; **Figure 2A**). Thus, participants deployed episodic memory to guide decision making throughout the task. Next, we examined whether participants altered their behavior in response to reversals in deck value. Immediately preceding a reversal trial t, participants based most choices on the higher-valued deck (β_{t-1} = 0.141, 95% CI = [0.07, 0.214]; **Figure 2B**). Reversal trials disrupted this tendency (β_t = -0.362, 95% CI = [-0.437, -0.289]) until performance recovered around four trials later (β_{t+4} = 0.132, 95% CI = [0.084, 0.18]). Thus, participants were sensitive to reversals in deck value, thereby indicating that they engaged in incremental learning throughout the task.

Having established that both episodic memory and incremental learning were recruited for choice, we next sought to determine the impact of uncertainty on episodic memory specifically. To accomplish this, we isolated trials on which episodic memory was most likely to have guided participants' decisions. We first identified trials where the predictions made by either memory system were in direct conflict with one another. These incongruent trials consisted of those on

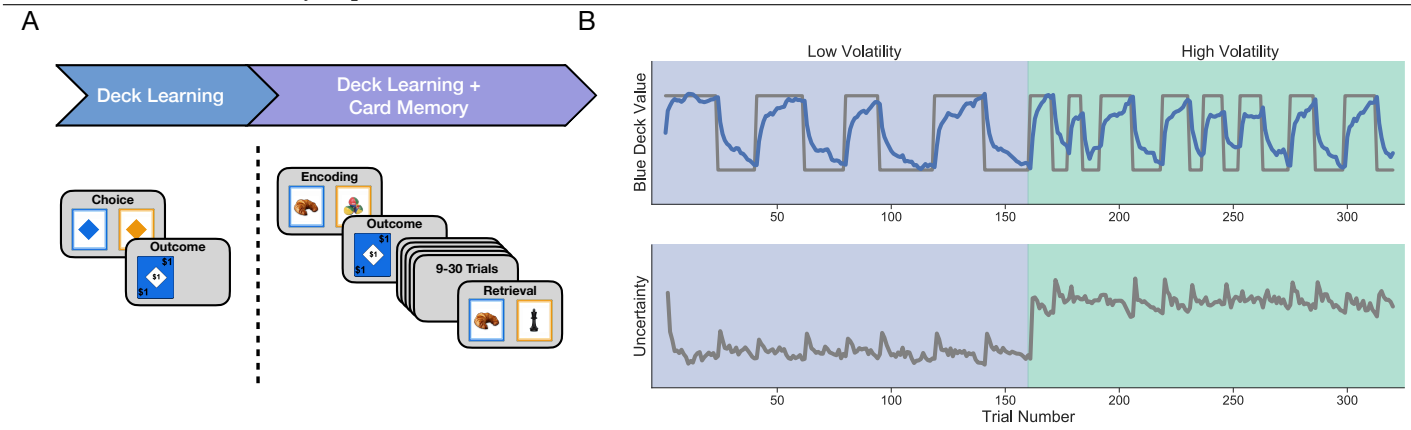


Figure 1: **A)** Participants first completed a *deck learning* task which consisted of choosing between two decks of cards and receiving an outcome. Second was the main task of interest, the combined *deck learning and card memory* task, which was identical to the deck learning task but cards also contained trial-unique objects. **B)** Environment design. Top: The true value of the blue deck is drawn in gray for an example trial sequence. In blue is estimated blue deck value from the RB model.[9] Bottom: Uncertainty about deck value as estimated by the RB model.

which the higher value deck had a card featuring an old object that was of low value (<50¢) or the lower value deck had a card featuring an old object that was of high value (>50¢). Consistent with our hypothesis, we found that participants based more decisions on episodic value in the high volatility environment compared to the low volatility environment (β_{env} = 0.101, 95% CI = [0.048, 0.155]; **Figure 2C**). In line with the idea that incremental learning, which is simpler and more time-efficient, should be used more frequently for decisions, participants deployed this system for the majority of incongruent trials across both environments (β_0 = -0.254, 95% CI = [-0.301, -0.208]). Furthermore, decisions based on episodic value took longer (β_{RT} = 0.183, 95% CI = [0.152, 0.216]; **Figure 2D**), suggesting that retrieving a single experience is indeed more costly in time when compared to relying on cached incremental value.

2.2 Uncertainty increases sensitivity to episodic value

We next sought to understand the circumstances under which incremental learning was eschewed in favor of episodic memory by developing a computational model that modulates its' sensitivity to episodic value as a function of posterior uncertainty about incremental value. We first fit a hierarchical version of the reduced Bayesian (RB) reinforcement learning model developed by Nassar and colleagues[9] to the deck learning task. This allowed us to assess incremental learning free of any contamination due to competition with episodic memory. The RB model modulates its learning rate for deck value as a function of two quantities: the probability that the most recent outcome received was indicative of a change in deck luckiness and how uncertain the model's beliefs about deck luckiness are. We focused on the second quantity, relative uncertainty (RU), as it specifically captures imprecision in incremental learning. We then used the parameters fit for each participant to generate estimates of subjective deck value and RU in the deck learning and card memory task. We next used a mixed effects logistic regression to create a combined choice model that used these estimates alongside episodic value to predict choices in the deck learning and card memory task. For each subject s and trial t, this model can be written as:

$$\theta_t = logit^{-1}(\beta_0 + b_{0,s[t]} + V_t(\beta_1 + b_{1,s[t]}) + O_t(\beta_2 + b_{2,s[t]}) + OV_t(\beta_3 + b_{3,s[t]}) + OV_t \times RU_t(\beta_4 + b_{4,s[t]})) \quad (1)$$

where θ_t is the probability that the orange deck was chosen on trial t, fixed effects are represented with β, and random effects for each subject are represented with b. The intercept captures a bias toward choosing either of the decks regardless of outcome, the effect of V captures sensitivity to deck value estimated from the RB model, the effect of O captures a bias toward choosing a previously seen object regardless of its value, and the effect of OV captures sensitivity to the value of a previously seen object. Critically, the effect of $OV \times RU$ captures sensitivity related to the interaction between episodic value and uncertainty: the more positive this term, the greater the impact of uncertainty on the role of episodic memory.

We first checked that participants were sensitive to the volatility manipulation by examining the RB model hazard rates fit to each environment, which capture the subjective probability that a participant believes a reversal in deck luckiness will occur on any given trial. Group-level hazard rates were indeed reliably larger for the high compared to the low volatility environment (Low = 0.051, 95% CI = [0.043, 0.059]; High = 0.077, 95% CI = [0.065, 0.092]; **Figure 3A**). Next, we examined the model's ability to estimate uncertainty as a function of reversals in deck luckiness. RU increased

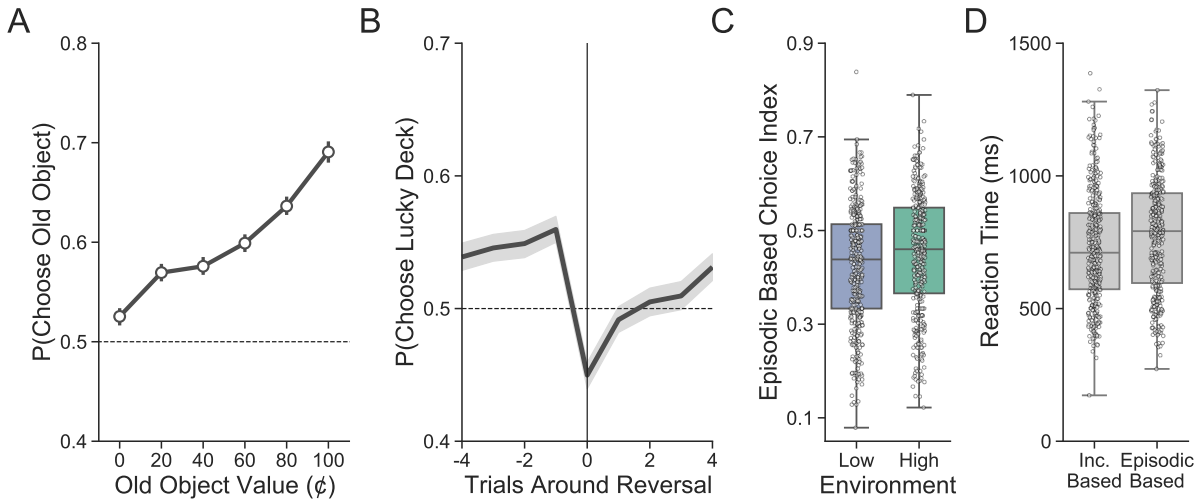


Figure 2: **A)** Participants' choices demonstrate sensitivity to the value of old objects. **B)** Reversals in deck luckiness altered choice such that the currently lucky deck was chosen less following a reversal. **C)** On incongruent trials, choices were more likely to be based on episodic memory in the high compared to the low volatility environment. **D)** Reaction time was longer for incongruent choices based on episodic memory compared to those based on incremental learning. Group-level averages are shown as large points and bands represent 95% CIs. Small points represent averages for individual subjects and box and whiskers represent quartiles of this distribution.

immediately following a reversal and stabilized over time and was greater in the high compared to the low volatility environment (β_{env} = 0.217, 95% CI = [0.109, 0.285]; **Figure 3B**). A primary prediction made by the combined choice model is that participants should increase their reliance on episodic memory with greater RU. In line with this prediction, episodic memory was indeed used more on incongruent trial decisions made under conditions of high RU (β_{RU} = 1.71, 95% CI = [0.55, 2.924]; **Figure 3C**).

Finally, we examined the bias and sensitivity parameters of the fit combined choice model. Participants were not unduly biased toward either deck color (β_0 = -0.03, 95% CI = [-0.068, 0.005]), but they were biased to choose cards that featured an old object regardless of that object's value (β_O = 0.291, 95% CI = [0.267, 0.316]). Participants used both sources of value: both deck value as estimated by the model (β_V = 0.549, 95% CI = [0.486, 0.616]) and the episodic value from old objects (β_{VT} = 0.146, 95% CI = [0.126, 0.165]) had strong impacts on choice. Lastly, we turned to participants' sensitivity to the interaction between episodic value and RU (**Figure 3D**), which captures the hypothesized effect. As predicted, episodic value did indeed impact choices more when relative uncertainty was high (β_{OVXRU} = 0.02, 95% CI = [0.004, 0.037]), thereby indicating that episodic value was relied on more when beliefs about incremental value were uncertain.

2.3 Posterior inference and statistical analyses

All analyses not described in the previous section were conducted using mixed effects regressions (linear for continuous and logistic for binary outcomes). All reinforcement learning and regression models were fit hierarchically with Bayesian inference such that the joint posterior was approximated using No-U-Turn Sampling as implemented in Stan. This approach improves parameter identifiability and predictive accuracy.[10] For all models, fixed effects are reported in the text as the median of each parameter's marginal posterior distribution alongside 95% credible intervals, which indicate where 95% of the posterior density falls. Parameter values outside of this range are unlikely given the model, data, and priors. Thus, if the range of likely values does not include zero, we conclude that a meaningful effect was observed.

3 Discussion

The majority of research on memory's role in value-based decision making has focused on simple strategies involving the storing and updating of cached estimates extracted over many experiences. While recent work has demonstrated that episodic memory also contributes to value-based decisions,[2,3] the circumstances under which episodic memory is recruited for decisions have yet to be elucidated. Here, we sought to determine whether the use of episodic memory is shaped by volatility, an aspect of the environment that has been extensively shown to exert control over incremental learning. We reasoned that the inaccuracies that are characteristic of a volatile environment would increase uncertainty, yielding separate effects on how incremental learning and episodic memory are used. First, as has been extensively

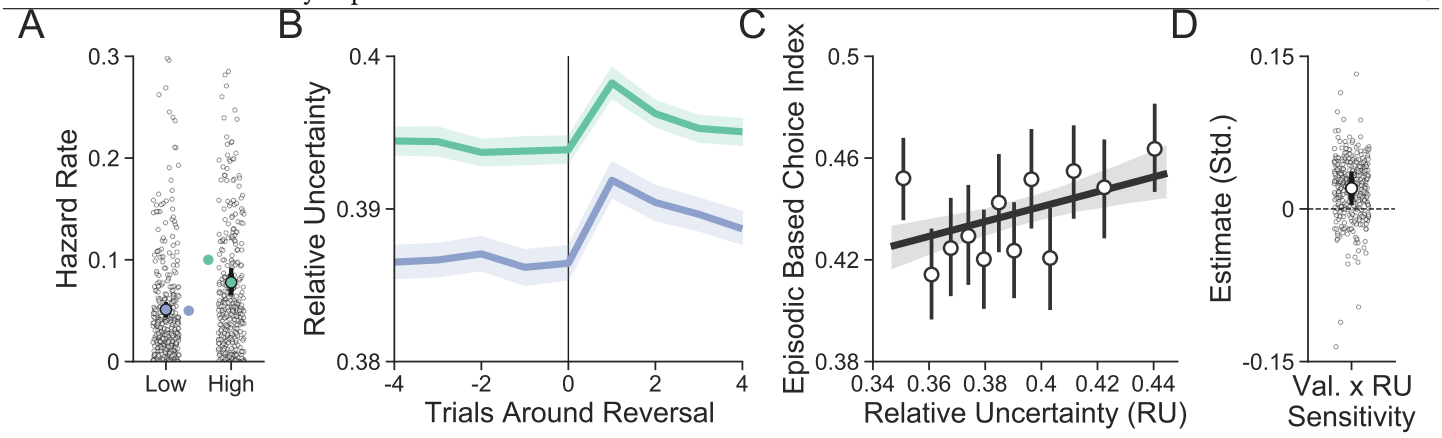


Figure 3: **A)** Hazard rates fit to each environment. The true hazard rates are shown on the interior of the plot. **B)** RU shown on each trial surrounding a reversal event. **C)** Participants' episodic-based choice increased with RU. **D)** Fit old object value x RU sensitivity parameter from the combined choice model. Large points are group-level means and small points are individual subject means. Error bars and bands represent 95% CIs.

shown in the incremental learning system, new outcomes should exert more influence while the past is downweighted in response to uncertainty.[9] Second, we hypothesized that episodic memory should be recruited to augment decisions during this period of poor incremental performance.

Our findings support this hypothesis. Participants based a greater number of decisions on single experiences in the high volatility environment. Within each environment, participants were sensitive to reversals in deck value. As predicted, single experiences were used more at these timepoints.

These results have important implications for understanding memory's role in guiding decisions, and suggest that rational principles of cost and benefit determine how and when different forms of memory will be used. While decision making can greatly benefit from episodic memory early in learning when little is known about the environment,[5,8] our results suggest a more general interpretation of these findings: that episodic memory is useful whenever uncertainty is high. There is strong precedent for this type of uncertainty-based arbitration in the brain, with the most well-known being the trade off between incremental, or model-free, learning and model-based learning.[6] Control over decision making by model-free and model-based systems has been found to shift in accordance with the accuracy of their respective predictions,[7] and humans adjust their reliance on either system in response to external conditions that provide a relative advantage to one over the other.[11] Tracking uncertainty provides useful information about when inaccuracy is expected and helps to maximize utility by deploying whichever system is best at a given time. Our results add to these findings and expand their principles to include episodic memory in this tradeoff.

In conclusion, we have demonstrated that uncertainty signaled by larger than expected prediction errors impacts whether incremental learning or episodic memory is recruited for decisions. Greater uncertainty increased the likelihood that single experiences were retrieved for decision making. This effect manifested as an overall increase in the number of choices that were based on episodic memory in an environment with higher volatility. Our results suggest that episodic memory aids decision making when simpler sources of value are less accurate.

References

1. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998), ISBN: 0-262-19398-1.
2. A. M. Bornstein, M. W. Khaw, D. Shohamy, N. D. Daw, *Nature Communications* **8**, 15958, ISSN: 2041-1723 (2017).
3. K. Duncan, A. Semmler, D. Shohamy, *Journal of Cognitive Neuroscience*, 1–13, ISSN: 0898-929X, 1530-8898 (2019).
4. S. J. Gershman, N. D. Daw, *Annual Review of Psychology* **68**, 101–128, ISSN: 0066-4308, 1545-2085 (2017).
5. M. Lengyel, P. Dayan, *Advances in Neural Information Processing Systems 20*, 889–896 (2008).
6. N. D. Daw, Y. Niv, P. Dayan, *Nature Neuroscience* **8**, 1704–1711, ISSN: 1546-1726 (2005).
7. S. W. Lee, S. Shimojo, J. P. O'Doherty, *Neuron* **81**, 687–699, ISSN: 0896-6273 (2014).
8. A. Santoro, P. Frankland, B. Richards, *Journal of Neuroscience* **36**, 12228–12242, ISSN: 0270-6474, 1529-2401 (2016).
9. M. Nassar, R. Wilson, B. Heasly, J. Gold, *Journal of Neuroscience* **30**, 12366–12378, ISSN: 0270-6474, 1529-2401 (2010).
10. C. van Geen, R. T. Gerraty, *Journal of Mathematical Psychology* **105**, 102602, ISSN: 0022-2496 (2021).
11. D. Simon, N. Daw, *Advances in Neural Information Processing Systems* **24** (2011).

The Primacy Bias in Deep Reinforcement Learning

Evgenii Nikishin* Mila, Université de Montréal evgenii.nikishin@mila.quebec

Max Schwarzer* Mila, Université de Montréal max.schwarzer@mila.quebec

Pierluca D'Oro* Mila, Université de Montréal pierluca.doro@mila.quebec

Pierre-Luc Bacon Mila, Université de Montréal pierre-luc.bacon@mila.quebec

Aaron Courville Mila, Université de Montréal aaron.courville@umontreal.ca

Abstract

This work identifies a common flaw of deep reinforcement learning (RL) algorithms: a tendency to rely on early interactions and ignore useful evidence encountered later. Because of training on progressively growing datasets, deep RL agents incur a risk of overfitting to earlier experiences, negatively affecting the rest of the learning process. Inspired by cognitive science, we refer to this effect as *the primacy bias*. Through a series of experiments, we dissect the algorithmic aspects of deep RL that exacerbate this bias. We then propose a simple yet generally-applicable mechanism that tackles the primacy bias by periodically resetting a part of the agent. We apply this mechanism to algorithms in both discrete (Atari 100k) and continuous action (DeepMind Control Suite) domains, consistently improving their performance.

Keywords: deep reinforcement learning, overfitting, forgetting

Acknowledgements

The authors thank Rishabh Agarwal, David Yu-Tung Hui, Ilya Kostrikov, Ankit Vani, Zhixuan Lin, Tristan Deleu, Wes Chung, Hattie Zhou, Mandana Samiei, Marc G. Bellemare for insightful discussions and useful suggestions on the early draft and Compute Canada for providing computational resources. This work was partially supported by Facebook CIFAR AI Chair, Samsung, Hitachi, IVADO, and Gruppo Ermenegildo Zegna.

*Equal contribution, correspondence to Evgenii Nikishin.

1 Introduction

The primacy bias is a well-studied cognitive bias in human learning [13]. Facing a sequence of experiences, humans often form generalizations based on early evidence which might negatively impact future decision making [18].

The central finding of our work is that deep reinforcement learning (RL) algorithms are susceptible to a similar bias. The primacy bias in deep RL is a tendency to overfit early interactions with the environment preventing the agent from improving its behavior on subsequent experiences. Through a series of controlled experiments, we first expose plausible causes of this phenomenon and show that common algorithmic features such as a high replay ratio [7, 4] and long n-step targets [20] amplify the primacy bias. As a remedy, we then propose a simple *resetting* mechanism, compatible with any deep RL algorithm equipped with a replay buffer, which allows the agent to forget a part of its knowledge.

Despite its simplicity, this resetting strategy consistently improves the performance of agents on benchmarks including the discrete-action ALE [2] and the continuous-action DeepMind Control Suite [21]. Resets impose no additional computational costs and require only two implementation choices: which neural network layers to reset and how often.

2 Preliminaries

We adopt the standard formulation of reinforcement learning [20] under the Markov decision process (MDP) and consider deep RL algorithms where the action-value function $Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) | s_0 = s, a_0 = a\right]$ and π (when needed) are modelled by neural network function approximators. We focus on off-policy methods that learn $Q_\pi(s,a)$ though *temporal-difference* (TD) learning [19] and reuse past experiences with a *replay buffer* [14]. The frequency of resampling experience from the buffer is controlled by the *replay ratio* [7, 4] which plays a critical role in the algorithm's performance: higher replay ratios may allow better sample efficiency but incur a risk of overfitting. TD learning can be generalized by using *n-step* targets $\mathbb{E}_\pi\left[r(s_t,a_t) + \gamma r(s_{t+1},a_{t+1}) + \cdots + \gamma^n Q_\pi(s_{t+n},a_{t+n})\right]$ for predicting $Q_\pi(s,a)$. Here, n controls a trade-off between the (statistical) bias of Q_π estimates and the variance of the sum of future rewards.

3 The Primacy Bias

The main goal of this work is to understand how the learning process of deep reinforcement learning agents can be disproportionately impacted by initial phases of training due to an effect called the primacy bias.

The Primacy Bias in Deep RL: *a tendency to overfit initial experiences that damages the rest of the learning process.*

This definition is wide-ranging: the primacy bias has multiple roots and leads to multiple negative effects on the training of an RL agent, but they are all connected to improper learning from early experiences.

The rest of this section presents two experiments intending to demonstrate the existence and behavior of the phenomenon in isolation. First, we show that excessive training of an agent on early interactions can fatally damage the rest of the learning process. Second, we show that data collected by an agent impacted by the primacy bias is adequate for learning, although the agent cannot leverage due to its accumulated overfitting.

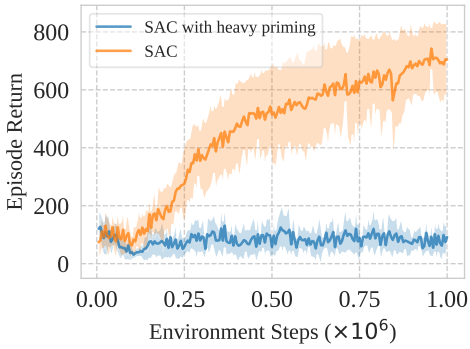


Figure 1: Undiscounted returns on `quadruped-run` for SAC with and without *heavy priming* on the first 100 transitions. An agent extremely affected by the primacy bias is unable to learn even after collecting thousands of new transitions. Mean and std are over 10 runs.

3.1 Heavy Priming Causes Unrecoverable Overfitting

One of the crucial algorithmic aspects impacting the primacy bias is degree of the reliance of an agent on early data. It is vital for sample efficiency to leverage initial experiences well, and to this end, the agent may sample from its buffer and update its neural network several times before interacting further with an environment. We hypothesize that such a practice may have severe consequences and probe it to its extreme: could *overfitting on a single batch of early data be enough to entirely disrupt an agent's learning process*?

To investigate this question, we train Soft Actor-Critic [6] on the `quadruped-run` environment from DeepMind Control suite (DMC) [21]. We use default hyperparameters, which imply a single update for policy and value functions per step in the environment. Then, we train an identical agent in an experimental condition that we refer to as *heavy priming*: after collecting 100 data points, we update the agent 10^5 times using its replay buffer, before resuming standard training. Figure 1 shows that even after training on almost one million new transitions, the agent with heavy priming is unable to solve the task.

This experiment conveys a simple message: overfitting to early experiences might inexorably damage the rest of the learning process. Indeed, we will see in Section 4 that even a relatively small number of updates per step can cause similar issues. The finding suggests that the primacy bias has compounding effects: an overfitted agent gathers worse data that itself leads to less efficient learning that further damages the ability to learn and so on.

3.2 Experiences of Primed Agents are Sufficient

Once the agent is heavily impacted by the primacy bias, it might struggle to reach satisfying performance. But is the data collected by an overfitted agent unusable for learning? We train a SAC agent with 9 updates per step in the MDP; due to the primacy bias, this agent performs poorly. Then, we initialize the same agent from scratch but use the data collected by the previous SAC agent as its initial replay buffer. Figure 2 demonstrates that returns collected by this agent improve rapidly approaching the optimal task performance.

This experiment articulates that the primacy bias is not a failure to collect proper data per se, but rather a failure to learn from it. The data stored in the replay buffer is in principle enough to have better performance but the overfitted agent lacks the ability to distill it into a better policy. In contrast, the randomly initialized neural networks are not affected by the primacy bias and thus capable of fully leveraging the collected experience. This intuition forms the basis of the algorithmic solution to the issues highlighted above.

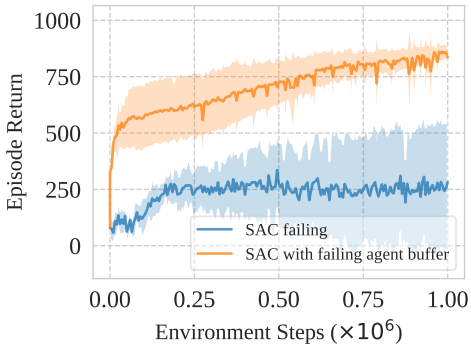


Figure 2: Undiscounted returns on `quadruped-run` for SAC trained with 9 updates per step. `SAC failing` is a standard SAC agent; `SAC with failing agent buffer` is a SAC agent initialized with the replay buffer of the first agent, which allows it to learn quickly. Mean and std are over 10 runs.

3.3 Have You Tried Resetting It?

We now present a simple technique that mitigates the primacy bias. The solution, which we dub *resetting* in the rest of the manuscript, is given by the following recipe:

Addressing the Primacy Bias: *periodically re-initialize the last layers of the agent's neural networks, preserving the replay buffer.*

The next section analyzes both quantitatively and qualitatively the performance improvements provided by resetting in addressing overfitting to early data.

4 Experiments

The goals of experiments are mostly twofold. First, we investigate across different algorithms and domains the effect on performance of using resets as a remedy for the primacy bias; next, we analyze the learning dynamics induced by resetting, including its interaction with critical design choices such as the replay ratio and n-step TD targets.

We focus our experimentation in two settings: discrete control, represented by the 26-task Atari 100k benchmark [9], and continuous control, represented by the DeepMind Control Suite [21]. We apply resets to three baseline algorithms: SPR [17] for Atari, and SAC [6] and DrQ [10] for continuous control from dense states and raw pixels respectively. For SPR, we reset the final layer of a 5-layer Q-network, using three resets spaced 2×10^4 steps apart; for SAC, we reset *the entire policy and value networks* every 2×10^5 steps; for DrQ, we reset last 3 layers of the policy and value networks every 4×10^5 steps. In every case, we also reset target networks. After each reset, we return directly to standard training.

To provide rigorous evaluations of all algorithms, we follow recommendations from [1] and use interquartile mean (IQM) for measuring performance.

Method	IQM
SPR + resets	**0.478** (0.46, 0.51)
SPR	0.380 (0.36, 0.39)
DrQ(ϵ)	0.280 (0.27, 0.29)
DER	0.183 (0.18, 0.19)
CURL	0.113 (0.11, 0.12)
SAC + resets	**656** (549, 753)
SAC	501 (388, 609)
DrQ + resets	**757** (698, 810)
DrQ	570 (473, 665)

Table 1: Point estimates and 95% bootstrap confidence intervals for the performance of SPR, SAC, and DrQ with resets. Results for SPR are computed over 20 seeds per task, and for SAC and DrQ are computed over 10 seeds. Other baselines are taken from [1] and use 100 seeds.

4.1 Resets Consistently Improve Performance

The empirical evidence in Table 1 suggests that resets mitigate the primacy bias and provide significant benefits across environments for the final performance of the agent. Remarkably, the magnitude of improvement provided by resets for SPR is comparable to improvements of prior advances while not requiring additional computation costs.

4.2 The Learning Dynamics of Resetting Agents

At the first glance, resetting may appear as a drastic (if not wasteful) measure as the agent must learn the parameters of the randomly initialized layers from scratch every time. Figure 3 shows a representative example of the learning trajectories induced by resets. Surprisingly, the agent quickly reaches or surpasses its prior performance after each reset.

After resetting, the agent is free from the negative priming provided by its past training iterations: it can better leverage the data collected so far, thus improving its performance and unlocking the possibility to generate higher quality data for its future updates. The crucial element behind the success of resetting resides in preserving the replay buffer across iterations allowing the agent to recover.

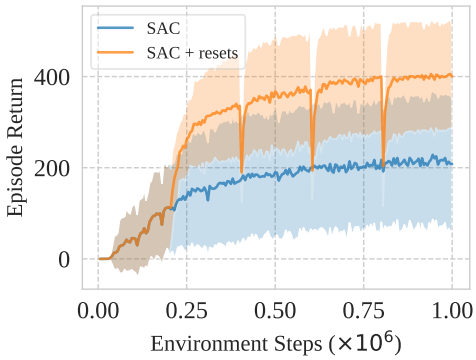


Figure 3: Effects of resets for SAC (32 updates per step) on `hopper-hop`. After each reset, performance recovers quickly and the resetted agent achieves higher overall returns. Mean and std are over 10 runs.

4.3 The Elements Behind the Success of Resets

We now provide an ablation study aiming at the question: under which conditions resets are maximally impactful?

Replay Ratio. Our initial experiments in Section 3 suggest that the degree of reliance on early data is a critical determinant of the strength of the primacy bias. We vary the replay ratio, the number of gradient steps per each environment step, in SPR and SAC. Figure 4 reports results for SAC while the conclusions for SPR are the same. With fewer updates, resets provide little or no benefit, implying that agents might be underfitting early data. With resets, however, SAC achieves its highest performance at the high replay ratio of 32, where resets increase performance by over 100%. Resets thus improve sample efficiency by performing more updates per each data point.

n-step targets. The effect of resets depends on the variance of TD targets. We observe that with high n the agent is more likely to overfit and hence the resetting effect increases. Figure 4 demonstrates the effect size for SPR; for SAC results are similar.

The results with varying replay ratios and n-step targets suggest that resets reshape the hyperparameter landscape creating a new optimum with higher performance.

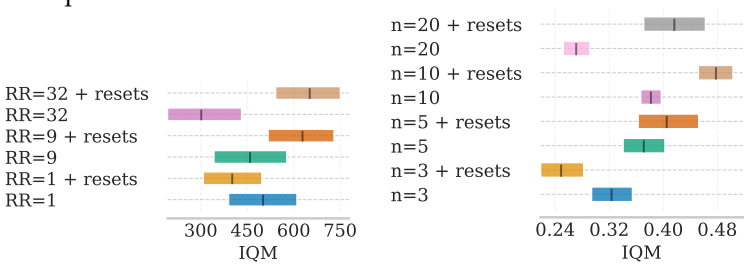


Figure 4: The impact of resets on SAC at different replay ratios (left) and SPR at different n-step return lengths (right). High replay ratio makes the agent overly reliant on the early experince and thus the resets have the largest influence. The effect size of resets is largest for high n since it increases the variance of targets exposing the agent to a risk of overfitting.

What to reset. The number of layers to reset is a domain-dependent choice. We observed the best performance when resetting only the last layer in SPR, while for DrQ resetting the last 3 (out of 7) layers was better. We conduct two additional ablations: resetting or preserving the optimizer state and replay buffer. We find that resetting the optimizer state has essentially no impact because moment estimates are update rapidly. Resetting the replay buffer, however, made it impossible for agents to quickly recover their prior performance.

TD failure modes. Temporal-difference learning can be prone to divergence and collapse to a trivial solution. Once in a failure mode, standard RL optimization struggles to recover from it. Adding resets solves this problem by giving the agent the second chance. We observe that on sparse reward tasks where Q_π might collapse, the buffer contains trajectories reaching the goal state suggesting that *the primacy bias is more an issue of optimization rather than exploration*.

5 Related Work

The primacy bias in deep RL is intimately related to memorization, optimization in RL, and cognitive science. Various aspects of our work have been studied in the literature.

Addressing overfitting in RL. [11, 12] show that an approximator for value function gradually loses its expressivity due to bootstrapping which might amplify the effects of the primacy bias. [8] uses an on-policy buffer-free algorithm and distills the previous network after resetting it to improve generalization. Forms of non-uniform sampling including re-weighting recent samples [22] and prioritized experience replay (PER) [16] can be seen as a way to mitigate the primacy bias. SPR, which already uses PER as a component, still benefits from resets.

Forgetting mechanisms. In contrast to the well-known phenomenon of catastrophic forgetting [5], several works have observed catastrophic memorization [15], similar to the primacy bias. [3] notices higher sensitivity of the trained networks to early data. Resetting subnetworks has recently received more attention in supervised learning. [23] shows that forgetting might improve generalization and draws a connection to the emergence of compositional representations. These works complement the evidence about the primacy bias in deep RL and add to our analysis of the regularizing effect of resets.

Cognitive science. The primacy bias (also known as the *primacy effect*) has been studied in human learning for many decades [13]. [18] argues that outcomes of the first experience have a substantial and lasting effect on subsequent behavior and affect the outcomes of future decision making. Even though humans and RL systems learn under different conditions, our findings provide evidence that artificial agents also exhibit this type of bias.

6 Conclusion

This work identifies the primacy bias in deep RL, a damaging tendency of artificial agents to overfit early experiences. We demonstrate the dangers associated with this form of overfitting and propose a simple solution based on resetting a part of the agent. The experimental evidence across domains and algorithms suggests that resetting is an effective and generally applicable technique. We are intrigued by the results: if something as simple as resetting drastically improves the performance, a room for advancements in deep RL is enormous. Overall, this work sheds light on the learning process of deep RL agents, unlocks training regimes that were unavailable without resets, and opens possibilities for further studies improving both understanding and performance of deep reinforcement learning algorithms.

References

[1] Rishabh Agarwal et al. "Deep reinforcement learning at the edge of the statistical precipice". In: *NeurIPS*. 2021.

[2] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[3] Dumitru Erhan et al. "Why does unsupervised pre-training help deep learning?" In: *AISTATS*. 2010.

[4] William Fedus et al. "Revisiting fundamentals of experience replay". In: *ICML*. PMLR. 2020, pp. 3061–3071.

[5] Robert M French. "Catastrophic forgetting in connectionist networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[6] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[7] Hado P van Hasselt et al. "When to use parametric models in reinforcement learning?" In: *NeurIPS*. 2019.

[8] Maximilian Igl et al. "Transient Non-stationarity and Generalisation in Deep Reinforcement Learning". In: *ICLR*. 2021.

[9] Łukasz Kaiser et al. "Model Based Reinforcement Learning for Atari". In: *ICLR*. 2019.

[10] Ilya Kostrikov et al. "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels". In: *ICLR*. 2021.

[11] Aviral Kumar et al. "Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning". In: *ICLR*. 2020.

[12] Clare Lyle et al. "Understanding and Preventing Capacity Loss in Reinforcement Learning". In: *ICLR*. 2022.

[13] Philip H Marshall and Pamela R Werder. "The effects of the elimination of rehearsal on primacy and recency". In: *Journal of Verbal Learning and Verbal Behavior* 11.5 (1972), pp. 649–653.

[14] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[15] Anthony Robins. "Catastrophic forgetting in neural networks: the role of rehearsal mechanisms". In: *Proceedings 1993 The First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*. IEEE. 1993, pp. 65–68.

[16] Tom Schaul et al. "Prioritized Experience Replay". In: *ICLR*. 2016.

[17] Max Schwarzer et al. "Data-Efficient Reinforcement Learning with Self-Predictive Representations". In: *ICLR*. 2020.

[18] Hanan Shteingart et al. "The role of first impression in operant learning." In: *Journal of Experimental Psychology: General* 142.2 (2013), p. 476.

[19] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Yuval Tassa et al. *dm_control: Software and Tasks for Continuous Control*. 2020. arXiv: 2006.12983 [cs.RO].

[22] Che Wang et al. "Striving for simplicity and performance in off-policy DRL: Output normalization and non-uniform sampling". In: *ICML*. PMLR. 2020, pp. 10070–10080.

[23] Hattie Zhou et al. "Fortuitous Forgetting in Connectionist Networks". In: *ICLR*. 2022.

Learning to Query Internet Text for Informing Reinforcement Learning Agents

Kolby Nottingham
UC Irvine
Irvine, CA 92697
knotting@uci.edu

Alekhya Pyla
UC Irvine
Irvine, CA 92697
apyla@uci.edu

Sameer Singh
UC Irvine
Irvine, CA 92697
sameer@uci.edu

Roy Fox
UC Irvine
Irvine, CA 92697
royf@uci.edu

Abstract

Generalization to out of distribution tasks in reinforcement learning is a challenging problem. One successful approach improves generalization by conditioning policies on task or environment descriptions that provide information about the current transition or reward functions. Previously, these descriptions were often expressed as generated or crowd sourced text. In this work, we begin to tackle the problem of extracting useful information from natural language found in the wild (e.g. internet forums, documentation, and wikis). These natural, pre-existing sources are especially challenging, noisy, and large and present novel challenges compared to previous approaches. We propose to address these challenges by training reinforcement learning agents to learn to query these sources as a human would, and we experiment with how and when an agent should query. To address the *how*, we demonstrate that pretrained QA models perform well at executing zero-shot queries in our target domain. Using information retrieved by a QA model, we train an agent to learn *when* it should execute queries. We show that our method correctly learns to execute queries to maximize reward in a reinforcement learning setting.

Keywords: Reinforcement learning, Natural language, Question answering

1 Introduction

Reinforcement learning agents have recently begun to be deployed in real world applications, but these systems still tend to be repetitive tasks with well defined state spaces. Better generalization is needed for applications that interact with the more dynamics parts of the real world. Towards this end, researchers have been attempting to assist reinforcement learning agents using natural language, an abundant type of data that is easy for humans to provide and understand. In this work we explore using domain data found in the wild to help agents learn better in given domain.

Using text data found in the wild is important for scaling reinforcement learning methods to real world applications (Luketina et al., 2019). Some of the most abundant sources of informative language in the wild are data specific to a domain but independent of any particular task in that domain (e.g. internet forums, wikis, and other documentation). This type of data tends to be noisy, large, and filled with information irrelevant for the current task. While most previous work informing reinforcement learning agents with natural language uses text that is generated from rules and templates (Chevalier-Boisvert et al., 2018; Zhong et al., 2019) or crowdsourced for a specific task (Chen et al., 2019; Shridhar et al., 2020; Hanjie et al., 2021), we use challenging task-independent domain text found in the wild.

We propose teaching reinforcement learning agents to actively query natural language sources for information to help in its current task. Humans often query the internet to help them generalize to new tasks. We identify three major components to teaching artificial agents to do the same. Those are *how*, *when*, and *what* to query. We address the first two in this work and leave the latter for future work.

To address the *how*, we use a pretrained QA model to extract structured representations from text in a zero-shot setting. Next, we teach an agent *when* to query by adding a query action in the agent's action space. Our agent successfully retrieves relevant information from text and only does so when required for the current task. We use the nethack learning environment for our experiments along with the game's online wiki pages (Küttler et al., 2020).

	Resistance				Attack			
	recall	precision	F1	IOU	recall	precision	F1	IOU
Keyword	0.62	0.64	0.63	0.46	**0.98**	0.15	0.26	0.15
UnifiedQA	**0.81**	**0.72**	**0.76**	**0.61**	0.74	**0.53**	**0.62**	**0.45**

Table 1: Metrics for predicting monster resistance and damage types from nethack wiki pages. The keyword baseline searches the page for the resistance and attacks type directly. The UnifiedQA method searches the QA model output for the resistance and damage keywords.

2 Related Work

2.1 Task & Domain Independent Language

Language is becoming a popular method for informing reinforcement learning agents. This is partly motivated by recent advancements in powerful pretrained language models. Language models have been used in reinforcement learning applications to process text data such as natural language instructions (Hill et al., 2020) or observations grounded in text (Ammanabrolu et al., 2020; Yin et al., 2020; Li et al., 2022). Language models are generally found to increase generalization capability.

Other work, more closely related to ours, has attempted to impart common sense to reinforcement learning agents from the language model itself (Dambekodi et al., 2020; Ahn et al., 2022). Because lanuage models are trained on vasts amount of data that are independent of both an agent's target domain and task, this approach limits knowledge to information that the language model was repeatedly exposed to during training. This is usually referred to as common sense knowledge and doesn't provide the same type of knowledge found in domain dependent language.

2.2 Task & Domain Dependent Language

Language that is both task and domain dependent is specific to the current task and can provide information about the agent's current reward and transition functions. For example, Zhong et al. (2019) and Hanjie et al. (2021) train agents conditioned on descriptions on the current environment and goal. These descriptions are gathered using generation templates and crowdsourcing respectively.

In an effort to work towards methods that scale, we choose to use language that is found in the wild. We found that language found in the wild is often domain dependent but usually task-independent. We believe that focusing on this type of language data will lead to valuable research as it is more applicable than domain independent data but more available in the wild than task dependent data.

3 Learning How to Query

Text in the wild often consists mostly of information that is irrelevant to our current task. Also, there are typically too few instances found in the data to train models from scratch. Thus, our goal is to suggest a method for extracting specific information from language with little to no training data. To this end we propose using pretrained QA models prompted with task specific questions.

3.1 Zero-shot Performance

We analyze the ability of the QA Model to perform in a zero-shot setting by testing its ability to recover monster resistance and damage types from the nethack wiki (`https://nethackwiki.com/`). We label a small evaluation set of monster data for 98 pages. Monsters are labeled as having a set of resistance and attack types. There are eight potential resistances and 17 potential attack types. Table 1 shows the performance of a three billion parameter pretrained UnifiedQA model (Khashabi et al., 2020) compared with a simple keyword search baseline.

For the resistance task we prompted UnifiedQA with the contexts "What is it resistant to?" and "What is it's resistance?". For the attack task we prompt the model with "What attack does it do?" and "What type of damage does it do?". We check the joint generated output for the questions for the set of resistance and attack types. Our reported metrics compare this predicted set with the labeled set.

The attack type task was the more of the two because of the greater number of attribute classes and because attack types of other monsters were commonly mentioned on a monster's wiki page. This made precision on this task suffer, but, for the most part, the UnifiedQA model was able to filter these irrelevant mentions of attack types.

466

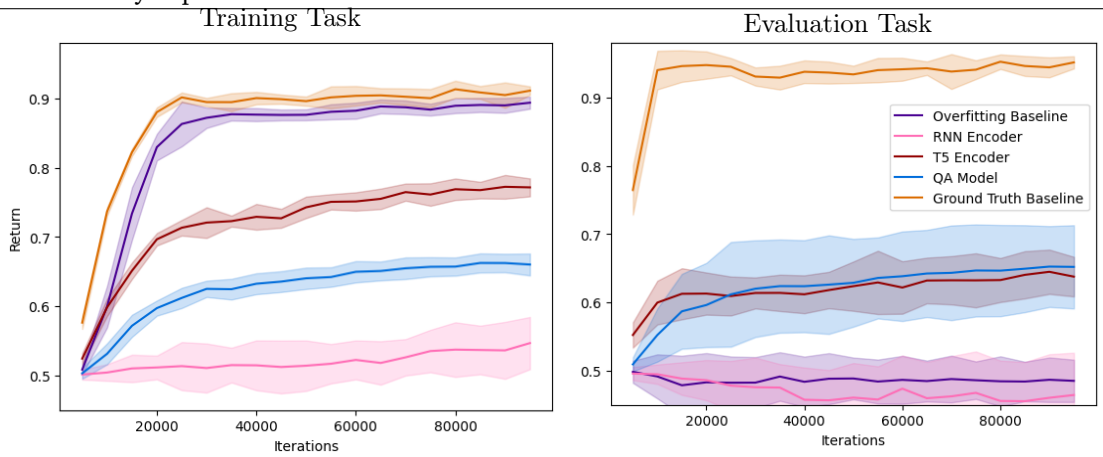


Figure 1: Returns for each method on the training and evaluation tasks. We perform an 80:20 training evaluation split over the nethack monsters. This way, the evaluation task only sees novel monsters. Error bars represent standard deviation over 10 random seeds.

3.2 Representation Generalization

To compare methods for extracting information from natural language data, we use a contextual bandit problem based on the nethack learning environment (Küttler et al., 2020) to compare the below text representation methods.

No External Source Baseline: This method trains a value function on a one-hot encoding of the state with no additional knowledge. It is expected to overfit.

RNN Encoding: We train a single layer LSTM to encode the provided language data. The final hidden layer of the LSTM is passed to the value function and both are trained end to end.

LM Encoding: This approach uses a pretrained LM with frozen weights to encode language data. The extracted text features is passed to the value function as a vector encoding.

QA Model: The QA approach defines a fixed set of questions to ask using the language data as context. The language data is then represented as a binary vector encoding the responses to each question. We write a set of questions manually to prompt the model.

Ground Truth QA Baseline: The ground truth baseline is identical to the QA model but with ground truth query responses rather than responses returned by the model. This method represents ideal generalization with the caveat that it may provide information that doesn't exist in the original external language source.

In our setup, the agent is given the goal of receiving a target resistance (fire, shock, sleep, poison, or cold) by consuming a monster corpse. It is then given a choice between two monsters (out of 388) along with the wiki pages each monster. One of the monsters is guaranteed to confer the target resistance. The agent receives a reward of one if it chooses the monster that will confer the target resistance and a zero otherwise.

Figure 1 shows the results of our contextual bandit over 100,000 iterations. As expected the baselines both perform well on the training task while only the ground truth baseline generalizes to the evaluation task. Due to the extra monster identifying information contained in the LM encoded representation, it outperforms the QA model on the training task, but it does not generalize any better than the QA model on the evaluation task. The RNN struggles to encode the needed information from the wiki pages on both tasks. The only methods that do not overfit to the training data are the QA model and the ground truth baseline.

4 Learning When to Query

In partially observable environments, a reinforcement learning agent is given an imperfect observation of the current environment state. Additional natural language data can be viewed as providing additional observations to an agent to help disambiguate between states. When actively querying language data, an agent should be able to learn to query only in states where the additional information will increase its expected return.

	Reward	Weapon Choice	Query Relevance
Baseline Agent	2.16	0.55	
Query Agent	2.25	0.89	0.64
Query+Explore Agent	**2.28**	**0.95**	**0.67**
Oracle Agent	2.34	0.98	

Table 2: Performance of reinforcement learning agents in nethack. Reward is the number of monsters killed in an episode. Weapon choice is the percent of time that the agent chose to attack with the correct weapon for a corresponding monster. Query relevance is the percent of queries the agent makes that were significant in determining the correct weapon choice. The querying agent is compared to agents that have all important information without needing to query (Oracle) and an agent that is deprived of that information and cannot query (Baseline). Finally, we experiment with adding a special exploration policy specifically for queries (+Explore).

We formulate querying as part of the reinforcement learning problem by adding a query action to the agent's action space. In our experiments, executing a query receives the same timestep penalty that all actions receive. This way the agent learns to only query when it is optimal to do so.

We test this method by designing custom nethack levels using the minihack library (Samvelyan et al., 2021). We spawn an agent with two choices of weapon in its inventory in a level full of four types of monsters. For two of these monsters, the choice of weapon the agent attacks with is significant because the monsters have resistances to one item or the other. For the other two, both weapons work equally well. We give the agent the ability to query the resistances of nearby monsters using the monster wiki pages and the QA method described in section 3. The agent is rewarded each time it successfully slays a monster until it dies or the episode horizon is reached.

Table 2 shows the results of our querying agent compared with baselines that either always or never have the monster resistance information. We report the reward along with how often the agent chooses to attack with the correct weapon and how often the agent's queries are for the monsters with different resistances. The agent should be able to learn to manage its inventory to always attack with the optimal weapon. It should also learn to only query the wiki when necessary, avoiding unnecessary queries about the monsters with identical resistances.

We found that the base querying agent took nearly eight times longer to learn to use resistance information in this task. To speed up this process we implement an exploration policy for taking the query action. At the start of training, the agent is forced to take the query action with 25% probability. This probability decreases to zero over the first five million steps of training. Doing this helps the agent learn to utilize queries sooner. As shows in table 2, the query+explore agent converged in the same amount of time as the baseline and oracle agents and learned a better policy.

5 Conclusion

In this work, we teach a reinforcement learning agent how and when to query for additional information about its environment. We use natural language data found in the wild for the agent to query and discuss how to overcome the challenges that come from this type of data. We find that pretrained QA models are good at retrieving structured zero-shot representations of text. Additionally, reinforcement learning agents can learn when to query this information and when not to.

In future work, we're excited to see reinforcement learning agents with more advanced methods for querying. In addition, further work should explore what to query in the current context of the environment in addition to how and when.

References

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

P. Ammanabrolu, E. Tien, M. Hausknecht, and M. O. Riedl. How to avoid being eaten by a grue: Structured exploration strategies for textual worlds. *arXiv preprint arXiv:2006.07409*, 2020.

H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

S. Dambekodi, S. Frazier, P. Ammanabrolu, and M. O. Riedl. Playing text-based games with common sense. *arXiv preprint arXiv:2012.02757*, 2020.

A. W. Hanjie, V. Y. Zhong, and K. Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 4051–4062. PMLR, 2021.

F. Hill, S. Mokra, N. Wong, and T. Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020.

D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, and H. Hajishirzi. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700*, 2020.

H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

S. Li, X. Puig, Y. Du, C. Wang, E. Akyurek, A. Torralba, J. Andreas, and I. Mordatch. Pre-trained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.

M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

X. Yin, R. Weischedel, and J. May. Learning to generalize for sequential decision making. *arXiv preprint arXiv:2010.02229*, 2020.

V. Zhong, T. Rocktäschel, and E. Grefenstette. Rtfm: Generalising to novel environment dynamics via reading. *arXiv preprint arXiv:1910.08210*, 2019.

Revisiting Model-based Value Expansion

Daniel Palenicek
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
palenicek@robot-learning.de

Michael Lutter
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
michael@robot-learning.de

Jan Peters
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
jan@robot-learning.de

Abstract

Model-based value expansion methods aim to improve the quality of value function targets and, thereby, the effectiveness of value function learning. However, to date, these methods are being outperformed by Dyna-style algorithms with conceptually simpler 1-step value function targets. This shows that in practice, the theoretical justification of value expansion does not seem to hold. We provide a thorough empirical study to shed light on the causes of failure of value expansion methods in practice which is believed to be the compounding model error. By leveraging GPU based physics simulators, we are able to efficiently use the true dynamics for analysis inside the model-based reinforcement learning loop. Performing extensive comparisons between true and learned dynamics sheds light into this black box. This paper provides a better understanding of the actual problems in value expansion. We provide future directions of research by empirically testing the maximum theoretical performance of current approaches.

Keywords: Model-based Reinforcement Learning, Value Expansion

Acknowledgements

This work was funded by the Hessian Ministry of Science and the Arts (HMWK) within the projects "The Third Wave of Artificial Intelligence - 3AI" and hessian.AI. Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

1 Introduction

In recent years a large fraction of the reinforcement learning (RL) community has been focused on model-based RL to improve the sample complexity. Model-based RL algorithms consist of an iterative process of jointly learning a dynamics model from data and then leveraging the learned model in a model-free RL training loop. The learned models have been used for data augmentation [8, 9, 11], improving the value targets [2, 4, 12, 13], improving the policy gradient [7] or any combination thereof. These works have proposed various approaches for training the model, new model architectures, (automatically) adapting the rollout horizons and computing better value targets.

A common understanding among most model-based RL approaches is that the compounding model error along modelled trajectories is one of the main problems to be solved or at least avoided. This compounding model error results in modelled trajectories drifting away from the true trajectory even though they start from the same states and execute the same action sequence. Learning more accurate dynamics models is often believed to be key.

Two key takeaways that have been used by many of these model-based RL papers and have manifested in recent literature are using (1) Short model-rollout horizons and (2) Heteroskedastic ensemble dynamics models.

Short model-rollout horizons As the learned models are at best approximately correct, model errors accumulate with the length of the rollout horizon. Therefore, one common practice introduced by Janner et al. [8] is to use shorter rollout horizons with learned models. Otherwise, one exploits the approximation error, and the RL agent fails to learn the task. This approach can be vaguely thought of as *treating the symptoms* of model errors by behaving pessimistic and cutting rollouts early before the accumulating error can become too large. In practice, this paradigm is often taken to the extreme by using 1-step model rollouts only. Furthermore, using shorter rollouts contradicts the theoretical insights we have into value expansion methods.

Heteroskedastic ensemble dynamics models Initially proposed by Chua et al. [3], this model has been widely adopted in most subsequent papers. The main benefit is that the model learns the aleatoric uncertainty separately from the epistemic uncertainty. It is an attempt to construct a more capable model architecture which is better suited to represent the environment dynamics. A further benefit is that by explicitly modelling uncertainties they can be used down the line.

Both of these takeaways are an attempt to treat model errors. The first, by reducing the length of the prediction horizon, and the second by explicitly learning uncertainty measures which can be leveraged later down the line. This leads us to two lines of questioning which might challenge the understanding of the current approaches.

First, if we learned a perfect dynamics model, would this solve all of the problems that current model-augmented actor-critic approaches struggle with? And if this were the case, could we then just simply use longer horizons and obtain even greater increases in sample efficiency? The relevance of different possible future research directions is linked directly to the answer of these questions. If the answer is *yes*, then we should focus future research onto learning more accurate models. If the answer turns out to be *no*, however, it might be the case that greater accuracy has diminishing returns for model-augmented actor-critic approaches or hinders necessary exploration for example. Striving for more accurate models in these approaches might then not be the most important priority and interesting new research directions could open up.

Second, are stochastic models really necessary to achieve good results? Or can deterministic models deliver comparable performance if built and trained carefully? Current benchmark environments usually feature deterministic dynamics. Naturally, the question arises, of whether a deterministic model should not be sufficient at learning to model these systems. If the answer is *yes*, we should revisit deterministic models with the possibility of cutting down complexity.

To come closer to answering these questions, we believe, that there is a need for extensive empirical analysis of the impact of model errors on the training algorithms. In this paper, we focus on the first line of questioning. We investigate the question of whether learning more accurate dynamics models can still increase the performance of value expansion methods [4]. Therefore, we create an experimental setup with a perfect dynamics model, by replacing the learned dynamics model with an oracle dynamics model. This allows us to study the theoretical performance of value expansion approaches in isolation without the negative impact of model errors. Only recently, with the development of GPU based physics simulators, this type of study has becoming computationally feasible. Simulators like BRAX [5], provide GPU accelerated simulation of dynamical systems scaling to thousands of parallel environments. It allows us to perform fast, oracle dynamics rollouts in the inner model-based RL training loop for entire batches in parallel. An additional performance benefit is that we are able to perform the training on the GPU only which limits the amount of costly memory transfers from CPU to GPU and back.

2 Maximum-Entropy Model-Based Value Expansion

We adapt Model-based Value Expansion (MVE) [12] for the maximum-entropy RL case in order to combine it with a model-free Soft Actor-Critic (SAC) [6] learner. For this, consider a Markov Decision Process (MDP) [10], defined by the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho, \gamma\}$ with state space $\mathcal{S} \subseteq \mathbb{R}^n$ and action space $\mathcal{A} \subseteq \mathbb{R}^m$. At each time step t, the agent observe a state $s_t \in \mathcal{S}$ and samples an action $a_t \in \mathcal{A}$ according to a policy $a_t \sim \pi(\cdot \mid s_t)$. The environment returns a next state $s_{t+1} \in \mathcal{S}$ according to the transition probability density function $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$ and the corresponding scalar reward $r_t = \mathcal{R}(s_t, a_t)$. The starting state of a trajectory is sampled from the initial state distribution $s_0 \sim \rho$. γ is a discount factor. The main objective of maximum-entropy RL is to find a policy π that maximizes $J(\pi) = \mathbb{E}_{s_0 \sim \rho}\{\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) - \alpha \log \pi(\cdot \mid s_t))\}$ with the initial state distribution ρ. The actor loss is defined as $J_\pi(s_t, a_t) = \alpha \log\big(\pi(a_t|s_t)\big) - Q(s_t, a_t)$ with $a_t \sim \pi(s_t)$. In the maximum-entropy case, the value expansion used within the critic loss is described by

$$V^H(s_0) = \sum_{t=0}^{H-1} \gamma^t \Big[r(s_t, a_t) - \alpha \log \pi(\cdot \mid s_t) \Big] + \gamma^H \Big[Q(s_H, a_H) - \alpha \log \pi(\cdot \mid s_H) \Big].$$

The corresponding critic loss is defined as $J_Q(s_t, a_t, s_{t+1}) = \frac{1}{2}[Q^H_{tar} - Q(s_t, a_t)]^2$ with $Q^H_{tar}(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \gamma V^H(s_{t+1})$. We define a learned dynamics model as an ensemble of N probabilistic neural networks $\hat{\mathcal{P}}_\Phi = \{p^i_\phi(s_{t+1}, r_t | s_t, a_t)\}_{i=0}^N$, which output mean and variance of the state transition and reward, similar to [3]. At inference time, one network in the ensemble is sampled uniformly to capture epistemic uncertainty.

3 Experiments

In this section, we compare the theoretical performance of MVE with its practical performance. For this purpose, we construct a version of MVE where we replace the learned dynamics model with an oracle dynamics model. For clarity, we will refer to the latter as *Oracle-based Value Expansion (OVE)*. OVE creates a well-defined, artificial environment for studying training performance by eliminating the negative impact of model errors. It lets us answer whether there is still room for performance gains by learning more accurate dynamics models and if, in the absence of model errors, ever longer rollout horizons can increase performance of value expansion methods further. Our experiments focus on five standard RL benchmark environments: InvertedPendulum, Cartpole Swingup, Hopper, Walker2d and HalfCheetah. As an efficient GPU based physics simulator, we use BRAX [5], which provides implementations of these benchmark environments. Our experiments are implemented in JAX [1] to integrate seamlessly with BRAX and take full advantage of the GPU.

3.1 Training Performance

We compare the training performance of MVE and OVE. Therefore, we train both algorithms with varying rollout lengths. Figure 1 shows the MVE and OVE training performance on the top and bottom row, respectively. For comparison, we provide a SAC baseline (corresponding to MVE/OVE with a rollout horizon of 0). We plot the mean and standard deviation across five random seeds.

OVE shows a clear trend that the theoretical improvements of increased horizons can be achieved in the absence of model errors. As expected, the improvements have diminishing returns with increased rollout horizons. Where Walker2d and HalfCheetah suffer a slight performance decrease for $H = 30$. From a practical perspective, there seems to be an optimal trade-off between the benefit of a longer rollout horizon and the increase in computational cost.

Overall, MVE results are split. While longer rollout horizons assist the training performance in InvertedPendulum, Cartpole and Walker2d, the training performance of Hopper and HalfCheetah suffers immensely. We assume that the learned model is not accurate enough to produce better value targets for the learning agent in these two environments due to the compounding model error.

3.2 Diminishing Returns of Longer Rollout Horizons

We further investigate the diminishing returns in training performance using longer rollout horizons. Figure 2 shows the number of environment steps that MVE/OVE with a certain rollout horizon requires to *first* reach a set threshold on the episode reward. The differently shaded lines represent different thresholds. The thresholds are linearly interpolated between a [min, max] episode reward which for the different environments we have picked as follows: InvertedPendulum = [50, 200], Cartpole = [100, 400], Hopper = [250, 1000], Walker2d = [500, 1900], HalfCheetah = [500, 3000].

OVE shows the tendency of diminishing improvements for longer rollouts. While most environments show notable improvements by increasing rollout horizons from $H = 0$ to $H = 5$ or even $H = 10$, the lines flatten for horizons $H = \{20, 30\}$. Even in the absence of model errors, increasing the rollout horizon appears to reach limitations.

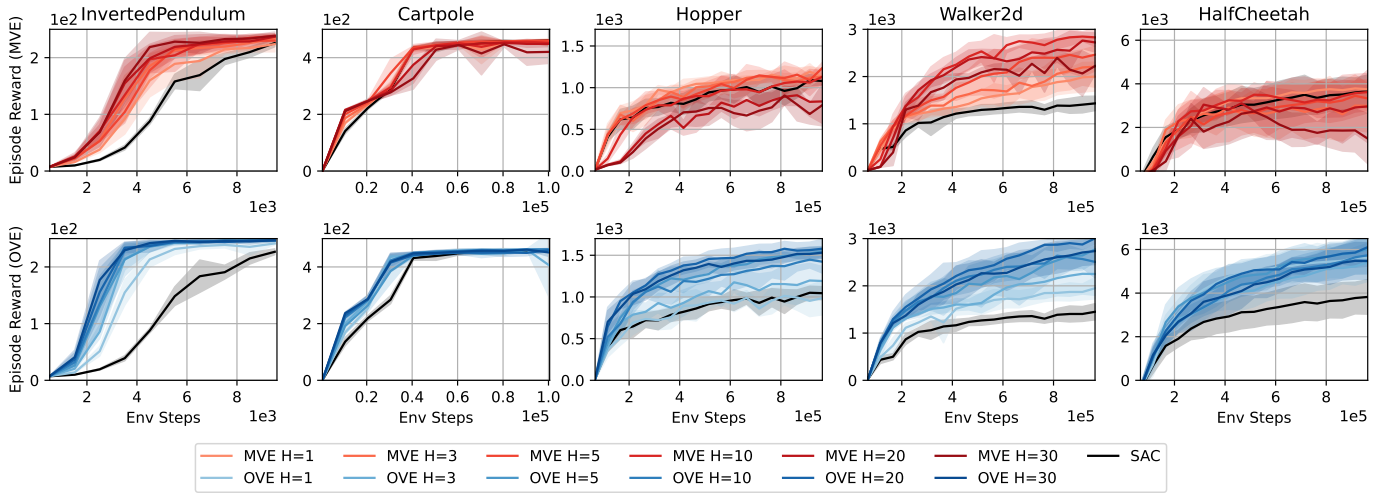


Figure 1: MVE training performance (top) and OVE training performance (bottom). We evaluate each for multiple rollout horizons $H \in \{1, 3, 5, 10, 20, 30\}$ and plot the mean and variance across 5 random seeds.

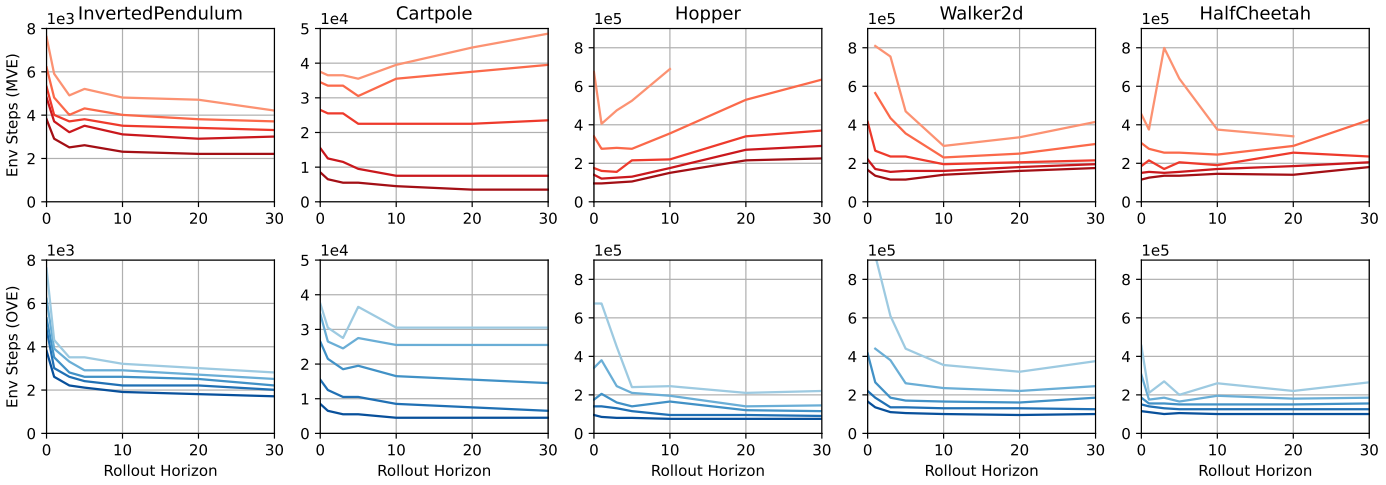


Figure 2: Number of environment steps until MVE/OVE with rollout horizons $H \in \{1, 3, 5, 10, 20, 30\}$ reach a certain threshold of episode reward. The different thresholds are represented by the different shades of red/blue.

Except for the InvertedPendulum environment, MVE experiments show that in the case of a learned dynamics model, rollout horizons above $H = 3$ or $H = 5$ indeed hurt the overall performance. Up to a point where it is not able to solve HalfCheetah for $H = 30$. This is clear evidence that more accurate dynamics models could improve MVE training performance for short rollout horizons. However, due to the diminishing returns of increasing rollout horizons it has its practical limitations for longer rollout horizons.

4 Conclusion

Our experiments have empirically shown that in the absence of model errors, MVE shows increased performance with longer rollout horizons. Therefore, we conclude that MVE can be made more sample efficient by training more accurate dynamics models. At the same time, we have seen diminishing returns of that improvement with increasing rollout horizons. Our empirical findings strengthen the theoretical justifications of MVE by Feinberg et al. [4] and allow for two streams of future research. First, improving model accuracy through better model training techniques and architectures. Second, understanding *how* model errors impact value expansion and how the negative impact can be mitigated. This needs more research into analyzing and understanding how these model errors negatively impact training.

In the future, we plan to take a detailed look at the exact nature of the impact of model errors on the learning process and on the generated value targets themselves. We hope that by understanding the effects, we can design more capably algorithms that are more robust to model errors and sample efficient at the same time.

References

[1] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. 2018. URL: http://github.com/google/jax.

[2] Jacob Buckman et al. "Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion". In: *Advances in Neural Information Processing Systems*. 2018.

[3] Kurtland Chua et al. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems*. 2018.

[4] Vladimir Feinberg et al. "Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning". In: *International Conference on Machine Learning*. 2018.

[5] C. Daniel Freeman et al. *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. Version 0.0.10. 2021. URL: http://github.com/google/brax.

[6] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *arXiv preprint arXiv:1801.01290* (2018).

[7] Nicolas Heess et al. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015.

[8] Michael Janner et al. "When to Trust Your Model: Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems*. 2019.

[9] Thanard Kurutach et al. "Model-Ensemble Trust-Region Policy Optimization". In: *arXiv preprint arXiv:1802.10592* (2018).

[10] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[11] Richard S Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine Learning* (1990).

[12] Junjie Wang et al. "Dynamic Horizon Value Estimation for Model-based Reinforcement Learning". In: *arXiv preprint arXiv:2009.09593* (2020).

[13] Chenjun Xiao et al. "Learning to combat compounding-error in model-based reinforcement learning". In: *arXiv preprint arXiv:1912.11206* (2019).

Adversarial poisoning attacks on reinforcement learning-driven energy pricing

Sam Gunn *
Department of Computer Science
University of California, Berkeley
gunn@berkeley.edu

Doseok Jang *
Department of Computer Science
University of California, Berkeley
djang@berkeley.edu

Orr Paradise *
Department of Computer Science
University of California, Berkeley
orrp@eecs.berkeley.edu

Lucas Spangher
Department of Computer Science
University of California, Berkeley
lucas_spangher@berkeley.edu

Costas J. Spanos
Department of Computer Science
University of California, Berkeley
spanos@berkeley.edu

Abstract

Complex controls are increasingly common in power systems. Reinforcement learning (RL) has emerged as a strong candidate for implementing various controllers. One common use of RL in this context is for prosumer pricing aggregations, where prosumers consist of buildings with solar generation and energy storage. Specifically, supply and demand data serves as the observation space for many microgrid controllers who are passed on a policy from a central RL agent, who is learning online. Each controller outputs an action space consisting of hourly "buy" and "sell" prices for energy throughout the day; in turn, each prosumer can choose whether to transact with the RL agent or the utility. The RL agent is then rewarded through its ability to generate a profit.

RL is known to be effective for this task. We ask: what happens when some of the microgrid controllers are compromised by a malicious entity? We demonstrate a novel attack in RL and a simple defense against the attack.

At a high level, our attack perturbs each trajectory to reverse the direction of the estimated gradient. We demonstrate that if data from a small fraction of microgrid controllers is adversarially perturbed, the learning of the RL agent can be significantly slowed. With larger perturbations, the RL aggregator can be manipulated to learn a catastrophic pricing policy that causes the RL agent to operate at a loss. We demonstrate other environmental characteristics are worsened too: prosumers face higher energy costs, use their batteries less, and suffer more transformer power violations when the pricing aggregator is adversarially poisoned.

We address this vulnerability with a "defense" module; i.e., a "robustification" of RL algorithms against this attack. Our defense identifies the trajectories with the largest influence on the gradient and removes them from the training data. Intuitively, it works because the non-poisoned trajectories are not expected to have out-sized gradients. It is computationally light and reasonable to include in any RL algorithm.

Acknowledgements

We are very thankful to Sergey Levine for thoughtful comments and guidance throughout all stages of this work. This research is supported by the *Simons Collaboration on the Theory of Algorithmic Fairness*, the *DARPA GRAIL project*, and by the *National Research Foundation, Prime Minister's Office, Singapore* under its *Campus for Research Excellence and Technological Enterprise (CREATE) programme*.

*Equal contribution

1 Introduction

Artificial Intelligence (AI) heralds great benefits to power system operation. In the future, AI-based controls could manage the use of passive appliances [19, 3], orchestrate demand response [2], and optimize power flow throughout networks [4, 5]. In the context of energy grids, local grid networks (i.e., microgrids) enable refined control at the cost of increased complexity, necessitating adoption of complex controls at scale.

At the same time, energy grids are known to be lucrative targets for cyberattacks (e.g., [8]). Our work investigates the robustness of an AI-based microgrid controller to malicious actors. We present a novel attack that enables a few compromised microgrid controllers to adversely affect the behavior of connected controllers by *poisoning the data* on which it is trained. This expands on a recent explosion of interest in adversarial attacks [10, 12, 6]. We pair this finding with a gradient-based defense that eliminates the threat of this attack.

More concretely, we examine a setting in which a network of microgrid controllers collect supply and demand data that are continually aggregated by a central agent. The agent uses online reinforcement learning (RL) to optimize its profits. In our attack, a few microgrid controllers are compromised by a malicious adversary. The adversary applies a perturbation to the collected data, severely impacting the provider and *the entire network* of controllers. The provider is made to operate at a loss, and all prosumers are made to pay higher energy costs, use their batteries less, and violate more transformer power constraints.

Our work is set against a backdrop of developments in energy grid control that hold both promise and peril: RL-based controllers allow for sophisticated control in unprecedented granularity. Yet, we must be careful to minimize risk enabled by the opaque nature of deep learning. Our attack stands out in its subtlety and its scope. Other forms of large-scale interference such as blackouts and line disruptions are, by definition, easily detectable and local. Yet our attack causes harm by interfering with the agent's learning, and may not be detected until significant financial damage has been incurred. Furthermore, by interfering with the central agent's learning, our methods can damage systems that are physically disconnected from the energy grid under attack.

Outline. In Section 2 we briefly contextualize our work within RL and energy controls. In Section 3 we describe the threat model, attack, and defense. In Section 4 we introduce our experimental setup, which is used in Section 5 to evaluate the efficacy of our attack and defense in an energy grid environment. Finally, in Section 6 we discuss limitations and future work.

2 Background: RL for prosumer energy pricing

RL has been applied to a number of demand response situations in prosumer microgrids; most work centers on agents that directly schedule resources [15, 16] or control appliances [19, 11, 20]. Recent works have used an RL controller as a price setter in a market: RL has been used to estimate dynamic prices in a multi agent environment of demand response assets in [9], as well as [7, 1, 18].

Demand response, an incentive mechanism geared towards moving consumption, is a no-material solution to variable wind and solar generation and is thus seen as an important technique in the energy transition. It has been demonstrated that learning local price controls is an effective demand response mechanism due to its generalizability and optimal local battery resource utilization [14, 13].

The literature on adversarial attacks for RL in demand response focuses on *responding* to prices [17] rather than *setting* them. To our knowledge, there are no works on adversarial attacks on dynamic price setting for demand response.

3 Techniques

3.1 Threat model

In our setting, N controllers continuously collect data to be aggregated by a centralized agent. Learning takes place over multiple *iterations*; in each iteration, each controller collects a trajectory $\tau \coloneqq (o_i, a_i, r_i)_i$ collected according to the agent policy π_θ. The agent's policy π_θ is described by a neural network. Nodes are required to feed observations through π_θ so as to collect policy-specified actions (pricing schemes), so we assume that the network parameters θ and architecture are shared with the controllers.

The attacker's power is determined by a fraction of *corrupted controllers* $\varepsilon \in (0, 1)$, and a *perturbation bound* $\rho > 0$, as follows: An attacker controls $\varepsilon \cdot N$ of controllers. The attacker *perturbs* the trajectories collected by each compromised controller, causing it to report back a trajectory $\widetilde{\tau}$ instead of the collected trajectory τ. Crucially, these perturbations are

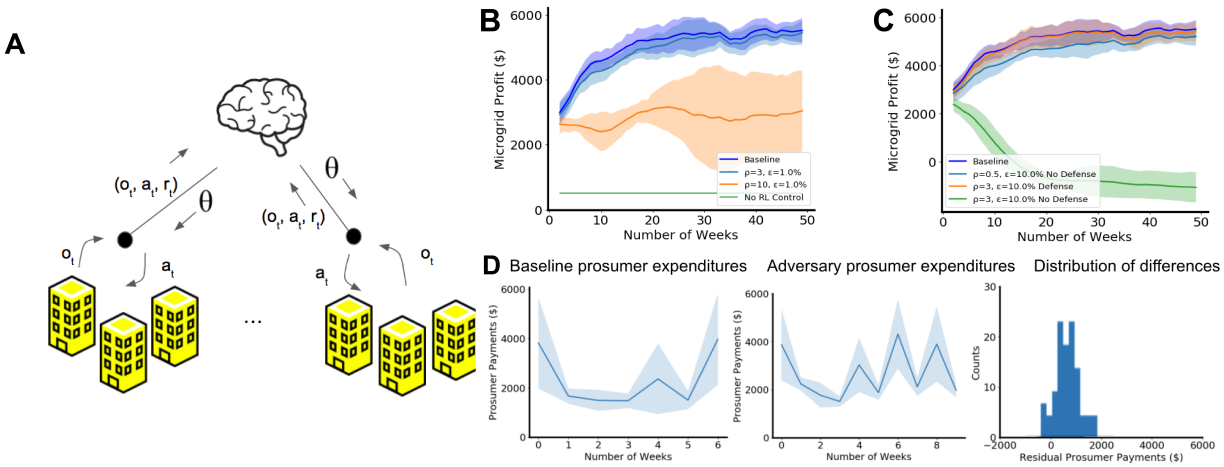


Figure 1: **A.** A description of the microgrid environment. In this figure, the brain is the RL agent, the black dot is the microgrid controller, and the adversary attacks the a_t that is sent back to the RL agent. **B.** Effect of the adversary on the agent's learning. Note that $\varepsilon = 1\%$ corresponds to only one adversarial microgrid. **C.** Effect of our defense in the presence of an adversary. **D.** Characterization of prosumer costs in the baseline and adversarial scenarios. The prosumer consistently pays more in energy when the adversary interferes.

of small norm, that is,

$$\|\widetilde{\tau} - \tau\|_\infty \leq \rho$$

for some *perturbation bound* $\rho > 0$. Note that our attacker adheres to the suggested policy π_θ, but lies about the result to the agent.

Remark 1. In our setting, the attacker may only perturb the actions of each trajectory. Observations and rewards remain unperturbed, because such perturbations would be expensive or easily noticed. This is in contrast to previous work in RL poisoning in which only rewards are poisoned [12].

3.2 The attack

At a high level, our attack aims to perturb each trajectory to reverse the direction of the estimated gradient $\nabla_\theta f(\theta)$. Let θ be the parameters of the agent's policy, τ_P be the unperturbed set of compromised trajectories (the trajectories collected by compromised controllers), $\tilde{\tau}_P$ be the set of perturbed adversarial trajectories (reported back to the agent), and τ_H be the set of honest trajectories (unaffected by the adversary). Our adversary minimizes the correlation of the gradient post-perturbation with the honest one by solving the following constrained optimization problem:

$$\begin{aligned} \min_{\tilde{\tau}_P} \quad & \langle \nabla_\theta f_\theta\left(\tilde{\tau}_P\right), \nabla_\theta\left(f_\theta(\tau_P) + f_\theta(\tau_H)\right)\rangle \\ \text{such that} \quad & ||\tilde{\tau}_P - \tau_P||_\infty \leq \rho. \end{aligned} \tag{1}$$

Since the compromised controllers report $\tilde{\tau}_P$ to the agent instead of τ_P, the agent will take gradient steps according to $\nabla_\theta\left(f_\theta(\tilde{\tau}_P) + f_\theta(\tau_H)\right)$. Therefore, choosing $\tilde{\tau}_P$ to minimize Equation (1) should maximally mislead the gradient towards a sub-optimal policy. Equation (1) is optimized by the adversary using the Fast Gradient Sign method (FGSM) [6]. Interestingly, we find that our adversaries can obtain nearly identical results by solving Equation (1) without the τ_H term, meaning that the adversary does not require any information about the honest (uncompromised) controllers.

3.3 The defense

We propose a defense to protect an online deep RL agent from the attack described in Section 3.2. Our defense works by identifying and removing the trajectories which have the largest influence on the gradient from the training data. Intuitively, this defense works because the honest trajectories are not expected to have out-sized gradients. Note that the poisoned trajectories are not easily identifiable without calculating the gradient through the policy; while the adversarial perturbations significantly influence the gradient estimate, the perturbations themselves are small. More formally, if the RL agent suspects that some fraction $\hat{\varepsilon}$ of the microgrids are adversarially controlled, then, when estimating the gradient $\nabla_\theta f(\theta)$, it ignores the $\hat{\varepsilon}$-fraction of trajectories τ with largest $||\nabla_\theta f_\theta(\tau)||_2$.

4 Experimental setup

4.1 The Price-Setting Microgrid Problem

Consider a setting of 100 microgrids. One RL agent sets the policy parameters θ of all 100 microgrid controllers, which transacts locally within each microgrid. Each microgrid consists of 7 prosumer office buildings. Every prosumer has a battery, solar panel array, and baseline energy consumption; each wants to minimize their energy cost. Prosumers see both grid-set hourly energy buy and sell prices and local microgrid controller-set hourly energy buy and sell prices. Prosumers choose to transact with either the grid or the RL aggregator at each hour. Prosumers also decide when to discharge their battery according to both their demand and the energy prices. The microgrid controller accepts all transactions the prosumers request of it. It does not produce or store energy, but sells energy it has bought from prosumers producing energy in a timestep to prosumers demanding energy in the same timestep. The aggregator balances the net load by purchasing from or selling to the energy utility under which they sit, usually at a loss. As the manager of the RL-aggregator, you see the grid's buy and sell prices, and wish to learn an automatic pricing strategy such that you consistently turn a profit. See Figure 1.A for a graphical depiction of the environment.

For a more precise description of the convex optimizations governing prosumer battery behavior and the reward function training the RL-aggregator, see [1].

4.2 Adversarial microgrid poisoning "in the wild"

We briefly present a potential real-world example of our adversary in action.

Suppose that Eastern Gas & Electric (EG&E) is piloting a dynamic, local pricing program. To do this, EG&E instantiates an RL agent to train across a sample of building clusters (i.e. microgrids grouped locally). Unfortunately, there is an attacker who wishes to disrupt the functioning of EG&E, and they intercept the outflow of data from one of the local microgrid controllers. In one attack strategy, the attacker wishes to minimize the extent to which the outgoing prices are perturbed so as to escape detection. In another attack strategy, the attacker considers high perturbations in order to maximally disrupt profitability.

5 Results

Next, we present experimental results demonstrating the gradient-reversing adversary's harmful potential, as well as the efficacy of the filtering defense.

All of our experiments used the MicrogridLearn environment [1] consisting of 100 microgrids of 7 buildings each. The RL agent is an Actor-Critic agent which updates every week over the course of one year.

The attack. Figure 1.B shows our attacker can significantly hinder the RL agent's learning by co-opting a single microgrid controller. The maximal difference between successive actions taken by the true policy is around 6, so the strongest attack in the single-trajectory setting requires a relatively high perturbation budget $\rho = 10$. However in Figure 1.C, our attack utilizes a smaller perturbation budget of $\rho = 3$ with ten ($\varepsilon = 10\%$) compromised controllers to achieve significant damage.

The defense. We find that our defense recovers the original performance of the RL agent, even under generous ε and ρ. See Figure 1.C.

Characterizations of environmental response. We investigated several ways in which the environment responded to adversarial attack beyond the sheer profit: individual prosumer energy costs (the sum of the building's energy expenditures with the adversary and without), battery utilization (the number of times batteries were charged and discharged, and the total capacities) and transformer power constraint violations. Under all measures, the environment performed worse with an adversary, even those not directly targeted: the prosumers paid on average *more* for the energy, the battery was used *less* when the microgrid controller was adversarially perturbed, and transformer power constraints were violated more. We present the prosumer prices in Figure 1.D and omit the rest due to space constraints.

6 Future Work

The goal of our work is to call attention to the threats made possible by adoption of RL in energy grid pricing. Towards this end, we focused on a narrow yet concrete setting, leaving much room for future work.

478

- The classical adversarial machine learning literature has an abundance of *targeted* attacks, in which the adversary is able to lead the agent towards a particular policy. One of our next objectives is to design an adversary to cause targeted damage, for example to lead the agent to learn a policy which puts the voltage constraints of the power grid at risk.
- Our proposed defense requires the RL agent to drop as many trajectories as could potentially be compromised. More sophisticated defenses could likely result in less dropped data and more robust learning.
- We would like to explore our attack in more environments. In parallel, we hope to achieve successful attacks with smaller settings of ρ and ε.

References

[1] Utkarsha Agwan, Lucas Spangher, William Arnold, Tarang Srivastava, Kameshwar Poolla, and Costas J. Spanos. 2021. Pricing in Prosumer Aggregations using Reinforcement Learning. In *e-Energy '21: The Twelfth ACM International Conference on Future Energy Systems, Virtual Event, Torino, Italy, 28 June - 2 July, 2021*, Herman de Meer and Michela Meo (Eds.). ACM, 220–224.

[2] Donald Azuatalam, Wee-Lih Lee, Frits de Nijs, and Ariel Liebman. 2020. Reinforcement learning for whole-building HVAC control and demand response. *Energy and AI* 2 (2020), 100020.

[3] Bingqing Chen, Zicheng Cai, and Mario Bergés. 2019. Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy. In *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 316–325.

[4] Bingqing Chen, Priya L Donti, Kyri Baker, J Zico Kolter, and Mario Bergés. 2021. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*. 199–210.

[5] Emiliano Dall'Anese, Hao Zhu, and Georgios B Giannakis. 2013. Distributed optimal power flow for smart microgrids. *IEEE Transactions on Smart Grid* 4, 3 (2013), 1464–1475.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[7] Byung-Gook Kim, Yu Zhang, Mihaela Van Der Schaar, and Jang-Won Lee. 2015. Dynamic pricing and energy consumption scheduling with reinforcement learning. *IEEE Transactions on smart grid* 7, 5 (2015), 2187–2198.

[8] Nir Kshetri and Jeffrey M. Voas. 2017. Hacking Power Grids: A Current Problem. *Computer* 50, 12 (2017), 91–95.

[9] Renzhi Lu, Seung Ho Hong, and Xiongfeng Zhang. 2018. A dynamic pricing demand response algorithm for smart grid: reinforcement learning approach. *Applied Energy* 220 (2018), 220–230.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[11] Giuseppe Pinto, Marco Savino Piscitelli, José Ramón Vázquez-Canteli, Zoltán Nagy, and Alfonso Capozzoli. 2021. Coordinated energy management for a cluster of buildings through deep reinforcement learning. *Energy* 229 (2021), 120725.

[12] Amin Rakhsha, Xuezhou Zhang, Xiaojin Zhu, and Adish Singla. 2021. Reward Poisoning in Reinforcement Learning: Attacks Against Unknown Learners in Unknown Environments. *CoRR* abs/2102.08492 (2021). arXiv:2102.08492

[13] Lucas Spangher. 2021. Transactive multi-agent reinforcement learning for distributed energy price localization. In *BuildSys '21: The 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Coimbra, Portugal, November 17 - 18, 2021*, Xiaofan Fred Jiang, Omprakash Gnawali, and Zoltan Nagy (Eds.). ACM, 244–245.

[14] Lucas Spangher, Akash Gokul, Manan Khattar, Joseph Palakapilly, Akaash Tawade, Adam Bouyamourn, Alex Devonport, and Costas J. Spanos. 2020. Prospective Experiment for Reinforcement Learning on Demand Response in a Social Game Framework. In *e-Energy '20: The Eleventh ACM International Conference on Future Energy Systems, Virtual Event, Australia, June 22-26, 2020*. ACM, 438–444.

[15] José R. Vázquez-Canteli, Jérôme Henri Kämpf, Gregor Henze, and Zoltán Nagy. 2019. CityLearn v1.0: An OpenAI Gym Environment for Demand Response with Deep Reinforcement Learning. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2019, New York, NY, USA, November 13-14, 2019*. ACM, 356–357.

[16] José R Vázquez-Canteli and Zoltán Nagy. 2019. Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied energy* 235 (2019), 1072–1089.

[17] Zhiqiang Wan, Hepeng Li, Hang Shuai, Yan Lindsay Sun, and Haibo He. 2021. Adversarial Attack for Deep Reinforcement Learning Based Demand Response. In *2021 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 1–5.

[18] Hanchen Xu, Hongbo Sun, Daniel Nikovski, Shoichi Kitamura, Kazuyuki Mori, and Hiroyuki Hashimoto. 2019. Deep reinforcement learning for joint bidding and pricing of load serving entity. *IEEE Transactions on Smart Grid* 10, 6 (2019), 6366–6375.

[19] Xiangyu Zhang, Xin Jin, Charles Tripp, David J Biagioni, Peter Graf, and Huaiguang Jiang. 2020. Transferable reinforcement learning for smart homes. In *Proceedings of the 1st International Workshop on Reinforcement Learning for Energy Management in Buildings & Cities*. 43–47.

[20] Yuekuan Zhou and Siqian Zheng. 2020. Machine-learning based hybrid demand-side controller for high-rise office buildings with high energy flexibilities. *Applied Energy* 262 (2020), 114416.

479

BLAST: Latent Dynamics Models from Bootstrapping

Keiran Paster*
Department of Computer Science
University of Toronto, Vector Institute
keirp@cs.toronto.edu

Lev McKinney*
Department of Computer Science
University of Toronto, Vector Institute
lev.mckinney@mail.utoronto.ca

Sheila A. McIlraith & Jimmy Ba
Department of Computer Science
University of Toronto, Vector Institute
{sheila, jba}@cs.toronto.edu

Abstract

State-of-the-art world models such as DreamerV2 [4] have significantly improved the capabilities of model-based reinforcement learning. However, these approaches typically rely on a reconstruction loss to shape their latent representations, which is known to fail in environments with high fidelity visual observations. Previous work has found that when learning latent dynamics models without a reconstruction loss by using only the signal provided by the reward, the performance can also drop dramatically. We present a simple set of modifications to DreamerV2 to remove its reliance on reconstruction inspired by the recent self-supervised learning method Bootstrap Your Own Latent [3]. The combination of adding a stop-gradient to the posterior, using a powerful auto-regressive model for the prior, and using a slowly updating target encoder, which we call BLAST, allows the world model to learn from signals present in both the reward and observations, improving efficiency on our tested environment as well as being significantly more robust to visual distractors.

Keywords: model-based reinforcement learning, deep reinforcement learning, representation learning

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (www.vectorinstitute.ai/partners).

*Equal contribution

1 Introduction

Many environments have simple dynamics and yet require millions or billions of potentially expensive agent interactions to be solved using deep reinforcement learning (DRL) algorithms. One promising way to improve the sample efficiency of DRL is by learning a model of the environment and then learning a policy within this model. Model-based RL (MBRL) is particularly efficient when learning a world model is relatively easy compared to learning the policy. Luckily, the state of the art in generative modeling is constantly improving, and as new techniques are discovered, these can bring significant improvements to our ability to learn world models for MBRL, making it a viable approach for learning a growing set of complex tasks.

With recent progress in generative modeling, MBRL can solve control tasks from pixels and master several Atari games. However, MBRL still struggles to solve problems with highly complex observations. This is largely because most MBRL techniques today rely on reconstruction to learn their models, where the world model is trained to predict future frames at a pixel level. This makes MBRL susceptible to changes in how an environment is rendered. While an environment may have fundamentally simple dynamics, it can be rendered with high fidelity textures, in 3D, or with a natural video playing in the background. All of these changes have been shown to cripple the performance of MBRL since a large amount of modeling capacity is taken up by modeling features that are irrelevant to the control task.

DreamerV2 [4], a state-of-the-art MBRL agent that learns a discrete world model, uses reconstruction to form its representations and thus suffers from this problem. The authors describe a modification that doesn't use a reconstruction loss, where it simply learns a latent representation that can predict rewards and the distribution of the next latent state. While learning a latent dynamics model this way should, in theory [2], be sufficient, empirical studies have shown that across multiple tasks, the signal from predicting future observations is critical and without it, performance suffers dramatically [4].

In our work, we show a simple way to regain this performance without using reconstruction or even contrastive [6] losses. Additionally, our method of leveraging image signals for representation learning without reconstruction is significantly more robust to distractors than DreamerV2 and can perform well even when a video is displayed in the background of the environment. Our method, which we call BLAST, consists of the following changes on top of DreamerV2:

- While DreamerV2 experiments with weighing the gradients that go to the prior and posterior in the KL loss, we show that **by completely stopping the gradient from going to the posterior**, the KL loss encourages representations that are informative of the predictable features of the observations.
- Since our method does not update the posterior to be predictable by the prior, we choose to use an **auto-regressive prior** to better fit the latent distribution.
- Inspired by similarities to recent advancements in self-supervised learning, we use techniques from Bootstrap Your Own Latent (BYOL) [3] such as a **slowly updating target network for prior targets to stabilize training** and using **batch normalization in the encoder**.

In order to evaluate our method, we run experiments on a highly-customizable grid world environment as well as continuous control experiments on the DeepMind Control Suite (DMC) [7]. In our grid world experiments, we evaluate the performance of our method on several rendering modifications, including environments with a small agent sprite as well as video backgrounds. We show empirically that DreamerV2 performs poorly on these environments while BLAST can learn a strong world model. On DMC, we find that BLAST can match the performance of several other MBRL agents without relying on reconstruction or contrastive losses. We conduct an extensive ablation study on the proposed changes and find that while all of the changes improve performance, batch normalization makes a substantial difference in several environments. Overall, BLAST represents a simple set of changes to DreamerV2 that enables practical reconstruction-free MBRL.

2 BLAST: Bootstrapped LAtents for Simulating Trajectories

2.1 Dreamer without Reconstruction

In this work, we build off of DreamerV2, a state-of-the-art MBRL algorithm. In Dreamer [4], a latent dynamics model called a Recurrent State-Space Model (RSSM) [4] is learned. This model consists of:

$$\begin{aligned}
&\text{Recurrent model:} && h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}) \\
&\text{Representation model:} && z_t \sim q_\theta(z_t|h_t, x_t) \\
&\text{Transition predictor:} && \hat{z}_t \sim p_\theta(\hat{z}_t|h_t) \\
&\text{Image predictor:} && \hat{x}_t \sim p_\theta(\hat{x}_t|h_t, z_t) \\
&\text{Reward predictor:} && \hat{r}_t \sim p_\theta(\hat{r}_t|h_t, z_t) \\
&\text{Discount predictor:} && \hat{\gamma}_t \sim p_\theta(\hat{\gamma}_t|h_t, z_t).
\end{aligned} \tag{2.1}$$

The world model is trained by minimizing the following loss, which corresponds to an ELBO of a hidden Markov model conditioned on the action sequence, making the RSSM a sequential VAE [4, 5]:

$$\begin{aligned}
\mathcal{L}(\theta) \doteq \mathrm{E}_{q_\theta(z_{1:T}|a_{1:T}, x_{1:T})} \Big[\sum_{t=1}^{T} & \underbrace{-\beta_{\text{img}} \ln p_\theta(x_t|h_t, z_t)}_{\text{image log loss}} \underbrace{-\beta_r \ln p_\theta(r_t|h_t, z_t)}_{\text{reward log loss}} \underbrace{-\beta_\gamma \ln p_\theta(\gamma_t|h_t, z_t)}_{\text{discount log loss}} \\
& \underbrace{+\beta_{\text{KL}} \mathrm{KL}(q_\theta(z_t|h_t, x_t) \| p_\theta(z_t|h_t))}_{\text{KL loss}} \Big]
\end{aligned} \tag{2.2}$$

In the original work [4], various representation learning ablations were done, including the removal of the image log loss. However, without image gradients to help form representations, Dreamer fails to learn an accurate world model.

While this naive reward-only version of Dreamer might not work well in practice, there have been several works that learn representations in Dreamer without reconstruction, primarily through contrastive learning. The original Dreamer paper used contrastive learning to learn a representation that has maximal mutual information with the encoded observation. Temporal Predictive Coding (TPC) [6] proposes to augment this loss with a contrastive approach that operates between timesteps, encouraging representations of predictable elements. While Dreamer with contrastive representations did not achieve performance on par with reconstruction-based representations, TPC is competitive with Dreamer with reconstruction on several tasks. Theoretically, TPC learns representations that are sufficient for control, but it is unclear whether the loss is still a lower bound on the log probability of the data. In practice, TPC requires carefully balancing four separate losses to avoid collapsed representations.

2.1.1 Learning Bootstrapped Representations

In order to learn a world model with neither reconstruction nor contrastive losses, we note that there are two ways to optimize the KL loss in DreamerV2. We can change the latent representations from the past to be more informative of future representations (forward) or we can change our future representation to be more predictable by the past (reverse). DreamerV2 [4] introduces an additional hyperparameter called the KL balance, where a parameter α is used to scale the gradients going into the prior and posterior in the KL loss. In experiments on Atari, DreamerV2 uses a KL balance of $\alpha = 0.8$, which updates both the posterior and prior distributions. In this section, we argue that by completely stopping the gradients that update the future representation ($\alpha = 1$), we are able to learn representations not only from the reward signal, but from bootstrapping off of the information present in the embeddings of future observations.

By only allowing gradients to flow into the prior and not the posterior, we recover a representation learning algorithm similar to BYOL [3], a self-supervised learning algorithm that has achieved strong performance on benchmarks without the use of the negative samples that are necessary for contrastive learning. Intuitively, even the representation produced by our encoder at initialization has some information about the observation it is encoding. By predicting this future representation, the past representations are encouraged to contain information that can help with the prediction task, which in turn increases the amount of information that is present in the representation.

In order to stabilize training, we use techniques from BYOL. Primarily, we employ a slowly updating target encoder with parameters ξ to produce the posterior, resulting in the following loss:

$$\begin{aligned}
\mathcal{L}(\theta) \doteq \mathrm{E}_{q_\theta(z_{1:T}|a_{1:T}, x_{1:T})} \Big[\sum_{t=1}^{T} & \underbrace{-\beta_r \ln p_\theta(r_t|h_t, z_t)}_{\text{reward log loss}} \underbrace{-\beta_\gamma \ln p_\theta(\gamma_t|h_t, z_t)}_{\text{discount log loss}} \\
& \underbrace{+\beta_{\text{KL}} \mathrm{KL}(\text{stop_grad}(q_\xi(z_t|h_t, x_t)) \| p_\theta(z_t|h_t))}_{\text{KL loss}} \Big]
\end{aligned} \tag{2.3}$$

The target encoder's parameters ξ are updated as an exponential moving average of θ as $\xi \leftarrow \tau\xi + (1-\tau)\theta$. Since we no longer update our encoder to be more predictable and fit an independent prior, we experiment with using an auto-regressive prior. We also find that batch normalization is important for the best performance of our model.

3 Experiments

We run experiments to evaluate the following hypotheses: **(H1)** The combination of proposed changes (BLAST) enable the learning of a strong world model without the use of reconstruction. **(H2)** BLAST is more robust to changes in environment rendering compared to DreamerV2. **(H3)** The most effective combination of the proposed changes is the combination of all five, which we call BLAST.

3.1 Environment

We primarily run experiments on a modified grid world environment as well as the DeepMind Control Suite (DMC) [7]. We choose to use the dynamic obstacles environment in gym-minigrid [1]. In this environment, three obstacles randomly move to adjacent tiles each time-step and the agent must navigate around them to reach a goal without colliding with any obstacles. This environment was primarily chosen since it is easy to modify while retaining characteristics that make it a good test-bed for testing world models, such as environment stochasticity and sparse reward. We also chose to evaluate on DMC to allow comparison to prior work which tested on this benchmark.

We modified the base environment `MiniGrid-Dynamic-Obstacles-6x6-v0` in several ways, as shown in Figure 1:

- `color_direction`: A different color is used to represent each of the possible directions that the agent could be facing.
  Scenario: *uncertainty in reconstruction can be confused with another possible observation.*
- `smaller_agent`: The agent is rendered as a triangle using one quarter of the pixels as in the original environment.
  Scenario: *important elements have a low weight in the reconstruction loss.*
- `random_frames`: The background is set to a random frame of a fixed video at each time-step.
  Scenario: *environment observations have temporally uncorrelated and uncontrollable elements in the background.*
- `video`: The background is set to a random frame of a fixed video upon resetting the environment. The video is then displayed in order in subsequent steps.
  Scenario: *environment observations have temporally correlated and uncontrollable elements in the background.*

For all experiments, we used a fixed horizon of 100 time-steps. The return is the number of times the agent reaches the green square in the episode minus the number of times it collides with an obstacle. For video background we use a video drawn from the the kinetics400 driving class across all experiments.

3.2 Experimental Setup

We ran all experiments using code forked from the official DreamerV2 [4] implementation. The code was modified to allow the use of batch normalization, a separate target encoder with weights set to an exponentially moving average of online weights, and an autoregressive prior within the RSSM. This allows us to configure DreamerV2 with any subset of our proposed changes and fairly compare the different configurations. Error bars in our plots represent represent standard deviation across 5 seeds.

3.3 Summary of Modifications

- **(-recon)** We remove the reconstruction loss from DreamerV2 by setting its weight to zero.
- **(+SG)** We add a stop gradient to the posterior in the KL loss by setting a KL balance of $\alpha = 1$.
- **(+EMA)** We use an exponential moving average for the encoder when we compute the posterior in the KL loss.
- **(+BN)** We add batch normalization to the encoder.
- **(+AR)** We use an autoregressive prior in the RSSM to better fit the discrete latent distribution.

3.4 Grid World Experiments

Figure 1 shows how performance differs on the grid world environments when the proposed modifications are added to DreamerV2 to create BLAST. DreamerV2 works well on the unmodified environment but struggles to learn in the presence of most of the rendering modifications. BLAST is able to achieve strong performance on all of the rendering modifications, confirming our hypothesis that BLAST would be more robust to adversarial rendering (H2). We found that only BLAST (+AR) and BLAST without an autoregressive prior (+BN) consistently performed well. We note that there are some environments such as `color_direction` where the use of batch normalization is vital to achieve strong performance.

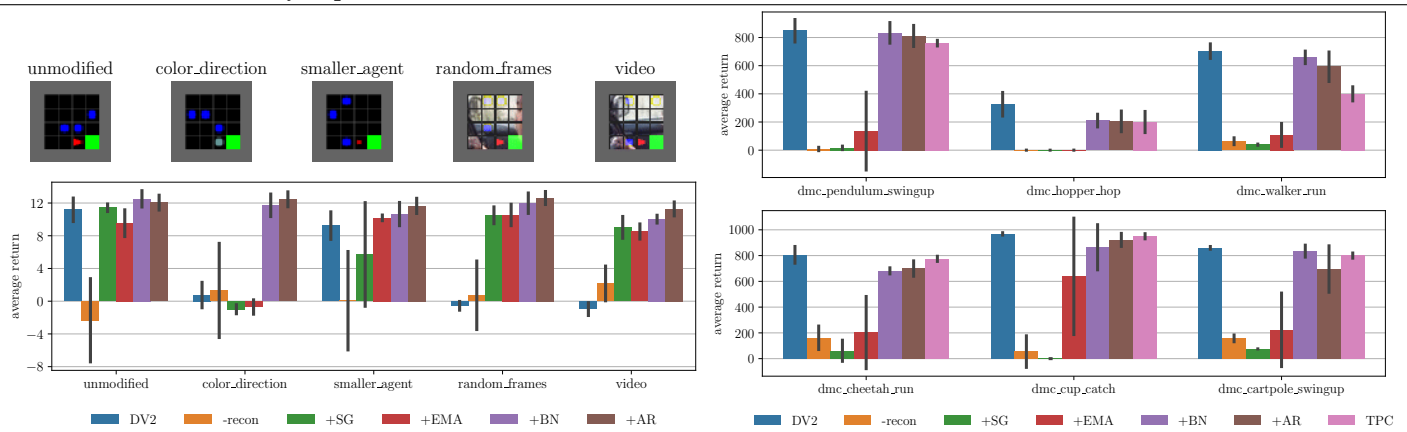


Figure 1: We performed a series of experiments to understand how our proposed changes affect performance across our various grid world modifications. Each subsequent change includes all of the modifications to the left. For example, the rightmost +AR represents the combination of all five changes.

3.5 DeepMind Control Suite

Our experiments on the modified grid world environments confirm that BLAST is significantly more robust to rendering changes than DreamerV2. In order to compare with prior state of the art model based RL algorithms, we ran continuous control experiments on several standard DMC environments. Figure 1 shows that BLAST can achieve performance that is essentially on par with vanilla DreamerV2 and Dreamer. Surprisingly, we found that batch normalization had an even larger effect on these environments, with several tasks achieving practically zero return until batch normalization was added. We also found that the autoregressive prior did not have a significant effect on performance in DMC environments, likely due to the deterministic nature of the environment. Figure 1 also shows that BLAST achieves comparable performance to TPC, both in terms of asymptotic performance as well as sample complexity. Despite using neither reconstruction nor contrastive losses, BLAST achieves performance that is on par with existing state-of-the-art MBRL approaches on DMC tasks.

4 Discussion

We present BLAST, a modification of DreamerV2 which adds a stop-gradient to the posterior, a powerful auto-regressive prior, and a slowly updating target network to learn representations without using reconstruction. BLAST performs far better than DreamerV2 on a range of differently rendered grid world environments, including ones with a video embedded in the background where DreamerV2 fails entirely to learn. We empirically justify our modifications and show evidence that adding the stop-gradient encourages the latent representations to contain more information about the observation.

While BLAST shows strong performance in our grid world environments as well as on continuous control tasks, we acknowledge several limitations and opportunities for future work. The primary limitation of this work is the need for an expressive prior such as an autoregressive model in stochastic environments. Since autoregressive models are considerably slower than the independent prior used in DreamerV2, our model takes longer to run simulations. We also believe that more work should be done to differentiate self-predictive approaches to representation learning such as BLAST to contrastive approaches in order to provide a stronger recommendation to practitioners looking to use reconstruction-free world models.

References

[1] Maxime Chevalier-Boisvert et. al., *". Minimalistic Gridworld Environment for OpenAI Gym,"* GitHub repository.

[2] Carles Gelada et. al., *"DeepMDP: Learning Continuous Latent Space Models for Representation Learning,"* ICML 2019.

[3] Jean-Bastien Grill et. al., *"Bootstrap your own latent - A new approach to self-supervised learning,"* NeurIPS 202.

[4] Danijar Hafner et. al., *"Mastering Atari with Discrete World Models,"* ICLR 2021.

[5] Diederik P. Kingma et. al., *"Auto-Encoding Variational Bayes,"* ICLR 2014.

[6] Tung D. Nguyen et. al., *"Temporal Predictive Coding For Model-Based Planning In Latent Space,"* ICML 2021.

[7] Yuval Tassa et. al., *"dm control: Software and Tasks for Continuous Control,"* GitHub repository.

What to learn next? Aligning gamification rewards to long-term goals using reinforcement learning

Reena Pauly
Max Planck Institute for Intelligent Systems
Tübingen, Germany
reena.pauly@tuebingen.mpg.de

Lovis Heindrich
Max Planck Institute for Intelligent Systems
Tübingen, Germany
lovis.heindrich@tuebingen.mpg.de

Victoria Amo
Max Planck Institute for Intelligent Systems
Tübingen, Germany
victoria.amo@tuebingen.mpg.de

Falk Lieder
Max Planck Institute for Intelligent Systems
Tübingen, Germany
falk.lieder@tuebingen.mpg.de

Abstract

Nowadays, more people can access digital educational resources than ever before. However, access alone is often not sufficient for learners to fulfill their learning goals. To support motivation, learning environments are often gamified, meaning that they offer points for interacting with them. But gamification can add to learners' tendencies to choose learning activities in a short-sighted manner. An example for a short-sighted choice bias is the preference for an easy task offering a quick sense of accomplishment (and in gamified environments often a quick accumulation of points) over a harder task offering to make real progress. The concept of optimal brain points demonstrates that methods from the field of reinforcement learning, specifically reward shaping, allow us to align short-term rewards for learning choices with their expected long-term benefit in a learning context. Building on that work, we here present a scalable approach to supporting self-directed learning in digital learning environments applicable to real-world educational games. It can motivate learners to choose the learning activities that are most beneficial for them in the long run. This is achieved by incentivizing each learning activity in a way that reflects how much progress can be made by completing it and how that progress relates to their learning goal. Specifically, the approach entails modelling how learners choose between learning activities as a Markov Decision Process and applying methods from reinforcement learning to compute which learning choices optimize the learners progress based on their current knowledge. We specify how our developed method can be applied to the English-learning App "Dawn of Civilisation". We further present the first evaluation of the approach in a controlled online experiment with a simplified learning task, which showed that the derived incentives can significantly improve both learners' choice behaviour and their learning outcomes.

Keywords: education; self-directed learning; reward shaping; optimal gamification; decision making; incentives

Acknowledgements

The authors thank the whole team from Solve Education for supporting us with valuable insights into their learning app "Dawn of Civilisations" and how users interact with it as well as for all the fruitful discussions concerning our research. Further thanks go to Valkyrie Felso for providing assistance with the analysis of users' interaction data. This project was supported by grant number 1757269 from the National Science Foundation, a fellowship from the IMPRS for Intelligent Systems to Reena Pauly, and grant number CyVy-RF-2019-03 from the Cyber Valley Research Fund.

1 Introduction

Digital educational resources have become increasingly available over the past years. However, many people struggle to fully exploit the offered resources in the pursuit of their personal learning goals [1, 2]. One possible reason for this is an aversion towards failure, leading to a preference for easy tasks over tasks that might allow them to really progress [2]. This can be viewed as a form of procrastination, as it stems from the tendency to prefer short-term pleasure (succeeding at an easy task) over long-term benefit (developing one's skills or knowledge) [2, 3]. Gamification has often been the tool of choice for helping learners persist through such motivational difficulties [4]. But just adding points or badges to an educational context without a principled theory can elicit negative effects [4]. Specifically, emphasizing momentary performance through game elements can intensify rather than alleviate the problem of striving to avoid failure [5]. Additionally, incentivizing schemes can often be gamed, meaning that a learner can engage in behaviour that will serve to accumulate points or badges without increasing the efficiency or success of their actual learning efforts [2, 5]. This issue can be addressed by leveraging a computational approach to calculating learning incentives in a way that aligns short-term rewards with expected long-term learning benefits. Xu et al. (2019) [2] showed that optimal brain points, derived from an application of optimal gamification to a mathematical model of skill acquisition, can help learners to overcome the described choice biases in a simple learning task. The term "brain points" was coined to describe a growth-mindset incentive system [6] and is meant to emphasize the opportunity to develop one's brain by seeking out challenges. However, optimal brain points cannot be computed for complex tasks and therefore do not translate to the kind of learning environments people encounter in their everyday life.

The work presented here extends the approach by [2] by developing a scalable method for computing brain points that can be applied to boost self-directed learning in the real world. Our goal is to eventually evaluate the developed method within the English learning app "Dawn of Civilisations" (DoC) by the non-profit organisation Solve Education. DoC specifically designed to address the needs of disadvantaged youth [7]. Users take on the role of a a city's mayor and play different minigames conveying different language lessons in order to gain resources to develop the city. Those resources are the in-game rewards, which have been purely performance-dependent so far. Here, we present a formal model of the choice between different learning activities and its application to the learning app. We describe how we compute brain points in a scalable manner and present the results from an online learning paradigm used to evaluate the method prior to its release in the learning app.

2 A scalable method for computing learning incentives

Choosing between educational activities can be modeled as a MDP For the purpose of developing a scalable principled method for supporting self-directed learning in digital learning environment, we model a learner's interaction with such an environment as a Markov Decision Process (MDP). The digital learning environment itself is defined by the following parameters:

- The number of skills to be trained, denoted N_s
- The number of activities a learner can choose between, denoted N_a
- An estimate of the effort required to complete each learning activity, denoted r_a
- A way to quantify a learner's competence at each skill, such that the set of competence values for the i^{th} skill is defined as $\mathcal{C}_i = \{c_1, c_2, ..., c_{N_{c_i}}\}$
- A learning goal (or sequence of learning goals) defining a goal competence value $c_{g_s} \in \mathcal{C}_s$ for each skill which has to be reached by the learner in order to consider the learning goal completed: $\boldsymbol{g} = (c_{g_1}, c_{g_2}, ... , c_{g_{N_S}})$. Goal completion is therefore defined as:

$$d(c, g) = \begin{cases} 1, \text{if } \forall i \in \{1..N_s\} : c_i \geq c_{g_i} \\ 0, \text{otherwise} \end{cases} \tag{1}$$

Using these parameters allows us to define the MDP in the following way:

- The action space of the MDP comprises one action for each learning activity the student could choose: $\mathcal{A} = \{a_1, a_2, ..., a_{N_a}\}$
- The state s encompasses the current competence values $c_s \in \mathcal{C}_s$ of each skill. Additionally, system-specific information β such as task availability or delay periods is needed to estimate the transition probability $P(s, a, s')$.
- The reward function $R(s, a, s')$ incorporates the estimation of the required effort r_a to carry out the chosen learning activity and a positive reward r_g for reaching the learning goal:

$$R(s, a, s') = \begin{cases} r_g - r_a, \text{if } d(c', g) = 1 \wedge d(c, g) = 0 \\ -r_a, \text{otherwise} \end{cases} \tag{2}$$

Applying the model to an educational game Our use case, the English-learning app "Dawn of Civilisations" (DoC) defines 6 skills to be trained: grammar, vocabulary, writing, reading, listening, and speaking ($N_s = 6$). Each skill at each of the 16 competence levels c_i is trained by a set of questions conveyed by 16 different minigames a learner can choose between ($N_a = 16$). Each minigame can train more than one skill depending on its content.

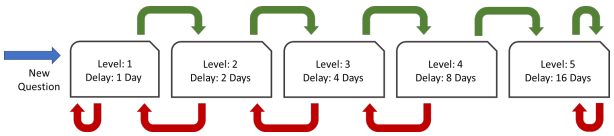


Figure 1: The spaced repetition schedule

As minigames are played, those questions go trough a spaced repetition schedule (see Figure 1). Questions move up a level when answered correctly and down a level when answered incorrectly. The delay between repetitions is longer for questions with higher levels, accounting for the assumption that the number of successful recalls flattens the forgetting curve [8]. The skill level of individual questions allows us to quantify a learner's competence at the trained skills. Specifically, a goal competence level c_{g_i} is considered reached by a learner once 80% of the corresponding questions have reached the highest level of the repetition schedule. Those rules also impact the transition probability $P(s, a, s')$, as it determines which questions will be presented by the chosen minigame and restricts possible state transitions. Consequently, the state s can be defined as the distribution of the questions across the different level- and delay categories. Knowing that distribution allows to both derive the current competence values and possible next states and therefore satisfies the conditions of the state definition laid out above. The probability of reaching a specific next state depends on how likely a learner is to solve the questions presented by the minigame correctly. Those probabilities were estimated from user data (for more details see [9]). The effort required to play the minigames r_a was parameterised as the time required to complete it, which is also estimated from the user data. Having defined this formal model of choosing between learning activities allows to apply the framework of optimal brain points [2].

Scalably computing learning incentives The optimal gamification method for decision support developed by Lieder et al. [3] computes incentives by adding a shaping function $F(s, a, s')$ to the MDP's reward function $R(s, a, s')$. The shaping function expresses the difference in value between two states of the MDP (Eq. 3).

$$F(s, a, s') = \gamma\phi(s') - \phi(s) \tag{3}$$

The potential function $\phi(s)$ estimates the value of being in state s. The optimal choice for $\phi(s)$ is therefore the optimal value function $V^*(s)$ [10]. Consequently, we define optimal brain points as

$$\text{OBP}(s, a) = \sum_{s'} P(s'|s, a) \cdot [R(s', a, s) + \gamma V^*(s') - V^*(s)] \tag{4}$$

Because computing the optimal value function is not tractable for large state-action spaces as that of the learning app, we have to define a different potential function in order to develop a scalable method for computing brain points. The optimal brain points serve as the benchmark against which we evaluate the success of the approximation. The potential function in Eq. 5 captures the progress towards a learning goal by looking at the ratio between the current competence value c_i and the goal competence value c_{g_i} for each skill. An overall measure of progress is obtained by summing over all skills.

$$\phi_{\text{ABP}}(s) = \sum_{i=1}^{N_s} \left[(c_{g_i} - 1) + \frac{c_i}{c_{g_i}} \right] \tag{5}$$

For calculating the approximate brain points (Eq. 6), we then maximize for the potential progress to be made by taking an action. That ensures that the action with the greatest possibility for progress will be incentivised most strongly, regardless of its perceived difficulty. The brain points thereby nudge the user away from games that only involve skills they have already mastered.

$$\text{ABP}(s, a) = \sum_{s'} P(s'|s, a) R(s, a, s') + \left(\max_{s'} \phi_{ABP}(s') - \phi_{ABP}(s) \right) \tag{6}$$

Applied to the model of the learning app DoC and its definition of c_i, the potential function (ϕ_{DoC}) sums the current levels $l(q)$ of the set of questions $\mathcal{Q}_{c_{g_i}-1}$ needed to complete to reach the goal competence value c_{g_i} and dividing it by the sum of maximal levels $m(q)$ for the same set, that is

$$\phi_{\text{DoC}}(s) = \sum_{i=1}^{N_s} \left[(c_{g_i} - 1) + \frac{\sum_{q \in \mathcal{Q}_{c_{g_i}-1}} l(q)}{\sum_{q \in \mathcal{Q}_{c_{g_i}-1}} m(q)} \right]. \tag{7}$$

As the approximate brain points only need to look one step ahead to estimate which action can lead to maximal progress, they can be computed regardless of the size of the state-action space and therefore scale to the level of complexity found in real-world learning games such as DoC. To test their effectiveness against the benchmark provided by the optimal brain points, we modeled a subset of DoC small enough to calculate the optimal value function. We then incentivized simulated agents which greedily chose the action yielding the largest immediate reward with either optimal or approximate brain points. We found that both kinds of brain points significantly improved the agent's performance in comparison to an agent choosing greedily based on the unchanged reward function and an agent choosing randomly. Approximate brain points led to equivalent increase in performance as optimal brain points and therefore commenced the next evaluation step with human learners [9].

3 Testing the method in an online experiment

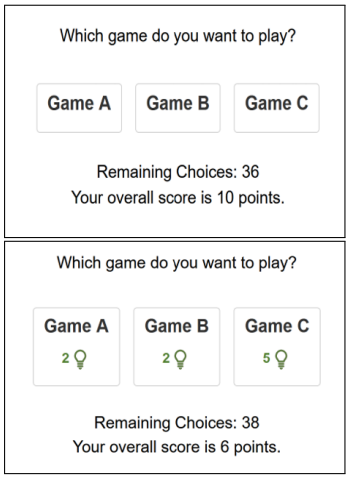


Figure 2: Choice screens with(out) brain points

To test the developed method in a controlled manner before the eventual field experiment within the actual learning app, we created an artificial learning task mimicking the digital learning environment of DoC. As learning material, a paired associative recognition paradigm was chosen. That means that participants were tasked to memorize associations between word pairs and recognize whether two words presented to them were associated or not. Non-words drawn from a database [11] were used to emulate words from an artificial language unknown to participants. Participants were offered three learning activities to choose from ($N_a = 3$). To that end, we created three distinct sets of word pairs with varying difficulty induced by varying levels of inter-word similarity. The successful manipulation of difficulty was confirmed in pre-tests, from which we also estimated the probability of answering correctly needed to construct $P(s, a, s')$ (for details, see [9]). Mastering each set was defined as a skill ($N_s = 3$). Each word pair got assigned levels according to the repetition schedule (Figure 1). As all three activities took an equal amount of time to complete, $r_a = 1\ \forall a$. In the experiments, participants were tasked to repeatedly choose which learning activity to engage in (see Figure 2). All participants were given a score based on their performance; this score mimicked the in-game reward structure of DoC.

Experimental Design The experiment used a between-subjects design with five conditions. In all conditions, participants saw a score based on the number of questions they had answered correctly and knew that their bonus payment depended on that score. In the control condition, participants chose between learning activities without any further incentives shown to them (see the top panel of Figure 2). In the second condition (OBP), participants were shown rounded optimal brain points (Eq. 4) when choosing between learning activities. In the third condition (ABP), participants were shown rounded approximate brain points (Eq. 6) when choosing between learning activities. There were two additional conditions, in which participants were not able to choose between learning tasks, but the choice associated with the highest number of optimal or approximate brain points was imposed on them. These conditions were included to disentangle the effects of the incentives on choice behaviour from the effect of optimal choices on learning progress. In all experimental conditions, participants were informed that the brain points convey information on how much each game is expected to help their learning progress. They were also made aware that the brain points did not count towards their score nor the monetary bonus they could earn.

Hypotheses Based on previous findings regarding the beneficial effects of optimal brain points [2], we hypothesized that participants in the OBP and ABP conditions would choose learning activities less short-sightedly, meaning that they choose more challenging games that they can learn from rather than exploit their knowledge to obtain a high score in a game they have already mastered. Specifically, they were expected to choose the game they scored the highest in so far less often than participants in the control condition, especially after having mastered the corresponding skill. Consequently, we expected participants supported by either kind of brain points (OBP or ABP) to learn more word pairs, which was defined as them having reached the highest level of the repetition schedule (see Figure 1). Furthermore they were expected to achieve higher overall sums of levels, representing a more continuous measure of learning progress. Lastly, based on our simulation results, we expected the approximated brain points (ABP) to have equivalent effects to the optimal brain points (OBP).

Participants 300 participants (57.5% female; *mean age* = 24.9 yrs, *SD* = 6.16 yrs), all older than 18 years and fluent in English, were recruited via Prolific [12]. All participants provided written informed consent and received a base compensation of 1.70 £ for the 16 minute experiment. Additionally, they were awarded a bonus payment based on their performance, with the average bonus set to be 0.40 £. 36 participants were excluded due to having failed two or more attention checks. New participants were recruited to replace them. This experiment was covered by the ethics approval from the IEC of the University of Tübingen under IRB protocol number 667/2018BO2.

Procedure Each of the experiment's 40 mini-blocks consisted of one choice trial (see Figure 2), four learning trials and a message informing participants of their score. In each choice trial, participants clicked on the button corresponding to the game they wanted to play (in the fourth and fifth condition, only one button was enabled). The assignment of the word pair sets to Game A, B and C was randomized. /In the learning trials, participants are first presented with the word pair and a reaction prompt and had 3.5 seconds to respond responses. Then, the appropriate feedback was shown for 2 seconds. After the 4 learning trials, participants were informed of the score they achieved, which equates to the number of correct answers given.

Results The type of incentives influenced how participants chose between games after having mastered at least one skill (H = 7.62, p = 0.022, $\eta^2 = 0.031$), see the up-most tile of Figure 3. As predicted, we found that participants in the control condition continued to choose the game they performed best in so far even after having mastered its contents at high rates (median = 75%, Interquartile Range (IQR) = 27%). Those rates were reduced strongly for participants incentivized with OBP (median = 9.5%, IQR = 14.4%, U = 39, p = 0.030^*, $\eta^2 = 0.709$) as well as for participants incentivized with ABP (median = 4.7%, IQR = 22.2%, U = 18, p = 0.035^*, $\eta^2 = 0.725$).

*Corrected p-value: The false discovery rate of the pairwise comparisons was controlled with the Benjamini-Hochberg procedure.

The different incentives also effected both learning outcome measures, namely the number of learned word pairs (H = 8.96, p = 0.011, $\eta^2 = 0.039$) and the progress measured in summed levels (H = 8.456, p = 0.015, $\eta^2 = 0.036$). Participants in the OBP condition learned more word pairs (median = 9, IQR = 5) than those without choice incentives (median = 7, IQR = 5, U = 1321, p = 0.006^*, $\eta^2 = 0.075$), as can be seen in the middle tile of Figure 3. OBP also led to higher sums of levels (median = 57, IQR = 9) in comparison to the control condition (median = 54, IQR = 9.25, U = 1393.5, p = 0.020^*, $\eta^2 = 0.058$), as depicted in the lower-most tile of Figure 3 An equivalent effect is also found to be elicited by ABP (median = 56, IQR = 10, U = 1503.0, p = 0.046^*, $\eta^2 = 0.037$). We found that ABP were as effective (102.25%) as OBP in reducing exploiting choice behaviour. The effects on the learning progress measures exhibited larger differences, with ABP having 26% of the effect of OBP on the number of learned words and 64% of the effect on overall achieved word levels. A complete analysis of all dependent variables and conditions can be found in [9].

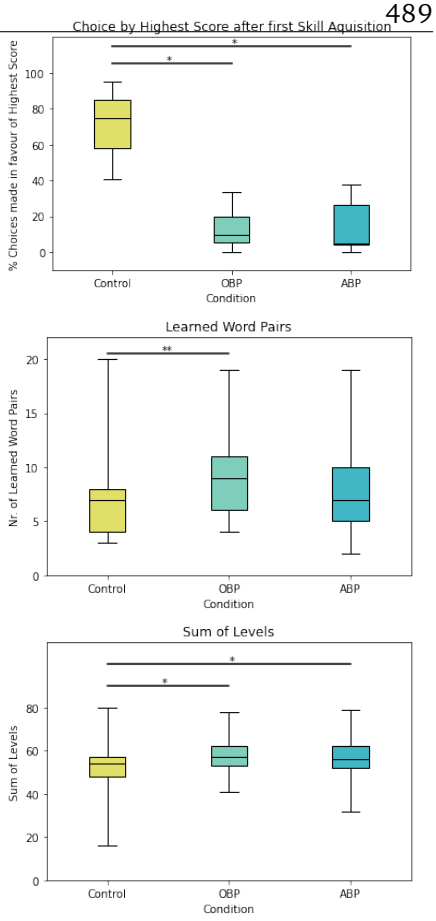


Figure 3: Choice ratios, number of learned word pairs and summed levels per condition

4 Discussion and Conclusion

Based on the presented results, we can conclude that our scalable computational approach to calculating incentives for self-directed learning can benefit learners choosing between different learning tasks. Specifically, the incentives lead to an improved choice behavior, meaning that learners were less inclined to exploit already mastered skills for gaining higher scores and more motivated to confront themselves with novel learning material. Consequently, the incentives led learners to make more learning progress on the material overall. The comparable effects of optimal and approximated choice incentives on the choice behaviour show the potential of the approximated points as a scalable alternative to optimal brain points. This is crucial for the translation of the approach from a controlled experimental task to the complexity of real-world learning environments. We observed that the approximate incentives did not increase the number of learned words within the experiment. This is most likely due to the fact that only a minority of participants mastered more than one skill whereas the approximated incentives minimize the number of games needed to complete all three. This finding reflects that the approximate incentives do not prioritize reaching the highest level of the most advanced skill over improving other skills. In the context of the artificial learning task, the approximated brain points did not reach the effectiveness of optimal brain points regarding the learning progress. Nevertheless, they did impact the learning progress positively in comparison to no incentives and with that justify their further evaluation in real learning environments. Future work will therefore evaluate the approximate brain points within the English learning app DoC. Acquiring data on the interaction of real users with an educational environment equipped with progress-based choice incentives will be immensely valuable in the pursuit to develop a general approach to incentivizing self-directed learning that could potentially help people make better use of the vast educational resources that are available to them.

References

[1] R. F. Kizilcec *et al.*, "Scaling up behavioral science interventions in online education," *Proceedings of the National Academy of Sciences*, 2020.

[2] L. Xu, M. Wirzberger, and F. Lieder, "How should we incentivize learning? An optimal feedback mechanism for educational games and online courses.," in *CogSci*, 2019, pp. 3136–3142.

[3] F. Lieder, O. X. Chen, P. M. Krueger, and T. L. Griffiths, "Cognitive prostheses for goal achievement," *Nature Human Behaviour*, 2019.

[4] A. M. Toda, P. H. D. Valle, and S. Isotani, "The dark side of gamification: An overview of negative effects of gamification in education," in *Higher Education for All. From Challenges to Novel Technology-Enhanced Solutions*, A. I. Cristea, I. I. Bittencourt, and F. Lima, Eds., vol. 832, Series Title: Communications in Computer and Information Science, Cham: Springer International Publishing, 2018, pp. 143–156. DOI: 10.1007/978-3-319-97934-2_9.

[5] R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner, "Off-task behavior in the cognitive tutor classroom: When students "game the system"," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 383–390. DOI: 10.1145/985692.985741.

[6] E. O'Rourke, K. Haimovitz, C. Ballweber, C. Dweck, and Z. Popović, "Brain points: A growth mindset incentive structure boosts persistence in an educational game," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 3339–3348. DOI: 10.1145/2556288.2557157.

[7] Solve Education! *Dawn of Civilisation Homepage*, 2021. [Online]. Available: https://dawnofcivilization.net/ (visited on 05/03/2021).

[8] N. J. Cepeda, H. Pashler, E. Vul, J. T. Wixted, and D. Rohrer, "Distributed practice in verbal recall tasks: A review and quantitative synthesis.," *Psychological Bulletin*, vol. 132, no. 3, pp. 354–380, 2006. [Online]. Available: https://escholarship.org/uc/item/3rr6q10c.

[9] R. Pauly, *How to incentivize efficient learning choices in digital learning environments? A reinforcement learning approach applied to an educational game*, 2022. [Online]. Available: osf.io/45ab2.

[10] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, 1999, pp. 278–287.

[11] K. Rastle, J. Harrington, and M. Coltheart, "358,534 nonwords: The ARC nonword database," *The Quarterly Journal of Experimental Psychology Section A*, vol. 55, no. 4, pp. 1339–1362, 2002. DOI: 10.1080/02724980244000099.

[12] S. Palan and C. Schitter, "Prolific. ac—A subject pool for online experiments," *Journal of Behavioral and Experimental Finance*, vol. 17, pp. 22–27, 2018. DOI: 10.1016/j.jbef.2017.12.004.

Decision Making in Non-Stationary Environments with Policy-Augmented Monte Carlo Tree Search

Geoffrey Pettet
Vanderbilt University
Nashville, TN, 37212
geoffrey.a.pettet@vanderbilt.edu

Ayan Mukhopadhyay
Vanderbilt University
Nashville, TN, 37212
ayan.mukhopadhyay@vanderbilt.edu

Abhishek Dubey
Vanderbilt University
Nashville, TN, 37212
abhishek.dubey@vanderbilt.edu

Abstract

Decision-making under uncertainty (DMU), i.e., taking actions with uncertain outcomes using (potentially imperfect) observations, is present in many important problems. An open challenge is DMU in non-stationary environments, where the dynamics of the environment can change over time. Reinforcement learning (RL), a popular approach for DMU problems, learns a policy by interacting with a model of the environment offline. Unfortunately, if the environment changes the policy can become stale and take sub-optimal actions, and relearning the policy for the updated environment takes time and computational effort. An alternative is online planning approaches such as Monte Carlo Tree Search (MCTS), which perform their computation at decision time. Given the current environment, MCTS plans using high-fidelity models to determine promising action trajectories. These models can be updated as soon as environmental changes are detected to immediately incorporate them into decision making. However, MCTS's convergence can be slow for domains with large state-action spaces. In this paper, we present a novel hybrid decision-making approach that combines the strengths of RL and planning while mitigating their weaknesses. Our approach, called *Policy Augmented MCTS* (PA-MCTS), integrates a policy's action-value estimates into MCTS, using the estimates to "seed" the action trajectories favored by the search. We hypothesize that by guiding the search with a policy, PA-MCTS will converge more quickly than standard MCTS while making better decisions than the policy can make on its own when faced with nonstationary environments. We test our hypothesis by comparing PA-MCTS with pure MCTS and an RL agent applied to the classical CartPole environment. We find that PA-MCTS can achieve higher cumulative rewards than the policy in isolation under several environmental shifts while converging in significantly fewer iterations than pure MCTS.

Keywords: Reinforcement Learning, Non-stationary Environment, Monte Carlo Tree Search

1 Introduction

Decision-making under uncertainty (DMU), i.e., taking actions with uncertain outcomes using (potentially imperfect) observations, is present in many important problems such as autonomous driving, emergency response, and medical diagnosis [3]. An open challenge is DMU in non-stationary environments, where the dynamics of the environment can change over time. A decision agent must adapt to these changes or take sub-optimal actions. To illustrate, consider emergency response management (ERM), a domain we have significant experience with [5]. ERM deals with optimizing the allocation and dispatch of mobile resources such as ambulances in the face of uncertain incident demand to minimize response times. The state-action space for an ERM setting can be very large; for example, in a city with 20 ambulances to allocate between 30 potential waiting locations, there are *Permutations*$(30, 20) = \frac{30!}{10!} = 7.31 \times 10^{25}$ possible assignments at each decision epoch. This complexity alone makes ERM a difficult problem, and non-stationarity compounds the challenge. The environments in which ERM systems operate shift over long time scales: road networks, congestion patterns, and demographics all change over time [5]. Sudden events such as road closures, inclement weather, and equipment failure can also unpredictably impact the environment. An effective ERM decision agent must adapt to such changes with minimal effects on service quality.

There are two common DMU approaches: reinforcement learning (RL) and online planning. In RL approaches, an agent learns a policy π, a mapping from states to actions, through interacting with the environment. Often, learning takes place offline using environmental models, and once a policy is learned, it can be invoked nearly instantaneously at decision time. Deep RL methods have achieved state-of-the-art performance in many applications [9, 3]. However, when faced with non-stationary environments, a policy can become stale and make suboptimal decisions. Moreover, retraining the policy on the new environment takes time and considerable computational effort, particularly in problems with complex state-action spaces such as ERM. While RL methods specifically designed to operate in non-stationary environments have been explored [6, 2], there is a delay between when a change is detected and when the RL agent converges to the updated policy.

The alternative approach is to perform online planning using algorithms such as Monte Carlo Tree Search (MCTS). Given the current environment, these approaches perform their computation at decision time by planning using high-fidelity models to determine promising action trajectories. These models can be updated as soon as environmental changes are detected and then such changes can be immediately incorporated in decision-making. MCTS has been proven to converge to optimal actions given enough computation time [4], but convergence can be slow for domains with large state-action spaces. The slow convergence is a particular issue for problem settings with tight constraints on the time allowed for decision-making, e.g., in ERM, when an incident occurs, any time used for decision-making increases the time until a responder is dispatched. In prior work, we have applied several approaches to scale MCTS to practical ERM problems, including decentralized [7] and hierarchical planning [8], but these approaches make assumptions about the environment to prune the state-action space that can lead to sub-optimal decisions.

Both RL and MCTS have weaknesses when applied to complex DMU problems in non-stationary environments. In this paper, we present a novel hybrid decision-making approach that combines the strengths of RL and planning while mitigating some of these weaknesses. The intuition behind our approach is that if the environment has not changed too much between when a policy was learned and when a decision needs to be made, the policy can still provide helpful information. Our approach, called *Policy Augmented MCTS* (PA-MCTS), integrates a policy's action-value estimates into MCTS, using the estimates to "seed" the action trajectories favored by the search. The impact of the policy can be tuned based on how similar the environment at decision time is to the one used during training. We hypothesize that by guiding the search with a policy, PA-MCTS will converge more quickly than standard MCTS while making better decisions than the policy can make on its own. The primary advantage of this approach compared to pure RL approaches for non-stationary environments is that it does not require any relearning—it can be applied as soon as changes are detected.

In this paper, we test our hypothesis by applying PA-MCTS to the OpenAI Gym Cartpole-v1 environment [1], a version of the classic inverted pendulum problem. We first learn a policy using RL on the standard environment and then perform several experiments in which we vary parameters affecting environmental dynamics, such as the force of gravity and the cart's mass. Finally, we compare the performance of pure MCTS, the RL policy in isolation, and PA-MCTS. Our results show that PA-MCTS can achieve higher cumulative rewards than the policy in isolation under several environmental shifts while converging in significantly fewer iterations than pure MCTS. While this exploratory paper shows the promise of PA-MCTS, there are still open questions to consider: how can we determine the influence the policy should have at a decision time based on the current environment? Are there specific types of environmental shifts where this approach performs poorly? Are there bounds on how much the environment can change while still benefiting from PA-MCTS? In future work, we will explore these questions and apply the approach to more complex environments such as ERM.

2 Approach

We consider a Markov Decision Process (MDP) which is represented by the tuple $(S, \mathcal{A}, P(s, a), R(s, a))$ where S is a finite state space, $\mathcal{A}$ is an action-space, $P(s, a)$ is a state transition function, and $R(s, a)$ is a reward function defining the instantaneous reward for taking action a in state s. We consider non-stationary environments, meaning that $P(s, a)$ and

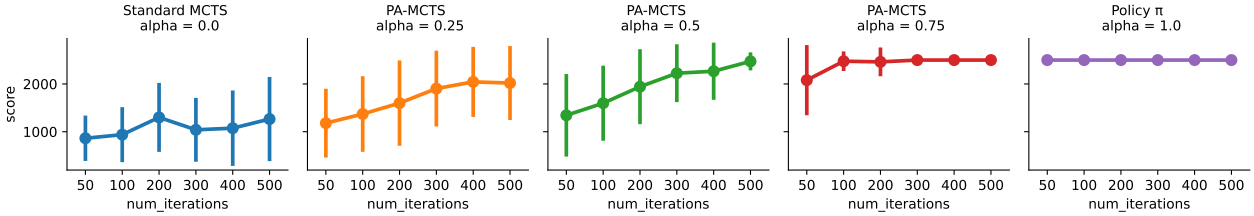


Figure 1: PA-MCTS results when applied to the default CartPole environment. Columns represent values for α. The individual plots' x-axis is the number of MCTS iterations per decision epoch. The y-axis is the cumulative reward. The dots represent the mean cumulative reward over 50 samples, while the vertical lines are the standard deviation.

$R(s, a)$ can change over time. Our agent's goal is to choose the sequence of actions that maximizes the cumulative reward G (referred to as the *return*) received while interacting with the environment. We assume that a policy π is learned offline under initial environmental conditions (defined by a transition and reward function). The policy can be used to compute the expected discounted reward of taking an action a in a state s and then following the policy (denoted by $Q(s, a)$).

Our approach is based on Monte Carlo Tree Search (MCTS), an iterative algorithm that builds a search tree online in an incremental and asymmetric manner. Each MCTS iteration consists of four stages: (1) selection, (2) expansion, (3) rollout, and (4) back-propagation. The stage relevant to our work is selection, in which the next node to expand is determined. Starting at the root node, a *tree policy* is recursively applied to descend through the tree to pick the most promising child, continuing until it reaches a non-terminal leaf node to expand. A popular tree policy is the Upper Confidence Bounds for Trees (UCT) algorithm [4], which selects the next node according to:

$$\underset{j \in \text{Children}(p)}{\arg\max} \{\overline{G}_j + c\sqrt{\ln(n_p)/n_j}\} \tag{1}$$

where p is the parent node, $\overline{G}_j$ is the average return of rollout simulations including child j, n_p and n_j are the number of times p and j have been visited respectively, and c is a hyperparameter controlling the trade-off between exploitation and exploration.

Our contribution is a novel version of UCT that is modified to incorporate the policy's action-value estimates $Q(s, a)$ when estimating the value of a node, which we call *Policy Augmented UCT* (PA-UCT). When PA-UCT is used as the tree policy for MCTS, we call the resulting algorithm *Policy Augmented MCTS* (PA-MCTS). PA-UCT selects nodes to explore using the following policy

$$\underset{j \in \text{Children}(p)}{\arg\max} \quad \alpha Q(s_p, a_{p,j}) + (1-\alpha)\overline{G}_j + c\sqrt{\frac{\ln(n_p)}{n_j}} \tag{2}$$

where s_p is the state at the parent node p, $a_{p,j}$ is the action which transitions state s_p to child $j's$ state when taken, and α is a weight hyperparameter that controls the trade-off between the policy and rollout returns: if $\alpha = 1$, PA-UCT reduces to greedy action selection based on the learned policy, whereas if $\alpha = 0$, it reduces to standard UCT. If $\alpha \in (0, 1)$, then both estimates are considered for action selection.

MCTS is proven to converge to the optimal action given infinite iterations [4]. As the tree search produces improved estimates with increasing iterations given the updated environmental dynamics, we can decrease the influence of the policy on PA-UCT. Therefore, an alternative version of PA-UCT, PA-UCT with α-decay, uses the following selection policy on iteration i:

$$\underset{j \in \text{Children}(p)}{\arg\max} \quad \alpha(\frac{1}{1+k \cdot i})Q(s_p, a_{p,j}) + (1-\alpha(\frac{1}{1+k \cdot i}))\overline{G}_j + c\sqrt{\frac{\ln(n_p)}{n_j}} \tag{3}$$

where k is a hyperparameter that controls the rate of decay.

3 Experiments

To evaluate the efficacy of PA-MCTS, we first learn a policy π for the OpenAI Gym cartpole-v1 environment [1], using a double deep Q RL algorithm [10]. Environment parameters are set to default values, except the maximum length of each episode which is increased from 500 to 2500 time steps, in order to evaluate the performance of PA-MCTS[1].

[1]The relevant default parameters are that gravity $g = 9.8$ and the cart's mass $m = 1.0$. The standard cartpole reward function of returning $R(s, a) = 1.0$ for each time step before reaching a terminal state was used. The double Q-learning agent's learning rate $= 0.001$, and a Boltzmann control policy was used. The agent learned for 300000 steps

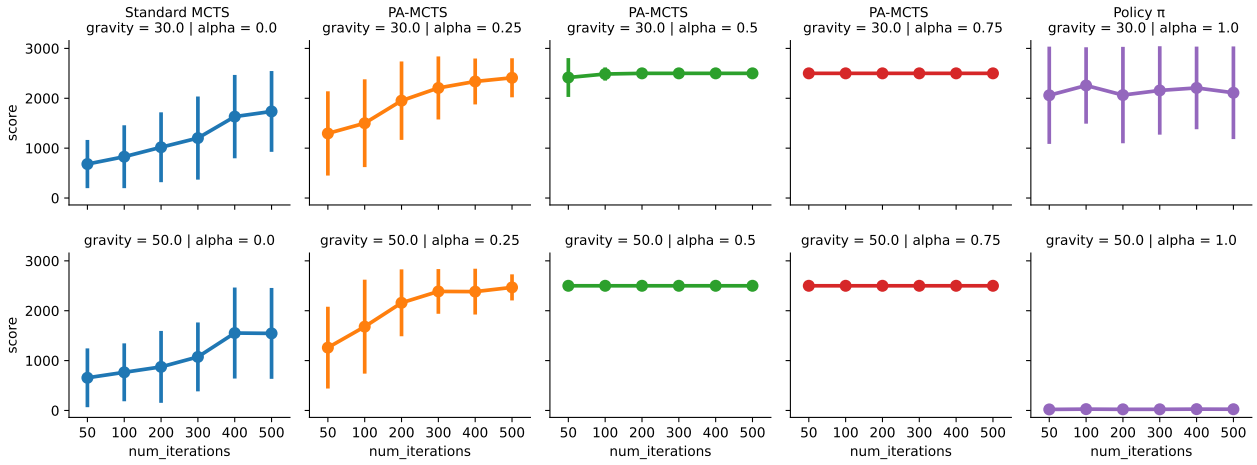


Figure 2: PA-MCTS results with gravity modified from its default value of 9.8 m/s^2. Rows represent different gravity values. Columns represent values for α. The individual plots' x-axis is the number of MCTS iterations per decision epoch. The y-axis is the cumulative reward. The dots represent the mean cumulative reward over 50 samples, while the vertical lines are the standard deviation.

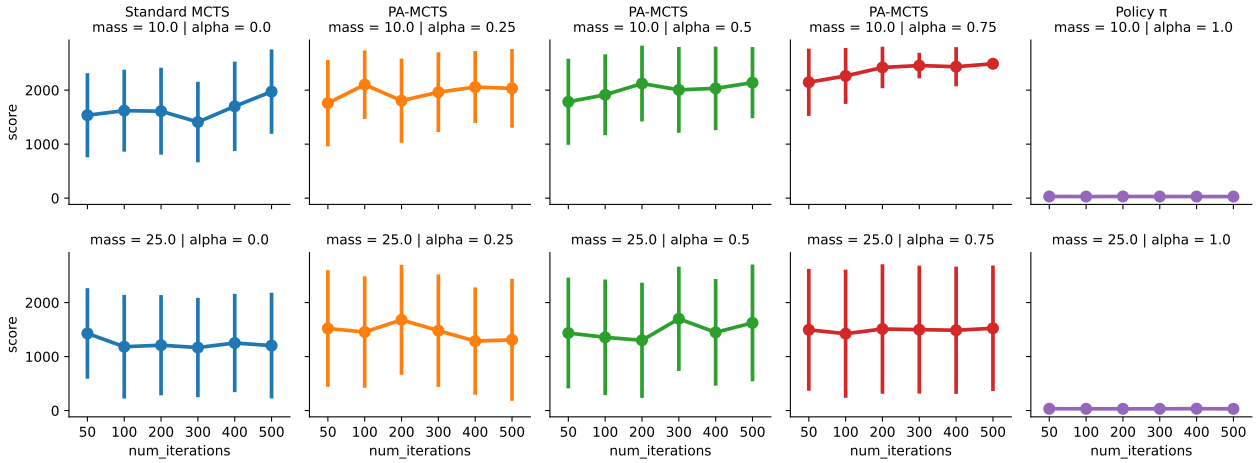


Figure 3: PA-MCTS results with the cart's mass modified from its default value of 1.0 kg. Rows represent different cart masses. Columns represent values for α. The individual plots' x-axis is the number of MCTS iterations per decision epoch. The y-axis is the cumulative reward. The dots represent the mean cumulative reward over 50 samples, while the vertical lines are the standard deviation.

We then perform experiments comparing π ($\alpha = 1.0$) and MCTS ($\alpha = 0.0$) [2] against PA-MCTS[3] with several values for α in the default environment used to learn π. We use the following MCTS iteration budgets to evaluate the convergence of each approach: {50, 100, 200, 300, 400, 500}. The results are shown in figure 1. Our first observation is that the policy π ($\alpha = 1.0$) achieves the best possible return of 2500, as expected. We also observe that PA-MCTS with $\alpha \in \{0.25, 0.5, 0.75\}$ converges in far fewer iterations than standard MCTS (i.e. with $\alpha = 0.0$), with $\alpha = 0.75$ converging to the optimal return at 300 iterations.

The second set of experiments introduce non-stationarity to the environment by modifying two environmental parameters: we change the gravitational constant g from the default value of 9.8 m/s^2 to 30.0 m/s^2 and 50.0 m/s^2, and change the cart's mass m from the default of 1.0 kg to 10.0 kg and 25.0 kg. The results are shown in figures 2 and 3. Our first observation is that the policy's performance in isolation degrades significantly: with $g = 30$ m/s^2 and $\alpha = 1.0$, the mean return does not reach the optimal value, while for $g = 50.0$ m/s^2 and $m \in \{10.0$ kg, 25.0 kg$\}$ the policy obtains returns close to 0. We also observe that PA-MCTS again converges in significantly fewer iterations than standard MCTS in most cases, notably achieving the optimal return within 50 iterations for $g \in \{30.0$ m/s^2, 50.0 $m/s^2\}$ and $\alpha = 0.75$ in figure 2.

[2] Recall that when $\alpha = 1.0$, PA-UCT reduces to greedy action selection based on the learned policy, and when $\alpha = 0.0$, it reduces to standard UCT.

[3] We use the following PA-MCTS hyperparameters in all experiments: the exploration-exploitation tradeoff parameter $c = 50$, the planning horizon is 500 time steps, the discount factor $\gamma = 0.999^{493}$, and the decay rate $k = 0.0$

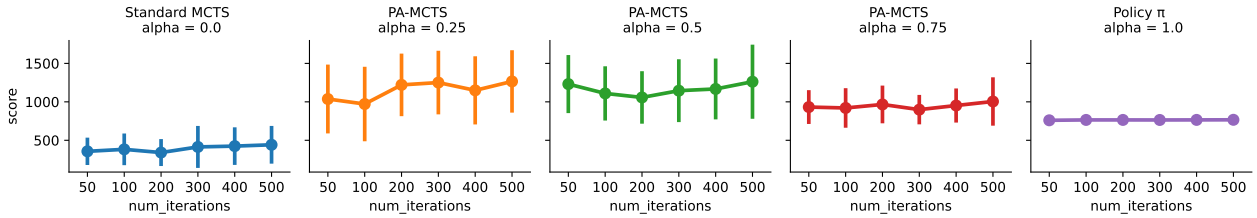


Figure 4: PA-MCTS results with a modified reward function which incentivizes staying near the center of the track. Columns represent values for α. The individual plots' x-axis is the number of MCTS iterations per decision epoch. The y-axis is the cumulative reward. The dots represent the mean cumulative reward over 50 samples, while the vertical lines are the standard deviation.

The exception is $m = 25.0$ kg in figure 3, where PA-MCTS does not significantly improve upon standard MCTS with any tested α. There are several possible explanations for this: we may not have tested the optimal α value, the environment may have shifted too much for the policy to be useful, or it may be that balancing the pole with a cart this massive is too challenging given the default action-space. More study is needed to determine the amount of environmental shift that PA-MCTS can handle without retraining the policy. Finally, we evaluate PA-MCTS with a modified reward function. In the standard cartpole environment, the agent receives a reward of 1 for each time-step before the episode is terminated. We modify the reward function to return $1 - (|x_p|/X_t)$, where x_p is the cart's x coordinate and X_t is the track's boundary. In the cartpole environment, the center of the track is at $x = 0$ and the boundaries are at $x = -X_t$ and $x = X_t$. Therefore, this new reward function provides higher rewards the closer the cart is to the center of the track. The results with this reward function are shown in figure 4. We again observe that PA-MCTS obtains higher returns than the policy while converging in fewer iterations than standard MCTS.

4 Conclusion

We present a novel hybrid decision-making approach that combines the strengths of reinforcement learning and planning for non-stationary environments called *Policy Augmented MCTS* (PA-MCTS). Using the classical cartpole environment, we show that PA-MCTS can achieve higher cumulative rewards than an RL agent in isolation under environmental shifts while converging in significantly fewer iterations than pure MCTS. In future work, we will apply PA-MCTS to more complex problems, further explore the effect of hyperparameters such as the policy's influence weight α and the decay rate k, and study how we can use observations of the current environment to determine α at decision time.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Non-stationary reinforcement learning: The blessing of (more) optimism. *Available at SSRN 3397818*, 2019.

[3] Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for Decision Making*. MIT Press, 2022.

[4] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[5] Ayan Mukhopadhyay, Geoffrey Pettet, Sayyed Vazirizade, Di Lu, et al. A review of incident prediction, resource allocation, and dispatch models for emergency management. *Accident Analysis and Prevention*, 2021.

[6] Ronald Ortner, Pratik Gajane, and Peter Auer. Variational regret bounds for reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 81–90, 2020.

[7] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Conference on Autonomous Agents and MultiAgent Systems*, 2020.

[8] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey. Hierarchical planning for dynamic resource allocation in smart and connected communities. *ACM Trans. Cyber-Phys. Syst.*, 2021.

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[10] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Why do some beliefs and action policies resist updating?

Sashank Pisupati
Princeton Neuroscience Institute
Princeton University
pisupati@princeton.edu

Angela Langdon
Princeton Neuroscience Institute
Princeton University
alangdon@princeton.edu

Yael Niv
Princeton Neuroscience Institute, Department of Psychology
Princeton University
yael@princeton.edu

Abstract

Why do some beliefs and action policies fail to update despite abundant contrary evidence, re-appearing long after they have stopped being adaptive? The failure to extinguish outdated beliefs, although inconsistent with many simple models of learning, is a widely observed empirical phenomenon proposed to arise from maladaptive inference of hidden structure - for instance, individuals assigning contrary evidence to new hidden causes, rather than updating previously learned associations. What parameters of the inference process might make agents susceptible to forming such maladaptive structures? Here, using simulations of latent cause inference during Pavlovian and instrumental learning tasks, we show that priors about randomness and change influence resistance to updating as well as learning dynamics. We show that prior beliefs that the world is highly deterministic or controllable yield fast learning of new associations at the cost of making old ones more resistant, while beliefs that the world is stochastic or uncontrollable yield slow but robust updating of old associations. Additionally, prior beliefs that latent causes persist with a certain timescale yield fast adaptation to contingency shifts occurring at that timescale, but also increase the possibility that outdated beliefs or action policies will resurface at a similar timescale. These results link observable features of update-resistant behavior in Pavlovian and instrumental tasks to inductive biases about the stochasticity, volatility and controllability of the world. Understanding the factors underlying update-resistance could yield insight into recurring conditions such as spontaneous recovery of fear in anxiety and relapse in addiction.

Keywords: Latent cause inference, belief updating, inductive biases, Anxiety, Addiction

Acknowledgements

We would like to thank the following people for helpful discussions: Ines Aitsahalia, Sam Zorowitz, Mingyu Song, Dan Bennett, Zack Dulberg, Yeon Soon Shin, Yoel Sanchez Araujo, Peter Hitchcock, Dan Mirea, Isabel Berwian, Oded Bein, Rachel Bedder, Anna Konova, Stephanie Groman and Nathaniel Daw. This work was funded by NIDA R01 DA042065, NIMH R01 MH119511 and a grant from the Templeton Foundation

1 Introduction

A central feature of learning theories is their ability to update beliefs, values and action policies in light of new evidence so that they are more accurate or adaptive. This ability is also the basis of paradigms such as extinction learning and exposure therapy, which assume that exposure to contrary evidence should be sufficient to correct outdated beliefs. However, it is a well-known empirical phenomenon that contrary evidence alone is not sufficient to properly update beliefs and such paradigms often fail to yield their desired results, with outdated beliefs or action policies continuing to resurface long after they are no longer adaptive (Dunsmoor et al., 2015). What makes some beliefs or action policies resistant to being updated?

One mechanism that has been proposed to explain such updating failures, is maladaptive inference of hidden-cause structure. Gershman et al., 2010 proposed that if agents attributed surprising or contrary evidence to new hidden (latent) causes, this would prevent them from updating previously existing associations ('old' latent causes), and those associations would be more likely to resurface in the future. For instance, after learning about a cue that is associated with a punishment, the lack of that punishment may be surprising. If the new state of affairs is attributed to a new "safe" latent cause, this will prevent the update of the old "dangerous" latent cause, therefore protecting it from extinguishing due to prediction-error driven learning (Figure 1a). Indeed, individual differences in inferred causal structure have been shown to correlate with differences in the ability to extinguish fearful associations (Gershman & Hartley, 2015; Norbury et al., 2020). What parameters of the inference process might make an agent more or less susceptible to learning such hidden structures, and consequently, to update-resistance?

Here, we explore the factors shaping latent-cause inference and update-resistance in Pavlovian (Figure 1a) and instrumental (Figure 1b) tasks. We demonstrate that the extent to which beliefs or action policies are update-resistant is shaped by two factors. First, priors about randomness (i.e. the stochasticity or controllability of observations) can influence the speed of learning and the extent to which surprising evidence updates old associations. Second, priors about change (i.e. the persistence or volatility of latent causes) can influence the speed of adaptation to changing evidence, and the extent to which old beliefs resurface long afterwards.

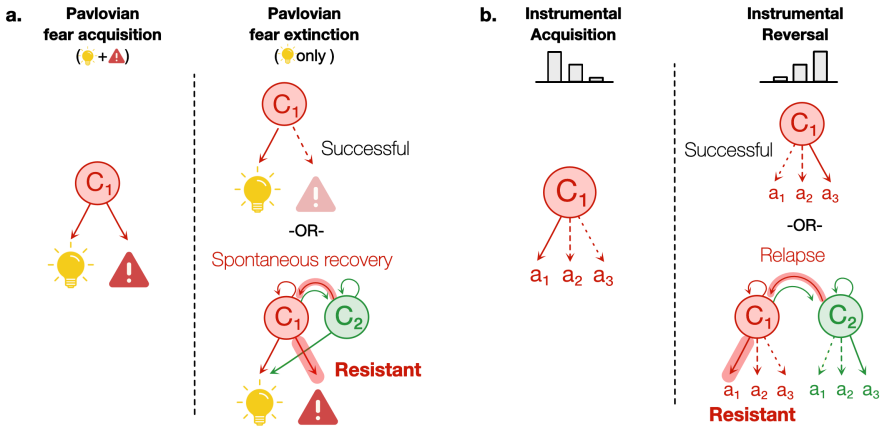


Figure 1: **Resistance to extinction due to inference of causal structure**: a) Extinction-resistant beliefs arising from maladaptive inference in a Pavlovian task. b) Extinction-resistant action policies in an instrumental task. C = latent cause.

2 Model description

We start from a model of associative learning proposed by Gershman et al., 2010, that assumes that rather than directly learning associations between cues and outcomes (i.e., cue values), individuals infer hidden or "latent" causes that produce "observations" in the form of both cues and outcomes, learning observation probabilities for each latent cause.

The model assumes that new observations are *a priori* more likely to be generated by latent causes that have already generated many observations, but new causes may be encountered with a small probability proportional to a "concentration" parameter ν (upto as many causes as trials). In addition, the model assumes that the currently active cause may persist into the next trial with a probability η, a parameter that reflects the agent's estimate of the **volatility** of latent causes. This amounts to a prior belief that latent causes are generated by a "persistent" variant of the Chinese Restaurant Process similar to Lloyd and Leslie, 2013. The prior probability of encountering latent cause k on trial t is therefore given by:

$$p(c_t = k | c_{t-1} = j, \mathbf{c}_{1:t-2}) = \begin{cases} \eta \cdot \frac{N_k}{t-1+\nu} & \text{if } k = j, \text{ i.e., the previously active latent cause} \\ (1-\eta) \cdot \frac{N_k}{t-1+\nu} & \text{if } k \neq j, \text{ i.e., any other old latent cause} \\ (1-\eta) \cdot \frac{\nu}{t-1+\nu} & \text{if } k \text{ is a new latent cause} \end{cases} \quad (1)$$

We assume that observations on a trial are a collection of binary events generated by latent causes through Bernoulli processes with independent "generative" probabilities $\phi_{i,k}$ for each feature $f_{i,t}$ (cue, outcome, etc.) in feature vector $\mathbf{F}_t$ conditioned on each latent cause $c_t = k$. As such, the same set of observations could have been generated by any one of the known old causes, or an entirely new cause. We use Beta priors with parameters α and β for these generative probabilities, whose variance reflects the agent's estimate of **stochasticity** in observations – when both these parameters are greater than 1, the prior is "stochastic" and expects events to occur randomly with probabilities close to $\frac{\alpha}{\beta}$, whereas when they are less than 1, it is "deterministic" and expects events to occur with probabilities close to 0 or 1, with a uniform prior corresponding to $\alpha = \beta = 1$.

$$p(\mathbf{F}_t|c_t = k) = \prod_i p(f_{i,t}|c_t = k) = \prod_i \phi_{i,k}; \quad p(\phi_{i,k}) = \text{Beta}(\alpha, \beta); \alpha, \beta \begin{cases} = 1, \text{Uniform prior} \\ < 1, \text{Deterministic prior} \\ > 1, \text{Stochastic prior} \end{cases} \tag{2}$$

The model performs approximates Bayesian inference using particle filtering (sequential importance resampling of m particles with importance weights w_t), to infer the possible partitioning of trials into causes, based on all of the observations it has made so far $\mathbf{F}_{1:t}$. Further, it uses this posterior to inform its predictions about future observations (e.g. predicted outcomes given a cue), as well as to update its estimates of the "generative" observation probabilities given each cause.

$$p(c_{1:t}|\mathbf{F}_{1:t}) = \frac{p(\mathbf{F}_{1:t}|c_{1:t})p(c_{1:t})}{p(\mathbf{F}_{1:t})}; \qquad p(c_{1:t} = c|\mathbf{F}_{1:t}) \approx \sum_{l=1}^{m} w_t^{(l)} \delta[c_{1:t}^{(l)}, c], \quad w_t^{(l)} \propto \prod_i p(f_{i,t}|c_{1:t}^{(l)}, \mathbf{F}_{1:t-1}) \tag{3}$$

We extended this model to incorporate actions (Figure 1b) in a similar way to Lloyd and Leslie, 2013, allowing the latent cause $c_t = k$ to act as a partially observable context and condition the outcome probabilities $\phi_{i_0,j,k}$ of different actions $a_t = j$, and using Thompson sampling to select actions. One key difference in our formulation is that we use multiple particles to represent the posterior over causes on every trial. Thus, rather than sampling a single cause (particle) on every trial and selecting actions through Thompson sampling on the action-value distribution conditioned on that cause (Lloyd & Leslie, 2013), we perform Thompson sampling on the marginal action value distribution across causes, to ensure that our policy converges to the optimal policy in the limit of sufficient exploration, when uncertainty about action values is negligible but uncertainty about causes may still remain.

$$a_t = \underset{j}{\text{argmax}} \sum_{l=1}^{m} w_t^{(l)} \tilde{\phi}_{i_0,j,k}^{(l)}, \quad \tilde{\phi}_{i_0,j,k}^{(l)} \sim p(\phi_{i_0,j,k}|c_{1:t}^{(l)}, \mathbf{F}_{1:t}) \tag{4}$$

The variance of Beta priors over the action-and-cause-conditioned outcome probabilities $\phi_{i,j,k}$ controls the outcome entropy, akin to a prior over **controllability** (Huys & Dayan, 2009). When these parameters are < 1, the prior is more "controllable" since it expects actions to sometimes lead to high outcome probabilities and other times not.

3 Results

3.1 Beliefs that resist updating

We simulated the model on a classic extinction-learning task (Figure 1a) in which a cue (A) is paired with a threatening outcome (+) for 20 trials (acquisition), after which it is no longer paired with the punishment for another 20 trials (extinction), and observed the effect of different parameters on three measures – the inferred posterior probability over causes, p(Cause); the learnt associations with cue and outcome p(Event | Cause); and the outcome prediction p(Outcome).

We found that the variance of the observation prior over p_{Obs} affected all three measures to varying degrees (Figure 2a). A stochastic prior over observations lead to stable inference of a single cause throughout the extinction learning task, and as a result, to the complete updating and extinguishing of the threat association (Fig 2a, 1st column, red box). A side effect of this prior was an overall lower effective learning rate, as evident from the outcome prediction. On the other hand, a deterministic prior led to an abrupt inference of a second cause during extinction, thus protecting the original association and forming a new safe association (Fig 2a, last column, green box). This prior also had the effect of speeding up learning but only during initial acquisition.

The persistence of latent causes (Figure 2b, p_{Stay}) had a slightly different effect on learning dynamics. Although higher values of persistence also led to stronger segmentation into two causes, and protected the old association from updating, these also led to faster "change-detection", i.e., faster adaptation to the change in contingencies from acquisition to extinction (Figure 2b, gray curves). A side effect of the persistence was that the lower outcome prediction during extinction was only temporary: querying the model with the same cue after a short gap (Figure 2c, retrieval) revealed a drastic increase in outcome expectations reminiscent of "spontaneous recovery" of fear, which decreased soon after ("habituation") following further exposure to no outcomes.

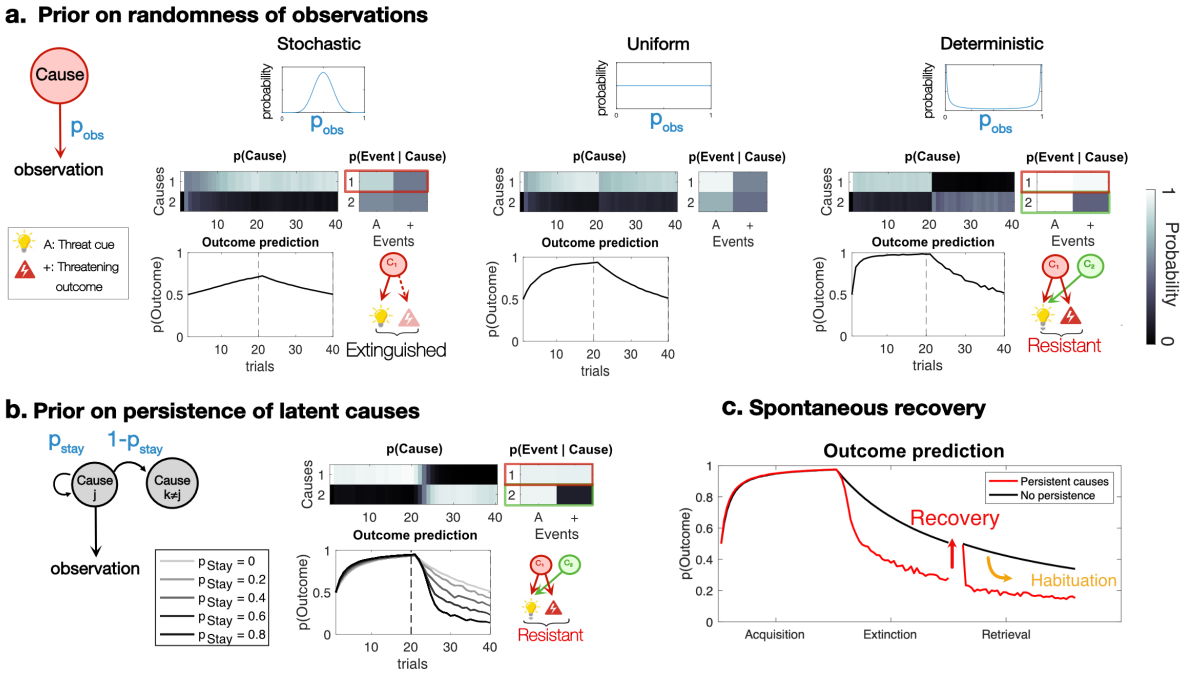


Figure 2: **Extinction resistant beliefs**: a) Effect of prior on randomness of observations, b) Effect of prior on persistence of latent causes, c) Spontaneous recovery phenomenon in the model. Shaded boxes: lighter shading = higher probability across trials or events

3.2 Action policies that resist updating

We next simulated an extension of the model that incorporated actions on a probabilistic reversal-learning task – a three-armed bandit task whose payoffs abruptly switch midway through the task, with the worst action becoming the best and vice versa. We measured the effect of prior parameters on the posterior over causes, p(Cause), the learnt contextual action value, p(Outcome | Action, Cause), the marginal action values across latent causes, p(Outcome | Action), and choices.

The effect of changing the variance of the outcome prior was similar in spirit to the Pavlovian simulations described above – a more "uncontrollable" prior (similar to a stochastic prior) that expects a close-to-chance probability of outcomes from all actions led to inference of a single stable cause, and therefore fully extinguished the initially learnt association by the end of reversal (Fig 3a, left column, red box). It also led to slower learning of action values, and more "perseverative" choices that were slow to reverse. Conversely, a more "controllable" prior (similar to a deterministic prior) that expects either very low or very high payoffs from actions, led to rapid inference of a new cause during reversal and therefore learning of two sets of conflicting action values, almost perfectly protecting the initial associations from updating (Fig 3a, right column, red and green boxes). It also led to much faster learning of action values, and a faster transition from exploratory to exploiting behavior after reversal.

Different priors on persistence over latent causes had a similar effect on inference, with higher values of persistence giving rise to better change-detection (up to a point), and a cleaner separation between acquisition and reversal causes, values and policies (Figure 3b, right column). However, a side effect of high persistence was that some time after reversal, the agent started inferring spurious switches back to the original cause, yielding spontaneous "relapse" events that were briefly evident in the policy (Figure 3c, red circle). These relapses were brief because the lack of outcomes from these mistaken actions quickly recalibrated the agent's beliefs, returning it to the correct policy. Such relapses can be seen in animal behavior on such a task (not shown).

4 Discussion

We showed that resistance to updating is shaped by two parameters that influence latent-cause inference – prior beliefs about randomness (stochasticity or controllability of observations) and prior beliefs about change (persistence or volatility of latent causes). These priors produce a trade-off between the stability of learning and the complexity of learnt representations, akin to a bias-variance tradeoff (Bear & Cushman, 2020; Dorfman & Gershman, 2019). Stochastic or uncontrollable priors produce simpler representations that are more robust to fluctuations in the evidence, with

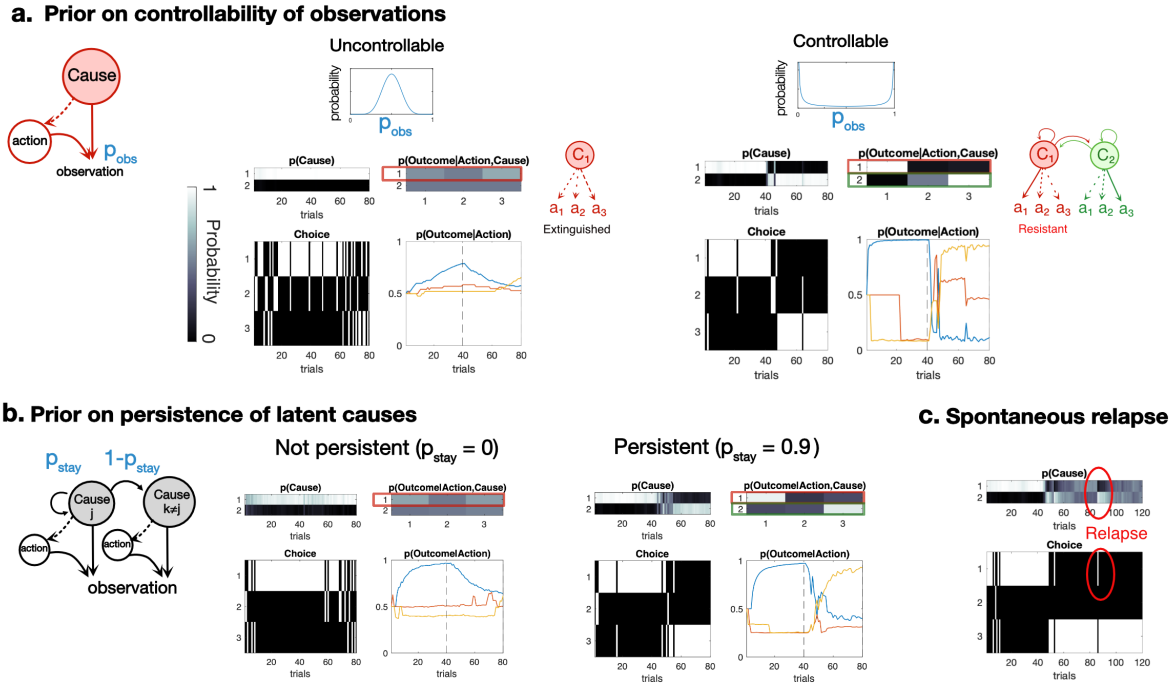


Figure 3: **Extinction resistant action policies**: a) Effect of prior on controllability of outcomes, b) Effect of prior on persistence of latent causes, c) Spontaneous relapse phenomenon in the model. Shading: lighter = higher probability.

the downside that they change more slowly. Conversely, deterministic or controllable priors produce more complex representations that adapt more quickly to changes in the evidence, but run the risk of being over-sensitive to sudden fluctuations and creating update-resistant beliefs.

These results help relate learning phenomena such as update resistance to high-level inductive biases about the stochasticity, volatility and controllability of the world, similar to past work using Kalman filters with (Gershman et al., 2014) and without memory of past states (Piray & Daw, 2021) in continuous-domain learning problems. These priors may themselves require updates to remain adaptive to changing environments (Huys & Dayan, 2009). Such mappings between learning phenomena and "core beliefs" (i.e., priors) may also have clinical implications, helping to improve predictions about therapeutic outcomes or personalized psychotherapy regimes by using precise models of learning (Niv et al., 2021).

References

Bear, A., & Cushman, F. (2020). Loss functions modulate the optimal bias-variance trade-off. *CogSci*.

Dorfman, H. M., & Gershman, S. J. (2019). Controllability governs the balance between pavlovian and instrumental action selection. *Nature communications*, *10*(1), 1–8.

Dunsmoor, J. E., Niv, Y., Daw, N., & Phelps, E. A. (2015). Rethinking extinction. *Neuron*, *88*(1), 47–63.

Gershman, S. J., Blei, D. M., & Niv, Y. (2010). Context, learning, and extinction. *Psychological review*, *117*(1), 197.

Gershman, S. J., & Hartley, C. A. (2015). Individual differences in learning predict the return of fear. *Learning & behavior*, *43*(3), 243–250.

Gershman, S. J., Radulescu, A., Norman, K. A., & Niv, Y. (2014). Statistical computations underlying the dynamics of memory updating. *PLoS computational biology*, *10*(11), e1003939.

Huys, Q. J., & Dayan, P. (2009). A bayesian formulation of behavioral control. *Cognition*, *113*(3), 314–328.

Lloyd, K., & Leslie, D. S. (2013). Context-dependent decision-making: A simple bayesian model. *Journal of The Royal Society Interface*, *10*(82), 20130069.

Niv, Y., Hitchcock, P., Berwian, I. M., & Schoen, G. (2021). Toward precision cognitive-behavioral therapy via reinforcement learning theory. *Precision Psychiatry: Using Neuroscience Insights to Inform Personally Tailored, Measurement-Based Care*, 199.

Norbury, A., Brinkman, H., Kowalchyk, M., Monti, E., Pietrzak, R. H., Schiller, D., & Feder, A. (2020). Latent cause inference during extinction learning in trauma-exposed individuals with and without ptsd. *Psychological Medicine*.

Piray, P., & Daw, N. D. (2021). A model for learning based on the joint estimation of stochasticity and volatility. *Nature communications*, *12*(1), 1–16.

Feature-based learning increases the generalizability of state predictions

Euan Prentis
Department of Psychology
University of Chicago
eprentis@uchicago.edu

Akram Bakkour
Department of Psychology
University of Chicago
bakkour@uchicago.edu

Abstract

Decisions have consequences that gradually unfold over time. To make effective decisions, it is therefore necessary to learn not only which states of the world are useful to be in (*value-based learning*) but also whether these states will be visited in the future (*state-predictive learning*). However, in real-world contexts, states are complex and vary along numerous feature dimensions. This reduces the likelihood that a given combination of features will reoccur, in turn limiting the extent to which past learning can be applied to relevant future experiences. This problem is known as the *curse of dimensionality*. Feature-based learning has been shown to mitigate the curse of dimensionality in the domain of pure value-based learning [1]; theoretically, feature-based learning should improve learning speed, generalizability, and compositionality. The present work addresses whether these advantages extend to the realm of predictive learning. We implement state- and feature-based successor representation models, and simulate their behavior on a novel sequential learning task in which sequences can be learned at either the state or feature level. We found that feature-based learning improves the speed, generalizability, and compositionality of predictive learning. Varying the amount of training each model received, we additionally observed that these advantages were most pronounced with less training. These results support the notion that feature-based learning (1) facilitates quick generalization in novel sequential learning problems, and (2) has the potential to mitigate the curse of dimensionality in real-world contexts. Continuing work will adapt the described task to probe whether humans use feature-based learning to make predictive inferences.

Keywords: Feature-based Learning, Successor Representation, Human Cognition, Compositionality

1 Introduction

Decisions have consequences that gradually unfold over time. For example, after choosing to drive to work instead of taking the subway, you may run into rush-hour traffic, fail to find parking near your workplace, arrive to work late, and ultimately get scolded by your boss for your tardiness. To make effective decisions, it is therefore necessary to learn not only which states of the world are useful to be in (e.g., at work on time; *value-based learning*) but also whether these states will be visited in the future (*state-predictive learning*).

States in real-world contexts are complex and high-dimensional. For example, as you consider which mode of transport to take to work, there will be differences in the weather, your mood, how long you slept, and so forth. This renders state-predictive learning non-trivial to perform. Due to variance along many numerous dimensions, it is unlikely that the exact same combination of features will reoccur. Learning predictions at the multidimensional state level therefore limits the extent to which learning can be applied to relevant future experiences. This problem is known as the *curse of dimensionality*.

In the domain of pure value-based learning, the curse of dimensionality can be mitigated by learning on the decomposed features of states (feature-based learning) rather than the states themselves (state-based learning, [1, 2, 3]). Theoretically, this improves learning along three dimensions: (1) learning speed, (2) generalizability, and (3) compositionality. (1) *Learning speed* is faster because tasks generally have fewer features than states, meaning that features are encountered more frequently. This allows for more opportunities to learn over the same number of training iterations. (2) *Generalizability* is greater because a single feature can be present across multiple states. Therefore, anything learned about a given feature in one state can be applied to any novel state partially composed of the familiar feature. (3) *Compositionality* is greater, since decomposed features encountered in different contexts may be re-combined into novel configurations with some inferred value. An agent can harness the power of compositionality to generate creative, goal-oriented action plans.

The present work addresses whether the benefits of feature-based learning extend to the realm of predictive learning. Using successor representation modeling, we simulate the behavior of state- and feature-predictive learners, and compare the speed, generalizability, and compositionality of their learning.

2 Successor Representation Modeling

State-predictive learning can be modeled using the successor representation (SR, [4]). SR learns a compressed version tasks' multi-step transition structures. This facilitates inference about distant outcomes without performing computationally intensive tree searches through the full state-space (e.g., as with model-based reinforcement learning [5]). Since searching through the large decision trees of real-world contexts is likely intractable, the computational efficiency of SR makes it a good candidate for human state-predictive learning. Work identifying signatures of SR in human behavioral and fMRI data supports this theory [6, 7, 8, 9].

SR learns estimated values and state predictions independently through temporal difference learning [10]. After a reward r is observed, the current state's estimated value V_s is updated according to the reward prediction error, plus the discounted estimated value of the next state $V_{s_{new}}$:

$$V_s = V_s + \alpha(r + \gamma V_{s_{new}} - V_s) \tag{1}$$

where free parameters α and γ respectively control the learning and discount rates. State predictions are represented in the successor matrix $M \in \mathbb{R}^{S \times S}$, where rows and columns correspond to current and new states, respectively. After a transition is observed, a count $e_{s_{new}}$ is added to the new state's position in the row vector. Since this update incorporates the discounted predictions of the new state, SR gradually learns to predict distant visitations. Formally:

$$M_s = M_s + \alpha(e_{s_{new}} + \gamma M_{s_{new}} - M_s) \tag{2}$$

When evaluating given state s, both the estimated values and visitation expectancies are incorporated:

$$O_s = M_s V_s \tag{3}$$

State values O_s are then turned into choice probabilities using a softmax with inverse temperature β:

$$p(a|s, s') = \frac{501 \quad e^{\frac{O_s}{\beta}}}{e^{\frac{O_s}{\beta}} + e^{\frac{O_{s\prime}}{\beta}}} \tag{4}$$

2.1 State-based SR

State-based SR learns and evaluates actions as described. It additionally caches visited states in memory as it interacts with the environment. To generalize to a novel state, the model identifies the most similar representation in memory, and retrieves the associated V_s and M_s. State-based SR also uses representational similarity to guide composition. When tasked with combining decomposed features into some rewarding state, the model considers all possible combinations of the provided features, and deterministically builds the combination which is most similar to high-value states in memory.

2.2 Feature-based SR

Feature-based SR is achieved by simply tweaking the model to learn V and M separately for each feature category f. In this case, final values F_{fi} are produced per feature instance i.

$$F_{fi} = M_{fi}V_{fi} \tag{5}$$

The state's final value O_s is the mean of these feature values. To generalize to novel states composed of partially familiar features, feature-based SR directly calculates O_s from the familiar feature values F_{fi}. When tasked with combining decomposed features into some rewarding state, the model also directly retrieves associated F_{fi}'s and deterministically builds the state that maximizes these values.

3 Sequential Learning Task

We implemented a sequential learning task (Figure 1), in which agents' goal was to maximize point earnings. The task consisted of three phases: training, test, and composition.

During training, choices were made between pairs of items that represented different states (Figure 1A). Each item was composed of three feature instances, sampled from five feature categories (ABCDE). Only three of the five categories would be seen in each training half (1st half: ABC; 2nd half: ADE; Figure 1C). After a choice was made, a single-step sequence was displayed followed by a reward (r = [-6, 6]). The successor item's

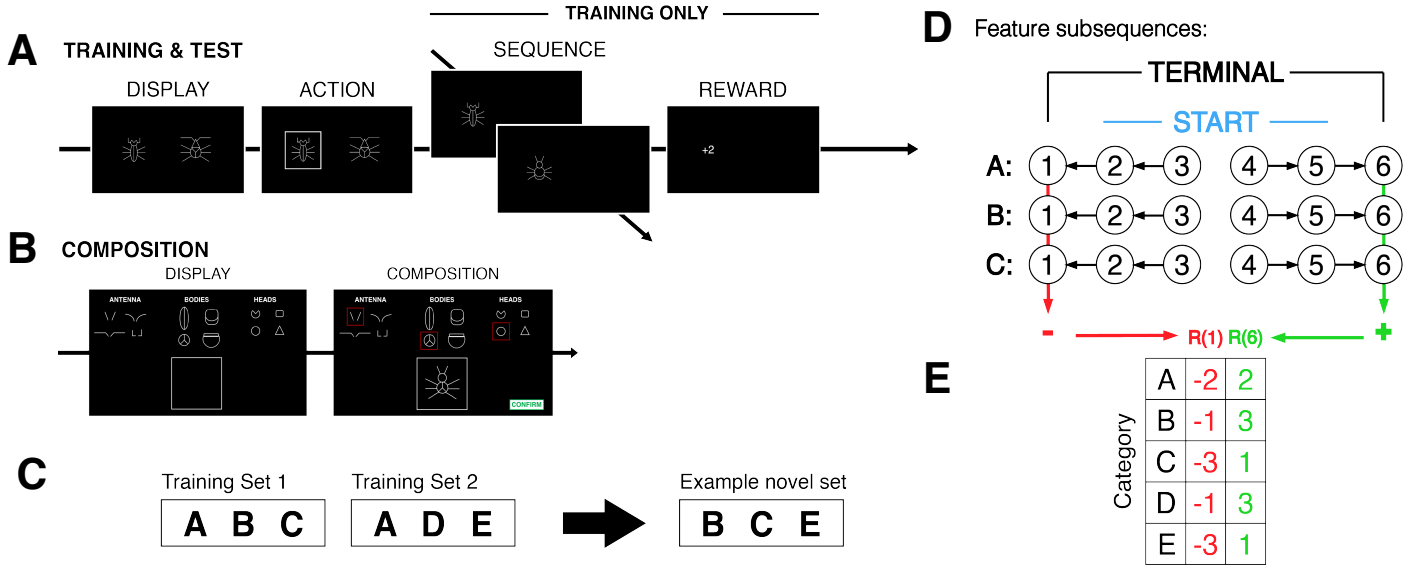


Figure 1: Task Design. A. Training and test trials. After an action was made during training, a one-step sequence and reward were observed (these were not seen at test). Example stimuli here are from the human subjects task. **B.** Composition trial. Agents selected decomposed features from different categories on each trial. **C.** Feature category sets. Whereas training items were directly sampled from a fixed set in each training half (i.e., ABC or ADE), novel items were composed of a mixture of features across halves (e.g., ACE, BDE, BCE). **D.** Sub-sequences associated with each feature category. Start items were composed of feature instances 2, 3, 4, or 5. Successor items were partially or fully composed of terminal feature instances (1, 6), that are associated with a reward value. **E.** Reward values of terminal feature instances. The reward values for terminal feature instances associated with a successor item would be summed to get the item reward value.

identity and reward value were determined by sub-sequences associated with each feature category (Figure 1DE). Thus, to maximize cumulative reward, agents had to predict which choices would lead to rewarding successor items.

Test was similar to training, but agents neither observed the one-step sequences nor the reward outcomes, preventing further learning. On each test trial, choices were between either training or novel items (Figure 1C). We hypothesized that feature-based learning is more generalizable, and thus expected the feature-based model to perform better on novel item trials. Trials also differed on whether choices were between terminal or start items (Figure 1D). In contrast to terminal items, start items were never worth points, and only indirectly led to reward outcomes through terminal items. Therefore, to make accurate inferences on start item trials, agents needed to apply both value and state-predictive learning.

Finally, agents completed a composition phase. On each trial, they were presented with decomposed feature instances, and had to recompose them into a reward-predictive item (Figure 1B). These trials also differed on whether training or novel items, and start or terminal items could be composed. Since the feature-based model learns directly on features, we predicted that it would be more effective at constructing rewarding items than the state-based model.

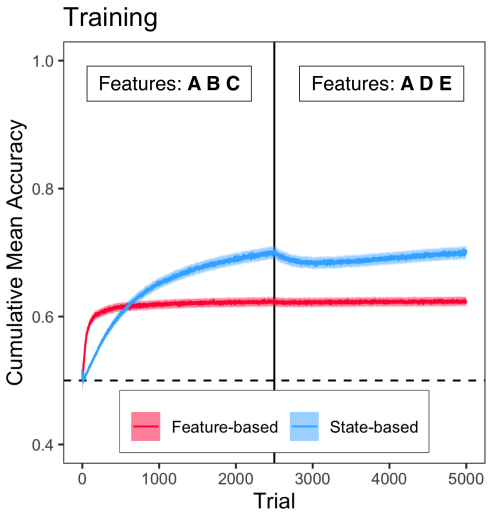


Figure 2: Training Curves. Curves are quantified by cumulative mean accuracy, and averaged over 1000 simulations of 5000 trials per model. Errors at 95% confidence intervals. The first 2500 trials involved features from categories ABC, and the remaining trials involved features from categories ADE.

4 Results

We simulated the feature- and state-based learning models on the sequential learning task, and compared the speed, generalizability, and compositionality of their learning.

First, we probed whether feature-based learning is faster by comparing the models' learning trajectories over 5000 training trials. We calculated the cumulative mean accuracy (CMA) of each agent's choices, where an accurate choice is defined as one that will lead to a more rewarding successor item. In each training half, since feature-based learners encountered each of the 18 feature instances more frequently than the state-based learners encountered each of the 128 unique items, we hypothesized that feature-based learners would achieve higher accuracy earlier in training. The results reflect this hypothesis (Figure 2). Whereas state-based learners surpassed a mean CMA of 0.6 on trial 465, feature-based learners surpassed a mean CMA of 0.6 on trial 150. However, on the final trial, state-based learners achieved a higher mean CMA (M = 0.70) than the feature-based learners (M = 0.62). These results indicate that while feature-based learning is faster, state-based learning is more accurate in the long run. There was also a more substantial drop in state-based learners' accuracy entering the second training half (where feature categories B and C were replaced with D and E). This indicates they were poorer at applying past learning to

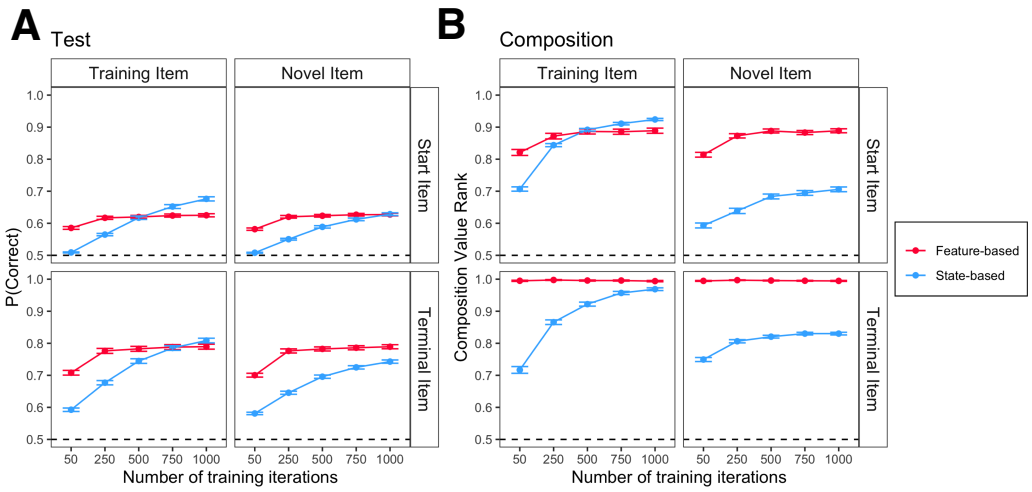


Figure 3: Test and Composition Performance. Errors are 95% confidence intervals. **A.** Proportion of correct choices made during test, by training amount and item type. **B.** Composition-value rank, by training amount and item type.

the new context and learning about the novel items.

Next, we assessed the models' abilities to generalize during the test phase with different amounts of training (50, 250, 500, 750, or 1000 trials). As predicted, we found that the feature-based model was better at generalizing, particularly with fewer training iterations (Figure 3A). With just 50 training iterations, the feature-based learners were more accurate on all trial types. However, the magnitude of this difference reduced with more learning, supporting the notion that while feature-based learning is faster, state-based learning performs better in the long-term. Notably, whereas the state-based learners' training accuracy came to surpass that of feature-based learners, their novel item accuracy did not. This suggests feature-based learning particularly facilitated generalization.

Lastly, we compared the models' performance at composing reward-predictive items during the composition phase. High performance is operationalized as the reward-value rank of the composed item, relative to all other possible combinations of features presented on a given trial. We predicted that the feature-based model would be able to leverage its feature-level learning to compose rewarding items more precisely, and thus achieve higher overall performance. The results support this hypothesis (Figure 3B). Notably, feature-based learners achieved higher composition accuracy for novel items across all numbers of training iterations.

5 Discussion

Simulating feature- and state-based predictive learning, we found that learning on the features of states rather than the states themselves bolsters learning speed, generalizability, and compositionality. These advantages were particularly pronounced with less training. Our results have two important implications.

Firstly, in novel tasks, agents would benefit from performing feature-based learning to achieve some degree of accuracy quickly. However, with extended experience, they may benefit from switching to more state-based mechanisms. Farashahi, Rowe, and colleagues [1] demonstrated similar advantages in the domain of pure value-based learning, finding that human subjects gradually switched from feature- to state-based learning in a low-dimensional environment. State abstraction also increases the generalizability of predictive learning [11]. However, since abstractions must be learned over multiple experiences, feature-based learning may be advantageous prior to the formation of stable abstractions.

Secondly, feature-based learning may help agents learn structural contingencies in real-world environments, where the curse of dimensionality renders state-based learning ineffective. For this reason, human beings may rely on feature-based learning to make predictive inferences. Continuing work will adapt the sequential learning task described here to test this hypothesis.

References

1. Farashahi, S., Rowe, K., Aslami, Z., Lee, D., & Soltani, A. (2017). Feature-based learning improves adaptability without compromising precision. *Nature Communications*, *8*(1), 1768.
2. Leong, Y. C., Radulescu, A., Daniel, R., DeWoskin, V., & Niv, Y. (2017). Dynamic Interaction between Reinforcement Learning and Attention in Multidimensional Environments. *Neuron*, *93*(2), 451–463.
3. Niv, Y., Daniel, R., Geana, A., Gershman, S. J., Leong, Y. C., Radulescu, A., & Wilson, R. C. (2015). Reinforcement Learning in Multidimensional Environments Relies on Attention Mechanisms. *Journal of Neuroscience*, *35*(21), 8145–8157.
4. Dayan, P. (1993). *Improving Generalisation for Temporal Difference Learning: The Successor Representation*. 14.
5. Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., & Dolan, R. J. (2011). Model-based influences on humans' choices and striatal prediction errors. *Neuron*, *69*(6), 1204–1215.
6. Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., & Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature Human Behaviour*, *1*(9), 680–692.
7. Russek, E. M., Momennejad, I., Botvinick, M. M., Gershman, S. J., & Daw, N. D. (2021). *Neural evidence for the successor representation in choice evaluation* [Preprint]. Neuroscience.
8. Stachenfeld, K. L., Botvinick, M. M., & Gershman, S. J. (2017). The hippocampus as a predictive map. *Nature Neuroscience*, *20*(11), 1643–1653.
9. Gershman, S. J. (2018). The Successor Representation: Its Computational Logic and Neural Substrates. *The Journal of Neuroscience*, *38*(33), 7193–7200.
10. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press.
11. Lehnert, L., Littman, M. L., & Frank, M. J. (2020). Reward-predictive representations generalize across tasks in reinforcement learning. *PLOS Computational Biology*, *16*(10), e1008317.

Robotic Planning under Uncertainty in Spatiotemporal Environments for Expeditionary Science

Victoria Preston*
MIT-WHOI Joint Program
Massachusetts Institute of Technology
Cambridge, MA 02139
vpreston@csail.mit.edu

Genevieve Flaspohler*
MIT-WHOI Joint Program
Massachusetts Institute of Technology
Cambridge, MA 02139
geflaspo@csail.mit.edu

Anna P. M. Michel
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
amichel@whoi.edu

John W. Fisher III
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
fisher@csail.mit.edu

Nicholas Roy
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
nickroy@csail.mit.edu

Abstract

In the expeditionary sciences, spatiotemporally varying enviironments — hydrothermal plumes, algal blooms, lava flows, or animal migrations — are ubiquitous. Mobile robots are uniquely well-suited to study these dynamic, mesoscale natural environments. We formalize expeditionary science as a *sequential decision-making problem*, modeled using the language of partially-observable Markov decision processes (POMDPs). Solving the expeditionary science POMDP under real-world constraints requires efficient probabilistic modeling and decision-making in problems with complex dynamics and observational models. Previous work in informative path planning, adaptive sampling, and experimental design have shown compelling results, largely in static environments, using data-driven models and information-based rewards. However, these methodologies do not trivially extend to expeditionary science in spatiotemporal environments: they generally do not make use of scientific knowledge such as equations of state dynamics, they focus on information gathering as opposed to scientific task execution, and they make use of decision-making approaches that scale poorly to large, continuous problems with long planning horizons and real-time operational constraints. In this work, we discuss these and other challenges related to probabilistic modeling and decision-making in expeditionary science, and present some of our preliminary work that addresses these gaps. We ground our results in a real expeditionary science deployment of an autonomous underwater vehicle (AUV) in the deep ocean for hydrothermal vent discovery and characterization. Our concluding thoughts highlight remaining work to be done, and the challenges that merit consideration by the reinforcement learning and decision-making community.

Keywords: Bayesian optimization, planning under uncertainty, robotics, informative path planning, adaptive sampling

Acknowledgements Support for scientific cruise RR2107 was supported by NSF OCE OTIC #1842053 and we thank the Captain and Crew of R/V Revelle, AUV SENTRY and ROV JASON teams, Scott Wankel, Peter Girguis, and colleagues for cruise assistance. V.P. is funded by a Martin Fellowship and in part by a NDSEG Fellowship. G.F. is funded by a Microsoft Research Fellowship and in part by award DE-NA000392 under the Department of Energy/National Nuclear Security Administration.

1 Introduction

Expeditionary science is the act of collecting *in situ* samples and observations of the natural world for the purposes of scientific discovery and model development. Transient, dynamic phenomena — hydrothermal plumes, algal blooms, lava flows — are of interest in many disciplines of observational science. It is often impossible for a single, static sensor to capture a comprehensive picture of these mesoscale spatiotemporal phenomena due to their large spatial scales and dynamic temporal nature. Large networks of static sensor nodes could address this sensing challenge, but the transience and sparsity of the target phenomena necessitates logistically impractical sensor density. Mobile robots are uniquely well-positioned to tackle this problem via extended deployments carrying heterogeneous sensor payloads that actively seek and perform targeted surveys of ephemeral environments. However, robots deployed for expeditionary science today currently execute open-loop, preset trajectories, like "lawnmowers" (back-and-forth grid-based pattern) hand-designed by human scientists. In dynamic environments, this open-loop execution often results in sparse measurements of the target phenomena or misses a short-lived target entirely. Given the cost of expeditionary field operations and the value of the data collected, it is critical to improve the efficiency and efficacy of robots as scientific tools.

Building robotic platforms for expeditionary science requires an autonomy stack that can integrate observations from heterogeneous sensors into a model of a spatiotemporal system and use this model to plan informative trajectories that target a specific scientific objective. This kind of autonomy poses many challenges for integrating probabilistic modeling and decision-making. Previous work in informative path planning (IPP) [1], adaptive sampling/experimental design [2], and decision-making under uncertainty [3] has tackled aspects of the expeditionary science problem, especially in static environments using data-driven models and information-based rewards. However, existing methodologies do not trivially extend to spatiotemporal environments, leaving key challenges such as model and dynamics learning and decision-making in large-scale environments unaddressed. In this abstract, we discuss these and other key challenges, and present some preliminary work conducted for hydrothermal plume discovery and mapping in the deep sea.

2 Expeditionary Science as a Decision-Making Problem

In an expeditionary science mission, a robot is tasked with collecting scientifically useful measurements of an unknown, partially-observable spatiotemporal environment under time, energy, and dynamical constraints. We formulate this sequential decision-making problem as a partially observable Markov decision-process (POMDP). Let $\Pi(\cdot)$ denote the space of probability distributions over the argument. A finite horizon POMDP can be represented as tuple: $(\mathcal{S}, \mathcal{A}, T, R, \mathcal{Z}, O, b_0, H, \gamma)$, where $\mathcal{S}$ are the states, $\mathcal{A}$ are the actions, and $\mathcal{Z}$ are the observations. At planning iteration t, the agent selects an action $a \in \mathcal{A}$ and the transition function $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ defines the probability of transitioning between states in the world, given the current state s and control action a. After the state transition, the agent receives an observation according to the observation function $O : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{Z})$, which defines the probability of receiving an observation, given the current state s and previous control action a. The reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ serves as a specification of the task. A POMDP is initialized with belief b_0 and plans over horizon H with discount factor γ.

Due to the stochastic, partially observable nature of current and future states, the realized reward in a POMDP is a random variable. Optimal planning is defined as finding a policy $\{\pi_t^* : \Pi(\mathcal{S}) \to \mathcal{A}\}_{t=0}^{H-1}$ that maximizes expected reward: $\mathbb{E}\Big[\sum_{t=0}^{H-1} \gamma^t R\big(S_t, \pi_t(b_t)\big) \mid b_0\Big]$, where b_t is the updated belief at time t, conditioned on the history of actions and observations. The recursively defined horizon-h optimal value function V_h^* quantifies, for any belief b, the expected cumulative reward of following an optimal policy over the remaining planning iterations: $V_0^*(b) = \max_{a \in \mathcal{A}} \mathbb{E}_{s \sim b}[R(s, a)]$ and

$$V_h^*(b) = \max_{a \in \mathcal{A}} \mathbb{E}_{s \sim b}[R(s, a)] + \gamma \int_{z \in \mathcal{Z}} P(z \mid b, a) V_{h-1}^*(b^{a,z}) \mathrm{d}z \qquad h \in [1, H-1], \tag{1}$$

where $b^{a,z}$ is the updated belief after taking control action a and receiving observation z, computed via Bayes rule using the transition T and observation O functions. The optimal policy at horizon h is to act greedily according to a one-step look ahead of the horizon-h value function. However, Eq. 1 is intractable for large or continuous state, action, or observation spaces and thus the optimal policy must be approximated. Well-designed algorithmic and heuristic choices are necessary to develop efficient, practical, and robust planning algorithms for expeditionary sciences.

3 Advances to Expeditionary Science

As a case study we highlight a scientific mission to the Gulf of California in November 2021, in which a research vessel deployed an autonomous underwater vehicle (AUV) for deep sea discovery and mapping of hydrothermal vent plumes (see Fig. 1). Characterizing hydrothermalism has implications for global nutrient and energy budgets, but nearly two-thirds of all hypothesized hydrothermal vents are yet undiscovered [4]. AUVs with point sensors and the ability to execute trajectories under operational constraints can "sniff" for vents in the water column and use observations to estimate seafloor vent characteristics, but tidal advection, diffusion, and turbulent mixing all contribute to complicated spatiotemporal observations. Additionally, accurate vent inference requires the disambiguation of the spatially-small, chemically-rich buoyant stem from the spatially-vast neutrally-buoyant layer that compose plumes [5]. Here, we present some of our work tackling this expeditionary robotics problem, and highlight key challenges we found in this context.

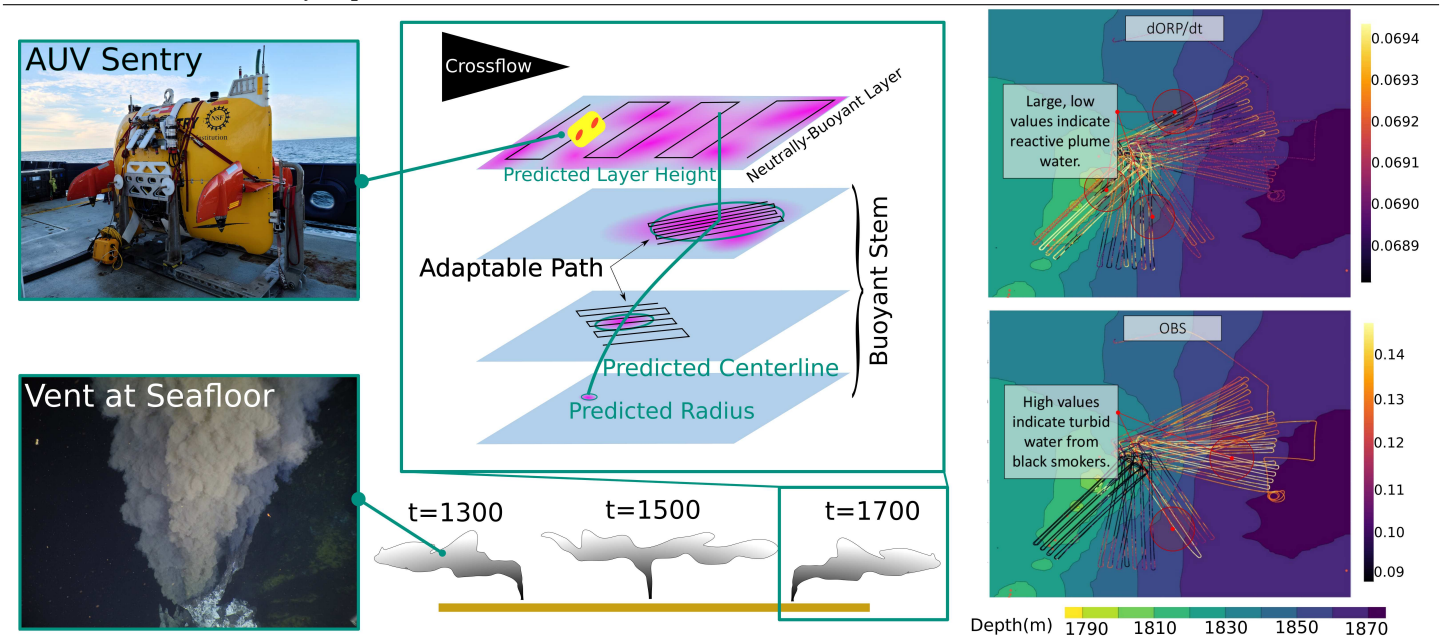


Figure 1: In November 2021, AUV Sentry was deployed in the Gulf of California to detect and map hydrothermal plume structures in the deep ocean. Plumes are composed of a buoyant stem and neutrally-buoyant layer. These plumes are dynamic, subject to tidal advection, diffusion, and turbulent mixing. We infer the spatiotemporal structure of plumes from sparse partial-observations and plan informative trajectories that respect operational constraints of AUV Sentry. Field data are shown on the right from two onboard science point-sensors. The dORP/dt sensor shows areas of reactive water, likely indicating buoyant stem detections. The OBS sensor shows areas of high turbidity, which can persist throughout a plume. Combining these sensor signals, we can infer the plume structure and dynamics.

3.1 Spatiotemporal IPP for the Maximum Seek-and-Sample Problem

We formulated the problem of finding the buoyant stem within the multi-modal non-buoyant plume layer using a robot as a POMDP and developed PLUMES : **P**lume **L**ocalization under **U**ncertainty using **M**aximum-valu**E** information and **S**earch [6], an autonomy stack for target-seeking in multi-modal environments. PLUMES utilized a Gaussian Process (GP) belief to model a single time-varying chemical signal, a progressively-widened Monte Carlo tree search, and a novel task-driven information-theoretic reward function. In spatiotemporal systems, PLUMES was able to find and track moving buoyant-stem targets and repeatedly explore an environment as uncertainty grew under stochastic plume dynamics [7]. PLUMES addressed challenges in continuous search for planning and balancing exploration-exploitation behavior under sparse reward signals, and demonstrated how the sum of algorithmic components can lead to a potent system. However, PLUMES also revealed the challenges of expeditionary science, particularly the difficulty of simultaneously training and utilizing a spatiotemporal belief model, and developing computationally tractable real-time nonmyopic planners.

3.2 Abstraction via Task-driven Macro-action Discovery

One key barrier to tractable, real-time nonmyopic planning in PLUMES was the need to search over long planning horizons, chaining together low-level action primitives to explore the environment. The development of abstract, high-level action representations would enable planners to search over shorter planning horizons, saving compute effort during real-time planning. Instead of using human-designed abstractions of the robot's continuous action space, our recent work learns a set of high-level actions that are useful for a given scientific task directly from a low-level specification of the POMDP model [8]. The algorithm takes into account the robot's current uncertainty about the state of the world, the stochasticity of the underlying environmental dynamics, and the current task to develop a set of high-level actions that can be used online while providing formal performance guarantees about the resulting "abstract" policies.

3.3 Iterative Mission Structures and Physically-Informed Uncertainty Models

PLUMES used a general, data-driven model of the environment and assumed that the robot's action space consisted of flexible, low-level motion primitives. To improve the operational effectiveness of PLUMES at sea, we proposed to: 1) make use of data-efficient probabilistic models that explicitly use physical equations that govern plume expression in the water column, and 2) develop planners that directly encode the physical and operational constraints of the AUV.

We developed and tested this new algorithmic expeditionary system during field trials in November 2021. The system made use of a belief model called PHUMES : **PH**ysically-informed **U**ncertainty **M**odel of **E**nvironment **S**patiotemporality, which sampled uncertain parameters of a simplified model of buoyancy-driven hydrothermalism (e.g., vent temperature, tidal crossflow) to generate multiple "envelopes" of possible plume-derived waters. These were aggregated into a probabilistic forecast over the spatiotemporal volume to be sampled by the AUV. A trajectory optimizer with knowledge of vehicle and ship operational constraints then scaled lawnmower primitives to maximize the number of "in plume" measurements that were to be collected at the continuum between buoyant stem and neutrally-buoyant layer. PHUMES addressed open challenges in robotic decision-making related to injection of prior knowledge or scientific inductive bias for data efficient training in complex domains and scalable planning with expensive belief models.

4 Core Challenges and Opportunities in Expeditionary Robotics

As identified in Sec. 3, there are a number of open challenges in reinforcement learning and decision-making in the context of expeditionary robotics. Here we present a brief survey of these open challenges and opportunities informed by years of collaborating with scientists to develop expedition frameworks in diverse application areas.

4.1 Belief Representations

Epistemic and aleatoric uncertainty: Reducing epistemic uncertainty of a spatiotemporal environment requires access to a model of the underlying dynamical system, or a data-driven technique that can uncover it. Extracting physically-meaningful quantities from observational data is typically performed post-expedition using computationally expensive numerical models "tuned" by observations. While this lends itself well to Bayesian inference formulations, it is intractable for practical decision-making. Data-driven techniques for model discovery [9] may be arguably more tractable, but generally suffer small-data challenges. Developing models that overcome the challenges of efficiently characterizing spatiotemporal dynamics from streaming, sparse observations would generally improve expeditionary robotics. Additionally, there is a unique opportunity to enable computation of proxies for aleatoric uncertainty, which are well-described in spatiotemporal environments with measures of chaotic motion (e.g., Lyapunov exponents) inferred from data [10]. The implication that aleatoric uncertainty can be estimated has yet to be utilized to, e.g., assess the attainable resolution of a model or set planning horizons.

Scientific knowledge as inductive bias: The kernel of a GP, the loss function in a neural network, or the activation functions between layers in a deep network can all be viewed as forms of inductive bias in a learning problem. For data-driven discovery of spatiotemporal dynamics, improving sample efficiency by leveraging opportunities to inject scientific knowledge to alleviate the learning burden is an open problem. While canonical numerical models of spatiotemporal phenomena are too computationally expensive to directly incorporate into e.g., GP kernels, the physical principles that underlie these models can be more easily summarized. "Physically-informed" data-driven probabilistic representations have been demonstrated outside of expeditionary robotics [9] and some work within IPP [11] shows rich opportunities for analyzing and extending these methods for larger environments and longer planning horizons.

Low-dimensional state embeddings: Expressing a spatiotemporal environment completely would require an exceedingly large, high-dimensional representation. Model order reduction (MOR) techniques reduce the dimensionality of spatiotemporal systems to a set of weights and vectors that sufficiently describe patterns in the dynamics. Uncovering low-dimensional state embeddings from partially-observed expedition data is a general challenge; uncovering a *useful* embedding for a specific decision-making problem is additionally challenging. Access to such an embedding would reduce the computational burden of representing belief in large environments for planning.

4.2 Decision-Making

Rollout-based planning with expensive belief models: State-of-the-art planners for POMDP problems often make use of rollout-based planning in tree search frameworks; continuous search variables are handled using strategies such as progressive widening or scenario sampling [3]. However, these planners require extensive online simulations for each rollout performed. Forward-simulating the dynamics and observational models for complex, spatiotemporal phenomena can be computationally intensive, which often limits the feasible look-ahead horizon in real-time operations on computationally-limited robotic platforms. Planners that selectively or adaptively perform expensive rollouts, automatically adjust the planning horizon based on the dynamics of the environmental system, or make use of continuous, offline planners would enable improved decision-making for expeditionary science.

Information rewards and task-driven exploration: Due to partial observability and stochastic dynamics in spatiotemporal contexts, a decision-maker must operate with significant and often growing state uncertainty. However, not all state uncertainty impacts task performance and uniform information gathering strategies can be inefficient. Understanding the value of information for accomplishing a task is a known challenge for planning under uncertainty and this is particularly true for expeditionary robotics. Recent works that develop heuristic information rewards [6] or task-driven value of information metrics [8] begin to build the tools necessary for expeditionary robotic planning.

Robust planning under model mismatch and uncertainty: Scientific models, whether data-driven or based on physical principles, are always imperfect representations of a robot's environment. Model mismatch or uncertainty in key model parameters leads to discrepancies between the environmental predictions that a robot uses during planning and its real-time observations. Planning robot trajectories that entirely miss a phenomenon due to overconfidence in an incorrect model is detrimental to scientific objectives. Planners must develop policies or trajectories that are robust to model mismatch and uncertainty, or are guaranteed to perform as well as a simple, naive data collection strategy.

Interpretable and operational decision-making: Decision-making algorithms must interface with and are constrained by a variety of stakeholders, including scientists, robot operators, and engineers. For example, when deploying an AUV from an oceanographic research vessel, the decision-making algorithm must account for ship scheduling, timing delays, weather, and multi-vehicle operations. This requires developing flexible planners that can understand and account for these complex constraints. Additionally, stakeholders are often concerned with robot safety and data quality. Producing plans that are interpretable for scientific and operational stakeholders is key for building trust and confidence in scientific autonomy.

5 Final Thoughts

Expeditionary science and robotics motivates a set of interesting reinforcement learning and decision-making problems that are not well-addressed by current state-of-the-art methods. There is increasing need to enable better *in situ* methodologies for monitoring and characterizing large spatiotemporal environments as accelerated climate change transforms delicate carbon budgets, ecosystem networks, and weather patterns. Robotic technologies can play a transformative role in understanding these evolving trends and assessing policy interventions. Novel developments towards the challenges listed here may have constructive and complementary effects in other application spaces, as many domains, such as material discovery and robotic manipulation, have overlapping challenges and opportunities. Our contributions, including PLUMES [6,7], macro-action discovery [8], the PHUMES model and trajectory optimizer for operational missions, and ongoing work in physically-informed deep kernel learning, represent algorithmic and systems contributions towards more effective autonomous systems in expeditionary science and begin to address some of the key challenges that we propose.

References

[1] G. Hitz, E. Galceran, M.-È. Garneau, F. Pomerleau, and R. Siegwart, "Adaptive continuous-space informative path planning for online environmental monitoring," *J. Field Robot.*, vol. 34, no. 8, pp. 1427–1449, 2017.

[2] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies." *Journal of Machine Learning Research*, vol. 9, no. 2, 2008.

[3] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for POMDPs with continuous state, action, and observation spaces," in *Proc. 24th Int. Conf. Automated Plan. Schedul.*, 2018.

[4] S. E. Beaulieu, E. T. Baker, and C. R. German, "Where are the undiscovered hydrothermal vents on oceanic spreading ridges?" *Deep Sea Research Part II: Topical Studies in Oceanography*, vol. 121, pp. 202–212, 2015.

[5] B. Morton, G. I. Taylor, and J. S. Turner, "Turbulent gravitational convection from maintained and instantaneous sources," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, pp. 1–23, 1956.

[6] G. Flaspohler, V. Preston, A. P. Michel, Y. Girdhar, and N. Roy, "Information-guided robotic maximum seek-and-sample in partially observable continuous environments," *IEEE Robotics and Automation Letters*, pp. 3782–3789, 2019.

[7] V. L. Preston, *Adaptive sampling of transient environmental phenomena with autonomous mobile platforms*. Massachusetts Institute of Technology, 2019.

[8] G. Flaspohler, N. A. Roy, and J. W. Fisher III, "Belief-dependent macro-action discovery in POMDPs using the value of information," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 108–11 118, 2020.

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[10] A. Blanchard and T. P. Sapsis, "Analytical description of optimally time-dependent modes for reduced-order modeling of transient instabilities," *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 2, pp. 1143–1162, 2019.

[11] T. Salam and M. A. Hsieh, "Adaptive sampling and reduced-order modeling of dynamic processes by robot teams," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 477–484, 2019.

What makes useful auxiliary tasks in reinforcement learning: investigating the effect of the target policy

Banafsheh Rafiee
University of Alberta, Canada,
Noah's Ark Lab, Huawei Technologies Canada
rafiee@ualberta.ca

Jun Jin
Noah's Ark Lab, Huawei Technologies Canada
jun.jin1@huawei.com

Jun Luo
Noah's Ark Lab, Huawei Technologies Canada
jun.luo1@huawei.com

Adam White
University of Alberta, Canada
amw8@ualberta.ca

Abstract

Auxiliary tasks have been argued to be useful for representation learning in reinforcement learning. Although many auxiliary tasks have been empirically shown to be effective for accelerating learning on the main task, it is not yet clear what makes useful auxiliary tasks. Some of the most promising results are on the pixel control, reward prediction, and the next state prediction auxiliary tasks; however, the empirical results are mixed, showing substantial improvements in some cases and marginal improvements in others. Careful investigations of how auxiliary tasks help the learning of the main task is necessary. In this paper, we take a step studying the effect of the target policies on the usefulness of the auxiliary tasks formulated as general value functions. General value functions consist of three core elements: 1) policy 2) cumulant 3) continuation function. Our focus on the role of the target policy of the auxiliary tasks is motivated by the fact that the target policy determines the behavior about which the agent wants to make a prediction and the state-action distribution that the agent is trained on, which further affects the main task learning. Our study provides insights about questions such as: Does a greedy policy result in bigger improvement gains compared to other policies? Is it best to set the auxiliary task policy to be the same as the main task policy? Does the choice of the target policy have a substantial effect on the achieved performance gain or simple strategies for setting the policy, such as using a uniformly random policy, work as well? Our empirical results suggest that: 1) Auxiliary tasks with the greedy policy tend to be useful. 2) Most policies, including a uniformly random policy, tend to improve over the baseline. 3) Surprisingly, the main task policy tends to be less useful compared to other policies.

Keywords: representation learning; auxiliary tasks; general value functions; reinforcement learning

1 Introduction

Learning about many aspects of the environment in addition to the main task of maximizing the discounted sum of rewards has been argued to be beneficial in reinforcement learning [Sutton et al., 2011]. A common view is that these additional tasks, also known as auxiliary tasks, can improve the data efficiency by shaping the representation [Jaderberg et al., 2016, Shelhamer et al., 2016, Mirowski et al., 2016]. In environments with sparse reward structures, auxiliary tasks provide instantaneous targets for shaping the representation in the absence of reward. It has also been argued that auxiliary tasks can function as regularizers, improving the generalization and avoiding representation overfitting in RL [Dabney et al., 2020]. Finally, what has been learned about an auxiliary task can be transferred to the main task, improving the data efficiency.

Many auxiliary tasks have been proposed and shown to accelerate learning on the main task. However, much of the work on auxiliary tasks have demonstrated their efficacy empirically and in many cases, the empirical results are mixed: in some cases auxiliary tasks result in substantial performance gain over the baselines whereas in other cases they achieve marginal improvements or even harm the performance [Jaderberg et al., 2016, Shelhamer et al., 2016]. Recent works have explored how auxiliary tasks benefit the representation learning in a systematic way [Lyle et al., 2021, Dabney et al., 2020]. More systematic studies are required to answer the question of what makes useful auxiliary tasks.

In this paper, we take a step toward answer the question of what makes useful auxiliary tasks. We specifically consider auxiliary tasks formulated as general value functions (GVF) [Sutton et al., 2011]. A core element of GVFs is the target policy as it both determines the target of prediction and the state-action distribution that the agent gets trained on. We empirically study the effect of the target policies associated with auxiliary tasks on their usefulness.

2 Background

In this paper, we consider the interaction of an agent with its environment at discrete time steps. At each time step t, the agent is in a state $S_t \in \mathcal{S}$, performs an action $A_t \in \mathcal{A}$ according to a policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, receives a reward $R_{t+1} \in \mathbf{R}$, and transitions to the next state S_{t+1}. We consider the problems of prediction and control.

For the prediction problem, the goal of the agent is to approximate the value function for a given policy π where the value function is defined by: $v_\pi(s) \doteq \mathbf{E}_\pi[G_t|S_t = s]$ where $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is called the return with $\gamma \in [0, 1)$ being the discount factor. In the control setting, the goal of the agent is to maximize the expected return. In this setting, it is common to use state-action value functions: $q_\pi(s, a) \doteq \mathbf{E}_\pi[G_t|S_t = s, A_t = a]$.

To estimate the value function, we use semi-gradient temporal-difference learning [Sutton, 1988]. More specifically, we use TD(0) to learn a parametric approximation $\hat{v}(s; \mathbf{w})$ by updating a vector of parameters $\mathbf{w} \in \mathbf{R^d}$ as follows:

$$\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + \alpha\delta_\mathbf{t}\nabla_\mathbf{w}\hat{\mathbf{v}}(\mathbf{S_t}; \mathbf{w})$$

where α denotes the step-size parameter and δ_t denotes the TD error: $R_{t+1} + \gamma\hat{v}(S_{t+1}; \mathbf{w_t}) - \hat{\mathbf{v}}(\mathbf{S_t}; \mathbf{w_t})$. $\nabla_\mathbf{w}\hat{v}(S_t; \mathbf{w})$ is the gradient of the value function with respect to the parameters $\mathbf{w_t}$.

For the control setting, to estimate the state-action value functions, we use the control variant of TD(0), Q-learning [Watkins and Dayan, 1992]. The update rule for Q-learning is similar to that of TD(0); however, $\hat{q}(S_t, A_t, \mathbf{w})$ is used instead of $\hat{v}(S_t, \mathbf{w})$ and the TD error is defined as $R_{t+1} + \gamma\text{max}_a\hat{q}(S_{t+1}, a; \mathbf{w_t}) - \hat{\mathbf{q}}(\mathbf{S_t}, \mathbf{A_t}; \mathbf{w_t})$. For action selection, Q-learning is commonly integrated with the epsilon greedy policy.

As the function approximator, we use neural networks. We integrate a replay buffer, a target network, and the RMSProp optimizer with TD(0) and Q-learning as is commonly done to improve performance when using neural networks as the function approximator [Mnih et al., 2013].

To formulate auxiliary tasks, a common approach is to use general value functions (GVFs). GVFs are value functions with a generalized notion of target and termination. More specifically, a GVF can be written as the expectation of the discounted sum of any signal of interest: $v_{\pi,\gamma,c}(s) \doteq \mathbf{E}[\sum_{k=0}^{\infty}(\prod_{j=1}^{k}\gamma(S_{t+j}))c(S_{t+k+1})|S_t = s, A_{t:\infty} \pi]$ where π is the target policy, γ is the continuation function, and c is a signal of interest. General state-action value function $q_{\pi,\gamma,c}(s, a)$ can be defined similarly with the difference that the expectation is conditioned on $A_t = a$ as well as $S_t = s$.

3 Investigating the effect of the auxiliary task's policy on its usefulness

As we discussed in Section 1, we study the effect of the target policy of the auxiliary tasks on their usefulness. Our experimental setup includes two phases: 1) Pre-training on the auxiliary tasks 2) Learning the main task. For the pre-training phase, a neural network is used as the function approximator for learning the auxiliary tasks. The behavior

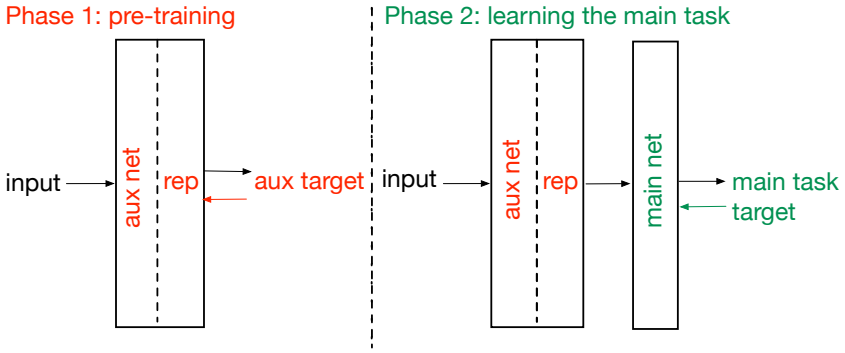


Figure 1: The two phases of pre-training on the auxiliary tasks and learning the main task.

policy is set to the target policy of the auxiliary tasks. In the second phase, the representation learned during the pre-training phase is kept fixed and fed to another network to learn the main task. The behavior policy in this phase is set to the policy learned for the main task. See Figure 1. Using this procedure, the representation learned for the auxiliary tasks in Phase 1 is transferred to be used in Phase 2 when learning the main task.

For the auxiliary tasks, we considered 4 policies: 1) Uniform random 2) Greedy 3) Sticky actions 4) The main task policy. The sticky actions policy is to follow the previous action with probability 0.9 and to select an action randomly with probability 0.1. Learning the auxiliary task corresponding to the greedy policy is a control problem whereas learning the auxiliary tasks corresponding to the other policies are prediction problems.

We experimented with pixel-based and non pixel-based environments. For the non pixel-based environments, we considered two types of cumulants: 1) The observation O_{t+1} 2) The observation difference $O_{t+1} - O_t$. We also used 6 different values of γ: $\{0, 0.5, 0.75, 0.8, 0.9, 0.99\}$ to cover a wide range of temporal horizons for the auxiliary tasks. Therefore, for each policy and each type of cumulant, $6 \times |\mathcal{O}|$ auxiliary tasks were learned in parallel where $|\mathcal{O}|$ denotes the observation size. To learn the auxiliary tasks in parallel during the pre-training phase, we used a multi-headed network with the representation learning layer being shared between the auxiliary tasks and each head corresponding to one auxiliary task. (In the case of control auxiliary tasks, multiple heads are assigned to each auxiliary task, each corresponding to one state-action value.). In the case of prediction auxiliary tasks, the behavior policy during the pre-training phase was set to the target policy of the auxiliary tasks; therefore, learning was on-policy. In the case of control auxiliary tasks, $6 \times |\mathcal{O}|$ target policies were being learned each maximizing the return corresponding to one cumulant and γ. The behavior policy, in that case, was round-robin over the $6 \times |\mathcal{O}|$ target policies, and learning was off-policy.

For the pixel-based environments, to specify the auxiliary tasks, we cropped the 40×40 observation space into a 36×36 region and subdivided it into a 6×6 grid of non-overlapping cells of size 4×4. We considered two types of cumulants: 1) Sum of the pixels in each cell 2) The absolute difference of the sum of the pixels in each cell from $t + 1$ to t. We used one value of γ equal to 0.9. Therefore, for each policy, 36 auxiliary tasks were learned in parallel for the pixel-based environments. Similar to the non pixel-based environments, for the prediction auxiliary tasks, the behavior policy during pre-training was set to the target policy of the 36 auxiliary tasks. For the control auxiliary tasks, on the other hand, the behavior policy was round-robin over the 36 target policies each corresponding to one of the auxiliary tasks.

4 Experimental setup and results

We experimented with classic control environments and pixel-based environments. For the classic control environments, we used the Mountain Car [Moore, 1991] and Acrobot [Sutton, 1995] problems. In the Mountain Car problem, an underpowered car should reach the top of a hill starting from the bottom of the hill. The observation space includes the position and velocity of the car with the position in $[-1.2, 0.6]$ and velocity in $[-0.07, 0.07]$. The action space includes three actions: 1) throttle forward 2) throttle backward 3) no throttle. The reward is -1 in all time steps and $\gamma = 1$.

Acrobot simulates a two-link underactuated robot with the goal of swinging the endpoint above the bar. The observation space includes two angles: 1) the angle between the first link and the vertical line 2) the angle between the two links. Note that in the original Acrobot environment, the observation space also includes the angular velocities corresponding to the two angles. In our experiments, however, to make the problem more challenging, we omitted the angular velocities. This makes the problem partially observable. The action space includes three actions: positive torque, negative torque, and no torque applied to the bottom joint. The reward is -1 at all steps and $\gamma = 1$.

For the pixel-based environments, we used two instances of the Minimalistic Gridworld Environment [Chevalier-Boisvert et al., 2018]: 1) Empty Minigrid 2) Door & Key Minigrid. The objective is to reach the goal cell. The observation

space is $40 \times 40 \times 3$ pixel input. The action space includes turning left, turning right, moving forward, picking an object, dropping something off, and opening the door. The reward is 0 except once the goal is reached in which case it is 1. γ is 0.99.

In the pre-training phase, for the classic control environments, we used a neural network with two feed-forward hidden layers of size 100 with ReLU nonlinearity. For Mountain Car, the position and velocity were fed into the network. For Acrobot, to deal with partial observability, we passed the angles and the action at the previous time step in addition to the angles at the current time step to the network as input. For Empty Minigrid, we used a neural network with one convolutional hidden layer and one feedforward hidden layer of size 64 with ReLU nonlinearity. For Door & Key Minigrid, we used a neural network with three convolutional hidden layers and one feedforward hidden layer of size 32 with ReLU nonlinearity. For the phase of learning the main task, we fed the representation learned during pre-training to a network with two feed-forward hidden layers of size 100.

We used learning curves to study the effect of different auxiliary tasks on performance. We first trained the representation using each of the auxiliary tasks for 500 episodes for Mountain Car, Acrobot, and Empty Minigrid, and 2000 episodes for Door & Key Minigrid. To create learning curves, we ran Q-learning with each of the learned representations 30 times (30 independent runs). To get the learning curve for Mountain Car, Acrobot, Empty Minigrid, and Door & Key Minigrid, we used 400, 200, 200, and 5000 episodes respectively.

In the case where the cumulant was equal to the observation, the representation learned for the auxiliary tasks resulted in performance gain over the baseline of no pre-training in almost all cases (Figure 2). In Mountain Car, the sticky actions policy and greedy policy resulted in the largest improvement over the baseline. In Acrobot and Empty Minigrid, the choice of the target policy did not have a substantial effect. In Door & Key Minigrid, the random policy resulted in the largest improvement over the baseline.

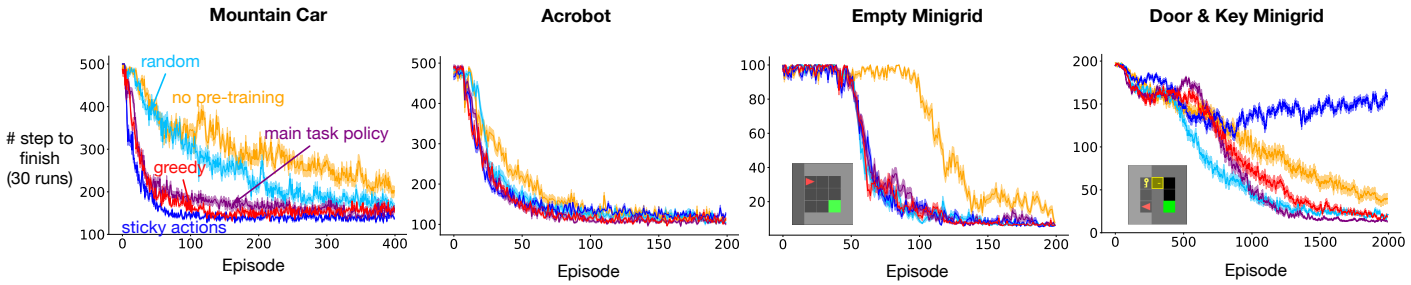


Figure 2: Learning curves for the case of cumulant equal to the observation. In the pre-training phase, the neural network was trained using the auxiliary tasks. The last hidden layer of the neural network was then fixed and used as the representation to learn the main task for 30 independent runs. In almost all cases, the representation learned for the auxiliary tasks improved over the baseline of no pre-training. In Mountain Car, the sticky actions policy and greedy policy resulted in the largest performance gain over the baseline. In Acrobot and Empty Minigrid, all auxiliary tasks resulted in similar performance. In Door & Key Minigrid, the random policy resulted in the largest performance gain.

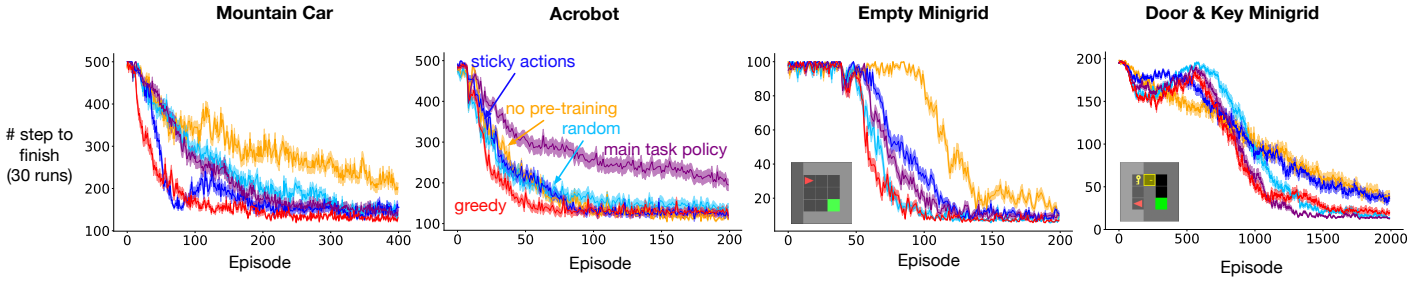


Figure 3: Learning curves for the case of cumulant equal to the observation difference. The results are for 30 independent runs. In almost all cases, the representation learned for the auxiliary tasks accelerated learning on the main task with the greedy policy resulting in the largest improvement in almost all cases.

In the case where the cumulant was equal to the observation difference, in almost all cases, the representation learned for the auxiliary tasks resulted in performance gain over the baseline of no pre-training, except for the case where the policy was the main task policy in Acrobot (Figure 3). The auxiliary task corresponding to the greedy policy resulted in the highest improvement across tasks. The results on the other policies were mixed: In the classic control environments, the sticky actions policy resulted in good performance gain whereas the main task policy was less useful or even harmful.

On the other hand, in the pixel-based environments, the main task policy resulted in good performance whereas the sticky actions policy resulted in worse performance compared to the other auxiliary tasks.

The results from both classic control and pixel-based environments suggest that the auxiliary tasks based on the greedy policy tend to be useful. Note that in the case of pixel-based environments where the cumulant is the observation difference, these auxiliary tasks are similar to the well-known pixel-control auxiliary task. Another conclusion is that most policies, even the random policy result in performance gain over the baseline. A surprising observation is that in most cases, the auxiliary tasks corresponding to the main task policy tend to fall into the group of less useful auxiliary tasks and even in some cases result in worse performance compared to the baseline.

5 Conclusions

Auxiliary tasks about different aspects of the environment can assist representation learning in reinforcement learning. The theories about what makes useful auxiliary tasks are still evolving. In this paper, we took a step toward answering this question by conducting an empirical study, investigating the role of the target policy in the utility of auxiliary tasks. Our study suggests that most policies, including a uniformly random policy, tend to improve over the baseline with the greedy policy resulting in the largest performance gain.

Further work is required to develop an understanding of how the target policy contributes to the usefulness of the auxiliary tasks. A natural future direction is to form concrete hypotheses regarding the observations that we had from our empirical results and answer questions such as: Why does the greedy policy tend to be useful? Why in some cases, the random policy results in the highest performance gain? Why in many cases, the main task policy results in lower performance gain?

References

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. *arXiv preprint arXiv:2006.02243*, 2020.

Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR, 2021.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Andrew Moore. Knowledge of knowledge and intelligent experimentation for learning control. 1991.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Exploring through Random Curiosity with General Value Functions

Aditya Ramesh[1] **Louis Kirsch**[1] **Sjoerd van Steenkiste**[1] **Jürgen Schmidhuber**[1,2,3]
[1]The Swiss AI Lab, IDSIA, University of Lugano (USI) & SUPSI, Lugano, Switzerland
[2]King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[3]NNAISENSE, Lugano, Switzerland
{aditya, louis, sjoerd, juergen}@idsia.ch

Abstract

Exploration in reinforcement learning through intrinsic rewards has previously been addressed by approaches based on state novelty or artificial curiosity. In partially observable settings where observations look alike, directly applying state novelty approaches can lead to intrinsic reward vanishing prematurely. On the other hand, curiosity-based approaches commonly require modeling precise transition dynamics which are potentially quite complex. Here we propose random curiosity with general value functions (RC-GVF), an intrinsic reward function that connects state novelty and artificial curiosity. Instead of predicting the entire environment dynamics, RC-GVF predicts temporally extended values through general value functions (GVFs) and uses the prediction error as an intrinsic reward. In this way, our approach generalizes a popular approach called random network distillation (RND) by encouraging behavioral diversity and reduces the need for additional maximum entropy regularization. Our experiments on four procedurally generated partially observable environments indicate that our approach is competitive to RND and could be beneficial in environments that require behavioural exploration.

Keywords: exploration, general value functions, curiosity, random network distillation

Acknowledgements

This research was supported by the ERC Advanced Grant (742870), the Swiss National Science Foundation grant (200021_192356), and by the Swiss National Supercomputing Centre (CSCS project s1090). We would also like to thank Kenny Young, Francesco Faccio, and Anand Gopalakrishnan for their valuable comments.

1 Introduction

Efficient exploration in reinforcement learning (RL) is a challenging problem, especially in high-dimensional observation spaces and sparse-reward environments [1, 2]. A common strategy to address this is by incorporating an *intrinsic* reward, in addition to the (sparse) extrinsic reward.

In the recent literature two main approaches have emerged: (1) approaches based on *state novelty*, where an intrinsic reward in the form of a 'novelty bonus' is awarded based on how often a state has been visited [2–4]; and (2) approaches based on *artificial curiosity*, where reward is constructed from the prediction error or information gain of a world model [5, 6].

A limitation of state novelty bonuses in partially observable settings is that observations may look alike. For example in random network distillation (RND) [4] the agent simply receives a novelty bonus based on the error in making predictions about the observation after applying a fixed randomly initialized neural network. The success of the RND novelty bonus would therefore rely on the generalization properties of a (random) neural network's mapping of observations to features. This can be especially problematic if the error (and thus the intrinsic reward) vanishes before the agent has reliably discovered the source of extrinsic rewards in the environment [7]. Similarly, curiosity-based approaches usually require modeling the complete environment dynamics, which are potentially very complex.

In this paper we explore a connection between state novelty and artificial curiosity that may address both limitations. We propose *random curiosity with general value functions* (RC-GVF), a novel approach to generating intrinsic rewards based on general value functions (GVFs) [8] and the prediction of abstract quantities about the environment [9]. We view GVFs as partially modeling the environment dynamics by predicting the temporally extended value of a quantity of interest (i.e. 'answering' a question) in the environment when following a given policy. This quantity of interest corresponds to random observation-dependent pseudo-rewards, akin to the random target features of RND. We then minimize the TD-error of these GVFs while using the error as an intrinsic reward. We argue that this is similar to the intrinsic reward derived from curiosity-based approaches, but now only modeling a random subset of the environment dynamics.

RC-GVF includes RND as a special case where the discount factor of the GVF is set to zero. Unlike in RND, where predictions about random target features may be viewed as predicting pseudo-rewards, RC-GVF takes longer horizon prediction errors into account that might be better suited for exploration. In particular, by integrating the policy as part of the prediction problem, we reward the agent for altering its behavior, reducing the need for additional exploration mechanisms such as maximum entropy regularization that is typically used in the case of RND.

We evaluate RC-GVF on a benchmark of sparse reward partially observable environments. Compared to RND we observe improvements in mean return in the absence of entropy regularization. Introspection reveals that our approach reaches states that provide external reward more frequently when learning only from intrinsic rewards in environments that benefit from behavioral diversity. In environments that require visiting most states, performance is similar to RND.

2 Preliminaries

We follow the standard POMDP formulation with time steps $t \in \mathbb{N}$, observations $o_t \in \mathcal{O}$, environment states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, extrinsic rewards $R_e(s_t)$, and policies $\pi(a_t|h_t)$ where $h_t = o_{1:t}$. The objective is to find the optimal policy π^* that maximizes the expected discounted return $J(\pi) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_e(S_k)]$, where $0 < \gamma < 1$ is the discount factor and upper case variables denote random variables.

Random Network Distillation In random network distillation (RND) [4] the agent receives a state novelty reward proportional to the error of predicting features $\hat{z}(o_t)$ generated by a randomly initialized neural network $Z_\phi : \mathcal{O} \to \mathbb{R}^d$. The RND intrinsic reward for an observation is given by

$$R_i(o_t) = \| Z_\phi(o_t) - \hat{z}(o_t) \|_2. \tag{1}$$

Entropy Regularization To encourage additional exploration, many RL algorithms employ entropy regularization. The maximum entropy objective [10] adjusts the RL objective to $J_{\text{MaxEnt}}(\pi) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_e(S_k) + \alpha \mathcal{H}(\pi(\cdot|H_k))]$, where $\alpha \in \mathbb{R}_+$ is a hyper-parameter that trades off rewards and entropy regularization. RND also typically employs this regularization, despite introducing its own exploration mechanism.

General Value Functions A general value function (GVF) [8] is defined by a policy π, a cumulant or pseudo-reward function $Z : \mathcal{O} \to \mathbb{R}$, and a discount factor γ_z. It can be expressed as $v_{\pi,z}(o) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma_z^k Z(O_{t+k}) | O_t = o\right]$. General value functions extend the concept of predicting expected cumulative values to arbitrary signals beyond the reward. They can be viewed as answers to questions about such quantities under a particular policy.

3 Random Curiosity with General Value Functions

This section describes *random curiosity with general value functions* (RC-GVF), a novel approach to intrinsic rewards based on GVFs of random pseudo rewards.

Random Pseudo-Rewards and GVFs Similar to artificial curiosity [5], in RC-GVF we model the environment dynamics. However, instead of modeling precise transition dynamics, we ask questions about future outcomes under a policy π [9]. In this paper, these questions are random and represented by a randomly initialized neural network Z_ϕ that maps observations to features $Z_\phi : \mathcal{O} \to \mathbb{R}^d$, here referred to as pseudo-rewards. At time step t, the current observation o_t is mapped to the pseudo-rewards $z_{t+1} \in \mathbb{R}^d$. To capture the outcome of a question across time, we are interested in the discounted pseudo-return G_t^z for a sequence of pseudo-reward random variables $Z_{t+1}, Z_{t+2} \dots$ given by $G_t^z = \sum_{k=0}^{\infty} \gamma_z^k \cdot Z_{t+k+1}$, where γ_z is a scalar discount factor. Pseudo-rewards and returns are random variables due to the stochasticity in the policy and/or environment dynamics. Similarly, the pseudo-value(s) of an observation under the policy π is given by $v_{\pi,z}(o) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma_z^k Z_\phi(O_{t+k}) \mid O_t = o \right]$. This value function is known as a *general value function* [8].

TD-Error as Intrinsic Curiosity Reward Our exploration mechanism rewards the agent for taking actions that generate unknown outcomes under fixed random questions. To that end, we train a separate (recurrent) neural network, which we call the *predictor*, to predict these pseudo-values. Concretely, the predictor $\hat{v}_{\pi,z} : \mathcal{H} \to \mathbb{R}^d$ maps histories of observations $\mathcal{H}$ to values. We focus on a predictor that is trained on-policy. One motivating factor for this is that it couples the prediction task to the current policy, which creates an incentive to vary the policy for additional exploration. As a target we use the (truncated) λ-return, which can be recursively expressed as $G_t^z(\lambda_z) = Z_{t+1} + \gamma_z(1-\lambda_z)\hat{v}_{\pi,z}(H_{t+1}) + \gamma_z\lambda_z G_{t+1}^z(\lambda_z)$. Here, $\lambda_z \in [0,1]$ is the usual parameter that allows us to balance the bias-variance trade off by interpolating between TD(0) and Monte Carlo estimates of the pseudo-return [11].

The intrinsic reward of RC-GVF at time step t is then defined as the error between the output of the predictor and the truncated λ-pseudo-return

$$R_i(o_t) = \| G_t^z(\lambda_z) - \hat{v}_{\pi,z}(h_t) \|_2, \tag{2}$$

where $h_t = o_{1:t}$. Using $G_t^z(\lambda_z)$ as the target links the prediction task and the intrinsic reward to the agent's policy, thereby encouraging *behavioral exploration*.

Effective Horizon The effective horizon over which predictions are considered depends on the choice of the discount factor γ_z. We obtain an intrinsic reward matching RND (Equation 1) for the special case of $\gamma_z = 0$, which yields $G_t^z = Z_{t+1}$. The larger γ_z, the more pronounced is the contribution of the current policy to the prediction problem.

4 Experiments

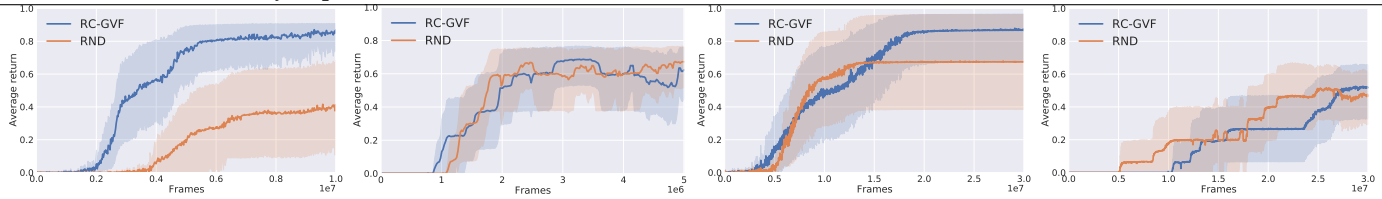
We compare RC-GVF and RND on four procedurally generated environments from MiniGrid [12]: KeyCorridor-S3R3, ObstructedMaze-2Dl, MultiRoom-N7-S8, and MultiRoom-N10-S4. Exploration in these environments is particularly challenging due to partial observability, sparse rewards, and the procedural generation of mazes and objects.

To avoid confounding factors from combining different exploration strategies, we mainly focus on the exploration behaviour in the absence of entropy regularization. For comparison with previous works, we also include an analysis with entropy regularization. Moreover, to study the benefit of RC-GVF in isolation, we also consider a setting where the agent only receives intrinsic rewards to guide its behaviour. Finally, we conduct an experiment to study the influence of the RC-GVF discount factor (γ_z) on the agent's performance.

Implementation We use Proximal Policy Optimization (PPO) [13] as our base agent. This agent is trained to maximize the expected sum of a weighted combination of intrinsic and extrinsic rewards. At each time step t, the agent receives a reward $R_t = R_e(s_t) + \beta R_i(o_t)$. Here $R_e(s_t)$ is the extrinsic reward from the environment, $R_i(o_t)$ is the intrinsic reward from either RND or RC-GVF, and $\beta \in \mathbb{R}_+$ is a hyperparameter to balance the weighting of intrinsic and extrinsic rewards.

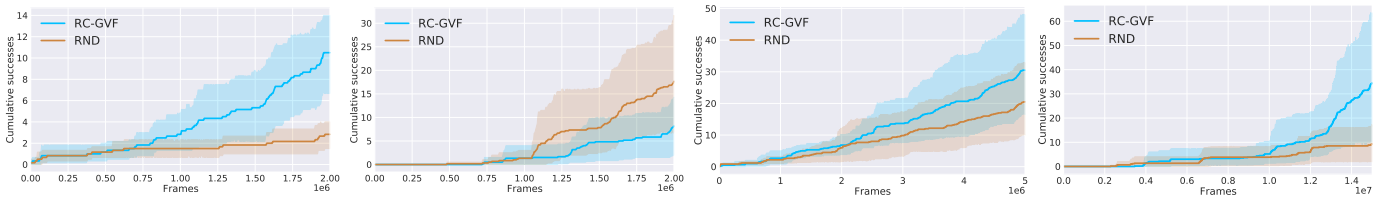
The agent consists of an actor and a critic, both share convolution layers followed by an LSTM [14], with separate multi-layer perceptron (MLP) heads for the actor and critic. The pseudo-reward generator is implemented as a convolutional neural network whose output is flattened to a vector. As per the original implementation of RND [4], the predictor has the same architecture as that of the pseudo-reward generator. In the case of RC-GVF, the predictor is recurrent, consisting of three convolutional layers followed by an LSTM with a linear output layer.

Evaluation We present the results averaged over 10 independent runs for each problem. Unless mentioned otherwise, all figures report the mean as a solid line and the shading indicates 95% bootstrapped confidence intervals for the 10 seeds.



(a) KeyCorridor-S3R3 (b) MultiRoom-N10-S4 (c) ObstructedMaze-2Dl (d) MultiRoom-N7-S8

Figure 1: Average return of RC-GVF and RND on the selected Minigrid environments (with no entropy regularization).



(a) KeyCorridor-S3R3 (b) MultiRoom-N10-S4 (c) ObstructedMaze-2Dl (d) MultiRoom-N7-S8

Figure 2: Cumulative number of times the agent successfully solves the environment, while only receiving intrinsic rewards. Shading indicates 95% bootstrapped confidence intervals over 6 seeds.

Results Figure 1 presents our results on the MiniGrid environments when no entropy regularization is used. We observe that RC-GVF appears more sample efficient than RND on the KeyCorridor environment. In the ObstructedMaze environment we see that our approach is more stable across independent runs. Both approaches are comparable and slightly unstable in the Multi-Room environments (N10-S4 and N7-S8). We note that in a previous empirical comparison with IMPALA [15] instead of PPO as the base agent (i.e. as in this paper), RND was not able to reach any extrinsic reward states on these MultiRoom tasks [7].

One explanation for the findings in Figure 1 could be that the tasks in the Multi-Room environments are inherently suited to RND's intrinsic reward of covering as many states as possible. The extrinsic reward in these environments requires visiting most states. Similarly, the better performance of RC-GVF in KeyCorridor and ObstructedMaze could be due to a greater need to try out *different* behaviours (finding the key, unlocking doors, picking up objects). Indeed, this variation in behaviors is encouraged by the policy dependence in RC-GVF.

Figure 2 presents an analysis of the number of episodes in which the agent successfully completes a task (i.e. it receives any extrinsic reward). To measure the exploration effect independent of extrinsic reward maximization, in this setting the agent receives only intrinsic rewards to guide its behaviour. It can be observed how RC-GVF completes the task more often on the KeyCorridor and ObstructedMaze environments, again indicating that RC-GVF could be better suited to explore the kinds of environments which benefit from policy diversity and higher entropy.

In the MultiRoom environments, we observe mixed results. It can be seen how an agent trained solely with the RND bonus reaches the goal state more frequently in the N10-S4 environment as compared to an agent trained with our bonus. We believe that the reason behind the reduced performance of RC-GVF is tied to the exploration mechanism driven by altering the policy. In these environments the goal state (with non-zero rewards) has a visually unique appearance that potentially produces large intrinsic rewards in the case of RND, whereas RC-GVF is also incentivized by *behavioral* exploration, reducing the tendency to follow the precise policy to revisit that state.

Additional Analysis We carry out an experiment to study the influence of the GVF discount factor γ_z on RC-GVF's performance in the KeyCorridor-S3R3 environment. Here γ_z can be viewed as interpolating between RND ($\gamma_z = 0$) and variations of RC-GVF with increasing emphasis on longer horizon predictions. We observe that the lower discount factors $\gamma_z \in \{0.007, 0.07, 0.25\}$ generally perform similarly to RND (shown as $\gamma_z = 0$), with only $\gamma_z = 0.007$ performing slightly better than expected (Figure 3). Increasing the GVF discount factor–and therefore the horizon considered for the value function predictions–appears to have a positive impact on the agent's performance in this setting.

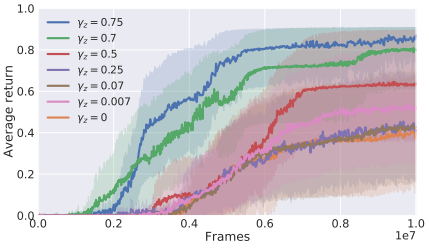


Figure 3: Performance of RC-GVF with different values of γ_z on the KeyCorridor-S3R3 environment (no entropy regularization).

In the previous experiments we have focused on settings without entropy regularization, to avoid confounding multiple exploration mechanisms. In Figure 4, we see that the performance of both approaches improves with the introduction of the entropy term. RND's improvement is more pronounced in comparison to RC-GVF. This suggests that RC-GVF requires less entropy regularization due to promoted behavioral diversity.

5 Conclusion

We developed RC-GVF, an intrinsic reward approach to exploration inspired by ideas from state novelty bonuses and artificial curiosity. Based on general value functions, it derives intrinsic rewards from the long term prediction error of random questions under the current policy. The discount factor allows us to control the horizon over which predictions are considered. Our approach includes RND as a special case when the discount factor is zero. Our experiments on four procedurally generated partially observable environments indicate that our approach could be beneficial in environments that require behavioural exploration.

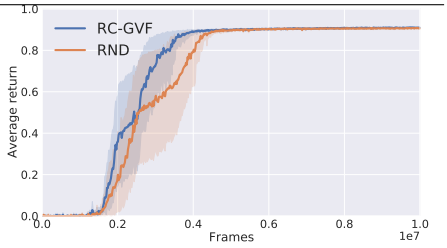


Figure 4: With entropy regularization, both methods improve and are comparable in KeyCorridor-S3R3.

While the incorporation of the current policy into the prediction task can have benefits in behavioral exploration, it can also lead to over-exploration. An off-policy or policy-conditioned [16] version of RC-GVF may mitigate this. Another potential improvement can come from a distributional perspective to obtain intrinsic rewards. This may be important as RC-GVF does not account for the inherent variance in the pseudo-return even for a fixed policy. Furthermore, in comparison to RND, where the predictor belongs to same model class as the pseudo-reward generator, we need to select a predictor of appropriate complexity.

In the future, we aim to compare RC-GVF with transition dynamics based curiosity approaches [17]. Further generalizations are possible; such as moving beyond random pseudo-rewards, general value functions under different policies, and introducing time or state dependent discounting.

References

[1] Sebastian B Thrun. Efficient exploration in reinforcement learning. 1992.

[2] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 2016.

[3] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[4] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations*, 2019.

[5] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

[6] Jan Storck, Sepp Hochreiter, Jürgen Schmidhuber, et al. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the international conference on artificial neural networks, Paris*, 1995.

[7] Roberta Raileanu and Tim Rocktäschel. RIDE: rewarding impact-driven exploration for procedurally-generated environments. In *8th International Conference on Learning Representations*, 2020.

[8] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[9] J. Schmidhuber. What's interesting? Technical Report IDSIA-35-97, IDSIA, 1997. ftp://ftp.idsia.ch/pub/juergen/interest.ps.gz; extended abstract in Proc. Snowbird'98, Utah, 1998;

[10] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[11] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[12] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

[16] Francesco Faccio, Louis Kirsch, and Jürgen Schmidhuber. Parameter-based value functions. In *9th International Conference on Learning Representations*, 2021.

[17] Yuri Burda, Harrison Edwards, Deepak Pathak, Amos J. Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *7th International Conference on Learning Representations*, 2019.

Hamiltonian Model Based Reinforcement Learning For Robotics

Adithya Ramesh
Robert Bosch Centre for Data Science and Artificial Intelligence,
Indian Institute of Technology Madras
Chennai, India
adithya.ramesh.1993@gmail.com

Balaraman Ravindran
Department of Computer Science and Engineering,
and
Robert Bosch Centre for Data Science and Artificial Intelligence,
Indian Institute of Technology Madras
Chennai, India
ravi@cse.iitm.ac.in

Abstract

Reinforcement Learning (RL) has a lot of potential in the robotics domain. However there remain some critical challenges that need to be overcome for us to see more successful deployments of RL based robotic systems in the real world. In this study we try to address one of these challenges, namely, sample efficiency, through the model-based RL approach. We learn a model of the environment, essentially its transition dynamics and reward function, and use it to generate imaginary trajectories which are then used to update the policy and value functions. Intuitively, the quality of the learnt policy will depend on the quality of the imaginary trajectories, which in turn will depend on the quality of the learnt environment model. Thus, learning better environment models should lead to better policies. In this study we investigate if this is true.

Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems, by utilizing the structure of the underlying physics. These studies however, have only focused on dynamics learning. In this study, we investigate if such structured models can also improve model-based RL. We train two versions of our model-based RL algorithm, a standard version and a physics-informed version. In addition, we also train a state-of-the-art model-free RL algorithm, Soft-Actor-Critic, to serve as a baseline. Our results show that physics-informed model-based RL outperforms standard model-based RL across environments in terms of dynamics learning, policy learning as well as sample efficiency. In simple environments, we find that physics-informed model-based RL, standard model-based RL and state-of-the-art model-free RL achieve similar average-return. In complex environments, we find that physics-informed model-based RL achieves significantly higher average-return compared to standard model-based RL and state-of-the-art model-free RL.

Keywords: Model-Based Reinforcement Learning, Robotics,
Physics-Informed Neural Networks, Hamiltonian Mechanics

1 Introduction

Reinforcement Learning (RL) can learn to solve sequential decision making problems through trial and error. In recent years, RL has been combined with powerful function approximators such as deep neural networks to successfully solve complex problems with high-dimensional, continuous state and action spaces across a range of domains, from video games [1] to robotics [2].

We are particularly interested in the application of RL to robotics. RL has a lot of potential in the robotics domain. Many tasks that are difficult to solve using traditional approaches, can be solved more easily using RL. However there remain some critical challenges that need to be overcome for us to see more successful deployments of RL based robotic systems in the real world. Some of these challenges include sample efficiency, safety, stability, explainability and generalization. In this study we focus on the problem of sample efficiency.

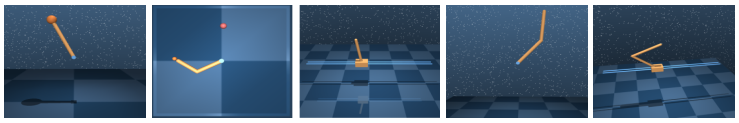
One of the drawbacks of traditional RL algorithms has been their poor sample efficiency – they require a large number of interactions with the environment to learn successful policies. In robotics, collecting such large amounts of training data using actual robots is not practical as the process would be extremely slow. Also, there would be significant wear and tear of parts over time, requiring frequent replacement. Hence it is preferred to train in simulation first and then deploy on actual robots. Usually, it is hard to capture all the salient aspects of the environment in a simulation, this is also known as sim-to-real mismatch. Hence it becomes necessary to fine tune the agent using data from the actual robot. In general, whether we are learning using simulations or actual robots, it is desirable to improve sample efficiency.

One approach to improving sample efficiency of RL algorithms is model-based RL, where we learn a model of the environment, essentially its transition dynamics and reward function, and use it to generate imaginary trajectories which are then used to update the policy and value functions. While most model-based RL approaches use the model simply to generate additional data, we make more effective use of the model by exploiting its differentiability. We efficiently learn behaviors by propagating gradients of learned state values back through the imagined trajectories. Intuitively, the quality of the learnt policy will depend on the quality of the imaginary trajectories, which in turn will depend on the quality of the learnt environment model. Thus, learning better environment models should lead to better policies. In this study we investigate if this is true.

Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems [3]. Consider for example, a simple pendulum swinging under just the force of gravity. In the absence of friction, it would keep swinging forever. Greydanus et al [4] learnt the dynamics of such an ideal pendulum using a standard deep neural network. They found that, given the current state, the network is able to predict the next state quite accurately. However when they tried to predict longer trajectories, they found that the network's predictions become progressively worse. When they estimated the total energy of the pendulum along the predicted trajectory, they found that it does not remain constant. Conservation of energy is a fundamental law of physics. In the case of the pendulum, it is a fundamental aspect of its dynamics, which the neural network has failed to learn. This is a simple example that motivates the need for better deep neural network based dynamics models for physical systems.

In general, we can learn better deep neural network based dynamics models for physical systems by utilizing the structure of the underlying physics, i. e., through better inductive biases. Zhong et al [3] summarizes and benchmarks the recent work in this area. These studies however, have only focused on dynamics learning. In this study, we investigate if such structured models, which are known to learn the dynamics better, can also improve model-based RL.

We focus on the robotics domain, in particular, systems undergoing pure rigid body motion. We consider the following environments,



(a) Pendulum (b) Reacher (c) Cartpole (d) Acrobot (e) Cart-2-pole

Figure 1: Environments considered

- Pendulum : Swing up and balance a pendulum. Gravity is present.
- Reacher : Control a two-link manipulator in the horizontal plane (no gravity), to reach a fixed target location. Both joints are actuated.
- Cartpole : Swing up and balance an unactuated pole by applying forces to a cart at its base. Gravity is present.
- Acrobot : Control a two-link manipulator in the vertical plane (gravity present), to swing up and balance. Only the second joint is actuated.
- Cart-2-pole : One extra pole is added to Cartpole.

We assume that there is no friction and that low-dimensional state information is available to the RL agent.

2 Methodology

2.1 Classical Mechanics

Classical mechanics is a branch of physics which studies the motion of macroscopic objects. The most familiar formulation of classical mechanics is Newtonian mechanics. Alternative formulations include Lagrangian mechanics and Hamiltonian mechanics. For complex mechanical systems such as robots, it is generally much easier to derive the equations of motion using Lagrangian or Hamiltonian mechanics rather than Newtonian mechanics.

In Lagrangian mechanics, the system's state is completely described by its position $\mathbf{q}$ and velocity $\dot{\mathbf{q}}$. The Lagrangian L is a scalar quantity defined as $L(\mathbf{q}, \dot{\mathbf{q}}, t) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q})$, where T is the kinetic energy and V is the potential energy. Assume that the external non-conservative force (e. g., motor torques) acting on the system is $\boldsymbol{\tau}$. The Lagrangian equation of motion is given by,

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}} - \frac{\partial L}{\partial \mathbf{q}} = \boldsymbol{\tau} \tag{1}$$

Note that the Lagrangian equation of motion is a single second-order differential equation. In Hamiltonian mechanics, the system's state is completely described by its position $\mathbf{q}$ and momentum $\mathbf{p}$. The Hamiltonian H is a scalar quantity, which for many physical systems is simply the total energy of the system, i. e., $H(\mathbf{q}, \mathbf{p}, t) = T(\mathbf{q}, \mathbf{p}) + V(\mathbf{q})$. The Hamiltonian equations of motion are given by,

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \;\; ; \;\; \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}} + \boldsymbol{\tau} \tag{2}$$

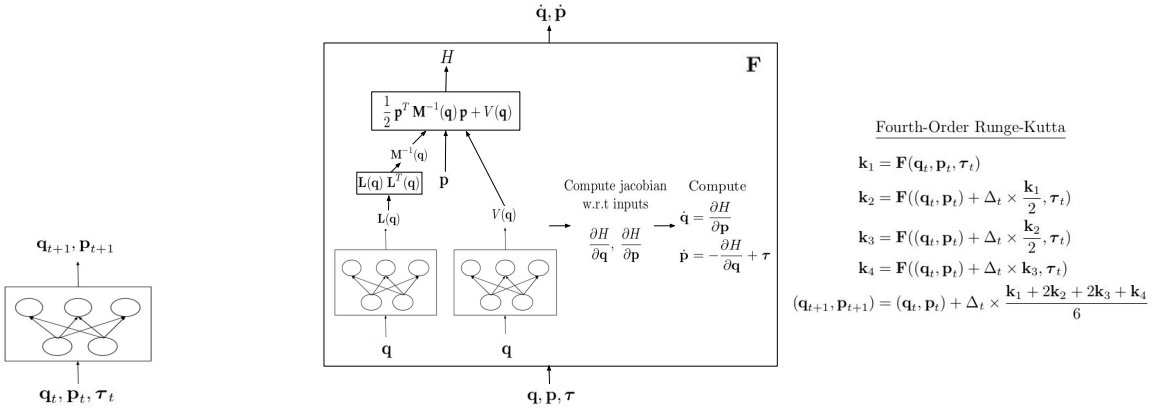
Note that the Hamiltonian equations of motion are two first-order differential equations. Conventionally, when simulating robotic systems, Lagrangian mechanics is used. We instead use Hamiltonian mechanics, because it is easier to learn first-order differential equations using deep neural networks compared to second-order differential equations. Common robotic simulators such as Mujoco [5] internally use Lagrangian mechanics and do not off-the-shelf provide the necessary data to study Hamiltonian mechanics. Hence we implement our own simulation.

2.2 Dynamics Learning

By learning dynamics, we essentially want to learn the transformation $(\mathbf{q}_t, \mathbf{p}_t, \boldsymbol{\tau}_t) \rightarrow (\mathbf{q}_{t+1}, \mathbf{p}_{t+1})$. The most straightforward solution is to train a standard deep neural network. We refer to this approach as DNN. This is shown in figure 2(a). Another approach is to utilize the structure of the underlying Hamiltonian mechanics. This approach builds upon recent work such as Hamiltonian Neural Networks [4] and Symplectic ODE-Net [6]. We detail our approach here. For systems undergoing rigid body motion, the Hamiltonian H can be written as

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\,\mathbf{p}^T\,\mathbf{M}^{-1}(\mathbf{q})\,\mathbf{p} + V(\mathbf{q}) \tag{3}$$

where $\mathbf{M}$ is the mass matrix. In this approach, one network learns the potential energy function $V(\mathbf{q})$ and another network learns a lower triangular matrix $\mathbf{L}(\mathbf{q})$, which we use to compute the inverse of the mass matrix as $\mathbf{M}^{-1}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\,\mathbf{L}^T(\mathbf{q})$, using the fact that the mass matrix and its inverse are symmetric and positive definite. Then, using equation (3) we compute the Hamiltonian H. Substituting H into equation (2), we compute the time derivative of the state. We then numerically integrate the time derivative of the state using a standard numerical integration technique, fourth-order Runge-Kutta, to compute the next state $(\mathbf{q}_{t+1}, \mathbf{p}_{t+1})$. We refer to this approach as HNN. The entire process is shown in figure 2(b).



(a) DNN (b) HNN

Figure 2: Dynamics Learning

2.3 Model-Based RL

While most model-based RL approaches use the model simply to generate additional data, we make more effective use of the model by exploiting its differentiability. We efficiently learn behaviors by propagating gradients of learned state values back through the imagined trajectories. We summarize our model-based RL algorithm below. It builds on the recent Dreamer algorithm [7].

Algorithm 1 MBRL Algorithm

Initialize networks with random weights.
Execute random actions for S episodes to initialize replay buffer.
for each episode **do**
  // **Environment Model Learning**
  for N_1 times **do**
    Draw mini batch $\{(o^i, a^i, r^i, o^{i+1}) \ : \ i = 1, ..., B_1\}$ from replay buffer.
    Fit transition dynamics and reward models.
  end for
  // **Behaviour Learning**
  for N_2 times **do**
    Draw mini batch $\{(o^j) \ : \ j = 1, ..., B_2\}$ from replay buffer.
    for each j **do**
      Imagine trajectory of length H starting from o^j.
    end for
    Actor Loss $= -\frac{1}{B_2} \sum_{j=1}^{B_2} \sum_{t=0}^{H-1} [\, V_\lambda^{target}(o_t^j) - \alpha \log \pi(a_t^j | o_t^j) \,]$
    Critic Loss $= \frac{1}{B_2} \sum_{j=1}^{B_2} \sum_{t=0}^{H-1} \frac{1}{2} \| \, V(o_t^j) - V_\lambda^{target}(o_t^j) \, \|^2$
    Update Actor, Critic.
    Update entropy weightage α, in a manner similar to Soft-Actor-Critic.
    Every 100 updates, $V^{target} \leftarrow V$
  end for
  // **Environment Interaction**
  Interact with environment for 1 episode using current policy. Add experience to replay buffer.
end for
Note : $S = 5$, $N_1 = 10^4$, $B_1 = 64$, $N_2 = 10^3$, $B_2 = 64$, $H = 16$

We train two versions of our model-based RL algorithm, one which uses the DNN approach for dynamics learning and the other which uses the HNN approach. In addition, we also train a state-of-the-art model-free RL algorithm, Soft-Actor-Critic (SAC) [8], to serve as a baseline.

3 Results

Environment	Method	Steps	Dynamics L1 Error ($\times 10^{-5}$)	Average Return (G)	Lyapunov Exponent	$\frac{G_{MBRL\text{-}HNN} - G_{MBRL\text{-}DNN}}{\lvert G_{MBRL\text{-}DNN} \rvert}$ ($\times 100$)	$\frac{G_{MBRL\text{-}HNN} - G_{SAC}}{\lvert G_{SAC} \rvert}$ ($\times 100$)
Pendulum	MBRL-HNN	0.25 M	3.7115	886	0.0372	0.63	0.09
	SAC	2.5 M	-	885.2			
	MBRL-DNN	0.5 M	200.62	880.5			
Reacher	MBRL-HNN	0.5 M	2.737	-120.8	0.0459	10.32	0.33
	SAC	5 M	-	-121.2			
	MBRL-DNN	1 M	126.08	-134.7			
Cartpole	MBRL-HNN	0.5 M	28.846	884.2	0.0725	12.21	0.48
	SAC	5 M	-	879.8			
	MBRL-DNN	1 M	2354	788			
Acrobot	MBRL-HNN	1 M	3.5471	-240.6	0.9208	69.2	39.27
	SAC	10 M	-	-396.2			
	MBRL-DNN	2 M	186.62	-781.1			
Cart-2-pole	MBRL-HNN	1 M	5.0984	783.4	1.0910	166.19	91.17
	SAC	10 M	-	409.8			
	MBRL-DNN	2 M	1432	294.3			

Table 1: Results

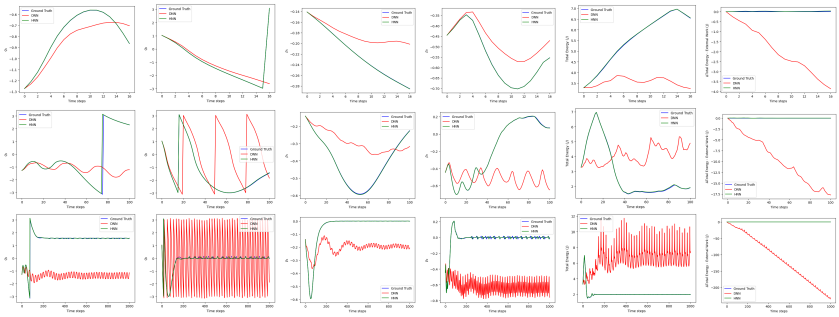


Figure 3: Imaginary trajectory vs ground truth, for Acrobot environment, for different imagination horizons H. Row 1 : $H = 16$, Row 2 : $H = 100$, Row 3 : $H = 1000$.

From table 1, we see that HNN is able to learn better dynamics models than DNN in terms of L1 error, across environments. We find that the imaginary trajectories of HNN show minimal deviation from the ground truth, while the imaginary trajectories of DNN show larger deviation. This is shown in figure 3, for the case of Acrobot. A similar trend is seen for other environments. Although during behaviour learning we use an imagination horizon of only 16 time steps, in figure 3, we plot imaginary trajectories for longer horizons as well, to see how far into the future our models can predict accurately.

From table 1, we see that MBRL-HNN achieves higher average-return compared to MBRL-DNN across environments, while requiring fewer samples. We find that the relative difference between the average-return of MBRL-HNN and MBRL-DNN is directly correlated to the Lyapunov exponent [9] of the environment. We try to explain why this is so. Lyapunov exponents measure the exponential rate of separation of trajectories which start from nearby initial states. Thus, they represent how quickly numerical errors will accumulate. Consider the dynamics model learnt by DNN. Its one-step prediction will have some error. Now consider an imaginary trajectory of horizon 16 time steps produced by DNN. If the environment has a large Lyapunov exponent, the prediction error will accumulate fast and the resulting trajectory will deviate significantly from the ground truth and we cannot learn a good policy from such trajectories. Whereas, if the environment has a small Lyapunov exponent, the prediction error will accumulate more slowly and the resulting trajectory will not deviate from the ground truth significantly and we can learn a reasonably good policy from such trajectories. Thus, the relative difference between the average-return of MBRL-HNN and MBRL-DNN is small when the Lyapunov exponent of the environment is small, and large when the Lyapunov exponent is large.

In simple environments, the average-return of MBRL-HNN and SAC are comparable. In complex environments, the average-return of MBRL-HNN is significantly higher than SAC.

4 Conclusion

We believe that model-based RL has an important role to play in unlocking the vast potential that RL has in the robotics domain. Our results show that physics-informed model-based RL outperforms standard model-based RL across environments in terms of dynamics learning, policy learning as well as sample efficiency. In simple environments, we find that physics-informed model-based RL, standard model-based RL and state-of-the-art model-free RL achieve similar average-return. In complex environments, we find that physics-informed model-based RL achieves significantly higher average-return compared to standard model-based RL and state-of-the-art model-free RL. In the future, we plan to extend our work to robotic systems involving contact dynamics and dissipation, and also attempt to learn directly from images.

References

[1] Mnih et al. Playing atari with deep reinforcement learning, 2013.

[2] Andrychowicz et al. Learning dexterous in-hand manipulation, 2019.

[3] Zhong et al. Benchmarking energy-conserving neural networks for learning dynamics from data, 2022.

[4] Greydanus et al. Hamiltonian neural networks, 2019.

[5] Todorov et al. Mujoco: A physics engine for model-based control. 2012.

[6] Zhong et al. Symplectic ode-net: Learning hamiltonian dynamics with control, 2020.

[7] Hafner et al. Mastering atari with discrete world models, 2022.

[8] Haarnoja et al. Soft actor-critic: Off-policy maximum entropy deep RL with a stochastic actor. 2018.

[9] Sandri. Numerical calculation of lyapunov exponents.5 1996.

Rainbow RBF-DQN

Sreehari Rammohan
Department of Computer Science
Brown University
sreehari@brown.edu

Bowen He
Department of Computer Science
Brown University
bowen_he@brown.edu

Shangqun Yu
Department of Computer Science
Brown University
shangqun_yu@brown.edu

Eric Hsiung
Department of Computer Science
Brown University
eric_hsiung@brown.edu

Eric Rosen
Department of Computer Science
Brown University
eric_rosen@brown.edu

George Konidaris
Department of Computer Science
Brown University
gdk@cs.brown.edu

Abstract

Deep reinforcement learning has been extensively studied, resulting in several extensions to DQN that improve its performance, such as replay buffer sampling strategies, distributional value representations, and double/dueling networks. Previous works have examined these extensions in the context of either discrete action spaces or in conjunction with actor-critic learning algorithms, but there has been no investigation of combining them for deep value-based continuous control. We adapted the methods discussed in Rainbow DQN to RBF-DQN, a deep value-based method for continuous control, showing improvements when evaluated over a set of 7 OpenAI Gym continuous control tasks in baseline performance and sample efficiency. Rainbow RBF-DQN outperforms both vanilla RBF-DQN and state-of-the-art actor-critic methods on the most challenging tasks such as Half-Cheetah and Humanoid.

Keywords: RBF-DQN, Value Based, Continuous Control, Rainbow

Acknowledgements

Part of this research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University.

1 Introduction

The introduction of RBF-DQN [Asadi et al., 2020] was a key neural network architectural breakthrough that made deep value based RL perform competitively on continuous action domains. In this paper we select four of the improvements to DQN—Distributional Representations [Dabney et al., 2017], Dueling Networks [Wang et al., 2015], Double Networks [van Hasselt et al., 2015], and Priority Experience Replay [Schaul et al., 2015]—and modify them to work in the continuous action space with RBF-DQN. The resulting agent, termed Rainbow RBF-DQN, overall demonstrates more stable learning and lower sample complexity on 7 OpenAI MuJoCo continuous control tasks: Humanoid, Ant, Half-Cheetah, Hopper, Bipedal Walker, Lunar Lander, and Pendulum.

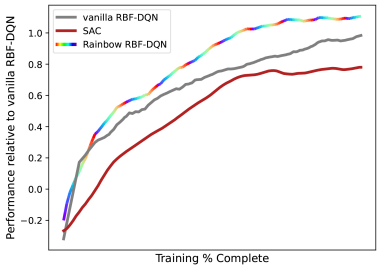


Figure 1: Rainbow RBF-DQN and SAC performance relative to vanilla RBF-DQN across the 7 MuJoCo control tasks.

2 Background

Deep value-based methods use deep learning to learn a set of parameters θ to approximate the state-action value function $Q_\theta(s,a)$. The vanilla Q-Learning objective is to minimize the TD-Error (also known as the Bellman error), which is defined for a single transition (s,a,r,s') as $r+\gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)$. Q-Learning is off-policy and therefore allows the agent to use a behavioral policy (such as epsilon greedy) to approximate a target policy (the optimal policy). Mnih et al. [2013] introduced DQN, allowing deep neural networks to play Atari using the TD-Error objective, a replay buffer to store previously experienced transitions, and a target network for stabilizing the TD target between learning updates. Note that in Q-Learning, the update rule relies on finding $\max_{a' \in \mathcal{A}} \hat{Q}(s',a';\theta)$. This is prohibitively expensive in continuous action spaces, due to an infinite search space, and approximations like discretizing the action space may produce sub-optimal solutions.

RBF-DQN [Asadi et al., 2020] is a value-based deep reinforcement learning approach that uses radial-basis functions to approximate the Q function for continuous action space tasks. RBF-DQN learns both the radial-basis centroids (which represent sampled actions) and their values, and uses a kernel to efficiently compute the action that maximizes the Q-value with a bounded error that depends on the temperature parameter β. Specifically, RBF-DQN approximates $Q^*(s,a)$ by learning centroid locations $a_i(s;\theta)$ and centroid values $v_i(s;\theta)$ as functions of state s and parameters θ and β according to:

$$\hat{Q}_\beta(s,a;\theta) := \frac{\sum_{i=1}^{N} e^{-\beta \|a - a_i(s;\theta)\|} v_i(s;\theta)}{\sum_{i=1}^{N} e^{-\beta \|a - a_i(s;\theta)\|}}. \tag{1}$$

The centroid locations $a_i(s;\theta)$ and state-dependent centroid values $v_i(s;\theta)$ are used to form the Q function output [Asadi et al., 2020], and are learned end-to-end during training by optimizing the Bellman error similar to the DQN loss [Mnih et al., 2013]. The action maximization property of RBF-DQN [Asadi et al., 2020] guarantees all critical points of $\hat{Q}_\beta$ can be well-approximated by a centroid location a_i. This makes action-maximization as simple as searching over all N centroids $\max_{i \in [1,N]} \hat{Q}_\beta(s,a_i;\theta)$ where a_i represents a centroid location. In multi-dimensional action spaces, the temperature parameter β can be tuned to ensure an upper bound on error for the optimal action [Asadi et al., 2020]:

$$\max_{a \in \mathcal{A}} \hat{Q}_\beta(s,a;\theta) - \max_{i \in [1,N]} \hat{Q}_\beta(s,a;\theta) \leq \mathcal{O}(e^{-\beta}). \tag{2}$$

The action-maximization property of Deep RBFs, paired with its universal function approximating properties, make RBF-DQN well-suited for continuous control tasks.

3 Extensions to RBF-DQN

Many of the recent successes in deep reinforcement learning have come from improvements to the vanilla Q-Learning approach. In this work, we specifically investigate a set of popular extensions to DQN that are included in Rainbow DQN [Hessel et al., 2018], which combines double Q-Learning [van Hasselt et al., 2015], prioritized experience replay [Schaul et al., 2015], and distributional RL [Bellemare et al., 2017]. In this section, we present new augmentations to accommodate applying Double Q-Learning, and Distributional Q-Learning to RBF-DQN. All other Rainbow augmentations (PER, Noisy Networks, and Multi-step Learning) are directly applicable without modification. [1]

[1] https://github.com/xdADq73Tur/rainbow_RBFDQN. Our Rainbow RBF-DQN implementation.

3.1 Double Q-Learning

To adapt double Q-Learning networks for RBF-DQN, the optimal action is selected using the online centroid and centroid value modules, parameterized by θ, and then a Q-value is produced using the centroid and centroid values from the target network, parameterized by θ^-, respectively. We implement double RBF-DQN according to

$$Q(s', a^*; \theta^-) = \frac{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||} v_i(s';\theta^-)}{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||}} \tag{3}$$

where $a^* = {}_{a_i} Q(s', a_i; \theta)$ is chosen according to the online network. The action a^* is the centroid location associated with the highest Q value according to Equation 1.

3.2 Distributional Q-Learning

In distributional Q-learning, rather than learning Q values in expectation, we learn the complete value distribution for a given state-action pair.

Applying this to RBF-DQN, the network's centroid value module is modified to instead output the quantile distribution for each centroid, denoted by the matrix $\mathbf{Z} \in R^{N_C \times N_Q}$ values, where N_C is the number of centroids and N_Q is the number of quantiles. Taking the expectation of the supports for the ith centroid results in the value v_i for the ith centroid, and can be summarized by $\mathbf{v}^\top = \mathbf{w}^\top \mathbf{Z}$, where $\mathbf{v} \in R^{N_Q}$, a is the proposed action and a_i is the location of the ith centroid, and the components w_i of the vector $\mathbf{w} \in R^{N_C}$ represent the normalized distances from the proposed action a to each centroid location a_i:

$$w_i = \left(\frac{\|a_i - a\|_2}{\sum_{i=1}^{N_C} \|a_i - a\|_2} \right) \tag{4}$$

$$[w_1 \quad w_2 \quad ... \quad w_{N_C}] \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1N_Q} \\ z_{21} & z_{22} & \cdots & z_{2N_Q} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N_C 1} & z_{N_C 2} & \cdots & z_{N_C N_Q} \end{bmatrix} \tag{5}$$

Each row in $\mathbf{Z}$ represents the quantiles for a given centroid. The element $z_{ij}(s, a; \theta')$ represents the jth quantile for the ith centroid. $\mathbf{v}$ represents the weighted quantile distribution for an action a, which can then be averaged to get $Q(s, a)$.

Quantile regression loss [Dabney et al., 2017] is used to update the network, where τ is a quantile probability associated with the given target distribution.

$$\rho_\tau(u) = |\tau - \delta_{u<0}| \mathcal{L}(u),$$

where

$$\mathcal{L}(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| < 1 \\ |u| - \frac{1}{2}, & \text{otherwise.} \end{cases}$$

u is the distance between each predicted support and each target support, with an additional argument of τ to specify which quantile point the prediction support is approximated to. Each quantile point $z_j(s, a; \theta') \; \forall j \in [1, N_Q]$ is independently moved toward the correct quantile position (according to the target quantile distribution formed by $\mathcal{T} z_j$ where $\mathcal{T}$ is the Bellman operator) $\mathcal{T} z_j \leftarrow r + \gamma z_j(s', a^*; \theta^-), \forall j$.

3.3 Integrated Agent

Our integrated RBF-DQN agent (Rainbow RBF-DQN) incorporates Distributional Quantile Regression, Double Networks, and Priority Experience Replay (PER). Empirically we found that Dueling networks, Noisy networks, and Multi-step returns did not lead to a consistent boost in performance over vanilla RBF-DQN so we exclude them from our Rainbow agent.

In order to incorporate PER to distributional RBF-DQN, the sampling weights are replaced with the regression loss for each batch of transitions.

$$p_t \propto (\sum_{i=1}^{N_Q} E_j[\rho_{\tau_i}(r + \gamma z_j(s', a^*; \theta^-) - z_i((s, a; \theta))])^w.$$

Double Q learning is added to Distributional RBF-DQN by having the online network parameterized by θ choose the centroid with the highest value at s', and return that centroid as the optimal action. This action is then passed into the target network parameterized by θ^- to return the target quantile distribution. $y_{\text{target}} = r + \gamma z_j(s', a^*; \theta^-)$, where $a^* \leftarrow E[z_i(s', \cdot; \theta)]$.

4 Experiments

We test our rainbow RBF-DQN agent on the 6 Open AI gym [Brockman et al., 2016] tasks presented in the vanilla RBF-DQN paper [Asadi et al., 2020] as well as on Humanoid-v2, a challenging 17 dimensional action space control task. For each task we use the low dimensional state space and default dense reward. We report the cumulative evaluation rewards across 10 trajectories.

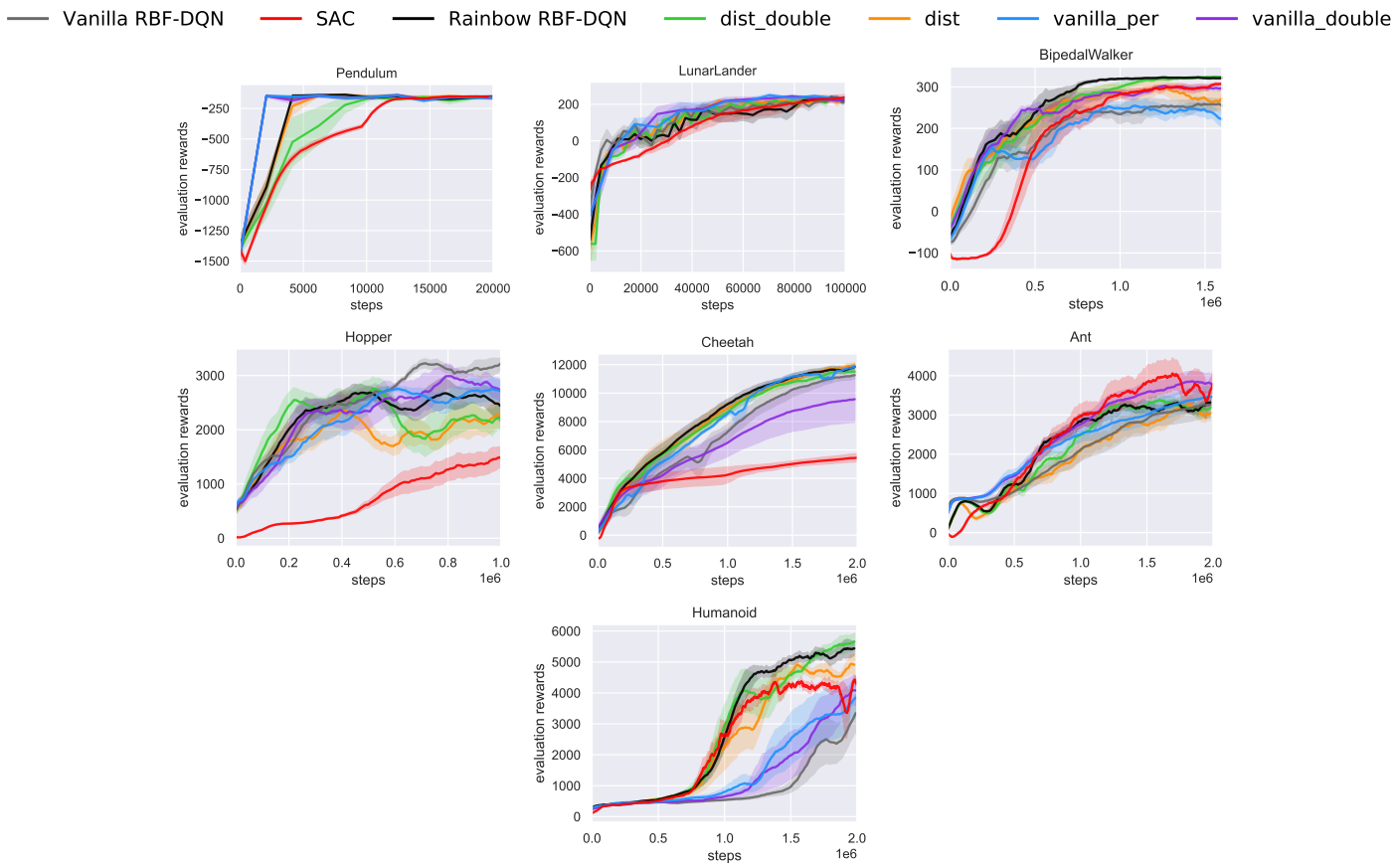


Figure 2: Total Evaluation Reward per Episode on 7 MuJoCo control tasks. All variants ran on 5 seeds and the 95% confidence interval across seeds is shaded.

5 Analysis and Discussion

In this section we review our experimental results and compare rainbow RBF-DQN to vanilla RBF-DQN and an existing state of the art actor-critic method, Soft Actor Critic (SAC) from Haarnoja et al. [2018]. We use the public implementation of vanilla RBF-DQN[2] and use the stable baselines 3 implementation of SAC[3].

5.1 Comparison to baseline vanilla RBF-DQN and SAC

In Figure 1 we plot rainbow RBF-DQN's performance across all 7 control tasks relative to vanilla RBF-DQN and SAC. Each task is treated equally, each variation is run for 5 seeds, and evaluation rewards are normalized relative to vanilla RBF-DQN.

According to Figure 1, when we average the performance of Rainbow RBF-DQN across all tasks, it is $\approx 10\%$ better in final performance than vanilla RBF-DQN and $\approx 30\%$ better than SAC (as well as more sample efficient as can be seen by the steeper learning curve). This improvement is in aggregate—on certain tasks like Humanoid, Rainbow RBF-DQN achieves almost 200% the reward of vanilla RBF-DQN, and nearly 25% better than current state of the art methods like

[2]https://github.com/kavosh8/RBFDQN_pytorch
[3]https://github.com/DLR-RM/stable-baselines3

SAC. Individual results are given in Figure 2. On a per-task basis, Rainbow RBF-DQN outperforms SAC on every task except Ant (where it is similar in performance), and only slightly underperforms vanilla RBF-DQN on Hopper. Rainbow RBF-DQN improves performance on Bipedal Walker and improves on vanilla RBF-DQN's already strong performance on the HalfCheetah task.

5.2 Ablation Studies

On high dimensional action space tasks, distributional value representations (with quantile regression) for RBF-DQN are crucial to achieving high reward, and are thus the most important ingredient to Rainbow RBF-DQN. In Figure 2, the Humanoid learning curve shows that both vanilla_per (vanilla with PER) and vanilla_double lag behind all the variations with distributional representations included (dist_double, dist, and Rainbow RBF-DQN). Double networks had only marginal positive impact on Distributional RBF-DQN hinting toward the fact that Distributional alone may have overestimation limiting properties. PER generally speaking led to only marginal improvements when paired with distributional RBF-DQN (we used $\alpha = 0.1$), but we included it because it is a relatively simple addition to the algorithm.

6 Conclusion

We presented algorithmic extensions for deep value-based learning for the RBF-DQN architecture, and empirically demonstrated improved performance on Humanoid, Bipedal Walker, and Half Cheetah. The most crucial ingredient in Rainbow RBF-DQN is quantile distributional value representations—allowing our agent to excel on high dimensional tasks like Humanoid, outperforming state of the art SAC results. Taken together, our results show that classical DQN enhancements in the discrete action domain can be applied with care to the continuous action domain. We hope Rainbow RBF-DQN becomes one of the go-to value-based methods that RL researchers use for continuous control tasks in high dimensional action spaces.

References

Kavosh Asadi, Neev Parikh, Ronald E. Parr, George D. Konidaris, and Michael L. Littman. Deep Radial-Basis Value Functions for Continuous Control. *arXiv e-prints*, art. arXiv:2002.01883, February 2020.

Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional Reinforcement Learning with Quantile Regression. *arXiv e-prints*, art. arXiv:1710.10044, October 2017.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1511.06581, November 2015.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv e-prints*, art. arXiv:1509.06461, September 2015.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3215–3222. AAAI Press, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204`.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 2017. URL `http://proceedings.mlr.press/v70/bellemare17a.html`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv e-prints*, art. arXiv:1606.01540, June 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv e-prints*, art. arXiv:1801.01290, January 2018.

Autonomous Open-Ended Learning of Tasks with Non-Stationary Interdependencies

Alejandro Romero [1], **Gianluca Baldassarre** [2], **Richard J. Duro**[1], **Vieri Giuliano Santucci**[2]
[1]Integrated Group for Engineering Research (GII)
CITIC research center
Universidade da Coruña, Spain
{alejandro.romero.montero, richard.duro}@udc.es
[2]Istituto di Scienze e Tecnologie della Cognizione (ISTC)
Consiglio Nazionale delle Ricerche (CNR), Roma, Italy
{gianluca.baldassarre, vieri.santucci}@istc.cnr.it

Abstract

Autonomous open-ended learning is a relevant approach in machine learning and robotics, allowing the design of artificial agents able to acquire goals and motor skills without the necessity of user assigned tasks. A crucial issue for this approach is to develop strategies to ensure that agents can maximise their competence on as many tasks as possible in the shortest possible time. Intrinsic motivations have proven to generate a task-agnostic signal to properly allocate the training time amongst goals. While the majority of works in the field of intrinsically motivated open-ended learning focus on scenarios where goals are independent from each other, only few of them studied the autonomous acquisition of interdependent tasks, and even fewer tackled scenarios where goals involve non-stationary interdependencies. Building on previous works, we tackle these crucial issues at the level of decision making (i.e., building strategies to properly select between goals), and we propose a hierarchical architecture that treating sub-tasks selection as a Markov Decision Process is able to properly learn interdependent skills on the basis of intrinsically generated motivations. In particular, we first deepen the analysis of a previous system, showing the importance of incorporating information about the relationships between tasks at a higher level of the architecture (that of goal selection). Then we introduce H-GRAIL, a new system that extends the previous one by adding a new learning layer to store the autonomously acquired sequences of tasks to be able to modify them in case the interdependencies are non-stationary. All systems are tested in a real robotic scenario, with a Baxter robot performing multiple interdependent reaching tasks.

Keywords: Autonomous Open-Ended Learning, Interdependent Tasks, Curriculum Learning, Intrinsic Motivations, Reinforcement Learning, Autonomous Robotics

Acknowledgements

*This work was partially supported by the the MCIU of Spain/FEDER (grant RTI2018-101114-B-I00), Xunta de Galicia (EDC431C-2021/39), Centro de Investigación de Galicia "CITIC" (ED431G 2019/01), by the Spanish Ministry of Education, Culture and Sports for the FPU grant of Alejandro Romero, and by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement no 713010, Project "GOAL-Robots – Goal-based Open-ended Autonomous Learning Robots"

1 Introduction

Autonomous open-ended learning (A-OEL) [5, 18] aims at the development of artificial agents able to solve a potentially unbounded set of different tasks in environments that might be unknown at design time. Similarly to multi-task reinforcement learning [6], a system has to learn multiple policies associated with different goals (i.e., the achievement of desired states/effects in the environment. In this sense "task" and "goal" can be used interchangeably, where a task consists in the achievement of the associated goal). However, in the A-OEL perspective the focus is not "simply" on the maximisation of the rewards, but on the development of a strategy that allows the agent to properly allocate the training time to maximise its competence over all the goals during the learning period. This reflects the scenario in which a system is left to explore the world for a limited time, while only in a second phase it will be assigned tasks that are useful for users: the greater the competence acquired, the higher the probability of being able to maximise the rewards for the subsequently assigned tasks.

In this perspective, intrinsic motivations (IMs) have been used in the field of machine learning and developmental robotics [11, 2], amongst other applications, to provide self-generated signals guiding the autonomous selection of tasks to be trained [16, 3, 4]. The majority of works within the intrinsically motivated open-ended learning framework are normally focused on scenarios where goal achievability does not depend on specific environmental states or preconditions. However, in real-world scenarios, tasks may require particular conditions to be fulfilled or, more interestingly, they may be interdependent so that one (or a sequence of them) is the precondition for the achievement of the other(s). As an example, consider a setting where the goal of arriving at a particular $[x, y]$ location is possible only if the intensity of illumination has already been set to a certain value. In that regard, the navigation goal is conditioned on the agent having caused the environment to reach a specific "illumination intensity goal".

The "interdependent tasks" scenario is of particular interest for both machine learning and robotics, and it has so far been scarcely studied in an A-OEL perspective [7, 15, 4]. Under the headings of curriculum learning [8, 9] and hierarchical reinforcement learning [1, 10], different works have focused on sequencing ever more complex tasks, with the aim of transferring knowledge from one to another or dividing the most difficult goals into sub-goals that can be learnt more easily. However, in most of these works, even when the agent autonomously creates and selects sub-tasks, these processes are based on an externally-assigned final goal. On the contrary, here we are interested in a situation where several possible interrelated tasks are presented to the agent, whose aim is to maximise its overall competence by selecting the goals it wants to learn and, where necessary, to learn the different curricula that are needed to acquire the skills of the hierarchically more complex goals. Furthermore, our aim is to address an even more complex scenario, which to date, especially in the field of autonomous robotics, is still poorly addressed: the scenario in which the interdependencies between goals may change over time, thus forcing the system to re-learn the different sequences in order to maintain a high competence in solving all the possible tasks.

In previous works, we presented the M-GRAIL architecture [15, 12] and we analysed how treating goal selection as a Markov Decision Process (MDP) results in better overall competence acquisition with respect to other approaches that treat goal selection as a bandit or contextual bandit problem [17, 7, 4]. Here we present a twofold study. On the one hand, we deepen the analysis of our approach by comparing M-GRAIL with a system which, although treating task selection as a bandit problem, is able to integrate the information about the dependencies between the goals directly into the low-level skills. On the other hand, we extend our system which, by relying only on intrinsic motivations, was not able to save the acquired sequences, and we present H-GRAIL, a new hierarchical system with 3 different learning processes. H-GRAIL is then tested in a scenario where the interdependencies between goals can change over time.

2 Problem Analysis and Suggested Solution

In multiple task learning the objective is to learn a set of different tasks (i.e., reaching a different goal), each associated to a core MDP. We assume a goal g is a specific subset $S^g \in S$, so that g has been achieved if the system enters any state in S^g. For each goal g there is a goal-dependent reward function r^g, determining a goal-dependent reward r_t^g at time t. Following [6], the overall objective of the system is then to find a policy π (or different goal-related policies π^g) such that

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}_{g\sim P}\left[r^g(\pi)\right], \tag{1}$$

where P is a probability distribution over the set G of possible goals g. As analysed in [15], in an OEL scenario the system objective is to maximise, in a finite and unknown learning time L, its competence C over G

$$C = \mathbb{E}_{g\sim P}\, C^g \tag{2}$$

where C^g is the goal-related competence for achieving goal g using π^g. In this sense, C^g reflects the expected goal-specific rewards $\mathbb{E}\{r_t^g + r_{t+1}^g + \ldots | \pi^g\}$ when executing π^g. Since L is finite and unknown, the agent has to maximise C as quickly as possible, efficiently distributing the learning time over G. The A-OEL of multiple goals is thus a training time allocation problem where the system has to build a meta-policy Π that at each time step t selects a goal to train for a certain (eventually fixed) amount of time t' so that at L the overall competence C will be maximal.

$$\Pi^*(t) = \arg\max_{\Pi} \; \mathbb{E}_{g\sim P}\left[C^g(t+L)\,|\,\Pi(t)\right]. \tag{3}$$

Given this objective, Π is not selecting goals to train with respect to their current competence $C^g(t)$, i.e. to the amount of returns expected for executing π^g, but with respect to the amount of competence the system can gain for practicing on g. The reward is thus the intrinsic motivation signal determined by the competence improvement $\Delta C^g = C^g(t+t') - C^g(t)$ obtained for training on g. Autonomously learning multiple tasks can thus be seen as a two-level problem: *(a)* the high-level goal-selection process to increase competence; and *(b)* the low-level learning of policies π^g. While we make no assumption about which algorithm is used to solve *(b)*, in the case where goals are independent (learning about one goal does not help the agent achieve other goals) and where the initial state of the environment is not affecting policy execution, task selection can be modelled as an N-armed bandit, as it has been typically addressed in the majority of OEL architectures, e.g. [3, 17, 4]. Differently, if tasks are interdependent (i.e. the achievement of some goals constitute the precondition for the achievement of other goals), Π has to take into consideration that goal-selection implies long-term consequences in terms of possible future rewards (i.e. future competence gain): it may be important to spend time practising goals that, on their own, do not provide competence improvement, but that allow the agent to train more advantageous goals. Thus, task selection now involves a credit assignment problem, since evaluating the returns gained from goal g requires backing up expected future competence improvement that may be achieved by further practising "distant" goals that have g as a precondition.

For this reason, similarly to [9], in M-GRAIL we proposed to no longer treat goal selection as a bandit problem but as an MDP, and to solve it through a Q-Learning algorithm that models the relation between the values of interrelated goals. However, given the transient nature of intrinsic motivations (which disappear when competence has reached a plateau), M-GRAIL has the problem that although it is able to learn all the policies π^g by properly assigning value to goals that constitute preconditions for others, it is not able to execute these sequences once the evaluations determined by intrinsic motivations have vanished. To cope with this limitation, here we propose H-GRAIL, a new architecture that adds a further layer to the previous system. M-GRAIL (as well as other systems in the literature) was essentially composed of two levels: one in which low level skills are learnt, through the maximisation of goal-specific rewards r^g; and a higher level in which goal selection takes place, treated as an MDP in which a Q-Learning algorithm maximises the competence improvement signal ΔC^g. H-GRAIL adds a further layer, changing the structure of the goal selector and dividing it into two components. The meta policy Π is again treated as a bandit problem based on the maximisation of competence improvement, but once the system has selected a goal g, it is used as an input to a second level which, structured as an MDP, must learn to sequence the sub-goals necessary to achieve the task selected by Π. This sub-goal selector, unlike the M-GRAIL selector, aims at the maximisation of goal-specific reinforcement r^g: in this way, the learnt interdependencies between the different goals will remain available to the system as "curricula", or better, as policies over sub-goals, even after the intrinsic motivations have disappeared.

3 Robotic Setup and Experiments

To test our system we implemented a robotic scenario (Fig. 1) where a Baxter robot has to learn to reach for different buttons that "light up" when pressed if their preconditions are satisfied. In a first experiment (Sec. 4.1), we compare M-GRAIL with a modified version of the e-MDB system [13], called Bandit-MDB, where the motivational system is implemented using a goal-selecting bandit mechanism based on competence improvement intrinsic reinforcements. On the contrary, M-GRAIL treats goal-selection as an MDP and solves it through a standard Q-Learning algorithm [19]. Both M-GRAIL and Bandit-MDB learn low-level skills via utility models, where each skill is an artificial neural network-based value function (see [14] for more details), however only the Bandit-MDB utility models receive contextual information as input (here a binary vector stating if a goal has been achieved within the current epoch, i.e. if a button is "on" or "off"), while in the case of M-GRAIL only the goal selector receives such information. This allow us to analyse how, and in particular at what level of the architecture, a robotic system should handle the dependencies between goals.

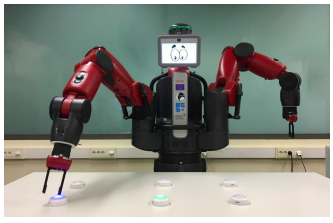


Figure 1: The experimental setup: buttons "light up" when pressed if all their preconditions are satisfied.

In a second experiment (Sec. 4.2) we test H-GRAIL in a similar robotic scenario, where interdependencies between goals are non-stationary (in particular, they change after a certain time during learning). H-GRAIL receives contextual information regarding the goals at the level of the sub-goal selector (implemented as a Q-Learning algorithm), while the high-level goal selector is implemented as a standard bandit maximising competence improvement.

In both experiments we use the right arm of the robot and we control its wrist final position (x, y, z) through Cartesian position control. Regarding perception, we used the images from an RGB-D camera located on the ceiling of the room and

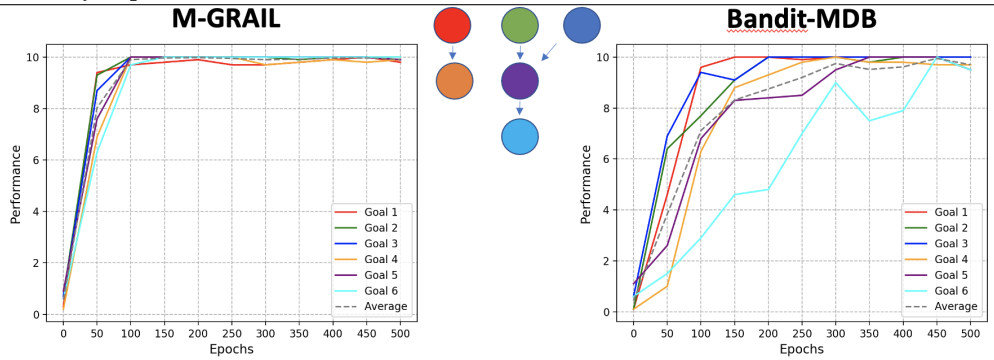


Figure 2: Configuration of the first experiment (arrows indicate dependencies) and the performance of the two systems.

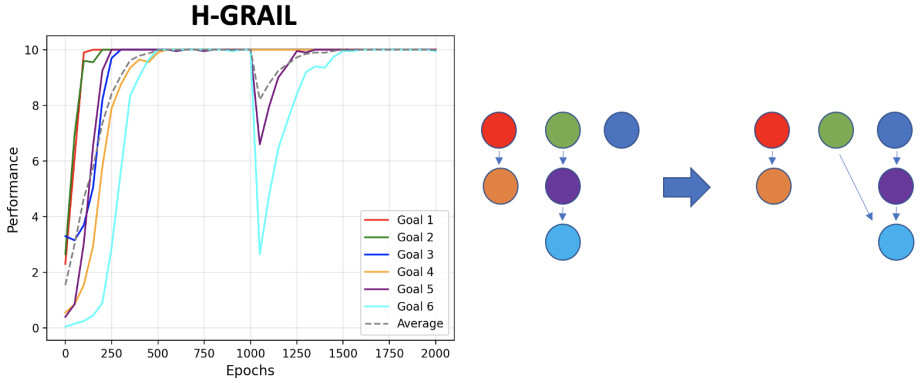


Figure 3: Configuration of the second experiment with non-stationary dependencies, and the performance of H-GRAIL.

binary sensors associated with buttons being pushed. This information is re-described in the form of distances between the detected objects and the end effector of the robot arm. Therefore, the perception of the robot is $(d_1, \ldots, d_n, s_1, \ldots, s_n)$, where d_j are the relative distances between the buttons and the robot end-effector, s_i are the states of the different buttons (active or not), and n is the number of buttons in the scenario (6 in the current experiment).

4 Results and Discussion

4.1 Testing where to incorporate interdependence knowledge

The first experiment was run for 500 epochs, each lasting 8 trials ending when the robot lights up the target button (it achieves the selected goal) or after a timeout of 70 time steps. At each trial the goal selector of the systems selects the goal to be achieved, while at the end of each epoch we reset the environment (the robot is set to home positions and the buttons are switched off). In this scenario there are two chains of dependencies: a simple one (with just one precondition) and a more complex chain where reaching the last goal (cyan button) requires the accomplishment of three other precondition goals. Fig. 2 shows the performance of the two systems (averages over 20 repetitions). M-GRAIL is very efficient in learning all the tasks, including those requiring preconditions, while Bandit-MDB needs 300 epochs to reach 90% performance and has more trouble learning the last task (activation of the cyan button). The longer time required by the Bandit-MDB system is the result of it having to learn more complex and longer skills, since, for example, to reach the blue button, the utility model corresponding to that skill must first learn to reach the red and green buttons. Unlike Bandit-MDB, when learning simple skills and concatenating them, M-GRAIL only has to learn to reach the blue button while the goal selector takes care of selecting the precondition goals before it: this ensures that the red and green buttons are active when the blue-button goal is selected. This experiment further corroborates the findings in [15], showing how our proposal to treat autonomous goal learning in a hierarchical manner yields positive results. In particular, here we analysed how it is more efficient to learn simple skills, not storing information on goal interdependencies, and concatenate goals at the level of the goal selector. This is because the time required for learning a single skill, given the appropriate preconditions guaranteed by the goal selection, is less than the time necessary to learn a skill to achieve a goal and also all those being the preconditions to it.

4.2 Testing H-GRAIL on tasks with non-stationary interdependencies

In the second experiment we test our new architecture H-GRAIL in an environment where interdependencies between goals are non-stationary. In particular, after 1,000 epochs the dependencies are modified as shown in Fig. 3 (right). Experiments are run for a total of 2,000 epochs, composed as in the first experiment. The results (averages over 20 repetitions) show that H-GRAIL was not only able to learn the skills necessary to achieve the different goal-dependent tasks, but also had the ability to dynamically adapt to a change in the structure of task relationships. Unlike M-GRAIL, the sub-goal selector added in H-GRAIL is able to store the acquired curricula through a Q-Learning algorithm based on the reinforcements obtained to achieve the tasks selected by the same sub-goal selector. However, when the interdependencies change the performance in achieving goals will drop (the curricula will not work anymore), thus the goal selector starts again a selection process aimed at maximising competence improvement that pushes the sub-goal selector towards finding new ways to sequence the different tasks to properly achieve those selected by the agent.

References

[1] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pages 438–445, 2004.

[2] G. Baldassarre and M. Mirolli. *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer Science & Business Media, 2013.

[3] A. Baranes and P.-Y. Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

[4] S. Blaes, M. Vlastelica Pogančić, J. Zhu, and G. Martius. Control what you can: Intrinsically motivated task-planning agent. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] C. Colas, T. Karch, O. Sigaud, and P.-Y. Oudeyer. Intrinsically motivated goal-conditioned reinforcement learning: a short survey. *arXiv preprint arXiv:2012.09830*, 2020.

[6] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1514–1523, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[7] S. Forestier, Y. Mollard, and P.-Y. Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

[8] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

[9] S. Narvekar and P. Stone. Learning curriculum policies for reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 25–33, 2019.

[10] R. Niel and M. A. Wiering. Hierarchical reinforcement learning for playing a dynamic dungeon crawler game. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1159–1166. IEEE, 2018.

[11] P.-Y. Oudeyer, F. Kaplan, and V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(6), 2007.

[12] A. Romero, G. Baldassarre, R. J. Duro, and V. G. Santucci. Analysing autonomous open-ended learning of skills with different interdependent subgoals in robots. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pages 646–651. IEEE, 2021.

[13] A. Romero, F. Bellas, J. A. Becerra, and R. J. Duro. Motivation as a tool for designing lifelong learning robots. *Integrated Computer-Aided Engineering*, 27(4):353–372, 2020.

[14] A. Romero, A. Prieto, F. Bellas, and R. J. Duro. Simplifying the creation and management of utility models in continuous domains for cognitive robotics. *Neurocomputing*, 353:106–118, 2019.

[15] V. G. Santucci, G. Baldassarre, and E. Cartoni. Autonomous reinforcement learning of multiple interrelated tasks. In *2019 Joint IEEE 9th international conference on development and learning and epigenetic robotics (ICDL-EpiRob)*, pages 221–227. IEEE, 2019.

[16] V. G. Santucci, G. Baldassarre, and M. Mirolli. Which is the best intrinsic motivation signal for learning multiple skills? *Frontiers in neurorobotics*, 7:22, 2013.

[17] V. G. Santucci, G. Baldassarre, and M. Mirolli. Grail: A goal-discovering robotic architecture for intrinsically-motivated learning. *IEEE Transactions on Cognitive and Developmental Systems*, 8(3):214–231, 2016.

[18] V. G. Santucci, P.-Y. Oudeyer, A. Barto, and G. Baldassarre. Intrinsically motivated open-ended learning in autonomous robots. *Frontiers in neurorobotics*, page 115, 2020.

[19] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Planning with Contraction-Based Adaptive Lookahead

Aviv Rosenberg*
Tel-Aviv University
avivros007@gmail.com

Assaf Hallak
Nvidia Research

Shie Mannor
Technion and Nvidia Research

Gal Chechik
Bar-Ilan University and Nvidia Research

Gal Dalal
Nvidia Research

Abstract

The classical Policy Iteration (PI) algorithm alternates between greedy one-step policy improvement and policy evaluation. Recent literature shows that multi-step lookahead policy improvement leads to a better convergence rate at the expense of increased complexity per iteration. However, prior to running the algorithm, one cannot tell what is the best fixed lookahead horizon. Moreover, per a given run, using a lookahead of horizon larger than one is often wasteful. In this work, we propose for the first time to dynamically adapt the multi-step lookahead horizon as a function of the state and of the value estimate. We devise two PI variants and analyze the trade-off between iteration count and computational complexity per iteration. The first variant takes the desired contraction factor as the objective and minimizes the per-iteration complexity. The second variant takes as input the computational complexity per iteration and minimizes the overall contraction factor. We then devise a corresponding DQN-based algorithm with an adaptive tree search horizon. We also include a novel enhancement for on-policy learning: per-depth value function estimator. Lastly, we demonstrate the efficacy of our adaptive lookahead method in a maze environment and in Atari.

Keywords: Planning, Reinforcement Learning, Adaptive Lookahead

*Research conducted while the author was an intern at Nvidia Research

1 Introduction

The classic Policy Iteration (PI) and Value Iteration (VI) algorithms are the basis for most state-of-the-art reinforcement learning (RL) algorithms. As both PI and VI are based on a one-step greedy approach for policy improvement, so are the most commonly used policy-gradient and Q-learning based approaches. In each iteration, they perform an improvement of their current policy by looking one step forward and acting greedily. While this is the simplest and most common paradigm, stronger performance was recently achieved using multi-step lookahead. Notable examples are AlphaGo [10] and MuZero [9], where the multi-step lookahead is implemented via Monte Carlo Tree Search (MCTS) [1]. Several recent works rigorously analyzed the properties of multi-step lookahead in common RL schemes [2, 3, 4, 5, 6]. This and other related literature studied a fixed planning horizon chosen in advance. However, both in simulated and real-world environments there is a large variety of states that benefit differently from various lookahead horizons. A grasping robot far from its target will learn very little from looking a few steps into the future, but if the target is within reach, much more precision and planning are required to grasp the object correctly. Similarly, at the beginning of a chess game, lookahead grants little information as to which move is better, while agents in intricate situations in mid-game benefit immensely from considering all future possibilities for the next few moves. Indeed, in this work, we devise a well-established methodology for adaptively choosing the planning horizons in each state, and show it achieves a significant speed-up of the learning process.

We propose two complementing approaches to determine the suitable horizon per state in each PI iteration. To do so, we keep track of the distance between the value function estimate and the optimal value. Our first algorithm Threshold-based Lookahead PI (TLPI) ensures a desired convergence rate and minimizes the computational complexity for each iteration. Alternatively, our second algorithm Quantile-based Lookahead PI (QLPI) takes the per-iteration computational complexity as a given budget and aims for the best possible convergence rate. We then prove that both TLPI and QLPI converge to the optimum and achieve significantly lower computational cost than its fixed-horizon alternative. Next, we devise QL-DQN: a DQN [7] variant of QLPI. In QL-DQN, the policy chooses an action by employing an exhaustive tree search [6] looking h steps into the future. The tree-depth h is chosen adaptively per state to achieve an overall improved convergence rate at a reduced computational cost. To sustain on-policy consistency while generalizing over the multiple depths, we use a different value network per depth, where the first layers are shared across networks. We test our method on Atari and show it improves upon a fixed-depth tree search.

2 Preliminaries and a Motivating Example

We consider a discounted MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a finite state space of size S, $\mathcal{A}$ is a finite action space of size A, $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition function, and $\gamma \in (0, 1)$ is the discount factor. Let $\pi : \mathcal{S} \to \mathcal{A}$ be a stationary policy, and let $V^\pi \in \mathbb{R}^S$ be the value function of π defined by $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s\right]$. The goal of a planning algorithm is to find the optimal policy $\pi^\star$ such that, for every $s \in \mathcal{S}$, $V^\star(s) = V^{\pi^\star}(s) = \max_{\pi:\mathcal{S}\to\mathcal{A}} V^\pi(s)$.

Policy Iteration (PI) starts from an arbitrary policy π_0 and performs iterations that consist of: (1) an evaluation step that evaluates the value of the current policy, and (2) an improvement step that performs a 1-step improvement based on the computed value. That is, for $n = 0, 1, 2, \ldots$: $\pi_{n+1}(s) = \arg\max_{a\in\mathcal{A}} r(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, a) V^{\pi_n}(s')$. By the contraction property of the Bellman operator, one can prove that PI finds the optimal policy after at most $\lceil (\log \frac{1}{\gamma})^{-1} S(A-1) \log \frac{1}{1-\gamma} \rceil$ iterations [8]. PI can be extended to h-PI by performing h-step improvements (instead of 1-step). Formally, the update rule of h-PI is $\pi_{n+1}(s) = \arg\max_{a\in\mathcal{A}} Q_h^{\pi_n}(s, a)$, where $Q_h^\pi(s, a) = \max_{\{\pi_t\}_{t=1}^h} \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t r(s_t, \pi_t(s_t)) + \gamma^h V^\pi(s_h) \mid s_0 = s, \pi_0(s) = a\right]$. The operator induced by h-step lookahead is γ^h contracting which allows to reduce a factor of h from the bound on the number of iterations until convergence [2], i.e., $\lceil (h \log \frac{1}{\gamma})^{-1} S(A-1) \log \frac{1}{1-\gamma} \rceil$.

Multi-step lookahead guarantees that the number of iterations to convergence is smaller than 1-step lookahead, but it comes with a computational cost. Computing the h-step improvement may take exponential time in h. In tabular MDPs, this can be mitigated with the use of dynamic programming [5], while in MDPs with a large (or infinite) state space, MCTS [1] or the alternative exhaustive tree-search [6] are used in a forward-looking fashion. We evaluate our algorithms by measuring their computational complexity. For this, let $c(h)$ be the computational cost of performing an h-step improvement.

To show the potential of adaptive lookahead, consider the following MDP example: Let $\mathcal{M}$ be an MDP with $n + 1$ states $s_0, s_1, \ldots, s_n$ and a single sink state s_{n+1}. Each of the $n + 1$ states transitions to the consecutive state by applying action u, and to the sink state with action d. All rewards are 0 except for state s_n where action u obtains reward $1 - \gamma$. Now consider the standard PI algorithm when initialized with $\pi_0(s_i) = d$ for all i. Since the reward at the end of the chain needs to propagate backward, in each iteration the value of only a single state is updated. Thus, PI takes exactly n iterations to converge to the optimal policy $\pi^\star(s_i) = u$. When instead a fixed horizon $h = 2$ is used, the reward propagates through two states in each iteration (instead of one) and therefore convergence takes $n/2$ iterations. Generally, h-PI takes n/h iterations to converge. While h-PI converges faster (in terms of iterations) as h increases, in most states, performing h-step lookahead does not contribute to the speed-up at all. For instance, taking $h = 2$, we can achieve the exact same convergence rate as 2-PI by using a 2-step lookahead in only a single state in each iteration. For general h, consider applying ℓ-step lookahead in only one state – the one that is ℓ steps behind the last updated state in the chain – for each $\ell = 2, \ldots, h$ and 1-step in the others. This guarantees the same number of iterations until convergence as h-PI, but with much less computation time. Namely, while the per-iteration computational cost of h-PI is $O\big(n \cdot c(h)\big)$, we can achieve the same convergence rate with just $O\big(n \cdot c(1) + \sum_{\ell=2}^{h} c(\ell)\big)$. In practice, when n is large and $c(h)$ can scale exponentially with h, this gap can be immense: $O\big(n \cdot 2^h\big)$ versus $O\big(n + 2^h\big)$.

Algorithm 1 TLPI(κ)

Initialization: Arbitrary π_0, $t \leftarrow 0$.
while π_t changes **do**
  Evaluation: compute V^{π_t}, and set $U(s,a) \leftarrow \infty$.
  1-step improvement: $U(s,a) \leftarrow Q_1^{\pi_t}(s,a)\ \forall s \in \mathcal{S}$.
  $h^{(\kappa)}$-step improvement: $U(s,a) \leftarrow Q_{h^{(\kappa)}}^{\pi_t}(s,a)$ for every $s \in \mathcal{S}$ such that $|V^\star(s) - \max_{a\in\mathcal{A}} U(s,a)| > \kappa \|V^\star - V^{\pi_t}\|_\infty$.
  Set $\pi_{t+1}(s) \leftarrow \arg\max_{a\in\mathcal{A}} U(s,a)\ \forall s \in \mathcal{S}$.
end while

Algorithm 2 QLPI($\theta_1, \ldots, \theta_H$)

Initialization: Arbitrary π_0, $t \leftarrow 0$.
while π_t changes **do**
  Evaluation: compute V^{π_t}, and set $U(s,a) \leftarrow \infty$.
  For $h = 1, \ldots, H$, h-step improvement: $U(s,a) \leftarrow Q_h^{\pi_t}(s,a)$ for every $s \in S$ such that $|V^\star(s) - \max_{a\in\mathcal{A}} U(s,a)| \geq q_h$, where q_h is the $(1-\theta_h)$ quantile of $\{|V^\star(s) - \max_a U(s,a)|\}_{s\in\mathcal{S}}$.
  Set $\pi_{t+1}(s) \leftarrow \arg\max_{a\in\mathcal{A}} U(s,a)\ \forall s \in \mathcal{S}$.
end while

3 Contraction-Based Adaptive Lookahead

In this section, we introduce the concept of dynamically adapting the planning lookahead horizon during runtime, based on the online obtained contraction. As shown in Section 2, h-PI convergence rate can be achieved when using lookahead larger than 1 in just h states. A prominent question is thus how to choose these states? In the example, the chosen states are evidently those with the maximal distance between their 1-step improvement and optimal value. Next, we show this approach also leads to theoretical convergence guarantees. To that end, we delve into the theoretical properties of PI. Since 1-step improvement yields γ contraction while h-step improvement gives γ^h, h-PI converges h times faster than standard PI [2]. Importantly, this contraction is with respect to the L_∞ norm; i.e., the states with the smallest contraction determine the convergence rate. This behavior is the source of weakness of fixed lookahead. The example in Section 2 shows that one state may slow down convergence, but it also hints at an elegant solution: use a larger lookahead value in states with small contraction. We leverage this observation and present two new algorithms: TLPI which aims to achieve a fixed contraction in all states in reduced computational cost, and QLPI which aims to achieve maximal contraction in every iteration within a fixed computational budget. While both algorithms seek to optimize a similar problem, their analysis differs and sheds light on the problem from different perspectives: TLPI depends on the actual value of the contraction per state, while QLPI considers the ordering with respect to the contraction factors of all states. Our vanilla algorithms assume knowledge of $V^\star$. This is a strong assumption. However, we make it only for the basis of our theoretical analysis. In the full version of the paper, we provide analysis after relaxing this assumption while in Sections 4 and 5 we present experiments that use alternative warm-start value functions.

Threshold-based Lookahead Policy Iteration. TLPI (Algorithm 1) takes as input the optimal value function $V^\star$ and a desired contraction factor κ. It ensures that in each iteration, the value in every state contracts by at least κ. This is achieved by first performing 1-step improvement in all states, and then performing $h^{(\kappa)}$-improvement in states whose measured contraction is less than κ, where $h^{(\kappa)}$ is the smallest integer h such that $\gamma^h \leq \kappa$. The following result states that TLPI converges at least as fast as $(h^{(\kappa)}-1)$-PI with improved computational complexity. To measure the trade-off between contraction factor (that determines convergence rate) and computational complexity needed to achieve it, Definition 3.1 presents $\theta^{(\kappa)}$ as the fraction of states in which we perform a large lookahead.

Definition 3.1. Let $\{\pi_t\}_{t=1}^T$ be the sequence of policies generated by TLPI. Let $\kappa \in (0,1)$, and define $\mathcal{X}_t = \{s : |V^\star(s) - T[V^{\pi_t}](s)| \leq \kappa\|V^\star - V^{\pi_t}\|_\infty\}$ as the set of states contracted by κ after 1-step improvement in iteration t. Then, denote by $\theta^{(\kappa)} = \max_{1\leq t\leq T} |S \setminus \mathcal{X}_t|/S$ the largest fraction of states with contraction less than κ, observed along all policy updates.

Theorem 3.2. *TLPI converges in* $\lceil \left((h^{(\kappa)}-1)\log\frac{1}{\gamma}\right)^{-1} S(A-1)\log\frac{1}{1-\gamma}\rceil$ *iterations, and* $S\cdot\left(c(1)+\theta^{(\kappa)}c(h^{(\kappa)})\right)$ *per-iteration complexity.*

Quantile-based Lookahead Policy Iteration. QLPI (Algorithm 2) resembles TLPI, but instead of a contraction coefficient κ, it takes as input a vector of quantiles (budgets) $(\theta_1, \ldots, \theta_H) \in [0,1]^H$ for some predetermined maximal lookahead H. QLPI attempts to maximize the contraction in every iteration while using ℓ-step lookahead in at most $\theta_\ell \cdot S$ states. This is achieved by performing ℓ-step improvement on the θ_ℓ portion of states that are furthest away from $V^\star$. Note that QLPI is a generalization of h-PI, obtained by setting $\theta_h = 1$ and $\theta_l = 0$ for all $\ell \neq h$. The following result is complementary to Theorem 3.2: instead of choosing the desired iteration complexity (via κ in TLPI), we choose the desired computational complexity per iteration via budgets $(\theta_1, \ldots, \theta_H)$. For the resulting iteration complexity, we define the induced contraction factor $\kappa^{(\theta)}$ – the effective contraction obtained by QLPI.

Definition 3.3. Let $\{\pi_t\}_{t=1}^T$ be the sequence of policies generated by QLPI. Let $h_t^\theta(s)$ be the largest lookahead applied in state s in iteration t when running QLPI with quantiles $(\theta_1, \ldots, \theta_H)$. For a given κ, define $\mathcal{Y}_t(\kappa) = \{s : |V^\star(s) - T^{h_t^\theta(s)}[V^{\pi_t}](s)| \leq \kappa\|V^\star - V^{\pi_t}\|_\infty\}$ as the set of states contracted by κ in iteration t. The induced contraction factor $\kappa^{(\theta)}$ is defined as the minimal κ such that $\mathcal{Y}_t(\kappa) = \mathcal{S}\ \ \forall t$.

Theorem 3.4. *QLPI converges in* $\lceil(\log\frac{1}{\kappa^{(\theta)}})^{-1} S(A-1)\log\frac{1}{1-\gamma}\rceil$ *iterations, and* $S\cdot\sum_{h=1}^H \theta_h c(h)$ *per-iteration computational complexity.*

To illustrate the merits of TLPI and QLPI, consider the chain MDP in Section 2. For TLPI set $\kappa = \gamma^h$ for some h. In every iteration, the states not contracted by κ after 1-step improvement are the h states closest to the end of the chain that have not been updated yet. Thus, $\theta^{(\kappa)} = h/S$ and the per-iteration computational complexity is $S\cdot c(1) + h\cdot c(h)$. For QLPI set $\theta_1 = 1$ and $\theta_2 = \cdots = \theta_h = 1/S$ for some h. In every iteration QLPI first performs 1-step lookahead in all states, and then, for each $\ell = 2, \ldots, h$, it performs ℓ-step lookahead in exactly one state – the state that is ℓ steps behind the last updated state in the chain. The induced contraction is thus $\kappa^{(\theta)} = \gamma^h$ and QLPI converges in n/h iterations with *optimal* per-iteration complexity of $S\cdot c(1) + \sum_{\ell=2}^h c(\ell)$.

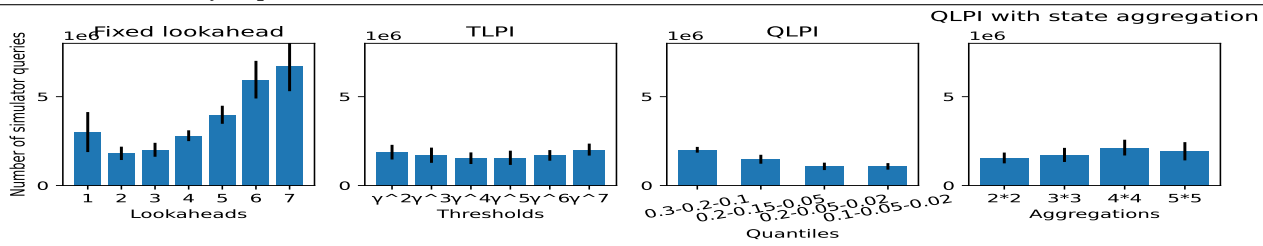


Figure 1: Number of queries to the simulator until convergence. Lower is better. The results are averaged across 10 runs and the error bars represented standard deviation. QLPI is run with lookaheads 1, 2, 4, and 8, where the quantiles in the x-axis represent $\theta_2, \theta_4, \theta_8$.

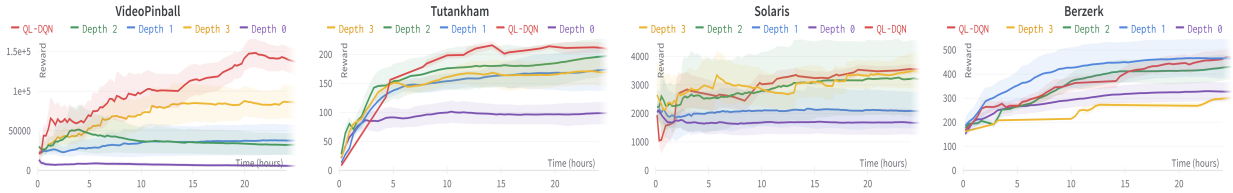


Figure 2: Average and std of training reward of QL-DQN (in red) and DQN with fixed tree-depths $0, 1, 2, 3$ in Atari environments.

4 Maze Experiments

In a first set of experiments, we evaluate our adaptive lookahead algorithms on a grid world with walls. Specifically, we used a 30×30 grid world that is divided to four rooms with doors between them. The agent is spawned in the top left corner and needs to reach one of four randomly chosen goals where the reward is 1, while avoiding the trap that incurs a reward of –1. There are four deterministic actions (up, down, right, left). Upon reaching a goal, the agent is moved to a random state. We set $\gamma = 0.98$. In short, the experiments show that with adaptive lookahead, we manage to reach the solution with better sample complexity compared to fixed-horizon h-PI. More importantly, our methods are robust to hyperparameter choices: improved results are obtained uniformly with *all* various tested parameters of TLPI and QLPI. This alleviates the heavy burden of finding the best fixed horizon for a given environment.

More specifically, we first test fixed-horizon h-PI with values $h = 1, 2, \ldots, 7$. In Figure 1, we compare the overall computational complexity, and not only the number of iterations. To measure performance, we count the number of *queries to the simulator* (environment) until convergence to the optimal value. More efficient lookahead horizons will require fewer overall calls to the simulator. Beginning with the fixed lookahead results in the leftmost plot, we see the trade-off when picking the lookahead. For a lookahead too short, convergence requires too many iterations such that even the low computational complexity of each iteration is not sufficient to compensate. Note that standard PI (i.e., $h = 1$) evidently performs worse than the best fixed lookahead although it is overwhelmingly the most popular version of PI. On the other extreme of very large lookahead, each iteration is too computationally expensive, despite the smaller number of iterations. Next, we run TLPI with $\kappa = \gamma^2, \gamma^3, \ldots, \gamma^7$. The results are given in Figure 1, second plot. By Theorem 3.2, when setting $\kappa = \gamma^h$, we expect the same number of iterations until convergence as h-PI but with better computational complexity. In fact, the results reveal even stronger behavior: TLPI(γ^h) for all $h = 1, 2, \ldots, 7$ achieves similar computational complexity compared to the *best* fixed lookahead witnessed in h-PI. Then, we run QLPI with $\theta_3 = \theta_5 = \theta_6 = \theta_7 = 0$ and $\theta_1 = 1$ in all our experiments. For $(\theta_2, \theta_4, \theta_8)$ we set the following values: $(0.3, 0.2, 0.1)$, $(0.2, 0.15, 0.05)$, $(0.2, 0.05, 0.02)$ and $(0.1, 0.05, 0.02)$, which respectively depict decreasing weights to depths $2, 4, 8$. The results are presented in Figure 1, third plot. Again we can see that for all the parameters, QLPI performs as well as the best fixed lookahead. Moreover, notice that for some choices of θ vectors, the performance significantly improves upon the best fixed horizon. Finally, we again run QLPI with $(0.1, 0.05, 0.02)$, but replace $V^\star$ with an approximation we obtain with state aggregation. Namely, we merge squares of $k \times k$ into a single state, solve the smaller aggregated MDP, and use its optimal value as an approximation for $V^\star$. We perform this experiment with $k = 2, 3, 4, 5$ and include the aggregated MDP solution process as part of the total simulator query count. This way, our final algorithm does not have any prior knowledge of $V^\star$. The results are presented in Figure 1, last plot. As expected, the performance is slightly worse than the original QLPI that uses the accurate $V^\star$, but for all different aggregation choices, the algorithm still performs as well as the best fixed lookahead in h-PI.

5 QL-DQN and Atari Experiments

In this section, we extend our adaptive lookahead algorithm QLPI to neural network function approximation. We present Quantile-based Lookahead DQN (QL-DQN): the first DQN algorithm that uses state-dependent lookahead that is dynamically chosen throughout the learning process. To extend QLPI to QL-DQN, we introduce three key features: First, we need to compute the quantile of the current state s in order to determine which lookahead h to use. However, we cannot go over the entire state space to find the position of h in

the ordering as in the tabular case. Instead, we propose to estimate the position of s using the replay buffer. Namely, we compute q_h : the $(1-\theta_h)$ ordering-based quantile of $|\tilde{V}^\star - T[\tilde{V}^{\pi_t}]|$ in the replay buffer and use lookahead of h if $|\tilde{V}^\star(s) - T[\tilde{V}^{\pi_t}](s)| \geq q_h$. For $\tilde{V}^\star$, we use a trained agent with depth 0, which we know performs poorly compared to larger lookaheads. Second, once a lookahead h was chosen, we need to perform h-step improvement. To implement h-step lookahead in Atari, we build upon the batch-BFS algorithm of [6]. In short, the lookahead mechanism relies on an exhaustive tree-search that at each step spans the tree of outcomes up to depth h from each state, done efficiently via parallel GPU emulation of Atari. Third, we introduce a *per-depth Q-function*. This feature is crucial to keep online consistency and achieve convergence. Without it, values across mixed depths suffer from off-policy distortion and the agent fails to converge. Technically, we maintain H parallel Q-networks (where H is the maximal tree depth) and use the h-th network when predicting the value of a leaf in depth h in the tree. To improve generalization and data re-use, the H networks share the initial layers (feature extractors). When storing states to the replay buffer, we attach their chosen lookahead depth and used it later for the target loss. All other parts of the algorithm and hyper-parameter choices are taken as-is from the original DQN paper [7].

We train QL-DQN on several Atari environments. Since the goal of our work is to improve sample complexity over fixed-horizon baselines, our metric of interest is the reward as a function of training time. Figure 2 presents convergence of QL-DQN versus DQN with fixed depths 0 through 3, as a function of time. The plots consist of the average score across 5 seeds together with std values. QL-DQN achieves better performance on VideoPinball and Tutankham, while on Solaris and Berzerk, it is on-par with the best fixed lookahead. The conclusion is again that we obtain a better or similarly-performing agent to a pre-determined fixed planning horizon. This comes with the benefit of robustness to the expensive hyper-parameter choice of the best fixed horizon per a given environment.

6 Discussion

In this paper, we propose the first planning and learning algorithms that dynamically adapt lookahead as a function of the state and the current value function estimate. We demonstrate the potential of adaptive lookahead both theoretically – proving convergence with improved computational complexity, and empirically – demonstrating their favorable performance in a maze and Atari. Our algorithms perform as well as the best fixed horizon in hindsight in almost all the experiments, while in some cases they surpass it. Future work warrants an investigation of whether the best fixed horizon can always be outperformed by an adaptive horizon. Theoretically, our guarantees rely on prior knowledge of the (approximate) optimal value, which raises the question of whether one can choose lookahead horizons adaptively without any prior knowledge, e.g., using transfer learning based on similarity between domains. Moreover, when the forward model used to perform the lookahead is inaccurate or learned from data, the adaptive state-dependent lookahead itself may serve as a quantifier for the level of trust in the value function estimate (short lookahead) versus the model (long lookahead). In essence, this can offer a way for state-wise regularization of the learning or planning problem. Our work is also related to the growing Sim2Real literature. In particular, consider the case of having several simulators with different computational costs and fidelity levels. The lookahead problem then translates to choosing in which states to use which simulator with which lookahead.

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[2] Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. Beyond the one-step greedy approach in reinforcement learning. In *International Conference on Machine Learning*, pages 1387–1396. PMLR, 2018.

[3] Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. Multiple-step greedy policies in online and approximate reinforcement learning. In *NeurIPS 2018-Thirty-second Conference on Neural Information Processing Systems*, 2018.

[4] Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. How to combine tree-search methods in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3494–3501, 2019.

[5] Y. Efroni, M. Ghavamzadeh, and S. Mannor. Online planning with lookahead policies. *Advances in Neural Information Processing Systems*, 33, 2020.

[6] A. Hallak, G. Dalal, S. Dalton, S. Mannor, G. Chechik, et al. Improve agents without retraining: Parallel tree search with off-policy correction. *Advances in Neural Information Processing Systems*, 34, 2021.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] B. Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. *Mathematics of Operations Research*, 41(3):758–774, 2016.

[9] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

All-persistence Bellman Update for Reinforcement Learning

Luca Sabbioni **Luca Al Daire** **Lorenzo Bisi** **Alberto Metelli** **Marcello Restelli**

Politecnico di Milano
Milan, Italy
{luca.sabbioni, luca.al, lorenzo.bisi, albertomaria.metelli, marcello.restelli}@polimi.it

Abstract

In Reinforcement Learning, the performance of learning agents is highly sensitive to the choice of time discretization. Agents acting at high frequencies have the best control opportunities, along with some drawbacks, such as possible inefficient exploration and vanishing of the action advantages. The repetition of the actions, i.e., *action persistence*, comes into help, as it allows the agent to visit wider regions of the state space and improve the estimation of the action effects. In this work, we derive a novel operator, the All-Persistence Bellman Operator, which allows an effective use of both the low-persistence experience, by decomposition into sub-transition, and the high-persistence experience, thanks to the introduction of a suitable *bootstrap* procedure. In this way, we employ transitions collected at *any* time scale to update simultaneously the action values of the considered persistence set. We prove the contraction property of the All-Persistence Bellman Operator and, based on it, we extend classic Q-learning and DQN. We experimentally evaluate our approach in both tabular contexts and more challenging frameworks, including some Atari games.

Keywords: Reinforcement Learning, Deep Q-Network,
Action Repetition, Persistence Selection

1 Introduction

In recent years, Reinforcement Learning (RL, [10]) methods have proven to be successful in a wide variety of applications, where sequential decision-making problems are typically modelled as a Markov Decision Process (MDP, [7]), a formalism that addresses the agent-environment interactions through *discrete-time* transitions. *Continuous-time* control problems, instead, are usually addressed by means of time discretization, which induces a specific control frequency f, or, equivalently, a time step $\delta = \frac{1}{f}$ [6]. This represents an environment hyperparameter, which may have dramatic effects on the process of learning the optimal policy [4]. Indeed, higher frequencies allow for greater control opportunities, but they have significant drawbacks. The most relevant one is related to the toned down effect of the selected actions. In the limit for time discretization $\delta \to 0$, the advantage of each action collapses to zero, preventing the agent from finding the best action [11] and leading to higher sample complexity. Moreover, a random uniform policy played at high frequency may not be adequate for exploration, as it tends to visit only a local neighborhood of the initial state [6]. This is problematic, especially in goal-based or sparse rewards environments, where the most informative states may never be visited. On the other hand, large time discretizations benefit from a higher probability of reaching far states, but they also deeply modify the transition process, hence a possibly large subspace of states may not be reachable.

One of the solutions to achieve the advantages related to exploration and sample complexity, while limiting the loss of control, consists in *action persistence*, called also *action repetition* [3, 4]. When the dynamics are very rapid, repeating an action is equivalent to acting at lower frequencies. Thus, the agent can achieve, in some environments, a more effective exploration, better capture the consequences of each action, and, as a final consequence, learn the optimal policy faster.

In this work, we propose a value-based approach in which the agent does not only choose the *action*, but also its *persistence*, with the goal of making the most effective use of samples collected at different persistences. The information collected at *one* persistence is then used to improve the action value function estimates of *all* the considered possible persistences, by decomposing the observed history in many sub-transitions of reduced length to update lower persistence values, and by using a suitable *bootstrapping* procedure of the missing information for higher persistences. This procedure is formalized with the introduction of the *All-persistence Bellman Operator*, which enjoys a contraction property analogous to that of the traditional optimal Bellman operator. This new operator is then embedded into the classic Q-learning algorithm, obtaining *Persistent Q-learning* (Per$Q$-learning). This novel algorithm, allowing for an effective use of the transitions sampled at different persistences, let us experience a faster convergence and fosters a better exploration of the state space. Furthermore, in order to deal with more complex domains, we consider the Deep RL scenario, extending the Deep Q-Network (DQN) algorithm to its persistent version *Persistent Deep Q-Network* (PerDQN). Finally, we evaluate the proposed algorithms, in comparison on both illustrative and complex domains, highlighting strengths and weaknesses.

2 Preliminaries

A discrete-time Markov Decision Process (MDP,[7]) is defined as a tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the finite action space, $P : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathcal{S})$ is the Markovian transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. A Markovian stationary policy $\pi : \mathcal{S} \to \mathscr{P}(\mathcal{A})$ maps states to probability measures over $\mathcal{A}$. We denote with Π the set of Markovian stationary policies. The *action-value function*, or Q-function, of a policy $\pi \in \Pi$ is the expected discounted sum of the rewards obtained by performing action a in state s and following policy π thereafter: $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s,\, a_0 = a \right]$, where $r_{t+1} = r(s_t, a_t)$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$ for all $t \in \mathbb{N}$. The optimal Q-function is given by: $Q^\star(s, a) = \sup_{\pi \in \Pi} Q^\pi(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. An *optimal policy* $\pi^\star \in \Pi$ is any policy greedy w.r.t. $Q^\star$, i.e., $\pi^\star(\cdot|s) \in \mathscr{P}\left(\arg\max_{a \in \mathcal{A}} Q^\star(s, a)\right)$.

Q-learning The *Bellman Optimal Operator* $T^\star : \mathscr{B}(\mathcal{S} \times \mathcal{A}) \to \mathscr{B}(\mathcal{S} \times \mathcal{A})$ is defined for every $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ and $(s, a) \in \mathcal{S} \times \mathcal{A}$ as [1]: $(T^\star f)(s, a) = r(s, a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s, a) \max_{a' \in \mathcal{A}} f(s', a')$. $T^\star$ is a γ-contraction in L_∞-norm and its unique fixed point is the optimal Q-function ($T^\star Q^\star = Q^\star$). When the P and r are known, the (action) value-iteration algorithm [7] allows to retrieve $Q^\star$ by means of the iterative application of $T^\star$. When the environment is unknown, Q-learning [13] collects samples with a *behavioral* policy (e.g., ϵ-greedy) and then uses them to update a Q-function estimate based on the updated rule: $Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a'))$, where $\alpha > 0$ is the learning rate.

Deep Q-Networks Deep Q-Network (DQN, [5]) employs a function approximator $Q_{\boldsymbol{\theta}}(s, a)$, parameterized by a deep neural network with weights $\boldsymbol{\theta}$ to estimate $Q^\star$. Interactions with the environment are stored in the *replay buffer* $\mathcal{D} = \{(s_t, a_t, r_{t+1}, s_{t+1})\}_{t=1}^n$. To improve stability, a *target network*, whose parameters $\boldsymbol{\theta}^-$ are kept fixed for a certain number of steps, is employed. The Q-Network is trained to minimize the mean squared temporal difference error $r + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}^-}(s', a') - Q_{\boldsymbol{\theta}}(s, a)$ on a batch of tuples sampled from the replay buffer $(s, a, r, s') \sim \mathcal{D}$.

Action Persistence The execution of actions with a persistence $k \in \mathbb{N}$ can be modeled by means of the k-persistent MDP [4], characterized by the k-persistent transition model P_k and reward function r_k. To formally define them, the *persistent transition model* is introduced: $P^\delta(\cdot, \cdot|s, a) = \int_{\mathcal{S}} P(\mathrm{d}s'|s, a) \delta_{(s', a)}(\cdot, \cdot)$, which replicates in the next state s' the previous action a. Thus, we have $P_k(\cdot|s, a) = \left((P^\delta)^{k-1} P\right)(\cdot|s, a)$ and $r_k(s, a) = \sum_{i=0}^{k-1} \gamma^i \left((P^\delta)^i r\right)(s, a)$. This framework eases the analysis of *fixed* persistences, but it does not allow the action repetition for a *variable* number of steps.

3 All-Persistence Bellman Update

We introduce our approach to make effective use of the samples collected at *any* persistence. In our framework, the agent chooses a primitive action a together with its persistence k with the introduction of the *persistence option*.

Definition 3.1 *Let $\mathcal{A}$ be the space of* primitive *actions of an MDP $\mathcal{M}$ and $\mathcal{K} := \{1, \dots, K_{\max}\}$, where $K_{\max} \geqslant 1$, be the set of persistences. A* persistence option *$o := (a, k)$ is the decision of playing primitive action $a \in \mathcal{A}$ with persistence $k \in \mathcal{K}$. We denote with $\mathcal{O}^{(k)} := \{(a, k) : a \in \mathcal{A}\}$ the set of options with fixed persistence $k \in \mathcal{K}$ and $\mathcal{O} := \bigcup_{k \in \mathcal{K}} \mathcal{O}^{(k)} = \mathcal{A} \times \mathcal{K}$.*

At any *acting step* t, the agent observes $s_t \in \mathcal{S}$, selects a persistence option $o_t = (a_t, k_t) \in \mathcal{O}$ and repeats the primitive action a_t for k_t times, observing the sequence of the states encountered and of the rewards collected. The next acting step is then $t + k_t$. During the execution of the persistence option, the agent is not allowed to change the primitive action. We now extend the policy and state-action value function definitions to consider this particular form of options. A *Markovian stationary policy over persistence options* $\psi : \mathcal{S} \to \mathscr{P}(\mathcal{O})$ is a mapping between states and probability measures over persistence options. We denote with Ψ the set of the policies of this nature. The state-option value function $Q^\psi : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ following a policy over options $\psi \in \Psi$ is defined as $Q^\psi(s, a, k) := \mathbb{E}_\psi \left[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s,\ a_0 = a, k_0 = k \right]$. In this context, the optimal action-value function is defined as: $Q^\star_{\mathcal{K}}(s, a, k) = \sup_{\psi \in \Psi} Q^\psi(s, a, k)$.

3.1 All-Persistence Bellman Operator

We recall that our goal is to leverage any $\overline{\kappa}$-persistence transition to learn $Q^\star_{\mathcal{K}}(\cdot, \cdot, k)\ \forall k \in \mathcal{K}$. Suppose that $\overline{\kappa} \geqslant k$, then, we can exploit any sub-transition of k steps from the $\overline{\kappa}$-persistence transition to update the value $Q^\star_{\mathcal{K}}(\cdot, \cdot, k)$. Thus, we extend the Bellman optimal operator to persistence options, introducing the operator $T^\star$. If, instead, $\overline{\kappa} < k$, in order to update the value $Q^\star_{\mathcal{K}}(\cdot, \cdot, k)$, we partially exploit the $\overline{\kappa}$-persistent transition, but then, we need to *bootstrap* from a lower persistence Q-value, to compensate the remaining $k - \overline{\kappa}$ steps. To this end, we introduce the *bootstrapping* operator $T^{\overline{\kappa}}$:

$$(T^\star f)(s,a,k) = r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s' | s, a) \max_{(a',k') \in \mathcal{O}} f(s', a', k'), \qquad \left(T^{\overline{\kappa}} f\right)(s,a,k) = r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s' | s, a) f(s', a, k - \overline{\kappa}). \quad (1)$$

By combining these two operators, we obtain the *All-Persistence Bellman operator* $\mathcal{H}_{\overline{\kappa}} : \mathscr{B}(\mathcal{S} \times \mathcal{O}) \to \mathscr{B}(\mathcal{S} \times \mathcal{O})$ defined for every $f \in \mathscr{B}(\mathcal{S} \times \mathcal{O})$ as $(\mathcal{H}^{\overline{\kappa}} f)(s, a, k) = \left(\mathbb{1}_{k \leqslant \overline{\kappa}} T^\star + \mathbb{1}_{k > \overline{\kappa}} T^{\overline{\kappa}} \right) f)(s, a, k)$. Thus, given a persistence $\overline{\kappa} \in \mathcal{K}$, $\mathcal{H}^{\overline{\kappa}}$ allows updating all the Q-values with $k \leqslant \overline{\kappa}$ by means of $T^\star$, and all the ones with $k > \overline{\kappa}$ by means of $T^{\overline{\kappa}}$. The following result demonstrates the soundness of the proposed operator.

Theorem 3.1 *The all-persistence Bellman operator $\mathcal{H}^{\overline{\kappa}}$ fulfills the following properties:*

(i) $\mathcal{H}^{\overline{\kappa}}$ is a γ-contraction in L_∞ norm;
(ii) $Q^\star_{\mathcal{K}}$ is its unique fixed point;
(iii) $Q^\star_{\mathcal{K}}$ is monotonic in k, i.e., for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ if $k \leqslant k'$ then $Q^\star_{\mathcal{K}}(s, a, k) \geqslant Q^\star_{\mathcal{K}}(s, a, k')$.

Thus, operator $\mathcal{H}^{\overline{\kappa}}$ contracts to the optimal action-value function $Q^\star_{\mathcal{K}}$, which, thanks to monotonicity, has its highest value at the lowest possible persistence. In particular, it is simple to show that $Q^\star_{\mathcal{K}}(s, a, 1) = Q^\star(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, i.e., by fixing the persistence to $k = 1$ we retrieve the optimal Q-function in the original MDP, and consequently, we can reconstruct a greedy optimal policy.

4 Persistent Q-learning

It may not be immediately clear what are the advantages of $\mathcal{H}^{\overline{\kappa}}$ over traditional updates. These become apparent with its empirical counterpart $\widehat{\mathcal{H}}^{\overline{\kappa}}_t = \mathbb{1}_{k \leqslant \overline{\kappa}} \widehat{T}^\star_t + \mathbb{1}_{k > \overline{\kappa}} \widehat{T}^{\overline{\kappa}}_t$, where:

$$\left(\widehat{T}^\star_t Q\right)(s_t, a_t, k) = r^k_{t+1} + \gamma^k \max_{(a',k') \in \mathcal{O}} Q(s_{t+k}, a', k'), \qquad \left(\widehat{T}^{\overline{\kappa}}_t Q\right)(s_t, a_t, k) = r^{\overline{\kappa}}_{t+1} + \gamma^{\overline{\kappa}} Q(s_{t+k}, a', k - \overline{\kappa}).$$

These empirical operators depend on the current *partial history*, which we define as: $H^{\overline{\kappa}}_t := (s_t, a_t, r_{t+1}, s_{t+1}, r_{t+2}, \dots, s_{t+\overline{\kappa}})$, used by Algorithm 1 to update each persistence in a backward fashion, to allow for an even faster propagation of values. At timestep t, given a sampling persistence $\overline{\kappa}_t$, for all sub-transitions of $H^{\overline{\kappa}}_t$, starting at $t + i$ and ending in $t + j$, we apply $\widehat{\mathcal{H}}^{j-i}_t$ to $Q(s_{t+i}, a_t, k + d)$, for all $d \leqslant K_{\max} - k$, where $k = j - i$. With these tools, it is possible to obtain the Persistent Q-learning algorithm (abbreviated as Per$Q$-learning), a persistent extension of Q-learning [13]: the agent follows a policy ψ^ϵ_Q, which is ϵ-greedy w.r.t. the option space and the current Q-function. This approach extends the MSA-Q-learning algorithm presented in [8], by bootstrapping higher persistence action values from lower ones.

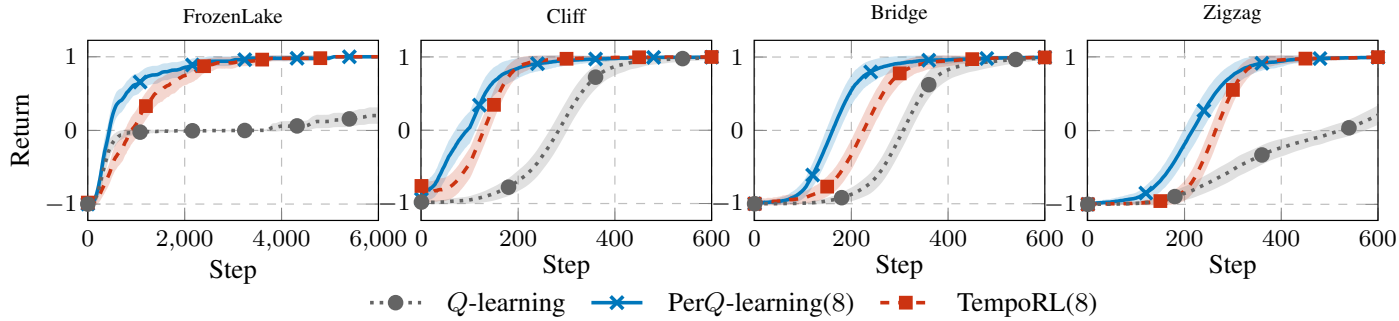


Figure 1: Performance evaluation on tabular environments, $K_{\max} = 8$. 50 runs (95% c.i.).

Algorithm 1 All Persistence Bellman Update

Require: Sampling persistence $\overline{\kappa}_t$,
partial history $H_t^{\overline{\kappa}_t}$, Q-function Q.
Ensure: Updated Q-function Q'

$Q' \leftarrow Q$
for $j = \overline{\kappa}_t, \overline{\kappa}_t - 1 \ldots, 1$ **do**
  for $i = j-1, j-2, \ldots, 0$ **do**
    $k \leftarrow j - i$
    $Q'(s_{t+i}, a_t, k) \leftarrow (1-\alpha) Q(s_{t+i}, a_t, k) + \alpha \widehat{T}^{\star}_{t+i} Q(s_{t+i}, a_t, k)$
    for $d = 1, 2, \ldots, K_{\max} - k$ **do**
      $Q'(s_{t+i}, a_t, k+d) \leftarrow (1-\alpha) Q(s_{t+i}, a, k+d) \alpha \widehat{T}^{k}_{t+i} Q(s_{t+i}, a_t, k+d)$
    end for
  end for
end for

The asymptotic convergence of Persistent Q-learning to $Q^{\star}_{\mathcal{K}}$ directly follows from the application of the results in [9], thanks to the fact that $\mathcal{H}^{\overline{\kappa}}$ is a contraction, provided that their (mild) assumptions are satisfied.

5 Persistent Deep Networks

In order to deal with high-dimensional settings, we develop an extension of Per$Q$-learning. It is straightforward to exploit Deep Q-Networks for learning in the options space $\mathcal{O}$. Standard DQN is augmented with $K_{\max}$ distinct sets of action outputs, to represent Q-value of the options space $\mathcal{O} = \mathcal{A} \times \mathcal{K}$, while the first layers are shared. The resulting algorithm, *Persistent Deep Q-Network* (PerDQN) is obtained by exploiting the application of the empirical all-persistence Bellman operator. The main differences between PerDQN and standard DQN consist in: (i) a modified ϵ-greedy strategy, which is equivalent to the one described for its tabular version; (ii) the use of *multiple* replay buffers accounting for persistence: whenever an option $o_t = (a_t, \overline{\kappa}_t)$ is executed, the generated partial history $H_t^{\overline{\kappa}_t}$ is decomposed in all its sub-transitions, which are stored in multiple replay buffers $\mathcal{D}_k$, one for each persistence $k \in \mathcal{K}$. Specifically, $\mathcal{D}_k$ stores tuples in the form $(s, a_t, s', r, \overline{\kappa})$, where s and s' are the first and the last state of the sub-transition, r is the $\overline{\kappa}$-persistent reward, and $\overline{\kappa}$ is a parameter to denote true length of the sub-transition, which will then be used to suitably apply $\widehat{\mathcal{H}}^{\overline{\kappa}}_t$.

Finally, the gradient update is computed by sampling a mini-batch of experience tuples from each replay buffer $\mathcal{D}_k$, in equal proportion. Given the current network and target parametrizations $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$, the temporal difference error of a sample $(s, a, r, s', \overline{\kappa})$ is computed as $\widehat{\mathcal{H}}^{\overline{\kappa}} Q_{\boldsymbol{\theta}^-}(s, a, k) - Q_{\boldsymbol{\theta}}(s, a, k)$. Our approach differs from TempoRL DQN [2], which uses a dedicated network to learn the persistence at each state and employs a standard replay buffer, ignoring the persistence at which samples have been collected.

6 Experimental Evaluation

Per$Q$-learning We present the results on the experiments in tabular environments, particularly suited for testing Per$Q$-learning because of the sparsity of rewards. We start with the deterministic 6x10 grid-worlds introduced by [2]. In these environments, the episode ends if either the goal or a hole is reached, with $+1$ or -1 points respectively. In all the other cases, the reward is 0, and the episode continues. Moreover, we experiment the 16x16 FrozenLake, with rewards and transition process analogous to the previous case, but with randomly generated holes at the beginning of the episode.

The results are shown in Figure 1, where we compared Per$Q$-learning with TempoRL (with the same maximum *skip-length* $J = 8$) and classic Q-learning. In all cases, Per$Q$-learning outperforms

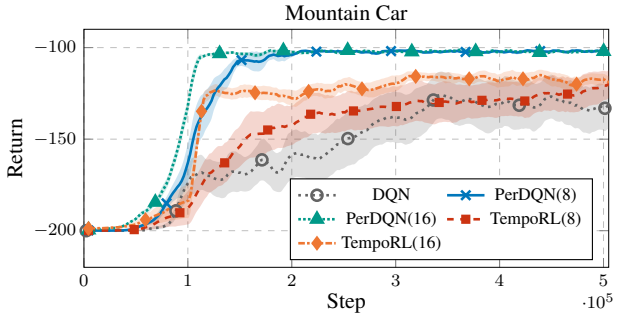


Figure 2: Performance on MountainCar. Parenthesis in the legend denote $K_{\max}$. 20 runs (avg$\pm$ 95% c.i.).

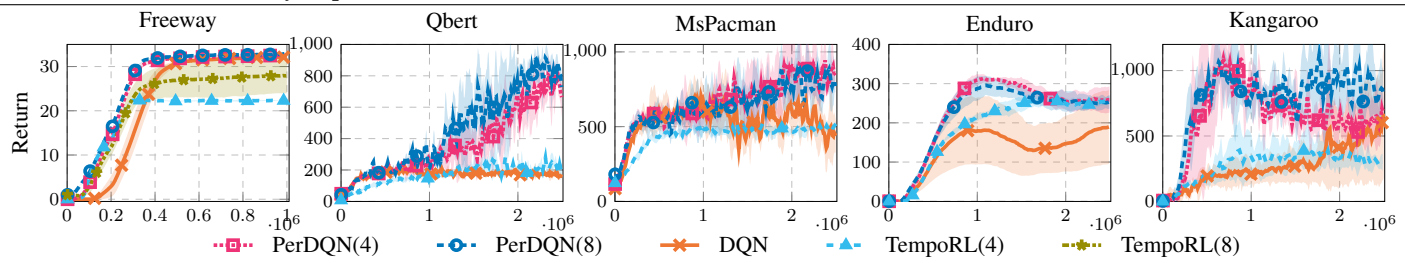


Figure 3: Atari games results for DQN and PerDQN. Parenthesis in the legend denote the maximum persistence K_{max}. 5 runs (avg$\pm$ 95% c.i.).

the other methods, especially Q-learning, whose convergence is significantly slower. In general, Per$Q$-learning shows faster rates of improvements than TempoRL, especially in the first learning iterations.

PerDQN In order to evaluate PerDQN performance, MountainCar is perhaps the most suited to evaluate the performance of persistence options, since 1-step explorative policies usually fail to reach the goal because of their low probability to commit to an action for long times [4]. Figure 2 shows that TempoRL and standard DQN cannot converge to the optimal policy, while PerDQN can reach the optimal solution, which consists in obtaining the minimum loss required to reach the top of the mountain. The algorithm is then tested in the challenging framework of Atari 2600 games. In Figure 3 we compare PerDQN, classic DQN and TempoRL. Our PerDQN displays a faster learning curve thanks to its ability of reusing experience, although in some cases (e.g. Kangaroo) PerDQN seems to inherit the same instability issues of DQN, we conjecture due to the overestimation bias [12].

7 Discussion and Conclusions

In this paper, we have considered RL policies that implement action persistence, modeled as *persistence options*, selecting a primitive action and its duration. We defined the *all-persistence* Bellman operator, which allows for an effective use of the experience collected from the interaction with the environment at any time scale. Thanks to this operator, action-value function estimates can be updated simultaneously on the selected persistence set: low persistences (and primitive actions) can be updated by splitting the samples in their sub-transitions; action value functions for high persistences can instead be improved by *bootstrap*, a procedure that takes into account the estimation of the partial missing information. After proving that the new operator is a contraction, we applied it to extend classic Q-learning and DQN with their persistent version. We performed an experimental campaign on tabular and deep RL settings demonstrating the effectiveness of our approach and the importance of considering temporal extended actions.

References

[1] Dimitri P Bertsekas and Steven E Shreve. *Stochastic optimal control: the discrete-time case*. Vol. 5. Athena Scientific, 1996.

[2] André Biedenkapp et al. TempoRL: Learning When to Act. In: *ICML*. 2021.

[3] Aravind S. Lakshminarayanan et al. Dynamic Action Repetition for Deep Reinforcement Learning. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2017, pp. 2133–2139.

[4] Alberto Maria Metelli et al. Control frequency adaptation via action persistence in batch reinforcement learning. In: *International Conference on Machine Learning (ICML)*. PMLR. 2020, pp. 6862–6873.

[5] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. In: *nature* 518.7540 (2015), pp. 529–533.

[6] Seohong Park et al. Time Discretization-Invariant Safe Action Repetition for Policy Gradient Methods. In: *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021).

[7] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[8] Ralf Schoknecht and Martin Riedmiller. Reinforcement learning on explicitly specified time scales. In: *Neural Computing & Applications* 12.2 (2003), pp. 61–80.

[9] Satinder Singh et al. Convergence results for single-step on-policy reinforcement-learning algorithms. In: *Machine learning* 38.3 (2000), pp. 287–308.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] Corentin Tallec et al. Making Deep Q-learning methods robust to time discretization. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6096–6104.

[12] Hado Van Hasselt et al. Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI conference on Artificial Intelligence (AAAI)*. Vol. 30. 1. 2016.

[13] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. PhD thesis. King's College, University of Cambridge, 1989.

Should Models Be Accurate?

Esra'a Saleh
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
esraa1@ualberta.ca

John D. Martin
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
jmartin8@ualberta.ca

Anna Koop
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
akoop@ualberta.ca

Arash Pourzarabi*
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
pourzara@ualberta.ca

Michael Bowling
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
mbowling@ualberta.ca

Abstract

Model-based Reinforcement Learning (MBRL) holds promise for data-efficiency by planning with model-generated experience in addition to learning with experience from the environment. However, in complex or changing environments, models in MBRL will inevitably be imperfect, and their detrimental effects on learning can be difficult to mitigate. In this work, we question whether the objective of these models should be the accurate simulation of environment dynamics at all. We focus our investigations on Dyna-style planning in a prediction setting. First, we highlight and support three motivating points: a perfectly accurate model of environment dynamics is not practically achievable, is not necessary, and is not always the most useful anyways. Second, we introduce a meta-learning algorithm for training models with a focus on their usefulness to the learner instead of their accuracy in modelling the environment. Our experiments show that in a simple non-stationary environment, our algorithm enables faster learning than even using an accurate model built with domain-specific knowledge of the non-stationarity.

Keywords: Model-based Reinforcement Learning, Planning, Model Learning, Meta Learning, Dyna, Sample Models

Acknowledgements

We are grateful for the generous support in funding this work. This work was funded by the Natural Sciences and Engineering Research Council, the Alberta Machine Intelligence Institute, and the Canadian Institute For Advanced Research. Computational resources were generously provided by Compute Canada.

*Arash Pourzarabi was killed when Flight PS752 was shot down by an Iranian surface-to-air missile on January 8, 2020. His unfinished research on training generative models to produce experience for model-based RL was a signficant influence on the direction taken in this work.

1 Introduction

A promising approach to sample efficiency in reinforcement learning is model-based RL, where an agent learns a model of the environment. The agent uses this model to generate its own *internal experience* with which it can plan, reducing the need for more expensive *veridical experience*, i.e., environment interactions. Learning effective models for planning remains a challenge in complex or changing environments [6]. Models for planning are traditionally trained to accurately simulate environment dynamics. In complex or changing environments, models will necessarily be imperfect, which can lead to detrimental effects on the learner.

Past work has used several strategies to grapple with imperfect models. The first is to compensate in the planning process. An imperfect model can be used selectively by estimating predictive uncertainty to avoid planning in states where it could be detrimental [1]. Similarly, while longer sequences of internal experience can often be beneficial in planning, their length needs to be limited due to increasing compounding model error [5]. Another strategy is to relax the requirement for models to accurately simulate environment transitions by focusing only on accuracy in the resulting returns. This has been formally articulated through the value equivalence principle [4, 3] and manifested in MuZero [7].

Our work questions the need for focusing on models that are concerned with accurately simulating the environment. We specifically study Dyna-style planning [9] in a prediction setting to provide a building block towards the control setting. We design a simple non-stationary environment that cannot be modelled perfectly without assumptions on how it changes. Then, we use meta-learning to train a model that generates synthetic transitions for fast adaptive learning. The resulting learning speed upon change in the environment makes it competitive even with a stable accurate model of environment dynamics for an agent with privileged access to regions of stability.

2 Should Models Be Accurate?

An important way RL systems can learn about their environment is by anticipating the outcomes of different behaviors using both veridical and internal experience. In the prediction setting, an RL system experiences a stream of observation vectors $\mathbf{o}_t \in \mathbb{R}^d$ and scalar rewards r_t. From this single stream of data, $\mathbf{h}_t \triangleq \mathbf{o}_0, r_1, \mathbf{o}_2, r_2, \mathbf{o}_3, \cdots r_{t-1}, \mathbf{o}_t$, the learner forms an approximate value function (i.e., a prediction) to estimate the expected sum of future discounted rewards. With linear function approximation, predictions are made by projecting a feature vector $\boldsymbol{\phi}_t = \mathbf{f}(\mathbf{h}_t) \in \mathbb{R}^n$ with weights $\boldsymbol{\theta}$:

$$\hat{v}(\boldsymbol{\phi}_t; \boldsymbol{\theta}_t) \triangleq \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t, \qquad \hat{v}(\boldsymbol{\phi}_t; \boldsymbol{\theta}) \approx v(\mathbf{h}_t) \triangleq \mathbf{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | H_t = \mathbf{h}_t].$$

Model-based learners update their prediction weights $\boldsymbol{\theta}$ by planning with internal experience produced by a model $m \in \mathcal{M} \subseteq \mathcal{P}(\mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n)$. In this work, models combine the selection of initial states[1] with the dynamics of a forward transition. At each planning step, the model generates an internal starting state $\tilde{\boldsymbol{\phi}}$, a next state $\tilde{\boldsymbol{\phi}}'$, and a reward $\tilde{r}$. Given this transition sample $(\tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}') \sim m$, an update rule can be applied to the prediction weights $\boldsymbol{\theta}$.

Dyna-style learning systems interleave planning steps with model-free updates on *veridical experience* $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$, which come from the environment's dynamics $m^* \in \mathcal{M}$. In our work, Dyna learners apply semi-gradient updates to reduce one-step temporal difference (TD) errors $\delta \triangleq r + \gamma \boldsymbol{\theta}^\top \boldsymbol{\phi}' - \boldsymbol{\theta}^\top \boldsymbol{\phi}$, using a fixed step size $\alpha \in \mathbb{R}^+$ at each time step: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \boldsymbol{\phi}$. This update is used with both veridical and internal experience.

In addition to learning predictions, model-based systems must also learn the planning model m. These systems are often designed with an *accuracy* objective—to approximate the environment dynamics so that eventually $m \approx m^*$. Using an accurate model, the learner stands to benefit from additional transition samples that simulate the veridical experience. Still, this raises an important question: should a learner strive for its model to be accurate? While accurate models might represent a subset of models that can be useful for learning, they do not represent the entirety of the set. Insisting on model accuracy, rather than model utility, can be a hindrance to fast adaptive learning. We expound on this with three observations.

First, **perfect accuracy in a model is not practically achievable**. Although accuracy is a well-defined objective for model learning, it represents a condition that cannot be achieved. Accuracy is only achieved when the planning model and environment agree—specifically when $m = m^*$. In realistic environments, a learning system can never achieve this condition, because the real world is too big and complex (including other learning agents such as itself) to be discovered and represented with a finite amount of compute. Therefore, fully-accurate models can never be learned.

Second, **even when possible, an accurate model is not necessary**. In spite of the barrier to perfect accuracy, prior work shows that other models can work just as well. For instance, the class of value-equivalent models are those which produce all the same value updates as m^*, but may transition under different dynamics [4]. Since they predict the same values, planning with value-equivalent models will lead to the same solution as an accurate model. Systems such as Predictron

[1] In Dyna-style planning the selection of an initial state (and action) for planning is sometimes called search control [9].

[8] and MuZero [7] are successful examples which also show the benefit of connecting the model learning process to the way it is used for planning and value improvement. More importantly, this line of work shows that even if accuracy could be achieved, fully-accurate models are not necessary.

Third, **an accurate model might not be the most useful**. In this work, planning models are used to approximate the true value function $v \in \mathcal{V}$ by reducing squared TD error with semi-gradient updates. Solutions to this objective come from the set of TD fixed points $\boldsymbol{\vartheta} \subset \mathcal{V}$. Fixed points can be indexed with the model whose experience produced them: $\boldsymbol{\theta}^{(m)} \in \mathcal{V}$.

Given that fully-accurate models cannot be achieved, nor are they needed to reach a fixed point, one may wonder whether there exists models that are *more useful* than others, or even more useful than the environment model! A natural measure of a model's utility could be the number of calls required to approach a TD fixed point. With this, consider a subset of models $\mathcal{M} \subset \mathbb{M}$ with the same TD fixed point as the environment model m^*: $\mathcal{M} \triangleq \{m \in \mathbb{M} | \boldsymbol{\theta}^{(m)} \in \boldsymbol{\vartheta}\}$. The following result shows that there can be models that find solutions faster than the environment model when starting from the same point.

Theorem 1. *There are domains where, for some initial $\boldsymbol{\theta} \in \mathcal{V}$, a TD fixed point $\boldsymbol{\theta}^* \in \boldsymbol{\vartheta}$ can be approached with fewer calls to $m \in \mathcal{M}$ than with the domain dynamics $m^* \neq m$.*

Proof Sketch. Consider a domain that contains k states, which share the same outgoing transitions and value. Suppose the agent is employing a linear function approximator with states represented with a one-hot encoding. Consider a model that produces internal experience from the k-hot vector containing these states, thus treating them as a single state. Clearly $m \neq m^*$, and existing results from the state aggregation literature show such aggregations exist and do not change the TD fixed point. Therefore, $m \in \mathcal{M}$. However, it would take k calls to m^* to produce the update produced by one call to m. □

In short, a perfectly-accurate model is not practically achievable depending on the complexity of an environment, is not necessary by the value equivalence principle, and it might not be the most useful for a learner's objectives. A useful model in the prediction setting is one that helps the learning system obtain an accurate value function.

3 A Demonstration of Useful Models

In this section, we demonstrate how useful models can be trained to aid fast learning. We focus on a prediction problem in a non-stationary environment where Dyna-style planning is used to rapidly adapt to changes in the environment. The environment is constructed so that, despite being simple, an agent's state representation is not sufficient to build an accurate model. We first describe the environment. Then, we present baselines that represent both model-free RL and model-based RL with models aimed at accuracy. After that, we introduce our algorithm that uses meta-learning to train a model aimed at usefulness. We seek to answer the question: *can an algorithm that learns a useful, but not necessarily accurate model, result in faster learning than either an accurate model or an equivalent model-free learning approach?*

Non-Stationary Windy Hallway. To empirically compare the learning performance of planning models, we use a non-stationary windy hallway: a gridworld of 6 columns and 3 rows. The last column contains 3 episode-terminating grid cells. The starting location is in row 1 and column 0. In every timestep, an agent either moves one grid cell North-East or South-East, each with a probability of 0.5 (or possibly just East if already in a northern or southern-most cell). Every 300 episodes, the environment switches between two different reward regimes, where the location of non-zero reward changes. In one regime, reward is +1 for terminating in row 0, and 0 otherwise. In the other, reward is -1 for terminating in row 2 and 0 otherwise. All algorithms use a discount factor of $\gamma = 0.9$, and planning algorithms perform 5 planning updates per environment step. Experiments run for 75,000 steps (15,000 episodes) and each algorithm's value-function estimates are compared against the true expected return.

Baselines. Based on the environment, we select baselines to highlight learning speed when no planning is done and when Dyna-style planning is done with models aiming for accuracy. For our representative algorithm without planning, we use TD(0), which employs value updates from veridical experience only. We call this algorithm *Model-free*. To keep things simple, our baseline representatives of algorithms with Dyna-style planning do not explicitly train a model at all, but replay back veridical experience as the extreme of optimizing for accuracy. We examine two alternatives for how past experience transition tuples are sampled. The simplest option is to store every experienced transition tuple. A model will simply sample transition tuples proportional to how often they were experienced in the past. We call the Dyna algorithm using this model *AllExperienceDyna*. If the environment were Markov in the agent's state representation, this model would approach a perfectly accurate model. As our environment is not Markov, this approach will result in obvious problems as internal experience from different reward regimes can distract it from the true expected return. Our second baseline uses domain-specific knowledge of the environment to only store and resample veridical transitions tuples that are Markov in the agent's state representation (i.e., the non-terminating state transitions). This experience is stable across changes in the reward regime and we call the Dyna algorithm with this model *StableExperienceDyna*. This approach, in many ways, is an

unfair comparison, as it is exploiting domain knowledge to construct a stable model. We could, though, see this as an idealized representative of Abbas and colleagues' approach that does selective planning based on "model inadequacy" [1], which is analogous to using a stable model.

SynthDyna. In Algorithm 2, we propose a proof-of-concept algorithm that we call *SynthDyna*. SynthDyna's model, unlike our planning baselines, is (i) learned, and (ii) generates synthetic internal planning experience that does not aim to accurately simulate environment dynamics. In designing SynthDyna, we seek to demonstrate that giving the model control over the representation of a transition for planning while having a model learning objective informed by utility to the learning system, can result in faster adaptation in our reward switching non-stationary environment. Like a typical Dyna architecture, SynthDyna updates its value function parameters using both veridical experience transitions and internal planning transitions from its model. Unlike our other Dyna representatives, SynthDyna's model is a generative model, using a 2-layer fully connected neural network with parameters $\boldsymbol{\eta}$ that takes in a Gaussian noise vector as input and produces a transition tuple $(\tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}')$. SynthDyna trains this generative model with a meta-learning procedure that uses a meta-loss tied to the learner's primary objective. As seen in line 6 of Algorithm 2, SynthDyna collects every experienced veridical transition $(\boldsymbol{\phi}_{t-1}, r_t, \boldsymbol{\phi}_t)$ and it also collects the value function parameters, $\boldsymbol{\theta}_p$ from the previous step before planning updates were performed. At a given frequency of steps in the environment, SynthDyna samples a batch of tuples where each is $(\boldsymbol{\theta}_p, \boldsymbol{\phi}, r, \boldsymbol{\phi}')$. It then uses that batch's samples as inputs to the metaloss; see Algorithm 1. The metaloss carries out the process of planning with SynthDyna's model for k steps updating $\boldsymbol{\theta}_p$ from the batch. It then evaluates the squared TD-error, on the stored veridical transition $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$ from the batch. To prevent the TD-target in the error from being manipulated before this evaluation step, the value function parameters in the target are set as the initial parameters $\boldsymbol{\theta}$. The gradient of the resulting meta-loss is then used to update the model parameters $\boldsymbol{\eta}$ to improve the model.

Algorithm 1 $\mathcal{L}$: SynthDyna Meta Loss

1: **input:** $\boldsymbol{\theta}$, $\boldsymbol{\phi}$, r,$\boldsymbol{\phi}'$
2: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
3: **for** $1, \cdots, k$ **do**
4: $\quad \tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}' \sim m(\boldsymbol{\eta})$
5: $\quad \tilde{\delta} \leftarrow \tilde{r} + \gamma \boldsymbol{\theta}'^{\top} \tilde{\boldsymbol{\phi}}' - \boldsymbol{\theta}'^{\top} \tilde{\boldsymbol{\phi}}$
6: $\quad \boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}' + \zeta \tilde{\delta} \tilde{\boldsymbol{\phi}}$
7: **return:** $(r + \gamma \boldsymbol{\theta}^{\top} \boldsymbol{\phi}' - \boldsymbol{\theta}'^{\top} \boldsymbol{\phi})^2$

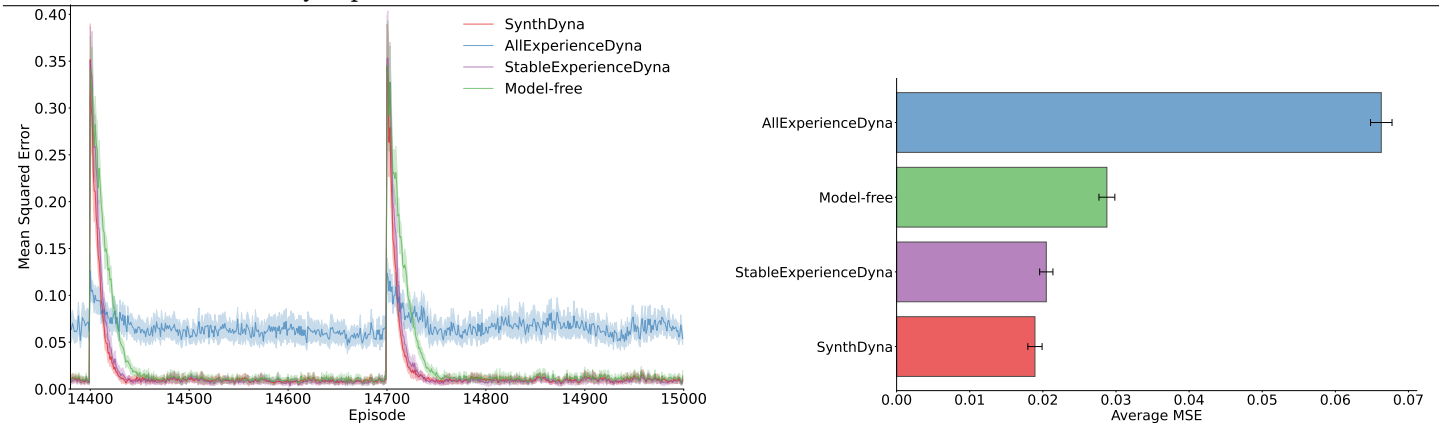
Algorithm 2 SynthDyna for Prediction

1: **input:** feature transform f
2: **initialize:** $\boldsymbol{\theta}_0$, $\boldsymbol{\eta}$, $\mathcal{D} \leftarrow \{\}$, $\boldsymbol{\phi}_0 \leftarrow f(\mathbf{h}_0)$
3: **for** $t = 1, 2, \cdots$ **do**
4: $\quad$ Observe $\mathbf{o}_t$ and r_t from the environment.
5: $\quad \boldsymbol{\phi}_t \leftarrow f(\mathbf{h}_t)$
6: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{\theta}_p, \boldsymbol{\phi}_{t-1}, r_t, \boldsymbol{\phi}_t\}$
7: $\quad$ // Update value with veridical experience.
8: $\quad \delta_t \leftarrow r_t + \gamma \boldsymbol{\theta}_{t-1}^{\top} \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^{\top} \boldsymbol{\phi}_{t-1}$
9: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \alpha \delta_t \boldsymbol{\phi}_{t-1}$
10: $\quad \boldsymbol{\theta}_p \leftarrow \boldsymbol{\theta}_t$ // Save $\boldsymbol{\theta}_t$ for model training
11: $\quad$ // Update value with internal experience.
12: $\quad$ **for** $1, \cdots, k$ **do**
13: $\quad\quad \tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}' \sim m(\boldsymbol{\eta})$
14: $\quad\quad \tilde{\delta} \leftarrow \tilde{r} + \gamma \boldsymbol{\theta}_t^{\top} \tilde{\boldsymbol{\phi}}' - \boldsymbol{\theta}_t^{\top} \tilde{\boldsymbol{\phi}}$
15: $\quad\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \beta \tilde{\delta} \tilde{\boldsymbol{\phi}}$
16: $\quad$ // Update SynthDyna model with metaloss $\mathcal{L}$ (Alg 1).
17: $\quad$ Sample minibatch $\mathcal{B}$ from $\mathcal{D}$.
18: $\quad \boldsymbol{\eta} \leftarrow \text{Adam}(\boldsymbol{\eta}, \mathcal{B}, \mathcal{L})$

We hypothesize that the combination of SynthDyna's meta-learned model that aims to be useful rather than accurate, and giving the model full control over the representation of the internal planning transitions will yield a model that can surpass the learning speed of all baselines including our strongest baseline, StableExperienceDyna, that is unfairly informed of stable transitions in the environment.

Results. We evaluate SynthDyna against our three baselines in the the non-stationary hallway environment described previously. Every algorithm's hyper-parameters are selected after a comprehensive grid search. The primary metric we focus on is the mean squared value error (MSE) per episode, which measures the average squared error of the predicted return compared to the expected return over the timesteps in an episode. Let $\boldsymbol{\theta}$ be the value function's parameters before an update based on a veridical transition, $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$, (i.e., line 9 of Algorithm 2). For an episode of length n, starting at timestep e, the mean squared value errors per episode is computed with: $\text{MSE} = \frac{1}{n} \sum_{t=e}^{e+n} (v(\mathbf{h}_t) - \boldsymbol{\phi}_t^{\top} \boldsymbol{\theta}_t)^2$.

Figure 1a shows the MSE per episode for the last two reward-regime switches. Figure 1b highlights the average MSEs for each algorithm for the last 600 episodes. AllExperienceDyna has the poorest performance across baselines due to its planning process that updates the value function with old experience that could be in direct opposition to the current reward regime. In an environment like this one, the much simpler Model-free algorithm performs substantially better as each update to the value function is exclusively done with the present moment's veridical experience. StableExperienceDyna is able to surpass the performance of AllExperienceDyna and Model-free demonstrating the value of model-based RL for fast adaptation in this domain. It can reap the benefits of planning by updating based on past transition tuples that are stable across switching reward regimes. SynthDyna outperforms even StableExperienceDyna with a statistically significant difference between the average MSEs over the last 600 episodes (two sample t-test, $p < 0.05$). This shows that a model like SynthDyna's can be built in reward switching non-stationary environment to produce



(a) Mean squared error per episode for each algorithm

(b) Average MSE per algorithm

Figure 1: We ran each algorithm in the non-stationary windy hallway environment for 15,000 episodes, with 30 trials. Fig. 1a, shows the mean squared value error per episode for the last 600 episodes. Fig. 1b, shows the average MSE over the last 600 episodes. Shaded regions and error bars show 95% confidence intervals.

completely synthetic transition tuples and yet update the value function more effectively than if a stable and accurate model of the world was used.

4 Conclusion

"All models are wrong but some are useful" [2]. In this work, we questioned whether models in MBRL should aim to accurately simulate environment dynamics. We posited an alternative approach that models should aim to be useful to the learning system, aiding in faster adaptation. As a proof of concept, we introduced SynthDyna, which uses meta-learning to train a model for Dyna-style planning such that it directly reduces prediction error. In our experiments, SynthDyna outperforms model-based baselines focused on accuracy. Focusing on model usefulness and the approach taken by SynthDyna are promising directions for MBRL to realize its potential for general, sample efficient RL.

References

[1] Z. Abbas, S. Sokota, E. Talvitie, and M. White. Selective Dyna-style planning under limited model capacity. In *International Conference on Machine Learning*, pages 1–10. PMLR, 2020.

[2] G. E. Box. Robustness in the strategy of scientific model building. In *Robustness in statistics*, pages 201–236. Elsevier, 1979.

[3] A.-m. Farahmand, A. Barreto, and D. Nikovski. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494. PMLR, 2017.

[4] C. Grimm, A. Barreto, S. Singh, and D. Silver. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020.

[5] G. Z. Holland, E. J. Talvitie, and M. Bowling. The effect of planning shape on Dyna-style planning in high-dimensional state spaces. *arXiv preprint arXiv:1806.01825*, 2018.

[6] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[8] D. Silver, H. Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, et al. The predictron: End-to-end learning and planning. In *International conference on machine learning*, pages 3191–3199. PMLR, 2017.

[9] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Option Discovery for Autonomous Generation of Symbolic Knowledge

Gabriele Sartor
University of Turin
Via Verdi, 8
10124 Torino, Italy
gabriele.sartor@unito.it

Davide Zollo
Roma Tre University
Via della Vasca Navale, 79
00146 Roma, Italy
davidedezollo@gmail.com

Marta Cialdea Mayer
Roma Tre University
Via della Vasca Navale, 79
00146 Roma, Italy
cialdea@ing.uniroma3.it

Angelo Oddi
ISTC-CNR
Via San Martino della Battaglia, 44
00185 Rome, Italy
angelo.oddi@istc.cnr.it

Riccardo Rasconi
ISTC-CNR
Via San Martino della Battaglia, 44
00185 Rome, Italy
riccardo.rasconi@istc.cnr.it

Vieri Giuliano Santucci
ISTC-CNR
Via San Martino della Battaglia, 44
00185 Rome, Italy
vieri.santucci@istc.cnr.it

Abstract

In this work we present an empirical study where we demonstrate the possibility of developing an artificial agent that is capable to autonomously explore an experimental scenario. During the exploration, the agent is able to discover and learn interesting options allowing to interact with the environment without any pre-assigned goal, then abstract and re-use the acquired knowledge to solve possible tasks assigned ex-post. We test the system in the so-called Treasure Game domain described in the recent literature and we empirically demonstrate that the discovered options can be abstracted in an probabilistic symbolic planning model (using the PPDDL language), which allowed the agent to generate symbolic plans to achieve extrinsic goals.

Keywords: Options, Intrinsic motivations, Planning

Acknowledgements

We are deeply indebted to George Konidaris and Steve James for making both the Skills to Symbols and the Treasure Game software available.

1 Introduction

In this work we want to show the possibility of developing an artificial system capable of "closing the loop" between low-level autonomous learning and high-level symbolic planning. In a previous work [7] the authors presented a promising approach, trying to combine learning in high-dimensional sensors and actuators spaces through the option framework with high-level, symbol-based, decision making through planning techniques. They proposed an algorithm able to abstract the knowledge of the system into symbols that can then be used to create sequences of operators to solve complex tasks in both simulated and real robotic scenarios. In particular, they assumed options were already given to the agent and then tested the capability of their algorithm to generate proper symbolic knowledge for a PDDL domain.

The general problem addressed in this paper is therefore to develop a system that effectively starts from learning options that can then be abstracted through the algorithm in [7] and used to solve through planning a task in which it is necessary to perform a sequence of different actions. Option learning has been thoroughly studied in the literature: in particular, when assigned with a goal, a system can leverage on this assigned task to discover or generate sub-goals that can be learnt and encapsulated into options [16]. In the Treasure Game domain that we are considering here (see [7]), given the assigned task of reaching for the treasure, the different passages of climbing up and down a ladder, pulling levers, picking keys, etc., can be identified as components of the sequence of actions needed to achieve the final goal, and thus learnt through any learning algorithm, described as options and chunked together in the proper sequence.

However, the perspective we take in this work is that of an agent who has to learn autonomously to interact with the environment, without necessarily being aware of what tasks will be later assigned to it. We make this assumption because it forces us to develop versatile and adaptive agents that can be used in unknown and unstructured environments. In this perspective, the general problem described above becomes the more specific one of autonomously identifying which options to learn. Similarly to what done in [15], we need to provide the agent with a general criterion by which it can identify states and/or events in the world that can be the target of specific options. Given that we do not know what task the agent will be asked to perform, we will leverage intrinsic motivations [9] to identify potentially interesting states and use them to build options.

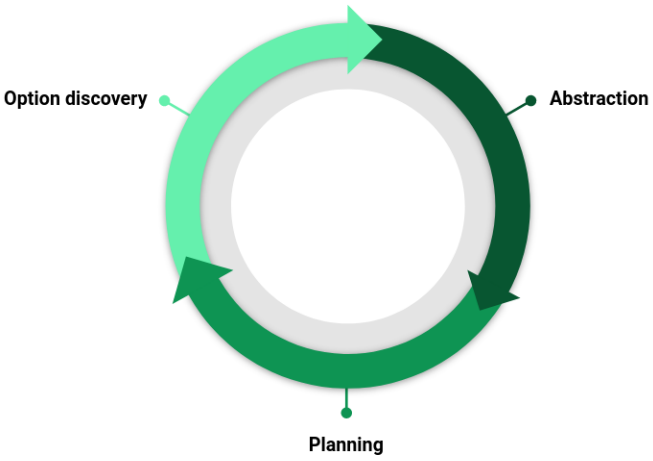


Figure 1: The proposed conceptual framework.

The field of Intrinsically Motivated Open-ended Learning (IMOL, [13]) is showing promising results in the development of versatile and adaptive artificial agents. *Intrinsic Motivations* (IMs, [10]) are a class of self-generated signals that have been used to provide robots with an autonomous guidance for several different processes, from state-and-action space exploration [5], to the autonomous discovery, selection and learning of multiple goals [12, 3]. In general, IMs guide the agent in the acquisition of new knowledge independently (or even in the absence) of any assigned task: this knowledge will then be available to the system to solve user-assigned tasks [14] or as a scaffolding to acquire new knowledge in a cumulative fashion [11, 4] (similarly to what have been called curriculum learning [1]).

To connect this process of autonomous option discovery with the generation of high-level planning procedures, following our preliminary works [8], we adopt a hierarchical approach aimed at developing a robotic architecture capable of holding together the different mechanisms needed to close the loop between low and high level learning representations. In details, the idea depicted in Figure 1 highlights the necessity of three different critical capabilities (or conceptual modules) that every robotic agent must have in order to operate autonomously.

The *option discovery* module combines primitives to create options and learns precondition and effect models. Then, these collected data are used by the *abstraction* module to generate a PPDDL domain containing operators following the *abstract sub-goal option* property. Finally, the PPDDL description can be used by an off-the-shelf planner to reach any sub-goal which can be described with the available high-level symbols. Potentially, the robotic agent can continue to execute these steps in a loop extending its knowledge and capabilities over time, exploitable by the human who can ask to reach a certain goal expressed in automatically generated symbols. In the following sections we experimentally show that the discovered options can be abstracted in a probabilistic symbolic planning model (in PPDDL language), which allowed the agent to generate symbolic plans to achieve extrinsic goals.

2 Option discovery

Options are temporally-extended actions defined as $o(I, \pi, \beta)$ [17], in which π is the policy executed, I the set of states in which the policy can run and β the termination condition of the option. The option's framework revealed to be an effective tool to abstract actions and extend them with a temporal component. The use of this kind of actions demonstrated to improve significantly the performances of model-based Reinforcement Learning compared to older models, such as one-step models in which the actions employed are the primitives of the agent [18]. Intuitively, these low-level single-step actions, or *primitives*, can be repeatedly exploited to create more complex behaviours.

In this section, we describe a possible way to discover and build a set of options from scratch. using the low-level actions available in the *Treasure Game* environment (see Fig. 2). In such environment, an agent starts from its initial position (home), moves through corridors and climbs ladders over different floors, while interacting with a series of objects (e.g., keys, bolts, and levers) to the goal of reaching a treasure placed in the bottom-right corner and bringing it back home.

Figure 2: The Treasure Game configuration used for the experimental analysis.

In order to build new behaviours, the agent can execute the following primitives: $1)go\_up$, $2)go\_down$, $3)go\_left$, $4)go\_right$, and $5)interact$, respectively used to move the agent up, down, left or right by 2-4 pixels (the exact value is randomly selected with a uniform distribution) and to interact with the closest object. In particular, the interaction with a lever changes the state (open/close) the doors associated to that lever (both on the same floor or on different floors) while the interaction with the key and/or the treasure simply collects the key and/or the treasure inside the agent's bag. Once the key is collected, the interaction with the bolt unlocks the last door, thus granting the agent the access to the treasure.

In our experiment, primitives are used as building blocks in the construction of the option, participating to the definition of π, I and β. In more details, we create new options from scratch, considering a slightly different definition of option $o(p, t, I, \pi, \beta)$ made up of the following components: p, the primitive used by the execution of π; t, the primitive which, when available, stops the execution of π; π, the policy applied by the option, consisting in repeatedly executing p until t is available or p can no longer be executed; I, the set of states from which p can run; β, the termination condition of the action, corresponding to the availability of the primitive t or to the impossibility of further executing p. Consequently, this definition of option requires p, to describe the policy and where it can run, and t, to define the condition stopping its execution, maintaining its characteristic temporal abstraction. For the sake of simplicity, the option's definition will follow the more compact syntax $o(p, t)$ in the remainder of the paper.

Algorithm 1 describes the process utilized to discover new options autonomously inside the simulated environment. The procedure runs for a number of episodes max_eps and max_steps steps. Until the maximum of steps of the current episode is not reached, the function keeps track of the starting state s and randomly selects an available primitive p, such that p can be executed in s (lines 8 and 9). Then, as long as p is available and there is no new available primitives (lines 10 and 11), the option p is executed, and the final state s' of the current potential option

Algorithm 1 Discovery option algorithm

```
1:  procedure DISCOVER(env, max_eps, max_steps)
2:      options ← {}
3:      ep ← 0
4:      while ep < max_eps do
5:          T ← 0
6:          env.RESET_GAME()
7:          while T < max_steps do
8:              s ← env.GET_STATE()
9:              p ← env.GET_AVAILABLE_PRIMITIVE()
10:             while (env.IS_AVAILABLE(p) and
11:                     not (env.NEW_AVAILABLE_PRIM())) do
12:                 env.EXECUTE(p)
13:                 s' ← env.GET_STATE()
14:             if s ≠ s' then
15:                 if env.NEW_AVAILABLE_PRIM() then
16:                     t ← env.GET_NEW_AVAILABLE_PRIM()
17:                     op ← CREATE_NEW_OPTION(p, t)
18:                 else
19:                     op ← CREATE_NEW_OPTION(p, {})
20:                 options ← options ∪ op
21:     return options
```

Figure 3: The algorithm discovering options in the environment.

```
1. go_down [to 5th floor]    2. go_left [to handle]       3. interact(handle)
4. go_right [to wall]        5. go_down [to 4th floor]    6. go_right [to handle]
7. interact(handle)          8. go_left [to key]          9. interact(key)
10. go_right [to stairs]     11. go_down [to 3rd floor]   12. go_left [to stairs]
13. go_down [to 1st floor]   14. go_left [to bolt]        15. interact(bolt, key)
16. go_right [to wall]       17. go_up [to 2nd floor]     18. go_right [to treasure]
19. interact(treasure)       20. go_left [to stairs]      21. go_down [to 1st floor]
22. go_left [to bolt]        23. go_right [to stairs]     24. go_up [to 3rd floor]
25. go_right [to wall]       26. go_left [to stairs]      27. go_up [to 4th floor]
28. go_right [to handle]     29. interact(handle)         30. go_left [to stairs]
31. go_up [to 5th floor]     32. go_left [to stairs]      33. go_up [home]
```

Figure 4: Plan generated by the mGPT planner with our autonomously synthesized PPDDL domain.

is updated. The function *NEW_AVAILABLE_PRIM* returns **True** when a primitive which was not previously executable becomes available while executing p; the function returns **False** in all the other cases. For instance, if the agent finds out that there is a ladder over him while executing the go_right option, the primitive go_up gets available and the function return **True**. In other words, *NEW_AVAILABLE_PRIM* detects the interesting event, thus implementing the surprise element that catches the agent's curiosity. For this reason, the primitive representing the exact reverse with respect to the one currently being executed is not interesting for the agent, i.e., the agent will not get interested in the go_right primitive while executing go_left. The same treatment applied to the (go_left, go_right) primitive pair is also used with the pair (go_up, go_down). When the stopping condition of the most inner while is verified and $s \neq s'$, a new option can be generated according to the following rationale. In case the **while** exits because of the availability of a new primitive t in the new state s', a new option $o(p, t)$ is created (line 17); otherwise, if the **while** exits because the primitive under execution is no longer available, a new option $o(p, \{\})$ is created, meaning *"execute p while it is possible"* (line 19). In either case, the created option op is added to the list $options$ (line 20), which is the output of the function. In our test scenario, the algorithm generated 11 working options (see Section 3), suitable for solving the environment, and collected experience data to be abstracted in PPDDL [19] format successively. Consequently, as we introduced above, the agent performs two learning phases: the first, to generate options from scratch and creating a preliminary action abstraction, and the second, to produce a higher representation partitioning the options and highlighting their causal effects. The latter phase, producing a symbolic representation suitable for planning, is analyzed in the next section.

3 Empirical Analysis

In this section we describe the results obtained from a preliminary empirical study, carried out by testing the Algorithm in Fig. 3 in the context of the Treasure Game domain [7]. The algorithm was implemented in Python 3.7 under Linux Ubuntu 16.04 as an additional module of the *Skill to Symbols* software, using the *Treasure Game* Python package. As previously stated, the Treasure Game domain defines an environment that can be explored by the agent by moving through corridors and doors, climbing stairs, interacting with handles (necessary to open/close the doors), bolts, keys (necessary to unlock the bolts) and a treasure. In our experimentation, the agent starts endowed with no previous knowledge about the possible actions that can be executed in the environment; the agent is only aware of the basic motion primitives at his disposal, as described in Section 2. The goal of the analysis is to assess the correctness, usability and quality of the abstract knowledge of the environment autonomously obtained by the agent. The experiment starts by using the algorithm showed in Figure 3, whose application endows the agent with the following set of learned options (11 in total):

$$\begin{aligned} O = \{&(go\_up, \{\}), (go\_down, \{\}), (go\_left, \{\}), (go\_left, go\_up), (go\_left, go\_down), (go\_left, interact), \\ &(go\_right, \{\}), (go\_right, go\_up), (go\_right, go\_down), (go\_right, interact), (interact, \{\})\} \end{aligned} \tag{1}$$

The test has been run on an Intel I7, 3.4 GHz machine, and the whole process took 30 minutes. All the options are expressed in the compact syntax (p, t) described in Section 2, where p represents the primitive action corresponding to the action's behavior, and t represents the option's stop condition (i.e., the new primitive action discovered, or an empty set). Once the set of learned options has been obtained, the test proceeds by applying the knowledge abstraction procedure described in [7]. In our specific case, the procedure eventually generated a final PPDDL domain composed by a set of 1528 operators. In order to empirically verify the correctness of the obtained PPDDL domain, we tested the domain with the off-the-shelf mGPT probabilistic planner [2]. The selected planning goal was to find the treasure, located in a hidden position of the environment (i.e., behind a locked door that could be opened only by operating on a bolt with a key) and bring it back to the agent's starting position, in the upper part of the Treasure Game environment. The agent's initial position is by the small stairs located on the environment's 5^{th} floor (up left). The symbolic plan depicted in Fig. 4 was successfully generated and, as readily observable, reaches the goal that has been imposed (note that the PPDDL

operators are named after their exact semantics manually, in order to facilitate their interpretation for the reader). The previous analysis is still ongoing work, in this respect, there are at least three research lines to investigate. The first line entails the study of different fine-tuning strategies of all the parameters utilized in the previously mentioned Machine Learning tools (such as DBSCAN, SVM, Kernel Density Estimator) involved in the knowledge-abstraction process. The second line is about analyzing the most efficient environment exploration strategy used to collect all the transition data that will be used for the classification tasks that are part of the abstraction procedure, as both the quantity and the quality of the collected data may be essential at this stage. The third one will be the exploration of innovative iterative procedures to incrementally refine [6] the generated PPDDL model.

References

[1] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning. pp. 41–48 (2009)

[2] Bonet, B., Geffner, H.: Mgpt: A probabilistic planner based on heuristic search. J. Artif. Int. Res. **24**(1), 933–944 (Dec 2005)

[3] Colas, C., Fournier, P., Chetouani, M., Sigaud, O., Oudeyer, P.Y.: Curious: intrinsically motivated modular multi-goal reinforcement learning. In: International conference on machine learning. pp. 1331–1340. PMLR (2019)

[4] Forestier, S., Portelas, R., Mollard, Y., Oudeyer, P.Y.: Intrinsically motivated goal exploration processes with automatic curriculum learning. arXiv preprint arXiv:1708.02190 (2017)

[5] Frank, M., Leitner, J., Stollenga, M., Förster, A., Schmidhuber, J.: Curiosity driven reinforcement learning for motion planning on humanoids. Frontiers in neurorobotics **7**, 25 (2014)

[6] Hayamizu, Y., Amiri, S., Chandan, K., Takadama, K., Zhang, S.: Guiding robot exploration in reinforcement learning via automated planning. Proceedings of the International Conference on Automated Planning and Scheduling **31**(1), 625–633 (May 2021), https://ojs.aaai.org/index.php/ICAPS/article/view/16011

[7] Konidaris, G., Kaelbling, L.P., Lozano-Perez, T.: From skills to symbols: Learning symbolic representations for abstract high-level planning. Journal of Artificial Intelligence Research **61**, 215–289 (2018), http://lis.csail.mit.edu/pubs/konidaris-jair18.pdf

[8] Oddi, A., Rasconi, R., Santucci, V.G., Sartor, G., Cartoni, E., Mannella, F., Baldassarre, G.: Integrating open-ended learning in the sense-plan-act robot control paradigm. In: ECAI 2020, the 24th European Conference on Artificial Intelligence (2020)

[9] Oudeyer, P.Y., Kaplan, F.: What is intrinsic motivation? a typology of computational approaches. Frontiers in Neurorobotics **1**, 6 (2009). https://doi.org/10.3389/neuro.12.006.2007, https://www.frontiersin.org/article/10.3389/neuro.12.006.2007

[10] Oudeyer, P.Y., Kaplan, F., Hafner, V.: Intrinsic motivation systems for autonomous mental development. IEEE transactions on evolutionary computation **11**(2), 265–286 (2007)

[11] Santucci, V.G., Baldassarre, G., Mirolli, M.: Biological cumulative learning through intrinsic motivations: a simulated robotic study on development of visually-guided reaching. In: Proceedings of the Tenth International Conference on Epigenetic Robotics (EpiRob2010. pp. 121–128 (2010)

[12] Santucci, V.G., Baldassarre, G., Mirolli, M.: Grail: A goal-discovering robotic architecture for intrinsically-motivated learning. IEEE Transactions on Cognitive and Developmental Systems **8**(3), 214–231 (2016)

[13] Santucci, V.G., Oudeyer, P.Y., Barto, A., Baldassarre, G.: Intrinsically motivated open-ended learning in autonomous robots. Frontiers in neurorobotics **13**, 115 (2020)

[14] Seepanomwan, K., Santucci, V.G., Baldassarre, G.: Intrinsically motivated discovered outcomes boost user's goals achievement in a humanoid robot. In: 2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob). pp. 178–183 (2017)

[15] Singh, S., Barto, A.G., Chentanez, N.: Intrinsically motivated reinforcement learning. In: Proceedings of the 17th International Conference on Neural Information Processing Systems. p. 1281–1288. NIPS'04, MIT Press, Cambridge, MA, USA (2004)

[16] Stolle, M., Precup, D.: Learning options in reinforcement learning. In: Koenig, S., Holte, R.C. (eds.) Abstraction, Reformulation, and Approximation. pp. 212–223. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[17] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (1998)

[18] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1-2), 181–211 (Aug 1999). https://doi.org/10.1016/S0004-3702(99)00052-1, http://dx.doi.org/10.1016/S0004-3702(99)00052-1

[19] Younes, H., Littman, M.: PPDDL1.0: An Extension to PDDL for Expressiong Planning Domains with Probabilistic Effects. Tech. rep., Carnegie Mellon University (2004), CMU-CS-04-167

Predictions Predicting Predictions

Matthew K. Schlegel*
Department of Computing Science
University of Alberta
mkschleg@ualberta.ca

Martha White
Department of Computing Science
University of Alberta
whitem@ualberta.ca

Abstract

Predicting the sensorimotor stream has consistently been a key component for building general learning agents. Whether through predicting a reward signal to select the best action or learning a predictive world model with auxiliary tasks, prediction making is at the core of reinforcement learning. One of the main research directions in predictive architectures is in the automatic construction of learning objectives and targets. The agent can consider any real-valued signal as a target when deciding what to learn, including the current set of internal predictions. A prediction whose learning target is another prediction is known as a composition. Arbitrarily deep compositions can lead to learning objectives that are unstable or not suitable for function approximators. This manuscript looks to begin uncovering the underlying structure of compositions in an effort to leverage and learn them more effectively in general learning agents. Specifically, we consider the dynamics of compositions both empirically and analytically. We derive the effective schedule of emphasis (or discounts) of future observations with compositions of arbitrary depth, leading to informative observations about the prediction targets. In the empirical simulations, we focus on the unintuitive behavior of compositions, especially in cases that are not easy to analyze. Overall, predictions predicting predictions which predict predictions have interesting properties and can add depth to an agent's predictive understanding of the world.

Keywords: reinforcement learning; prediction; GVFs

Acknowledgements

We would like to thank the Alberta Machine Intelligence Institute, IVADO, NSERC and the Canada CIFAR AI Chairs Program for the funding for this research. We would also like to thank Andrew Patterson for his insightful comments about the connection to digital signal processing and Adam White for sharing the Critterbot dataset.

*mkschleg.github.io

1 Introduction

Reinforcement learning is built on predicting the effect of behavior on future observations and rewards. Many of our algorithms learn predictions of a cumulative sum of (discounted) future rewards, which is used as a bedrock for learning desirable policies. While reward has been the primary predictive target of focus, TD models (Sutton 1995) lay out the use of temporal-difference learning to learn a world model through value function predictions. Temporal-difference networks (Sutton and Tanner 2004) take advantage of this abstraction and build state and representations through predictions. Sutton, Modayil, et al. 2011 and White 2015 further the predictive perspective by developing a predictive approach to building world knowledge through general value functions (GVFs).

GVFs have been pursued broadly in reinforcement learning: Günther et al. 2016 used GVFs to build an open loop laser welder controller, Linke et al. 2020 used predictions and their learning progress to develop an intrinsic reward, Edwards et al. 2016 used GVFs to build controllers for myoelectric prosthetics, using gvfs for auxiliary training tasks to improve representation learning (Jaderberg et al. 2017; Veeriah et al. 2019), to extend a value function's approximation to generalize over goals as well as states (Schaul et al. 2015), and to create a scheduled controller from a set of sub-tasks for sparse reward problems (Riedmiller et al. 2018). Successor representations and features are predictions of the state, learned or given, which have been shown to improve learning performance (Barreto et al. 2018; Dayan 1993; Russek et al. 2017; Sherstan et al. 2018).

Learning predictions of any real-valued signal the agent has access to also opens the possibility of asking compositional predictive questions (White 2015). A compositional question is one whose target is dependent on another prediction internal to the agent. Compositions expand the possible range of predictive questions we can specify as a GVF (Rafols et al. 2006; Schlegel et al. 2021; Sutton and Tanner 2004; White 2015; Zheng et al. 2021). While this may suggest the GVF framework is limited in what questions can be asked, the limitations are necessary so the predictions can be trained *independent of span* (van Hasselt and Sutton 2015). Learning independent of span means the target can be learned using online algorithms regardless of the effective horizon of the prediction. Adding layers of compositional questions have improved the learning in predictive representations (Rafols et al. 2006; Schlegel et al. 2021), and improved the performance of deep reinforcement learning through auxiliary tasks (Zheng et al. 2021). In the automatic specification of learning targets compositions are thought to provide a way for the agent to build complexity (Kearney 2022; Schlegel et al. 2021; Veeriah et al. 2019; Zheng et al. 2021), but often these architectures don't leverage compositions for stability concerns (Schlegel et al. 2021).

As well as improving behavior empirically, compositions can provide semantic depth. An excellent example of this can be seen in option-extended temporal difference networks (Rafols et al. 2006), and later explored again in Schlegel et al. 2021. The example is centered in an environment where the agent has a low-powered visual sensor and needs to learn its directionality from the painted walls. Each cardinal direction has a different colored wall. The first layer of predictions the agent makes is to predict what color it will observe if it were to drive straight. The second layer are myopic predictions which ask what the first layer's prediction will be after turning clockwise (or counter-clockwise). The second layer allows the agent to predict which walls are to its sides as well as the wall in the direction the agent is facing. These predictions cannot be specified in the usual GVF framework, but can be easily constructed through compositions. While this may be "repeated information" in a sense, the extra learning objectives makes the learning properties of the predictive representation better as compared to other specifications (Schlegel et al. 2021).

As algorithms for the automatic discovery of complex question networks continue to push the boundaries of what questions are considered by the agent, the properties of compositions should be better studied. When searching for what to learn the questions an agent eventually retains will be dependent on the agent's ability to learn the predictions. While it is clear questions that naturally diverge (say setting the discount $\gamma = 1$) should be avoided, other problems, such as the scale of a target, could be equally as problematic when using function approximation (i.e. end-to-end neural networks). This could mean important predictions are disregarded because the agent is unable to learn the answer without proper strategies to normalize the prediction's magnitude. Better strategies for learning and normalizing predictive targets will come from understanding the effective discount schedule (or emphasis) compositional predictions will have on the targets.

In this report, we consider the effect of compositions on the sequence of discounts, and relegate the effect of off-policy importance weights to future work. We first analyze the sequence of discounts over any number of compositions and constant discounts. We then analyze this sequence to better understand how it emphasizes parts of the data stream. Surprisingly, the effective discount for constant discount compositions have a form which can be described analytically. While this does not include the full spectrum of discount functions, it provides a first step towards understanding compositions. Next we look at simulations using more complex state-dependent discount functions using a simple consistent sequence and two timeseries datasets. In these simulations we focus on the effect of applying the same discount function a large number of times, looking to see if the shape of the returns become regular over the compositions. Finally, several future directions and questions are posed.

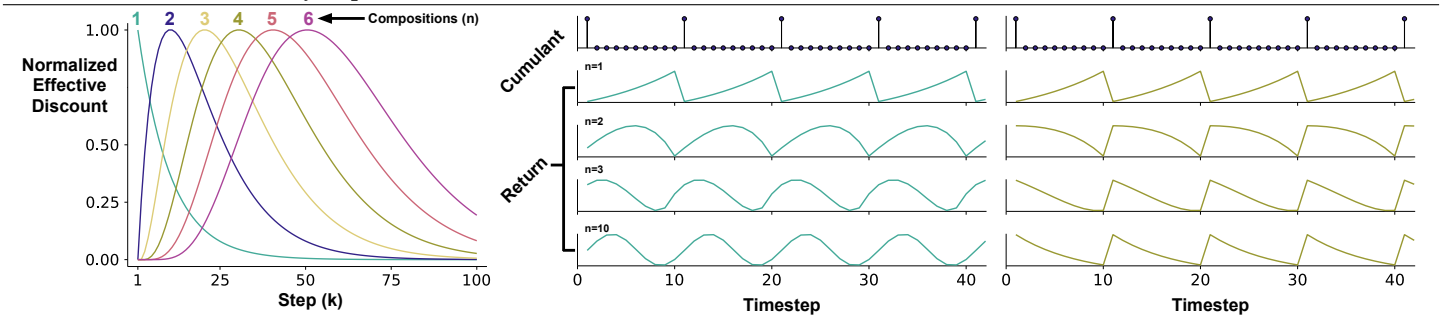


Figure 1: (**left**) The effective discount for n compositions normalized by the maximum value found in section 2. (**middle, right**) The cycle world simulations, with top graph as the cumulant and subsequent plots n compositions with constant and terminating discounts respectively.

2 Analyzing the sequence

In this section, we restrict to the setting where we have an infinite sequence of sensor readings $\mathbf{x} = \{x[0], x[1], \ldots, x[t], \ldots, x[\infty]\}$ where $x[i] \in [x_{\min}, x_{\max}]$ and a constant discount γ. The return of this signal starting at a time step t is $V[t] = \sum_{k=0}^{\infty} x[k]\gamma[k-t]$ where $\gamma[k] = \gamma^{k-1}$ for $k >= 1$ and 0 otherwise. This framing of the return is slightly different from the typical presentation. Specifically, we reinterpret the return as a convolution beteween γ and x [1] and shift the discount sequence over the sensor readings. This implicitly defines an infinite sequence of predictions $y[t]$. In the above equation, if we replace the sequence x with the sequence of predictions y, we get a new set of predictions and for any number of compositions n we have $V^n[t] = \sum_{k=0}^{\infty} V^{n-1}[k]\gamma[k-t]$. Expanding this equation we can define the general sequence of effective discounts for n compositions and the corresponding return as

$$\gamma^n[k] = \begin{cases} 0 & \text{if } k < n \\ \frac{\prod_{i=1}^{n-1}(k-i)}{(n-1)!}\gamma^1[k-n] \end{cases} \qquad V^n[t] = \sum_{k=0}^{\infty} x[k]\gamma^n[k-t]$$

where $\gamma^1[k] = \gamma[k]$ defined above and $V^n[t]$ is the target of the nth composition at timestep t. For any value n there are two sequences multiplied together. The original discounting shifted by the number of applications $\gamma^1[k-n]$ and a diverging series

$$Q^n[k] = \frac{\prod_{i=1}^{n-1}(k-i)}{(n-1)!} = \frac{\Gamma(k)}{\Gamma(k-n+1)\Gamma(n)}$$

where $\Gamma(k) = (k-1)!$ for $k \in \mathbb{Z}$ is known as the Gamma function, and can be used to analyze the function with $k \in \mathbb{R}$.

We know for any particular application of the convolution γ on a series with known domain $[x_{\min}, x_{\max}]$ the value function can take values bounded by $V^1[t] \in [\frac{x_{\min}}{1-\gamma}, \frac{x_{\max}}{1-\gamma}]$. This extends to n compositions in a straightforward way where the range of the value function becomes $V^n[t] \in [\frac{x_{\min}}{(1-\gamma)^n}, \frac{x_{\max}}{(1-\gamma)^n}]$. While normalizing the value function to take values within in the range $[0, 1]$ has been used in various settings (Schlegel et al. 2021), as we add more compositions we see the effective range of values shrinking considerably.

Given the effective discounting sequence above, we can begin to piece together the which observations are emphasized in the predictions. The first 100 steps of the effective discount function for several values of n can be seen in figure 1. These sequences are normalized to be in the range $[0, 1]$ for a visual comparison. The emphasis becomes increasingly spread as n increases, with the peak of this function moving further to the future at a consistent rate.

To find the maximum value we take the derivative of the log of the sequence with respect to k getting

$$\frac{\delta}{\delta k} \ln \gamma^n[k] = \psi(k) - \psi(k-n+1) + \ln \gamma$$

where $\psi(z+1) = H_z - C$ is the digamma function, $H_z = \sum_{i=1}^{z} \frac{1}{i} \leq \int_1^z \frac{1}{x} dx = ln(z)$ is the Euler harmonic number, and C is the Euler-Mascheroni constant. Using the approximation above, we can find where we should expect the maximal value is (to an approximation) $k = hn - (h-1) = h(n-1) + 1$, where $h = \frac{1}{1-\gamma}$ is sometimes known as the horizon of discount γ. Of course this is an approximation from above and the real value falls in $k \in [h(n-1), h(n-1)+1)]$.

[1] In digital signal processing (Oppenheim and Schafer 2010) often the convolution, in this case γ, is mirrored across t and the inifinte sequence of sensor readings is $\mathbf{x} = \{x[-\infty], \ldots, x[t], \ldots, x[\infty]\}$. The corresponding convolution would be $V[t] = \sum_{k=-\infty}^{\infty} x[k]\gamma[t-k]$ which would change how we define the sequence of γ. To be consistent with the reinforcement learning literature, we don't follow this here and instead implicitly define γ as the mirrored version and only consider the sequence starting at $k = 0$.

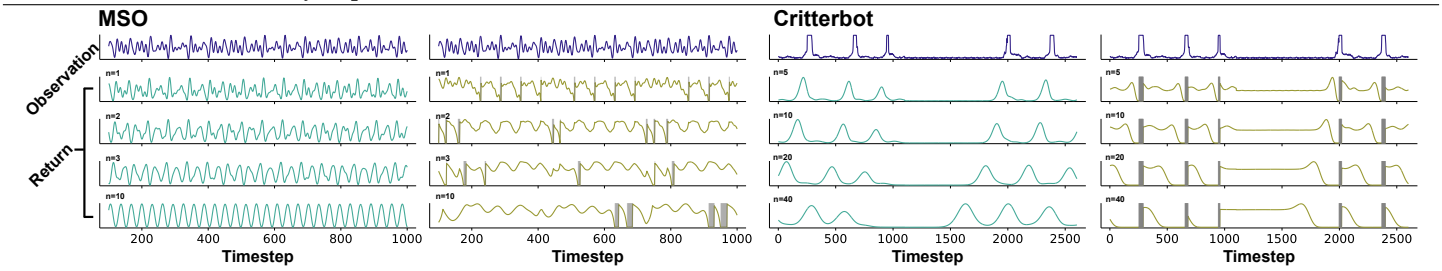


Figure 2: (**left two**) Returns of the multiple sinusoidal oscillator (MSO) synthetic data set with constant and terminating discount respectively. The gray vertical lines are where the return terminates. (**right two**) Returns of Critterbot data set over the light3 sensor with constant and terminating discount respectively.

3 Empirical observations

While we can describe the effective discount for composing constant discount predictions, the same techniques are difficult to apply to a non time-invariant discount (i.e. state-dependent discounts (Sutton, Modayil, et al. 2011; White 2015)). Instead, in this section we look at the ideal returns of various signals using constant discounting and a terminating discounting functions. We use three datasets moving from highly synthetic to real-world robot sensori-motor data. The goal of this section is to show the non-intuitive behavior of compositions to motivate further analysis and exploration. All code can be found at `https://github.com/mkschleg/CompGVFs.jl`. Below $\gamma = 0.9$ unless otherwise stated.

The first series is based off the cycle world, where the agent observes a sequence of a single active bit followed by 9 inactive bits, where the length of the sequence is $m = 10$. The cumulant is the observation itself, and in this report we learn using TD($\lambda = 0.9$) with learning rate $\alpha = 0.1$ and an underlying tabular representation where each component is the place in the sequence. We learn two chains of compositions. The first is that of the continuous discounting described above, and the second is a series of discounts which terminate (i.e. $\gamma[t] = 0$) when the observation is active. The predictions of a single run can be seen in figure 1. For the constant discount, as the number of compositions increases we see the prediction sequence converge to what looks to be a sinusoid with frequency of 10, and amplitude driven by the analysis above. We expect this to be the case following from the central limit theorem. For the terminating discount, the wave form is more interesting. The first layer of predictions look very similar to the constant discount with amplitude shifted by $\frac{\gamma^m}{1-\gamma^m}$. But as there are more compositions the effect seems to be the prediction is at its height farther away from the active bit. As the agent gets closer to the observation, the sequence of summed values is shorter leading to smaller values. Given the sequence we use it is easy to mistake this as the agent creating a trace of the cumulant, but we must remember the prediction is about future cumulants.

Next we use a subset of the Critterbot dataset (Modayil et al. 2014; White 2015), focusing on light sensor 3. This gives a sequence of spikes similar to the cycle world sequence and a long pause in-between consistent saturations of the light sensor. We are able to see with the current setting the predictions look more like shifted and spread spikes. But with many more compositions, the return reverts to a similar form as before. The terminating discounts (with termination at sensor saturation $x[t+1] > 0.99$) provides a nice demonstration of how the returns are predicting the signal, just with a decaying prediction instead of the usual growing prediction. The results are similar in the multiple sinusoidal oscillator (Jaeger and Haas 2004). We use a slightly different terminating discount where the return terminates when the previous normalized prediction is $y^{n-1}[t+1] > 0.9$ rather than when the observation is saturated. While there are decays as the MSO sequence peaks, as we increase the depth of the composition, these periods are less frequent. Deep compositions may indicate parts of the sequence where there are fewer saturations in the original sequence.

4 Future Directions

This work suggests a number of interesting research directions and questions. While we mostly analyzed the sequence on discrete steps and applications of the filter, the general form does lend itself to continuous and complex values of n and k. In a similar vein, we focused on real valued exponential discounting while several discounting schemes exist which could be applied to our formulation. We are particularly interested in complex discounting (De Asis et al. 2018) and hyperbolic discounting (Fedus et al. 2019). Applying a diverse set of discounting schemes in compositions provide an interesting way to extend the power of value functions while maintaining learnability through efficient algorithms like temporal-difference learning.

The approach used in this paper is unable to analyze state-dependent discount functions. One way around this might be in analyzing truncated sequences and taking an expectation over a distribution of sequence lengths. This might lead to a expected effective discounting sequence, but how this will interact with an underlying Markov process is unclear. This

is an important next step for understanding the effects of compositions in general value functions, and could also help in analyzing off-policy compositions.

Finally, the return can be re-interpreted as a convolution over the infinite sequence of observations. While this interpretation was only used to better the notation in this manuscript, further connections to convolutions and digital signal processing should be explored. Better filter designs might inspire different discounting schedules to squeeze more information from the data stream. We also have only analyzed these convolutions in the time domain. The frequency domain might give us more insight into how consistent signals like the cycle world dataset will be effected by compositions.

References

Barreto, Andre et al. (2018). "Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement". In: *International Conference on Machine Learning*. PMLR.

Dayan, Peter (1993). "Improving Generalization for Temporal Difference Learning: The Successor Representation". In: *Neural Computation*.

De Asis, Kristopher, Brendan Bennett, and Richard Sutton (2018). "Predicting Periodicity with Temporal Difference Learning". In: *arXiv preprint arXiv:1809.07435*.

Edwards, Ann L et al. (2016). "Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching". In: *Prosthetics and orthotics international* 40.5, pp. 573–581.

Fedus, William et al. (2019). "Hyperbolic discounting and learning over multiple horizons". In: *arXiv preprint arXiv:1902.06865*.

Günther, Johannes et al. (2016). "Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning". In: *Mechatronics* 34, pp. 1–11.

Jaderberg, Max et al. (2017). "REINFORCEMENT LEARNING WITH UNSUPERVISED AUXILIARY TASKS". In: *International Conference on Representation Learning*.

Jaeger, Herbert and Harald Haas (2004). "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication". In: *science*.

Kearney, Alex (2022). "What Should I Know? Using Meta-Gradient Descent for Predictive Feature Discovery in a Single Stream of Experience". In: *Unpublished*.

Linke, Cam et al. (2020). "Adapting Behavior via Intrinsic Reward: A Survey and Empirical Study". In: *Journal of Artificial Intelligence Research* 69, pp. 1287–1332.

Modayil, Joseph, Adam White, and Richard Sutton (2014). "Multi-Timescale Nexting in a Reinforcement Learning Robot". In: *Adaptive Behavior*.

Oppenheim, Alan V and Ronald W Schafer (2010). *Discrete-time Signal Processing*. Pearson Higher Education.

Rafols, Eddie, Anna Koop, and Richard Sutton (2006). "Temporal Abstraction in Temporal-difference Networks". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y. Weiss, B. Schölkopf, and J. C. Platt. MIT Press.

Riedmiller, Martin et al. (2018). "Learning by playing solving sparse reward tasks from scratch". In: *International conference on machine learning*. PMLR, pp. 4344–4353.

Russek, Evan M. et al. (2017). "Predictive Representations Can Link Model-Based Reinforcement Learning to Model-Free Mechanisms". In: *PLOS Computational Biology*.

Schaul, Tom et al. (2015). "Universal Value Function Approximators." In: *International Conference on Machine Learning*.

Schlegel, Matthew et al. (2021). "General value function networks". In: *Journal of Artificial Intelligence Research* 70, pp. 497–543.

Sherstan, Craig, Marlos C. Machado, and Patrick M. Pilarski (2018). "Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation". In: *arXiv:1803.09001 [cs, stat]*. arXiv: 1803.09001 [cs, stat].

Sutton, Richard (1995). "TD models: Modeling the world at a mixture of time scales". In: *Machine Learning Proceedings 1995*. Elsevier, pp. 531–539.

Sutton, Richard, Joseph Modayil, et al. (2011). "Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, Richard and Brian Tanner (2004). "Temporal-Difference Networks". In: *Advances in Neural Information Processing Systems*.

van Hasselt, Hado and Richard Sutton (2015). "Learning to predict independent of span". In: *arXiv:1508.04582*.

Veeriah, Vivek et al. (2019). "Discovery of useful questions as auxiliary tasks". In: *Advances in Neural Information Processing Systems*, pp. 9306–9317.

White, Adam (2015). "Developing a Predictive Approach to Knowledge". In: *University of Alberta*.

Zheng, Zeyu et al. (2021). "Learning State Representations from Random Deep Action-conditional Predictions". In: *Advances in Neural Information Processing Systems* 34.

Active Inference for Robotic Manipulation

Tim Schneider
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
tim@robot-learning.de

Boris Belousov
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
boris@robot-learning.de

Hany Abdulsamad
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
hany@robot-learning.de

Jan Peters
Intelligent Autonomous Systems
Technical University of Darmstadt
64289 Darmstadt, Germany
mail@jan-peters.net

Abstract

Robotic manipulation stands as a largely unsolved problem despite significant advances in robotics and machine learning in the last decades. One of the central challenges of manipulation is partial observability, as the agent usually does not know all physical properties of the environment and the objects it is manipulating in advance. A recently emerging theory that deals with partial observability in an explicit manner is Active Inference. It does so by driving the agent to act in a way that is not only goal-directed but also informative about the environment. In this work, we apply Active Inference to a hard-to-explore simulated robotic manipulation tasks, in which the agent has to balance a ball into a target zone. Since the reward of this task is sparse, in order to explore this environment, the agent has to learn to balance the ball without any extrinsic feedback, purely driven by its own curiosity. We show that the information-seeking behavior induced by Active Inference allows the agent to explore these challenging, sparse environments systematically. Finally, we conclude that using an information-seeking objective is beneficial in sparse environments and allows the agent to solve tasks in which methods that do not exhibit directed exploration fail.

Keywords: Model-Based Reinforcement Learning, Robotic Manipulation, Active Inference

1 Introduction and Related Work

A common belief in cognitive science is that the evolution of dexterous manipulation capabilities was one of the major driving factors in the development of the human mind [1]. Performing manipulation is cognitively highly demanding, forcing the actor to reason not only about the impact of its actions on itself but also about the impact on its environment. This inherent complexity leaves autonomous robotic manipulation a largely unsolved topic, despite significant advances in robotics and machine learning in the last decades.

One of the central challenges of manipulation is partial observability. While we are manipulating an object, we rarely know all of its physical properties in advance. Instead, we must resort to inferring those properties based on observations and touch. To deal with this issue as effectively as possible, humans have developed various active haptic exploration strategies that they constantly apply during manipulation tasks [2].

A recently emerging theory from cognitive science that tries to explain this notion of constant active exploration is Active Inference (AI) [3]. AI formulates both action and perception as the minimization of a single free-energy functional, called the Variational Free Energy (VFE). In doing so, Friston et al. [4] derive an objective function that consists of an extrinsic, goal-directed term and an intrinsic, information-seeking term. The combination of these two terms drives the agent to act in a way that is both goal-directed and informative, in that the agent learns about its environment through its actions.

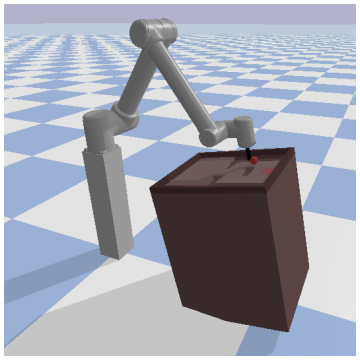


Figure 1: Robot using Active Inference to solve a challenging manipulation task.

In this work, we show how AI can be used to learn challenging robotic manipulation tasks without prior knowledge. For now, we assume that the environment is fully observable and only consider epistemic uncertainty[1]. To implement AI in practice, we use a neural network ensemble and deploy Model Predictive Control for action selection. We show that agents driven by AI explore their environments in a directed and systematic way. These exploratory capabilities allow the agents to solve complex sparse manipulation tasks, on which agents that are not explicitly information-seeking fail.

Related to our approach is PETS [5], which also trains ensemble models for the transition and reward distributions and selects actions with a Cross-Entropy Method planner. The key difference to our approach is that PETS does not use an intrinsic term and instead greedily select the actions they predict to yield the highest reward.

An approach similar to ours is Tschantz et al. [6], who also tackle RL tasks with AI. The difference to our approach is that they use a different free energy functional used for planning and chose a different approximation of their intrinsic term, which requires them to make a mean-field assumption over consecutive states. They evaluate their approach on multiple RL benchmarks, including *Mountain Car* and *Cup Catch*.

2 Active Inference

According to the Free Energy Principle (FEP) [3], any organism must restrict the states it is visiting to a manageable amount. Mathematically, AI implements this restriction as follows: Every agent maintains a generative model p of the world and avoids sensations o that are surprising, hence have a low marginal log-probability $\ln p(o)$. Thus, the objective can be written as

$$\min_{\pi} \; -\ln p(o) \tag{1}$$

where o is generated by some external process that can be influenced by changing the policy π.

The agent's generative model is assumed to consist of not only observations o, but also contain hidden states x, giving $p(o) = \int p(o,x)\,dx = \int p(o \mid x)\,p(x)\,dx$. To make Eq. (1) tractable, we apply variational inference and obtain the ELBO using Jensen's inequality:

$$-\ln p(o) = -\ln \int p(o,x)\,dx = -\ln \int \frac{q_\phi(x)}{q_\phi(x)} p(o,x)\,dx \leq D_{\mathrm{KL}}[q_\phi(x) \parallel p(x \mid o)] - \ln p(o) =: \mathcal{F}(o,\phi)$$

where $q_\phi(x)$ is the variational posterior, parameterized by ϕ, and $\mathcal{F}(o,\phi)$ is termed the Variational Free Energy (VFE) in the AI literature.

Minimizing $\mathcal{F}(o,\phi)$ w.r.t. the variational parameters ϕ corresponds to minimizing the KL divergence between the variational posterior $q_\phi(x)$ and the true posterior $p(x \mid o)$. In other words, by minimizing the VFE w.r.t. ϕ, the agent is solving the perception problem of mapping its observations to their latent causes.

[1] Epistemic uncertainty is the uncertainty the agent has over its model of the world. In contrast, aleatoric uncertainty is uncertainty over the agent's state.

To facilitate planning into the future, the VFE can be modified to incorporate an expectation over future states, yielding the Expected Free Energy (EFE) Friston et al. [4]:

$$\begin{aligned} G_\pi(\phi) &= -\mathbb{E}_{q_\phi(o_{t+1:T}, x_{t+1:T} \mid \pi)}[\ln p(o_{t+1:T}, x_{t+1:T}) - \ln q_\phi(x_{t+1:T} \mid \pi)] \\ &\approx -\underbrace{\mathbb{E}_{q_\phi(x \mid \pi)}[D_{\mathrm{KL}}[q_\phi(o \mid x, \pi) \parallel q_\phi(o \mid \pi)]]}_{\text{intrinsic term (expected information gain)}} \underbrace{-\mathbb{E}_{q_\phi(o \mid \pi)}[\ln p(o)]}_{\text{extrinsic term}} \end{aligned}$$

where we omitted subscripts for readability and defined $q_\phi(o \mid x) \coloneqq p(o \mid x)$, such that q_ϕ and p follow the same observation model.

The minimization of the EFE w.r.t. the policy π causes the agent to act in a way that maximizes both information gain and the extrinsic term. Here, the extrinsic term acts as an external signal that allows us to make the agent prefer or disprefer certain observations. While it is common in RL literature to use a reward function to give the agent a notion of "good" and "bad" behavior, in the AI framework, we define a prior distribution over target observations $p(o)$ that we would like the agent to make. Note that by making the reward part of the observation and setting the maximum reward as target observation [6], we can transform any reward-based task to fit into the AI framework.

3 Method

In this work, we propose a model-based Reinforcement Learning algorithm that uses AI to efficiently explore challenging state spaces. Therefore, we assume that the environment is fully observable, governed by unknown dynamics $P(x_\tau \mid x_{\tau-1}, a_\tau)$ and provides the agent with a reward $P(r_\tau \mid x_\tau, a_\tau)$ in every time step. We model both the dynamics and the reward with neural network conditioned Gaussians $p(r_\tau \mid x_\tau, a_\tau, \theta) \coloneqq \mathcal{N}(x_\tau \mid \mu_\theta^x(x_{\tau-1}, a_\tau), \sigma^x I)$ and $p(r_\tau \mid x_\tau, a_\tau, \theta) \coloneqq \mathcal{N}(r_\tau \mid \mu_\theta^r(x_\tau, a_\tau), \sigma^r I)$, resulting in the following generative model:

$$p(x_{0:T}, a_{1:T}, r_{1:T}, \theta) = p(x_0)\, p(a_{1:T})\, p(\theta) \prod_{\tau=1}^{T} p(r_\tau \mid x_\tau, a_\tau, \theta)\, p(x_\tau \mid x_{\tau-1}, a_\tau, \theta)$$

Since the environment is fully observed, the only hidden variables are the neural network parameters θ. Thus, we are left with the following minimization problem for selecting a policy $\pi \coloneqq a_{t+1:T}$ at time t:

$$\min_\pi G_\pi(\phi) \coloneqq -\underbrace{\mathbb{E}_{q_\phi(\theta \mid \pi)}[D_{\mathrm{KL}}[p(x_{t+1:T}, r_{t+1:T} \mid \theta, \pi) \parallel q_\phi(x_{t+1:T}, r_{t+1:T} \mid \pi)]]}_{\text{expected parameter information gain}} - \underbrace{\mathbb{E}_{q_\phi(r_{t+1:T} \mid \pi)}\left[\sum_{\tau=t+1}^{T} r_\tau\right]}_{\text{expected cumulative reward}} \tag{2}$$

where we defined the observation preference distribution such that $p(o_\tau) \propto e^{r_\tau}$, and defined $q_\phi(x, r \mid \theta, \pi) \coloneqq p(x, r \mid \theta, \pi)$. Hence, by this definition, q and p differ only in the marginal probability of the model parameters θ.

Similar to other methods utilizing Model Predictive Control [5], by minimizing this objective function we select a policy that maximizes the expected cumulative reward over a fixed horizon. However, additionally we are maximizing the expected parameter information gain, driving the agent to seek out states that are informative about its model parameters θ. This term causes the agent to be curious about its environment and explore it systematically, even in the total absence of extrinsic reward. The optimization of this objective can now theoretically be done by any planner that is capable of handling continuous action spaces. In this work, similar to Chua et al. [5], we use a variant of the Cross-Entropy Method to find an open loop sequence of actions $a_{t+1:T}$ that maximizes Eq. (2).

A major challenge in computing $G_\pi(\phi)$ is that neither the intrinsic, nor the extrinsic term can be computed in closed form. While the extrinsic term can straightforwardly be approximated with sufficient accuracy via Monte Carlo, the intrinsic term is known to be notoriously difficult to compute [7]. Thus, instead of maximizing it directly, many methods maximize a variational lower bound of it [8]. However, due to the high-dimensional nature of θ, these approaches are too expensive to be executed during planning in real time.

Hence, instead we propose to use a Nested Monte Carlo estimator that reuses samples from the outer estimator in the inner estimator to approximate the intrinsic term:

$$\mathrm{IG}((x, r), \theta) \approx \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ln p(x_i, r_i \mid \theta_i) - \ln \underbrace{\frac{1}{n}\sum_{\substack{k=1 \\ k \neq i}}^{n} p(x_i, r_i \mid \theta_k)}_{\text{inner estimator}}}_{\text{outer estimator}}$$

Although using the same samples $\theta_1, \ldots, \theta_n$ in the inner estimator as in the outer estimator violates the i.i.d. assumption, we found this reuse of samples to increase the sample efficiency substantially. Since this estimator only requires samples of θ, we represent $q_\phi(\theta)$ by a set of particles $\theta_1, \ldots, \theta_n$, making our model a neural network ensemble.

4 Experimental Results

A central feature that sets our method apart from other purely model-based approaches [5, 9] is the intrinsic term, that explicitly drives the agent to explore its environment in a systematic manner. To evaluate the exploratory capabilities of our method, we designed two hard-to-explore manipulation tasks: *Tilted Pushing* and *Tilted Pushing Maze*. In both tasks, the agent has to push a ball up a tilted table into a target zone to receive reward. The agent can move the gripper in a plane parallel to the table and rotate the black end-effector around the Z-axis (Z-axis being orthogonal to the brown table and pointing up). As input, the agent receives the 2D positions and velocities of both the gripper and the ball, and the angular position and velocity of the end-effector. To add an additional challenge, in the *Tilted Pushing Maze* task we add holes to the table, that irrecoverably trap the ball if it falls in. For a visualization of these tasks, refer to Fig. 2.

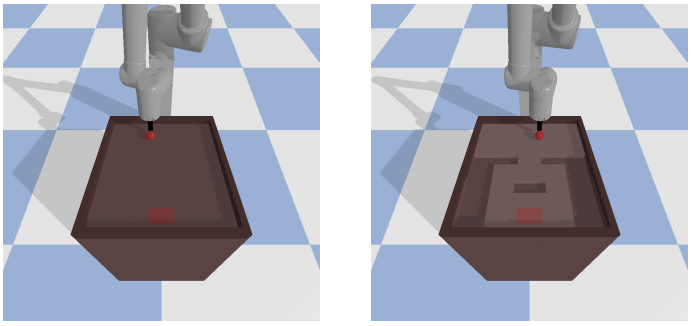


Figure 2: Visualization of the two environment configurations we test our methods on: *Tilted Pushing* (left) and *Tilted Pushing Maze* (right). The target zone is marked in red.

There are two aspects make these tasks particularly challenging: First, the reward is sparse, meaning that the only way the agent can learn about the reward at the top of the table is by moving the ball there and exploring it. Second, balancing the ball on the finger and moving it around requires a fair amount of dexterity, especially given the low control frequency of 4 Hz[2] we operate our agent on. Once the agent drops the ball, it cannot be recovered, giving the agent no choice but to wait for the episode to terminate to continue exploring. Both of these aspects make solving these tasks with conventional, undirected exploration methods like Boltzmann exploration or adding Gaussian noise to the action extremely challenging. Consequently, the agent has to learn to balance the ball without receiving any extrinsic reward, purely driven by its own curiosity.

As visible in Fig. 3, our method is able to solve the *Tilted Pushing*. Both SAC [10] and our method without an intrinsic term fail to find the reward within 10,000 episodes. The holes of *Tilted Pushing Maze* make this environment significantly harder to explore, as the ball has to be maneuvered around two corners in order to reach the target zone. In this experiment, only our method finds the reward within 30,000 episodes. As can be seen in Fig. 4, the reason for the bad performance of the non-intrinsic agent is its failure to explore the full state space. While our agent continues to systematically maneuver the ball around the holes in unseen locations, the non-intrinsic agent rarely passes the lower holes and leaves the upper half of the table unexplored.

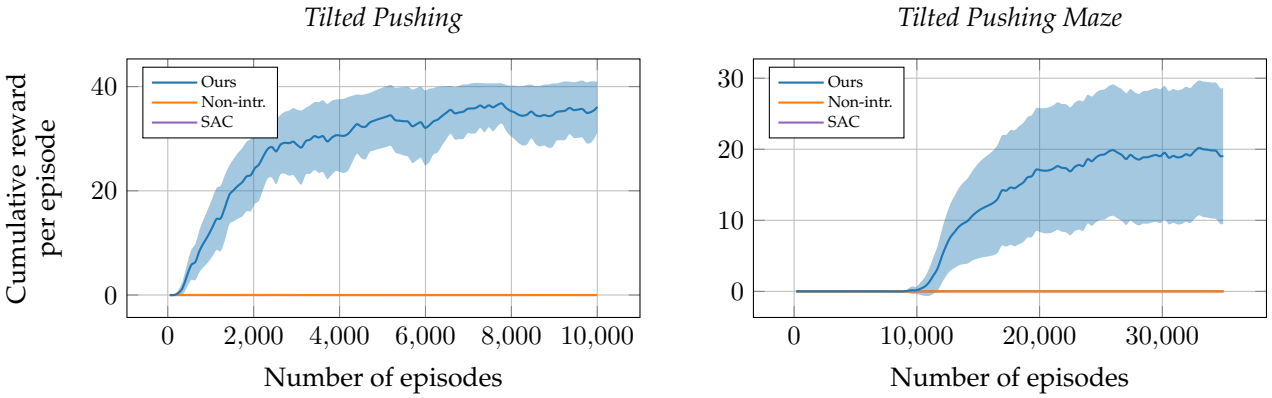


Figure 3: Cumulative per-episode reward for two different versions of our agent (one with intrinsic term, one without) and SAC on both variants of our environment. This graph displays the evaluation reward, which is obtained by rolling out the learned model without considering the intrinsic reward. Both non-intrinsic configurations and SAC failed to find the objective and converged to local minima.

These experiment show that our method is able to systematically explore a complex, contact-rich environment with many dead-ends. Without any extrinsic feedback, our agents learned to balance the ball on the end-effector and systematically move it around the environment until the target zone was found. The sole reason for this behavior to occur in the first

[2]The computation of the intrinsic term is computationally heavy, limiting us to this rather low control frequency.

place is that our agents understood they could only explore the entire state space if they kept balancing the ball and move it to unseen locations.

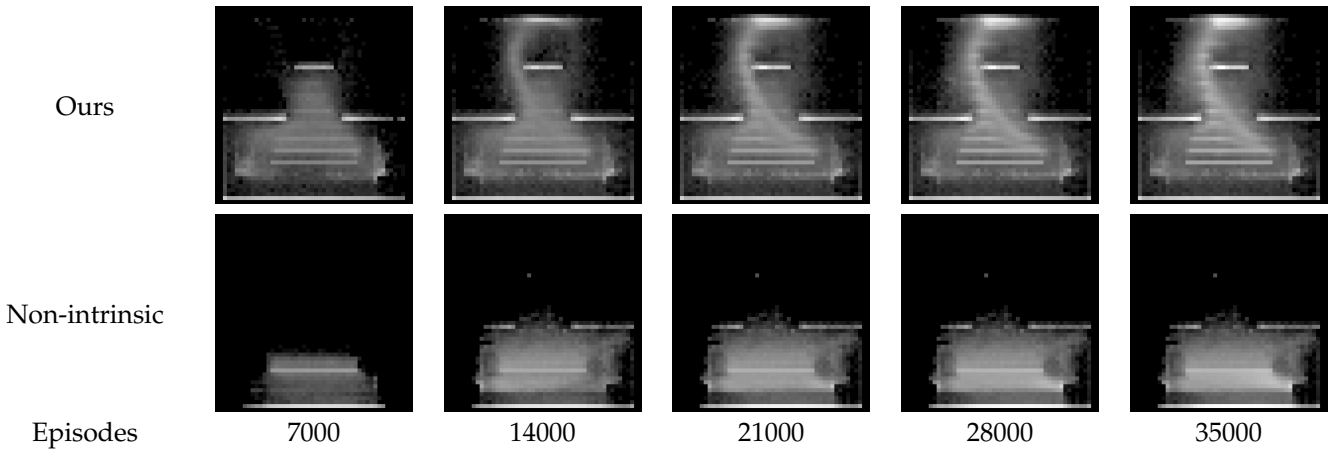


Figure 4: Comparison of the states visited by our method and an agent using no intrinsic term, relying on Gaussian exploration instead. The brightness of each pixel indicates how often the ball has visited the respective point of the table at the given point in the training. The coordinate origin is at the bottom of each image, meaning that the images are rotated 180°compared to the top-down view in Fig. 2. Each configuration was run once.

5 Conclusion

In this work, we developed a method capable of applying Active Inference to complex Reinforcement Learning tasks. We evaluated our method in two challenging robotic manipulation task, both designed to be particularly hard-to-explore. Throughout our experiments, we showed that our method induces systematic exploration behavior and is capable of solving even the most challenging of these environments. Neither the non-intrinsic configurations nor the maximum entropy method SAC managed to solve the robotic manipulation tasks. Hence, we conclude that the information-seeking behavior of our agents is beneficial for solving challenging exploration problems with sparse rewards.

Finally, in future work we plan to apply our method to a real robot and evaluate whether Active Inference can be used in real robotic manipulation tasks.

References

[1] Robert MacDougall. "The significance of the human hand in the evolution of mind". In: *The American Journal of Psychology* 16.2 (1905), pp. 232–242.

[2] Agnes Lacreuse and Dorothy M Fragaszy. "Manual exploratory procedures and asymmetries for a haptic search task: A comparison between capuchins (Cebus apella) and humans". In: *Laterality: Asymmetries of Body, Brain and Cognition* 2.3-4 (1997), pp. 247–266.

[3] Karl J Friston et al. "Action and behavior: a free-energy formulation". In: *Biological cybernetics* 102.3 (2010), pp. 227–260.

[4] Karl Friston et al. "Active inference and epistemic value". In: *Cognitive neuroscience* 6.4 (2015), pp. 187–214.

[5] Kurtland Chua et al. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *arXiv preprint arXiv:1805.12114* (2018).

[6] Alexander Tschantz et al. "Reinforcement learning through active inference". In: *arXiv preprint arXiv:2002.12636* (2020).

[7] David McAllester and Karl Stratos. "Formal limitations on the measurement of mutual information". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 875–884.

[8] Ben Poole et al. "On variational bounds of mutual information". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5171–5180.

[9] Danijar Hafner et al. "Learning latent dynamics for planning from pixels". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[10] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

SmartHome: An Interactive Environment for Procedural Learning

Maxime Gazeau
LG Electronics AI Lab
Toronto
maxime.gazeau@lge.com

Royal Sequiera
LG Electronics AI Lab
Toronto
royal.sequiera@lge.com

Harmanpreet Singh
LG Electronics AI Lab
Toronto
harmanpreet.singh@lge.com

Abstract

Sequential decision making is the process where a player takes a decision conditioned on past observations, that will further impact future observations and decisions. Therefore, generalization in sequential decision making cannot be empirically measured with tools built on the i.i.d. (independent and identically distributed) assumption. In Reinforcement Learning, generalization is often studied via environment overfitting, where the learning algorithm itself is overspecialized to the environment. The research community has studied generalization in text-based games by creating environments aimed at training learning algorithms. TextWorld is one such environment, where a large number of text-based games can be generated from a predefined procedure. The performance of the learning algorithm is measured via completion of novel games generated by the same procedure. In such a framework, the concept of generalization implies that a player has learnt a procedure hidden in the environment dynamics by interacting with objects derived from the same class.

We propose SMARTHOME, a framework with a variety of procedures related to controlling smart devices in a home layout. Following object-oriented programming paradigm, the framework provides support for building procedures of the following capabilities: *object abstraction*, *task abstraction*, *encapsulation*, and *compositionality*. Each class of device contains a rich set of attributes and values. To solve these games, the player will have to interact with all objects to understand their properties and functionalities. In addition, the player must identify the procedure for each type of tasks and queries in order to generalize to new games with unseen objects.

We propose two baselines for the framework: an LSTM and a large-language model based DQN. We show that while rich contextual representations improve *task completion rate*, they alone are not enough to achieve generalization in the proposed framework.

Keywords: Procedural learning, Text-based games, Generalization

1 Introduction

Generalizing to novel situations is a goal of machine learning systems. The knowledge acquired while learning to generalize can be divided into two types: *declarative knowledge* and *procedural knowledge*. Declarative knowledge is static and can be retrieved from data in a one-off manner. Tasks like answer extraction in question answering or natural language inference fall under this category. In contrast, *knowing how to do something* is called procedural knowledge and is by nature sequential. Procedures abstract a goal-oriented task by extracting high-level actions necessary to complete a task.

Following Goyal et al. [6], we compare procedural learning to object-oriented programming (OOP) concepts: while class instances (objects) may vary from one another, the set of methods are common across different objects of the same class. Methods can therefore be thought of as knowledge at the level of class but not specific to an instance [6]. Procedures would often share the following characteristics: i) *object abstraction*: a procedure should be valid for all objects in the same class and not specific to one instance. ii) *encapsulation*: the internal details (attributes and values) of an object are only accessible via *get* and *set* methods. iii) *task abstraction*: a procedure orchestrates object methods to achieve a global objective iv) *compositionality*: a procedure reuses knowledge from simpler procedure within a complex one.

Similarly, we define generalization as the ability to interact with novel objects belonging to the same class, which directly relates to a procedure's *object abstraction* capabilities. During the training phase, the player acquires class-level knowledge by interacting with unique instances from the same class. In contrast, during the testing phase, the player is expected to transfer this knowledge to unforeseen objects generated from the same class.

Objective	**Hey, you are a personal assistant! There is something our customer needs you to do! Modify Two in One Ultra Lightweight Laptop to 40 in corridor.**
Game	*-= Livingroom =-* *You arrive in a livingroom. You begin to take stock of what's in the room. You make out a low table. The low table is ordinary. On the low table you can make out a remote.* *Hey, want to see a tv? Look over there, a tv. Looks like someone's already been here and taken everything off it, though.* *There is an exit to the east. Don't worry, it is unblocked. You don't like doors? Try go west.* *There is an Expensive C1 77 Inch 4k Smart OLED Television on the floor.*
Agent	**> Go east**
Game	*-= Corridor =-* *You've just walked into a corridor. Let's see what's in here. There is an exit to the east. Don't worry, it is unblocked. You should try going west. There is a stylish Two in One Ultra Lightweight Laptop on the floor.*
Agent	**> Examine stylish Two in One Ultra Lightweight Laptop**
Game	*The stylish Two in One Ultra Lightweight Laptop is currently switched off.*
Agent	**> Turn on stylish Two in One Ultra Lightweight Laptop**
Game	*You switch the stylish Two in One Ultra Lightweight Laptop on.*
Agent	**> Examine stylish Two in One Ultra Lightweight Laptop**
Game	*The stylish Two in One Ultra Lightweight Laptop is currently switched on. Attributes are: battery and volume.*
Agent	**> Query stylish Two in One Ultra Lightweight Laptop about volume**
Game	*The admissible values for the volume are 20 percent, 30 percent, 40 percent and 50 percent. The current value for the volume is 20 percent.*
Agent	**> Set volume of stylish Two in One Ultra Lightweight Laptop to 40**
Game	*You set the volume of stylish Two in One Ultra Lightweight Laptop to 40.* *Your score has just gone up by one point.* *You scored 1 out of a possible 1, in 6 turn(s).*

Figure 1: A game from the SMARTHOME environment

Since learning object interactions well leads to better generalization, it is imperative that a learning environment support rich interactions with objects in the environment. To this end, we propose SMARTHOME, a novel benchmarking environment where all procedures pertain to controlling devices in a house. The framework provides support for *object abstraction*, *task abstraction*, *encapsulation*, and *compositionality* through rich object interactions and variety of procedures. For example, the framework also includes information-seeking procedures, where the player needs to interact with the user to gather information that might be missing from the given objective. A typical SMARTHOME scenario is shown in Figure 1. Our contributions can be summarized a follows:

1. We propose SMARTHOME, a framework to test generalization in text based games.
2. We propose two baseline agents: LSTM and BERT-based DQN and we test for generalization in different configurations (number of rooms, devices, etc., in the house).
3. We show that when compared to representations learnt from LSTM, large language models help in improving the *task completion rate*. However, they are not sufficient for achieving generalization in SMARTHOME environment, making it a strong benchmark environment.

2 Related Work

Several interactive frameworks have been proposed, mainly in the textual and the visual modalities. In Interactive Question Answering (IQA) [5], for example, an agent is given an initial image, a scene, and a question. The agent needs

to plan for the goal, conditioned on the question by navigating the scene, understanding the objects through interactions. Similarly, Chevalier-Boisvert et al. [1] introduced the BabyAI research platform to support human in the loop grounded language learning. The framework comprises of 19 levels of increasing difficulty, with each level aimed toward acquiring a richer synthetic language.

On the other hand, a few text-based frameworks have been proposed using Microsoft TextWorld [2], an open-source, extensible engine that both generates and simulates text-based games. A base scenario is designed using Inform7, and the TextWorld library creates interactive plays of similar games to train a Reinforcement Learning (RL) agent. One of the most interesting features in TextWorld is the full control of the game generation process. Several research questions have been addressed using the TextWorld environment. For instance, in contrast to the conventional question answering task, Yuan et al. [8] pose a procedural learning-based task, where the agent arrives at an answer by interacting with a textual environment. ALFWorld [7], is an interactive-visual framework, where an agent navigates the environment to satisfy a given high-level objective such as `make coffee`, `clean floor`, etc. The authors employ the TextWorld environment to learn low-level actions such as navigating in the environment, which will then be transferred to the visual framework.

3 The Environment

SMARTHOME environment: We developed a new text-based game environment called SMARTHOME, a family of procedurally generated games in which the player needs to satisfy a user-specified goal. The proposed framework simulates a user formulating a smart home query to a virtual assistant (such as Alexa). Using the environment, games of different task and query types can be created, each containing a large number of devices (objects).

The objective of a game is determined by the task type it belongs to. Currently, there are three type of tasks: *simple*, *control* and *IFTTT* (If This, Then That). The tasks are compositional and are designed to be in increasing level of complexity. *Simple* tasks consist of finding a device in a home layout, and turning it on/off (`turn on the television`). *Control* tasks consist of modifying the value of an attribute for a specific device (`set the washing machine cycle in the laundry to normal`). All control task objectives require that a player must first execute a simple task i.e., find the device and turn it on. *IFTTT* tasks execute a simple or control task only when certain conditions are met (`turn on the lights after 9 pm`). The type of query determines the amount of relevant information given in the objective. There are currently four types of queries: *complete-valid, complete-invalid, incomplete-valid*, and *incomplete-invalid*. *Complete* indicates that the query contains enough information about the task, *incomplete* means that the player has to retrieve the missing information. On the other hand, *valid* indicates that the information from the query is correct and the player does not need to verify it. *Invalid* means that some information is wrong and the player must either correct it or express that the task cannot be done. It is important to note that the information about the completeness and validity of the games is not made explicit to the player.

All games are procedurally generated following two steps: an objective is uniquely sampled based on the type of tasks and queries. Given the objective, we create the corresponding device, attributes, and location. This is to ensure that the task becomes feasible. We then randomly sample other smart objects and locations.

Complex object interaction and generalization: TextWorld engine originally comes with a few challenges implemented such as the Cooking Challenge. Different scenarios were later developed [8] to test if an agent could answer questions related to an object's attribute. In all games, the agent must interact with the surrounding objects to learn their functionalities. However, the interaction is limited to few actions. The objects also have limited capabilities and only boolean attributes. In SMARTHOME, each object is uniquely identified by a tuple (adjective, name). At instantiation, an object is assigned a list of attributes, a location in the layout and each attribute with a value.

All objects share the same functionalities: they can be queried, turned on or off, examined and the setting can be modified. However, their location, attributes, and possible values depend on the object's class. We believe that an agent can only thrive in such an environment if the agent learns how to interact with complex objects. Memorizing the games will lead to poor generalization as the agent will not be able to transfer its knowledge to a new object.

Action space: The player progresses through the game by entering text commands that differ from one game to another. In text-based games, actions relate to navigation or object interaction. Inform7, contains default commands for checking the current location `look`, `inventory` or to navigate the layout (`go east, go west, go north, go south`). Players can interact with a device via `turn on/off` commands. In SMARTHOME, we simplified the layout to form a straight line. The goal is to limit the influence of navigation on *task completion rate* and to better assess the agent's abilities to interact with objects. In several of the previously proposed approaches, navigation controls the most of the difficulty of the games, while in SMARTHOME the game complexity is mostly driven by task and query types.

Additionally, we add a new set of commands. All games contain an application that lists devices grouped by location. The player can access this list using the command `examine app`. The player can check for the existence of a device and `examine {device}` to learn about its attributes. However, the device must first be turned on to reveal its attributes.

The current value of an attribute can be obtained from `query {device} about {attribute}`. A virtual user is also present in all games to answer questions related to the objective via the command `ask {user} about {something}`. Finally, all object attributes can be modified via `set {attribute} of {object} to {value}`.

Procedures and task compositionality: Other major differences between existing challenges in TextWorld and SMARTHOME are the types of procedure learned and the overall objective of the games. In Cooking Challenge, for example, there is only one procedure (prepare a meal) and is mostly based on navigation and interaction with the object is only done via simple actions (`take pan, drop pan etc.`). SMARTHOME contains several procedures all relating to different devices, involving more complex interaction with objects and less navigation. Additionally, it supports the learning of compositional procedures; for example, the procedure for a simple task with complete and valid queries is: `find the location of a device, turn on a device`, whereas for control tasks, the procedure is given by `execute procedure for a simple task, find all attributes of the device, find admissible attribute-values configurations, change the setting of the device`. In case the query is incomplete and valid, the procedure requires gathering extra information from the environment: `execute procedure for simple tasks, find missing information from the user, application, or directly on the device, execute the procedure for control task.`

4 The Agent

Partially Observable Markov Decision Process (POMDP): SMARTHOME are POMDP defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, r)$, where $\mathcal{S}$ is the set of environment states containing complete internal information of the game, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is the set of state transition probabilities and $\mathcal{O}$ is the set of observations provided to the agent. The reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$ is a sparse binary reward assigned at task completion. The agent's goal is to maximize its task completion rate. SMARTHOME games are also epistemic POMDP [4]; i.e., the agent observes only limited number of games during training and must transfer its knowledge to unseen games sampled from the same distribution.

Deep Q-Learning: We propose a neural DQN baseline agent with LSTM and BERT encoders [3]. The agent consists of three main components: encoders, an aggregator, and a feedforward DQN network. The encoders generate hidden representations based on environment state information, such as observation and objective, at game step t. The encoded information is then aggregated and passed on to DQN to predict q-values. Our current game state information consists of variable length sequences of observations including raw text, description, objective, inventory, and one action command to evaluate. The admissible commands are received as part of metadata information from the game engine for that particular game. We encode each of these inputs using LSTM or BERT and aggregate them into a single vector to pass it to the DQN network. Finally, a feed-forward network predicts a single q-value for the current game state and for this command in particular. Note that the game state doesn't include *game id* information.

Type of Task	Game Setting	Completion Rate
Simple, complete	r5-e5-n100	50.0
	r5-e5-n200	37.5
	r10-e10-n200	52.5
Control, complete	r5-e5-n100	40.0
	r5-e5-n200	27.5
	r10-e10-n200	22.5
Control, incomplete	r5-e5-n100$^{\diamondsuit}$	15.0
	r5-e5-n100$^{\heartsuit}$	32.5

Table 1: Results with BERT-DQN,
Tested on control incomplete games ($\diamondsuit$) and control complete games ($\heartsuit$)

Experimental Results: The training dataset is composed of games with five rooms (r5), five entities (e5) and different number of games ($n = 10, 100, 200$). The test games consist of unseen games. Entities, attributes, and values however were seen during training. We compared the training and test performance of encoders with different capacities (LSTM and BERT based) and varied state inputs. Agents were trained under different game settings and evaluated on three sets of 40 test games—one each for *simple-complete*, *control-complete* and *control-incomplete* tasks.

In all scenarios, the LSTM-based agent failed to solve more than 10% of the evaluation games. The BERT-DQN agent is more sample efficient (requires fewer episodes to train), trains faster in terms of wall-clock time, and has a higher

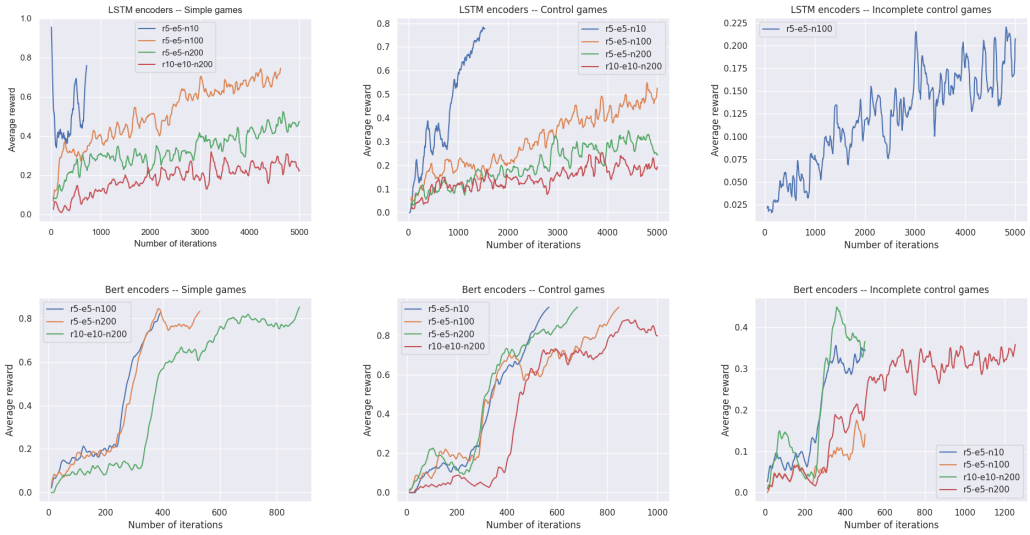


Figure 2: DQN baseline on SMARTHOME games

task completion rate for both simple and control games with *complete-valid* and *incomplete-valid* queries. Training results are displayed in Figure 2. This performance gap can be attributed to the larger model capacity and a richer contextual representation via the cross-attention computed between the objective and observations. However, the generalization gap with respect to *task completion rate* is still large even for the BERT-DQN model (see Table 1). We observe that *simple games* are easier to solve than *control games* and generalize better owing to the limited object interaction. We demonstrate decent generalization performance for an agent trained on *control-incomplete* tasks and tested on complete queries, where the agent can solve about 33% of the games. However, there is a performance drop when tested on *incomplete games*.

Conclusion: We proposed the SMARTHOME environment, interfaced with the OpenAI gym API, in which object affordance are more complex than existing text-based games like Zork or TextWorld. We provided two baselines and showed preliminary results towards generalization via learning to interact with complex objects and procedures. Further analysis should compare the agent's performance in existing text-based games with simpler object affordance.

References

[1] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

[2] Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pages 41–75. Springer, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4089–4098, 2018.

[6] Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

[7] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[8] Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Christopher Pal, Yoshua Bengio, and Adam Trischler. Interactive language learning by question answering. *arXiv preprint arXiv:1908.10909*, 2019.

Diverse partner creation with partner prediction for robust K-Level Reasoning

Jarrod J. Shipton
Department of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg
Jarrod.Shipton@wits.ac.za

Benjamin Rosman
Department of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg
Benjamin.Rosman1@wits.ac.za

Abstract

In Multi-agent Reinforcement Learning (MARL) there has been a substantial move towards creating algorithms which can be trained to work cooperatively with partners. In general this is done in a self play (SP) setting, where the agents are set to play and train with copies of themselves in a Decentralized Partially Observable Markov Decision Process setting. Agents trained with SP often result in behaviour such that arbitrary conventions, or ”handshakes”, will be formed in order to more efficiently achieve their goal. These arbitrary handshakes can be seen as unwanted behaviours as they creates the issue that when agents are paired with novel agents they will often not be able to complete a task cooperatively, even when paired with different training runs of the same algorithm. A valuable architecture to help tackle this problem is synchronous K-level reasoning with a best response (SyKLRBR), which creates agents that have policies based on grounded information which are robust to various handshakes. Weaknesses are still shown in that certain agents with specific handshakes still outperform this agent when paired with one another as compared with the SyKLRBR agent. This work expands on the SyKLRBR framework by factorizing the action-observation histories to fit a belief over a diverse set of agents created with multiple different runs of a modified SyKLRBR algorithm. These modifications allow the algorithm to create and identify a robust set of agents with various handshakes that could exist in potential novel partners, ultimately allowing it to take advantage of these handshakes for better results.

Keywords: Multi-Agent Reinforcement Learning
Partial Observability
Zero-shot Coordination
K-Level Reasoning

Acknowledgements

The authors would like to thank the members of RAIL Lab based at The University of the Witwatersrand, Johannesburg for valuable discussion and support.

1 Introduction

In Multi-agent Reinforcement Learning (MARL) there has been a substantial move towards creating algorithms which can be trained to work cooperatively with partners. This is generally done in self play (SP), where the agents are set to play and train with copies of themselves in a decentralized Partially Observable Markov Decision Process (Dec-POMDP) setting.

Agents trained with SP often result in behaviour such that arbitrary conventions over actions will be formed in order to more efficiently achieve their goal. An example of such a convention can be observed in the Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning (SAD) [4] where the agents are trained to play the game Hanabi. Hanabi is a card game comprising of cards with five different colours and five different ranks. Players take turns without communicating, besides through allowed gameplay actions, to play cards from their hands to form five piles of cards with matching colour and increasing rank without making more than three mistakes throughout the game. Each player does not have full information of the contents of their own hands and has to infer what each card is from the actions of other players. Each player may perform one of three main actions on each of their turns: give a hint about cards of specific quality to a single player, play a card, or discard a card. It has been observed that the trained agents used hints for purposes unintended by the game rules, such as, hinting the colour red meant the next player should play their first card instead of just signalling the red cards. Such an arbitrary convention, which we refer to as a "handshake", would be a desired and welcomed result if these agents are only to be paired with copies of themselves, or another agent with understanding of these handshakes. However, such a constraint detracts from reality, where the agent could have formed a handshake for any of the available colours, leading to a situation where a novel (previously unseen) partner would not know the handshake, and possibly act on their own, differing handshake leading to an undesired outcome.

It quickly becomes obvious that agents with specific handshakes will perform poorly outside of SP, that is, paired with novel agents. It is not even guaranteed that such an agent will perform well in cross-play (XP), where agents that are trained by the same algorithm, but in different training runs are paired at testing time. Notably, the take away from this is that well-trained SP agents, potentially using their own unique handshakes, will not always perform well with other well-trained, novel partners at test time. In works such as Other Play (OP) [6], Synchronous K-Level Reasoning with a Best Response (SyKLRBR) [3], and Off Belief Learning [5] the authors have developed methods to aid agents to be robust to handshakes and work cooperatively with novel agents at test time, or termed otherwise, aiding them to achieve zero-shot coordination (ZSC). These works show that agents which are robust to handshakes will perform better than those that have been trained with specific handshakes in XP or when paired with novel agents. However, this avoids a factor that can be taken advantage of: some of these novel partners perform better with their own handshake than with robust agents. It should thus be a goal to have an agent that is able to learn a policy that is both robust to many different handshakes, but also capable of identifying any known handshakes and responding to these in an optimal fashion.

In this work we take the idea that agents should be trained with a variety of agents which differ in behaviour to keep them robust to specific handshakes, however, we also wish for the agent to play in a specific manner when paired with an agent that has a similar behaviour to an agent it has trained with previously. To do this we propose a method to create a diverse set of training agents, and we train our agent to identify which of these agents it is playing with so that it can respond to their behaviour it recognizes to be a known handshake with the correct response, while still being robust to various play patterns.

2 Background

2.1 Dec-POMDPs

In the Dec-POMDPs setting we have N agents $i \in \{1, \ldots, N\}$. These agents receive observations $o_t^i = O(s_t)$ which are derived from the underlying state $s_t \in S$ for each agent at each timestep t. The agent then, at each timestep, performs an action, u_t^i from its policy $\pi_i \sim \pi(u^i|\tau_t^i)$, where τ_t^i is the action-observation history (AOH) of the agent. We define $\tau_t^i = \{o_0^i, u_0^i, r_1, \ldots, r_{t-1}, o_t^i, u_t^i\}$, where r_t, in this work, is the shared reward of the environment at each timestep defined by $R(s,t)$, which is the environment's reward function. The goal of the agents is to maximise the total expected reward, $\mathbb{E}_{\tau \sim P(\tau|s,u)}[R_t(\tau)]$. In this context $R_t(\tau)$ is the discounted sum of rewards for the entire AOH, $R_t(\tau) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, where γ is the discount factor. In general this could be for an infinite horizon, but in this work the environment is bounded by a maximum timestep limit of t_{max}.

2.2 SyKLRBR

The method SyKLRBR is a modified combination of two previously studied ideas, Cognitive Hierarchies (CH) [1] and K-Level Reasoning (KLR) [2], which deal with how to create a policy that will produce the best response (BR) to another policy's actions. The BR is the action taken by an agent which leads to the maximal reward given the other agent's action.

In CH k agents are trained to act in a given environment. The first of these agents is trained to act in the environment with versions of itself and is referred to as the first of k-level agents. Each of the subsequent k-level agents are trained with versions of themselves and a distribution of the previous $k-1$-level agents. A Poisson distribution of the previous k-level agents is commonly used. Each will then have a policy, π_k, which is a best response to the all previous k-level agents. In KLR agents are trained in a similar fashion, however they are only trained as a BR for the k-level policy directly before it, $k-1$-level. In both of these methods the agents of each k-level are trained until a predefined halting point, and only then is the next level trained.

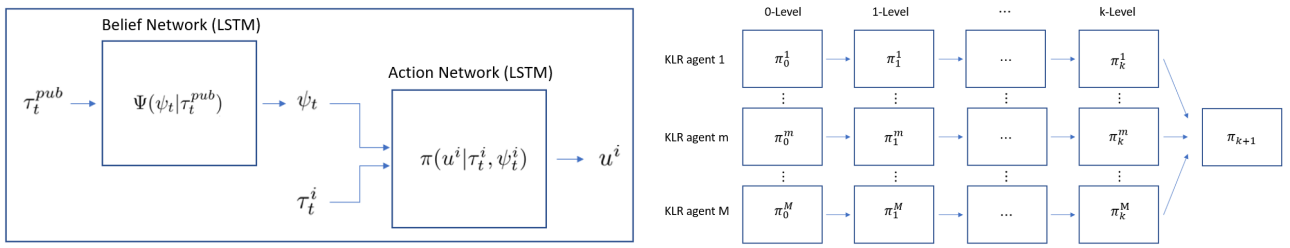
A Recurrent Replay Distributed Deep Q-Network (R2D2) [7] is used in SyKLRBR to train all k policies, π_k, synchronously, initializing the parameters, θ, of all policies randomly. The first policy of the k-level policies, π_0, picks an action randomly from each of the available actions each turn, with each action having the same probability. That is, the action is chosen at random from a uniform distribution over the action space. This ensures that there is no implied handshake in any of its actions, meaning all information it shares through its actions is grounded information. The policies for $k \geq 1$ are then trained with this as their $k = 0$ policy, sampling training partners from the k-levels below it using a Poisson distribution as in CH. The synchronous training is shown to produce a more robust agent than when training the levels sequentially as in KLR.

3 Proposed Method

Our method proposes that the policy of the agent should depend on its belief of the agent type it is playing with as well as on the AOH. In order to have a belief over the types of agents we are playing with, we would need to have a set of diverse agents to train with and for the environment episodes to be sufficiently long such that behaviour can be observed to form such a belief. We thus need to produce agents which are diverse, and keep a belief over which of these agents our agent is playing with.

In a Dec-POMDP setting the policy of an agent is dependent on its AOH, τ_t^i. Our work proposes that this should instead be based not only on the AOH, but with the belief of the agent type it is playing with as well. In order to infer a belief of the type of agent we are working with, we need to fit this belief only to observations and actions the other agent executes. Thus the belief is based on a distribution dependent on a factorized view of the AOH τ_t^i. The specific factorization of the AOH is into a public AOH, $\tau_t^{i,pub}$, and private AOH, $\tau_t^{i,priv}$. The public AOH contains all information about the trajectory which is available to be observed by all agents at each time step t, while the private AOH is the information only observable to agent i at each time step t. Using τ_t^{pub} we fit a distribution, $\Psi(\psi_t|\tau_t^{pub})$, where ψ_t is the belief of which of the agents we are playing playing with. This distribution is learned using an LSTM and is trained alongside the main policy, $\pi_i \sim \pi(u^i|\tau_t^i, \psi_t^i)$ which now incorporates ψ_t^i. A visualisation of this policy can be seen in Figure 1a.

To produce the set of diverse agents with which ours trains, we train M different SyKLRBR training cycles replacing their policy structure with ours. That is, we train $M \times K$ different policies $\pi_k^m \sim \pi(u^k|\tau_t^k, \psi_t^k)$. In each training run $m \in M$ the policy π_0^m is set to have randomized and differing probability weights for each action in the action space resulting in M differing sets of k-level agents. This is a particularly useful feature of the SyKLRBR training structure since for a given policy π_0^m the kth level policy will differ from the policy of another training run m, provided the initial distribution for π_0^m between the two runs m differs enough. The final policy, π_{k+1} is then trained uniformly with each of the sets of M agents, each set having a Poisson distribution over its respective k-levels. This has been visually represented in Figure 1b.



(a) (b)

Figure 1: a) The structure of the policy network. Using the public AOH as the input for the belief network, we produce the belief ψ_t, which is then paired with the AOH as inputs to the action network which chooses an action u^i. b) The structure of the M K-level policies with the K+1-level final policy. Each row represents a SyKLRBR policy run, with each arrow showing a preceding level of that run. Each successive policy is trained with a Poisson distribution of the previously connected policies, and π_{k+1} being uniformly paired with each of the m training runs.

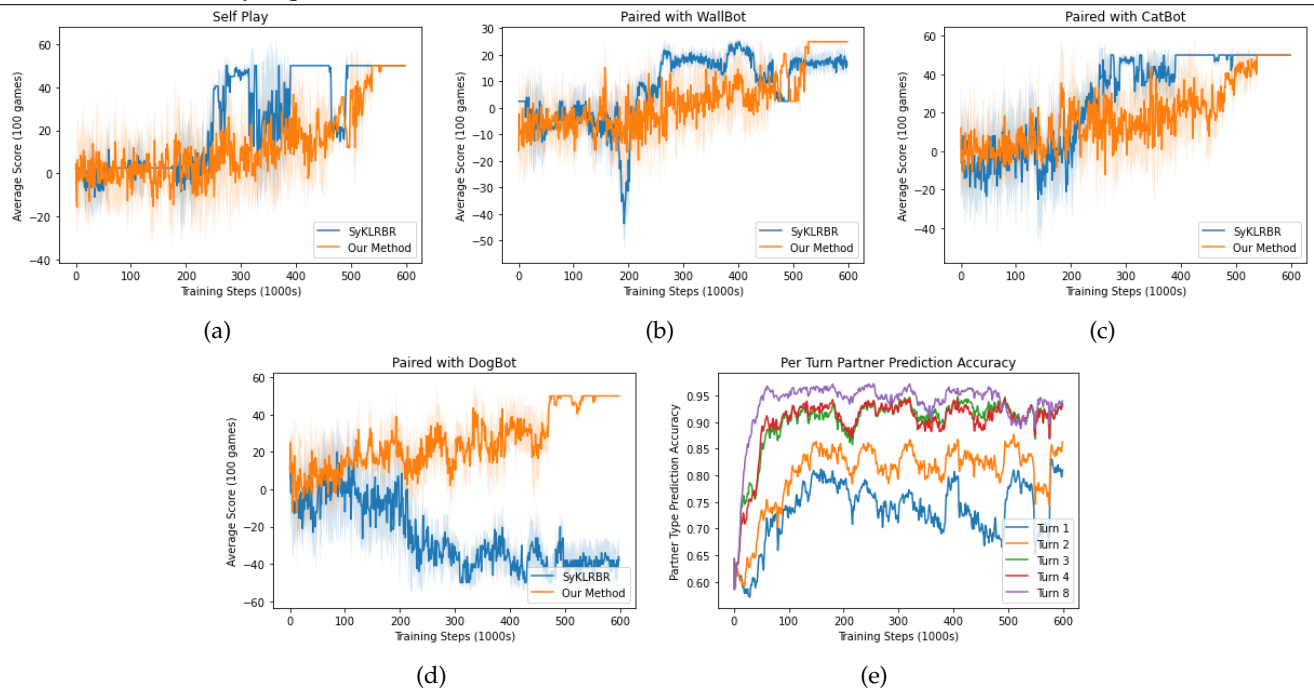
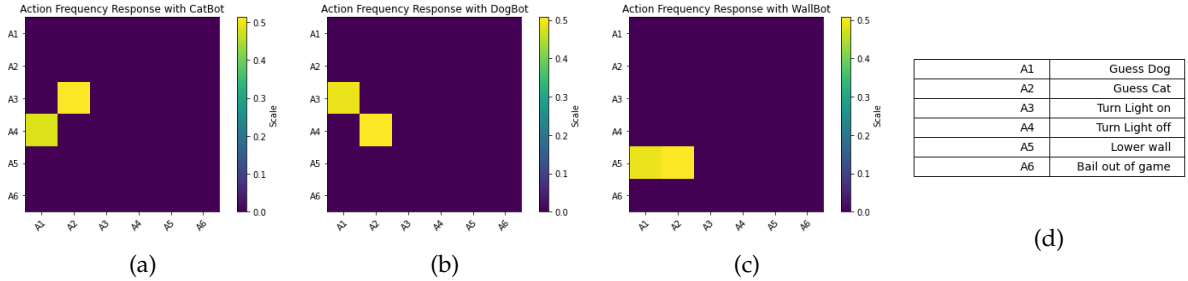


Figure 2: The comparison in scores between our proposed method and SyKLRBR during training is shown for (a) self play, (b) paired with WallBot, (c) paired with CatBot, and (d) paired with DogBot. (e) The per turn partner prediction accuracy during training of our proposed method.



A1	Guess Dog
A2	Guess Cat
A3	Turn Light on
A4	Turn Light off
A5	Lower wall
A6	Bail out of game

(d)

Figure 3: (a) The action response pairs of our proposed method when paired with CatBot. (b) The action response pairs of our proposed method when paired with DogBot. (c) The action response pairs of our proposed method when paired with WallBot. (d) Action reference table.

4 Evaluation

To evaluate this architecture and to allow for a prolonged interaction to create a belief of the player type we take a repeated version of the toy environment from OBL [5], repeating the game 5 times for each episode. This is a team game which involves two players with shared reward. In this game, there is a binary variable $pet \in \{dog, cat\}$ which player 1 can observe, however player 2 cannot due to a wall between the two players. This wall can be lowered by player 1 to reveal to player 2 which pet is with player 1. If this action is taken all reward values are halved, resulting in a maximum reward of 5. Furthermore there is a light which can be turned on by player 1 and observed by player 2, which is provided as a communication avenue to allow for the formation of handshakes beyond that of just lowering the wall. The goal of the game is for player 2 to correctly guess what pet player 1 has: if they guess correctly the team of two get a reward of 10, if they guess incorrectly the team gets a reward is -10. There is also a third option of avoiding the guess and to bail out of the game for a team reward of 1.

For the purpose of testing our algorithm on this environment we created three rules-based bots: one which always lowers the wall, and chooses the correct option if the wall was lowered (WallBot), one which turns on the light if the pet is a cat, and picks cat if the light is turned on (CatBot), and the third turns on the light if the pet was a dog, and picks dog if the light is turned on (DogBot). Our training run using our proposed method had $M = 4$ and $K = 4$. We trained a SyKLRBR version of the agent, with $K = 4$ for comparison. We found that both our algorithm and SyKLRBR achieved 50 points in

SP. However, when we tested these trained agents with the rules based bot, a difference in results was apparent, as seen in Figures 2a-d. The standard SyKLRBR agent achieves a perfect score of 50 points with either CatBot or DogBot, and a negative score with the other, occurring with an approximately even distribution across training runs. The SyKLRBR agent trained for this comparison, scores 50 points with CatBot, and -38 ± 5.9178 with DogBot. Our proposed method consistently achieves 50 points with both CatBot and DogBot.

Furthermore it can be seen that when our bot accurately predicts the agent it is playing with, the action it responds with matches the convention of that agent, as is shown in Figure 3. In Figure 3a we observe that for the CatBot, the response to turning on the light (action A3) is guessing cat (action A2), and the response to seeing a cat is turning on the light (action A3) with the CatBot responding by guessing cat (action A2). Similarly the correct responses can be observed for DogBot and WallBot.

While identifying the agent type is important, there is no accurate measure of how well it can predict if it is playing with CatBot, DogBot or WallBot. This is due to the fact that it is trained to identify which of the $M \times K$ partners it is playing with, and the accuracy of the prediction can lower if any of those $M \times K$ partners behave similarly at the end of training, which is probable for an environment such as this one where there are three optimal strategies. However, if the behaviour is similar between training partners, the best response will be similar, and thus while the measured accuracy decreases, it does not necessarily adversely affect the final agent. The accuracy of the belief of which partner the final agent is playing with increases over the course of the game and is shown, on a per turn basis, in Figure 2e.

In this environment we have discovered this algorithm is capable of learning to identify diverse agents created by the M training runs of our proposed architecture, identifying their type after as little as 4 actions, which is only possible due to the repeating version of this toy environment, which would otherwise end in 2 actions. It is important to reiterate that, in line with the assumptions of the model, this model requires an environment in which trajectories of a minimum sufficient length exist, to identify behaviour of these agents, acting in a similar fashion to how an agent trained by regular SyKLRBR would until it has successfully identified the agent type it is partnered with, after which it behaves in a manner that is able to play successfully with that agent.

5 Conclusion

By using M different distributions over the action space to initialize π_0 for each of the M SyKLRBR training runs, we are capable of creating a diverse set of agents. The addition of the belief network allows the agent to fit AOH trajectories to specific agent types. With these features our proposed method becomes both the source and identifier of diverse agents. These agents can then be used as training partners during the training process of this algorithm resulting in a robust agent that is capable of identifying the type of partner it is working with. This allows it to be able to match the handshake of an unknown agent if its behaviour mimics that of a known agent, while still maintaining a high level of robust behaviour for agent types it has not come across. This does come with the cost of long initial training times. These preliminary results on the toy environment presented in this paper are promising and offer an exciting avenue to apply this to environments such as Hanabi.

References

[1] Colin F Camerer, Teck-Hua Ho, and Juin-Kuan Chong. A cognitive hierarchy model of games. *The Quarterly Journal of Economics*, 119(3):861–898, 2004.

[2] Miguel A Costa-Gomes and Vincent P Crawford. Cognition and behavior in two-person guessing games: An experimental study. *American economic review*, 96(5):1737–1768, 2006.

[3] Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. K-level reasoning for zero-shot coordination in hanabi. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1912.02288*, 2019.

[5] Hengyuan Hu, Adam Lerer, Brandon Cui, Luis Pineda, Noam Brown, and Jakob Foerster. Off-belief learning. In *International Conference on Machine Learning*, pages 4369–4379. PMLR, 2021.

[6] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[7] Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2d2: repeatable and reliable detector and descriptor. *arXiv preprint arXiv:1906.06195*, 2019.

AppBuddy: Learning to Accomplish Tasks in Mobile Apps via Reinforcement Learning (Extended Abstract)*

Maayan Shvo
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
maayanshvo@cs.toronto.edu

Zhiming Hu
Samsung AI Center
Toronto, Ontario, Canada
zhiming.hu@samsung.com

Rodrigo Toro Icarte
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
rntoro@cs.toronto.edu

Iqbal Mohomed
Samsung AI Center
Toronto, Ontario, Canada
i.mohomed@samsung.com

Allan Jepson
Samsung AI Center
Toronto, Ontario, Canada
allan.jepson@samsung.com

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract

Human beings, even small children, quickly become adept at figuring out how to use applications on their mobile devices. Learning to use a new app is often achieved via trial-and-error, accelerated by transfer of knowledge from past experiences with like apps. The prospect of building a *smarter* smartphone — one that can learn how to achieve tasks using mobile apps — is tantalizing. In this paper we explore the use of Reinforcement Learning (RL) with the goal of advancing this aspiration. We introduce an RL-based framework for learning to accomplish tasks in mobile apps. RL agents are provided with states derived from the underlying representation of on-screen elements, and rewards that are based on progress made in the task. Agents can interact with screen elements by tapping or typing. Our experimental results, over a number of mobile apps, show that RL agents can learn to accomplish multi-step tasks, as well as achieve modest generalization across different apps. More generally, we develop a platform[1] which addresses several engineering challenges to enable an effective RL training environment. Our AppBuddy platform is compatible with OpenAI Gym and includes a suite of mobile apps and benchmark tasks that supports a diversity of RL research in the mobile app setting.

Keywords: AI Applications, Reinforcement Learning, Machine Learning, Mobile Applications

Acknowledgements

This work was all done at Samsung AI Center.

*The full paper appeared in the Proceedings of the 34th Canadian Conference on AI. 2021 [6].
[1]Code, Technical Appendix, and Video Demonstration can be found in https://www.cs.toronto.edu/appbuddy.

1 Introduction

Billions of people around the world use mobile apps on a daily basis to accomplish a wide variety of tasks. Building *smarter* smartphones that can learn how to use apps to accomplish tasks has the potential to greatly improve app accessibility and user experience. We explore the use of Reinforcement Learning (RL) to advance this aspiration.

RL has been applied in a diversity of simulated environments with impressive results [4, 7]. However, a myriad of challenges can prohibit the application of RL in real-world settings [1]. Learning to accomplish tasks in mobile apps is one such setting due to the usually large action space (the agent can interact with many elements on every screen), sparse rewards, and slow interaction with the environment (i.e., a physical phone or an emulator) that together make the collection of a large number of experience samples both necessary and arduous.

Recent work proposed to use supervised learning techniques to train computational agents to accomplish tasks in mobile apps [3]. A shortcoming of this approach is that it requires the creation of large training sets of labeled data. In contrast, RL agents can learn to solve tasks without human supervision by autonomously learning from interacting with mobile apps, and they can potentially learn better solutions than supervised learning approaches, as shown by previous work (e.g., [8]). Closer to our work are recent efforts that use RL to solve tasks using web interfaces [5] or that test mobile apps [2]. Coincident with the publication of the full paper describing our work and platform [6], Toyama et al. released a related open-source platform for RL research built on top of the Android ecosystem [9]. Perhaps the most notable difference between the two platforms is that Toyama et al.'s RL agents interact with screen pixels, whereas we exploit Android's view hierarchy to define the RL agent action space so that actions are interacting with different user interface (UI) elements on the screen. While our platform is amenable to a pixel-based action space, in our experience, defining the action space based on on-screen elements significantly improves sample efficiency for learning. This is particularly important here because of the relatively slow execution time in a phone environment.

In this paper, we explore whether an RL agent can learn policies that consistently solve tasks in real-world mobile phone apps. Our main contributions are as follows:

- We formulate the app learning task as an RL problem where the state and action space is derived from the phone's internal representation of screen elements and reward is modeled so as to incentivize intermediate task steps, while learning policies that complete tasks.
- We construct a mobile app learning environment that is engineered to collect experiences from multiple emulators simultaneously. The environment is made compatible with OpenAI Gym to support various RL algorithms. We also build several tools for efficient provisioning of Android emulators, obtaining emulator states, and interacting with the emulators.
- We experimentally evaluate our RL agent on a suite of benchmarks comprising a number of apps and tasks of varying difficulty. Results (i) demonstrate that RL agents can be successfully trained to accomplish multi-step tasks in mobile apps; (ii) expose the impact of design decisions including reward modeling and number of phone emulators used in training; and (iii) demonstrate the ability of our approach to generalize to similar tasks in unseen apps.
- We develop the AppBuddy training platform that includes the aforementioned mobile app learning environment together with a suite of mobile app-based benchmarks, allowing researchers and practitioners to train RL agents to accomplish tasks using various apps.

This paper represents an important step towards endowing smartphones with the ability to learn to accomplish tasks using mobile apps. The release of the AppBuddy training platform and suite of benchmarks opens the door to further work on this impactful problem by the broader research community.

2 System Design

Our objective is to explore whether an RL agent can learn to accomplish tasks in mobile apps by interacting with (either a physical or an emulated) phone environment. Mobile apps are interesting and challenging real-world RL benchmarks. Since they are optimized for accessibility, most tasks can be solved after executing a short sequence of actions. However, the branching factor (i.e., action space) is much larger than in a standard RL benchmark, which leads to a very sparse reward signal. Moreover, interacting with Android emulators is slow. These ingredients make mobile apps a challenging benchmark with interesting structure that, hopefully, RL agents will learn to exploit when solving these problems.

In this section, we present the basic building blocks to use RL in mobile apps: *action space, state representation and reward specification.* The overview of the system is shown in Figure 1. The agent (a neural network trained using PPO) interacts with several Android emulators in parallel to collect experiences. At each step, for each emulator, the agent chooses to tap on (or type into) a particular element on screen. After the agent performs the actions, the environments return *states* derived from the available on-screen information, together with *rewards* that depend on the current task the agent is solving.

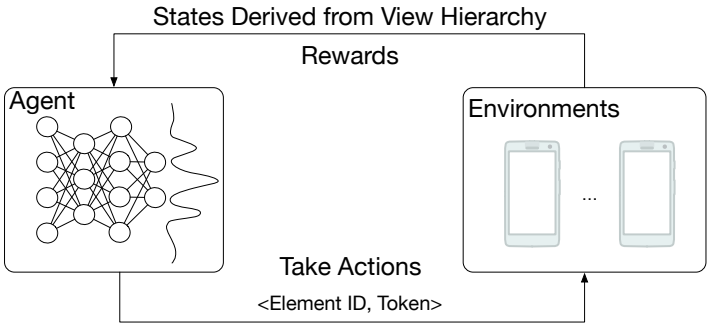


Figure 1: Overview of the proposed RL framework. The environment contains a group of emulators and the states are derived from the view hierarchy of the current screen. In each step, the agent will choose an action, which consists of an element ID and a token, and receive corresponding rewards based on progress made in the task.

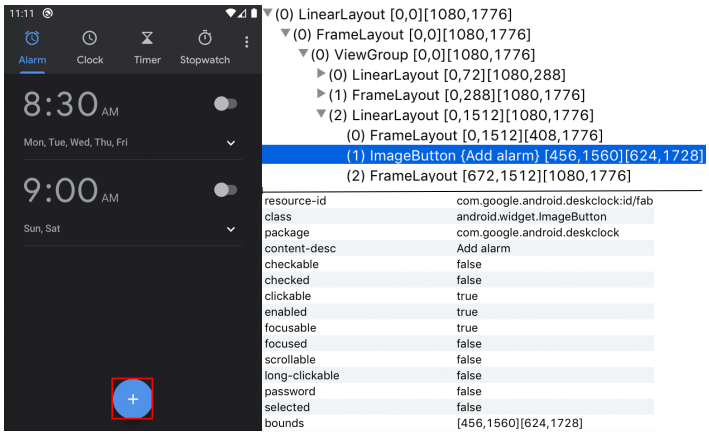


Figure 2: An example of the view hierarchy for a given screen. The '+' button with the red border on the left-hand side directly corresponds to the highlighted element ('Add alarm') in the view hierarchy on the right hand side.

Some tasks might require typing some particular text. For instance, the task "*add a new Wi-Fi network named Starbucks*" will require the agent to type *Starbucks* at some point. However, it is practically impossible for an RL agent to discover that typing *Starbucks*, letter by letter, will cause it to receive a reward. To handle this issue, we follow the same standard as in the web-based task literature [5] and provide the agent with a list of tokens to choose from when typing text. With that, we now describe the action space, state representation, and rewards used in our work.

2.1 Action Space

The main challenge towards successfully applying RL in smartphones is the unreasonably large action space. Technically, a user might tap on any pixel on the screen. But allowing that level of granularity would make learning infeasible. Fortunately, we can reduce the action space to the set of elements on screen by exploiting the information from the *view hierarchy*.

In Android applications, each view is associated with a rectangle on the screen. The view is responsible both for displaying the pixels on the screen as well as handling events in that rectangle. All the views on a particular screen are organized in a hierarchical tree structure, which is also called the view hierarchy. We show a simple example of a view hierarchy in Figure 2. In this figure, the left part shows a screenshot of the native Android alarm clock app. On the right, we can see part of the view hierarchy at the top and detailed attributes for the selected view called 'ImageButton {Add alarm}' at the bottom. The selected view is also highlighted in a red rectangle in the screenshot on the left.

Since the view hierarchy is always available for any Android application, we use its information to define the action space. From this hierarchy, we automatically extract the list of user interface (UI) elements on the screen. Then, the actions at the agent's avail are tuples comprising a UI element index (w.r.t. that list) and a token (as shown in Figure 1). The element index ranges from 0 to n-1, where n is a fixed upper bound for the maximum number of elements on any screen. We say that a UI element is *clickable* if it reacts when tapped and that a UI element is *editable* if text may be typed

into it. When the agent chooses to interact with some UI element, the agent will *type* into that element if the element is *editable* and *tap* on that element if the element is *clickable*.

In addition to selecting a UI element index, the agent also chooses a single token from a set of k tokens (in our experiments, k is 4) that are predefined for each task. For instance, if the task is "*add a new Wi-Fi network named Starbucks*," then *Starbucks* will be among the k tokens for that task. Then, the chosen token is typed into the selected UI element if that element is editable.

2.2 State Representation

As explained above, the action space is defined by the list of UI elements in the current screen, which is extracted from the screen's view hierarchy. To represent the state, we use the same list of UI elements. Concretely, the state is represented by an $n \times m$ matrix, where each row is a vector of m features representing a particular UI element from the list and n is an upper bound for the maximum number of elements that we expect to see on any screen. In the matrix, each UI element is represented using m features. These features include the *textual description* of the UI element (which is embedded using a pretrained BERT model). This description is available in the view hierarchy and specifies the purpose of the UI element (e.g., *'Add alarm'* for the '+' button in Figure 2). We also include information about whether the UI element is *clickable* or *editable* and its relative location in the view hierarchy (defined as the element's pre-order tree traversal index). The relative location features help capture the spatial correlations across different UI elements.

2.3 Reward Specification

When learning to accomplish tasks in mobile apps, reward is extremely sparse if the agent is only given a positive reward when accomplishing the task and a reward of 0, otherwise. To mitigate for reward sparsity, we specify the reward function R such that the agent is given an intermediate reward for reaching certain states in the app that correspond to sub-goals on the way to accomplishing the task. R is calculated as follows:

$$R = \sum_{i=0}^{k-1} (w_i \times r_i) + w_k \times r_k, \tag{1}$$

where $\mathbf{w} = \{w_0, w_1, ..., w_k\}$ indicates whether the agent has reached a certain state and $\mathbf{r} = \{r_0, r_1, ..., r_k\}$ represents the rewards. k is the total number of intermediate steps where the agent will receive positive intermediate rewards. The value of k is task-specific. Note that an intermediate reward r_i may only be given once in an episode and w_i is set to 0 after that. r_k is the reward returned by the environment when the task is complete (based on the view hierarchy extracted from the emulator).

For example, in order to accomplish a task in the settings app in our benchmarks, an agent must add a new Wi-Fi network named 'Starbucks'. $w_0 = 1$ when the agent reaches the 'Wi-Fi settings' screen and $w_1 = 1$ when the agent reaches the 'add new Wi-Fi network' screen. $w_2 = 1$ if the agent has added a new Wi-Fi network called 'Starbucks' and it now appears on the screen. In the next section, we discuss our experiments where we show the impact of excluding intermediate rewards (i.e., where only $w_k = 1$ and, hence, the agent receives a sparse reward for accomplishing the task).

3 Experimental Evaluation

The objectives of our evaluation were: 1) to show that RL can be used to learn policies that accomplish tasks in mobile apps; 2) to expose the impact of design decisions including reward modeling and number of phone emulators used in training; and 3) to demonstrate our approach's potential for generalization to similar tasks in unseen apps.

We ran experiments using PPO2 (an implementation of PPO made for GPU). The emulators were provisioned using Docker-Android with headless mode and KVM acceleration enabled. With this setup, we were able to train the agent with tens of emulators on a single machine.

Benchmarks. We experimented with 4 mobile apps (expense splitting, shopping list, alarm clock, and settings), where each app includes 3 tasks of varying difficulty.

Experiment Protocol. In each experiment, we ran PPO2 for some number of policy updates and evaluated the agent's current policy after each update. To evaluate the policy, we estimated its success rate by running the policy 100 times and counting how many times the policy was able to accomplish the task within 25 steps.

Summary of Results. Our results indicate that RL can be used to learn policies that accomplish tasks in mobile apps. Moreover, our results show that providing the agent with intermediate rewards was crucial in learning to accomplish many of the tasks. Plots of our results can be found in the full paper [6].

3.1 Applying Learned Skills to Unseen Apps

Despite our positive results, deploying our approach on real phones is still a ways away. In particular, imagine a human user of an alarm clock app. After learning how to use alarm clock app A, the user will typically have an easier time learning how to use alarm clock app B, assuming some similarity between the apps. We would like an RL agent to possess similar capabilities – learn a policy that accomplishes a task on one app and then use that trained policy to accomplish a similar task on a different, yet similar app.

To experiment with this idea, we began with a naive approach: we take a policy trained on the easy alarm clock task in our experiments and deploy this trained policy in a different app – the native Android alarm clock app. The unseen native app has many commonalities to the training app, both in form and in function, however, this naive approach *failed* to accomplish the same task (i.e., setting an alarm clock) in the unseen app, likely because the RL agent did not learn to focus on the appropriate features and instead memorized the element ID and token combination that should be chosen in each state. To remedy this, we *shuffled* the on-screen UI elements in the state representation given to the agent during training. In this way, the agent can no longer simply memorize the element ID that should be selected and must, instead, learn and attend to the relevant features of each element in the state representation. We moreover hypothesized that augmenting the accessibility text in the training app will help the agent generalize to the unseen app.

To test our hypothesis, we trained a policy on the open source alarm clock app while also shuffling the state representation, which now included relevant text in the accessibility field. Our results show that by shuffling the state representation, the agent was able to generalize to a similar task in the unseen app.

4 Concluding Remarks

Building smarter smartphones has the potential to broaden accessibility of phone applications and improve the user experience. In this paper, we explored the use of RL to learn to accomplish tasks using mobile apps. RL agent states were derived from the underlying representation of on-screen elements, rewards were based on progress made in the task, and agents could interact with elements on the phone screen by tapping or typing. Our experiments showed that our RL agents could learn to accomplish multi-step tasks in a number of mobile apps, as well as achieve modest generalization across different mobile apps. An important contribution of our work was the development of a mobile app RL environment that is compatible with OpenAI Gym and its provision through the AppBuddy training platform. The release of this training platform together with a suite of benchmarks opens the door to further research into learning to accomplish a diversity of tasks using mobile apps.

References

[1] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

[2] Y. Koroglu, A. Sen, O. Muslu, Y. Mete, C. Ulker, T. Tanriverdi, and Y. Donmez. Qbe: Qlearning-based exploration of android applications. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 105–115. IEEE, 2018.

[3] Y. Li, J. He, X. Zhou, Y. Zhang, and J. Baldridge. Mapping natural language instructions to mobile UI action sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[5] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. World of bits: An open-domain platform for web-based agents. In *ICML*, pages 3135–3144, 2017.

[6] M. Shvo, Z. Hu, R. Toro Icarte, I. Mohomed, A. Jepson, and S. A. McIlraith. Appbuddy: Learning to accomplish tasks in mobile apps via reinforcement learning. In *34th Canadian Conference on Artificial Intelligence*, 2021.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017.

[9] D. Toyama, P. Hamel, A. Gergely, G. Comanici, A. Glaese, Z. Ahmed, T. Jackson, S. Mourad, and D. Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

1 Introduction

In a changing environment, decision makers often face a choice between exploiting old but known options or switch to exploring lesser known but potentially more rewarding alternatives. In contrast to stable environments, the quality of options varies over time and exclusively choosing a known option entails the risk of missing alternatives that are more rewarding. Accordingly, gathering information through exploration enables adaptation to environmental contingencies and the potential of earning more reward than with exploitation. However, excessive exploration reduces long-term payoffs. By extension, agents may explore for distinct but hard to distinguish reasons – to maximize reward, to reduce uncertainty or simply randomly. While computational modeling allows to disentangle these forms of exploration to some extent (Gershman 2018; Wilson et al. 2021; but see below), from a model-free perspective the investigator is typically only left with counting behavioral switches, leaving the motivation for a given switch unclear.

Relatedly, in classical multi-armed bandit tasks used to study exploration-exploitation dilemma reward and information are confounded (Wilson et al. 2014). Specifically, continuous selection of the most rewarding arm automatically makes this arm relatively more certain (i.e., less informative) while the informativeness of less rewarding arms automatically increases. From a modeling perspective, this pollutes computation of the uncertainty bonus and impedes interpretation of whether a low uncertainty bonus reflects uncertainty avoidance or reward seeking. In the same vein, selection of a low value arm may reflect information seeking, random exploration or a simple mistake. To overcome these issues, we propose a modification of the standard dynamic multi-armed bandit task. In our version, only two arms per trial provide information about the reward gained from choosing them and thereby reduce uncertainty. A switch to a non-informative, lower-than-maximum value option is clearly a mistake as the option is dominated in terms of reward and entirely uninformative and thereby can be viewed as decision noise, i.e., random behavior. Moreover, a switch to the bandit with highest informativeness unequivocally reflects uncertainty-driven exploration. Thus, our task allows identifying uncertainty-driven switches independent of models and distinguishing them from mistakes.

Selecting options that are more uncertain provides more information but incurs the risk of a small reward. Given the well documented human aversion to uncertain gains (Ellsberg 1961), one might expect people with stronger uncertainty aversion to be less likely to explore highly uncertain options (Payzan-LeNestour and Bossaerts 2011). Indeed, several studies found that some individuals actively avoid the options with highest uncertainty in the multi-armed bandit task (e.g., Daw et al. 2006; Chakroun et al. 2020). However, the choices of these individuals were not entirely exploitative either, suggesting that not all exploration is uncertainty-driven. One possibility is for agents to select an arm that is suboptimal in terms of estimated payoff but associated with less uncertainty than other bandits with more suboptimal values. This choice will allow the agent to find out if one *high*-value bandit has become the *highest*-value bandit without the risk of receiving a small reward. Thus, we aim to apply our modified multi-armed bandit task to categorize choices as value-driven explorative (i.e., selecting the more rewarding (but not the most rewarding overall) and less uncertain of the two informative arms) rather than uncertainty-driven explorative or random.

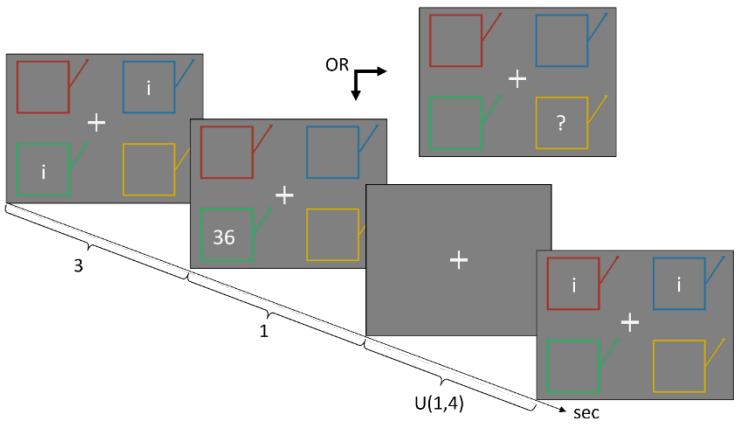
2.1 Methods: task structure

In our modified version of the classic multi-armed bandit task, participants selected one arm and received points in every trial and aimed to maximize their payoffs. The number of points paid by each arm gradually changed from trial to trial independently from other arms. The payoff μ for the arm a on each trial t varied between 1 and 100 points and was generated according to the decaying Gaussian random walk equation:

Equation 1. Decaying Gaussian random walk. The decay parameter λ and the decay center θ were set at 0.9836 and 50, respectively. The diffusion noise ν was drawn from a zero-centered Gaussian distribution with a standard deviation $\sigma_d = 2.8$.

$$\mu_{a,t+1} = \lambda * \mu_{a,t} + (1 - \lambda) * \theta + \nu$$

Figure 1. Example trial. At trial onset, the two informative arms were denoted with an "i". After they chose an arm, participants saw either the number of points won (informative arm chosen) or a question mark (non-informative arm chosen).



On each trial, two arms were defined as informative and two non-informative (Figure 1). The first informative arm was randomly determined from the two arms with currently the highest payoff. The second informative arm was set to be one of the two arms that have been seen (i.e., selected when it was informative) the least in previous trials. In case of ambiguity in arm categorization (e.g., on the first trial), the informative arms were determined randomly. Selection of informative arms was followed by feedback about the number of points received. Points won were not shown after selection of non-informative arms but contributed to the overall payout.

2.2 Methods: computational modeling

Whereas our task is intended to categorize choices as different forms of exploration, in a model-free way this separation relies on *objective* values (i.e., payoffs associated with each arm by design). To capture individual propensity to explore and exploit more precisely, one may want to compute *subjective* estimates about arm payoffs while accounting for various explorative strategies. To this end, we modeled participant's behavior with Bayesian Mean Tracker as value updating rule (Daw et al. 2006; Speekenbrink and Konstantinidis 2015). Action selection was modeled using the Softmax function and regulated by the inverse temperature parameter β, which captures the extent of choice stochasticity in agent's decisions (i.e., *random selection* with regard to their subjective value estimates). Then, several models were considered with different boni added to the predicted value in the Softmax (Equation 2). In total, four boni were added and model fit compared (Table 1). The first bonus favors selection of the most uncertain arm and captures individual propensity for *uncertainty-driven exploration* (Daw et al. 2006). The second bonus boosts selection of the informative arm with the highest value and lowest informativeness from the two informative arms. Introduction of this bonus allows us to test the hypothesis that some exploratory choices cannot be explained only by a desire to reduce uncertainty or by random choice but rather by willingness to gather information with minimal payoff loss, i.e., *value-driven* exploration. Third, we consider uncertainty-dependent random behavior by including a parameter that captures the propensity to behave randomly under increasing overall uncertainty (Gershman 2018; 2019). Finally, we include a perseveration bonus, which favors the arm selected on the previous trial and accounts for influence of past choices (Bornstein et al. 2017).

Equation 2. Softmax function for action selection in the simultaneous task. The Softmax function determines the probability P of selecting arm a on trial t. Q corresponds to the estimated value of the payoff and V to the estimated bonus associated with arm a on trial t. Description of other parameters can be found in **Table 1**.

$$P_{a,t} = \frac{\exp\left(\beta * (V_{a,t} + \varphi * V_{a,t}^{UncDirect} + \pi * V_{a,t}^{ValDirect} + \gamma * V_{a,t}^{UncRand}) + \omega * V_{a,t}^{Persev}\right)}{\sum_j \exp\left(\beta * (V_{a,t} + \varphi * V_{a,t}^{UncDirect} + \pi * V_{a,t}^{ValDirect} + \gamma * V_{a,t}^{UncRand}) + \omega * V_{a,t}^{Persev}\right)}$$

Table 1. Model parameters.

Parameter	Behavioral strategy	Scaling
β	Random selection	Not applied (NB: scaling by the expected value or uncertainty worsened model fits)
ϕ	Uncertainty-driven exploration	$V_{a,t}^{UncDirect}$ is calculated as σ, Kalman filter uncertainty associated with the arm
π	Value-driven exploration	$V_{a,t}^{ValDirect}$ is = 1 when added to the informative arm with the highest value among the two informative arms (but not the highest one overall) and scaled by its uncertainty.
γ	Uncertainty-dependent random exploration	$V_{a,t}^{UncRand} = Q_{a,t}/\sqrt{\sum\sigma_t^2}$ where $Q_{a,t}$ is the value of the arm a in trial t and $\sigma_{a,t}^2$ is the uncertainty associated with the arm
ω	Perseveration	$V_{a,t}^{Persev}$ = 1 if the arm chosen in the current trial was chosen in the previous trial as well and 0 otherwise

3 Results

Table 2. Expected log-predictive densities (ELPD), their differences and standard deviations with respect to the winning model. The winning model (in green) contains parameters accounting for perseveration as well as random selection, uncertainty-driven, and value-driven exploration but not uncertainty-dependent random exploration.

Model	Difference in ELPD	SD of ELPD
β	-5435.1	87.4
β + φ	-897.8	40.8
β + φ + ω	-54.9	8.4
β + φ + ω + π	0	0
β + φ + ω + π + γ	-35.4	2.5
β + ω + π	-3572.8	78
β + π	-4288.5	84.2

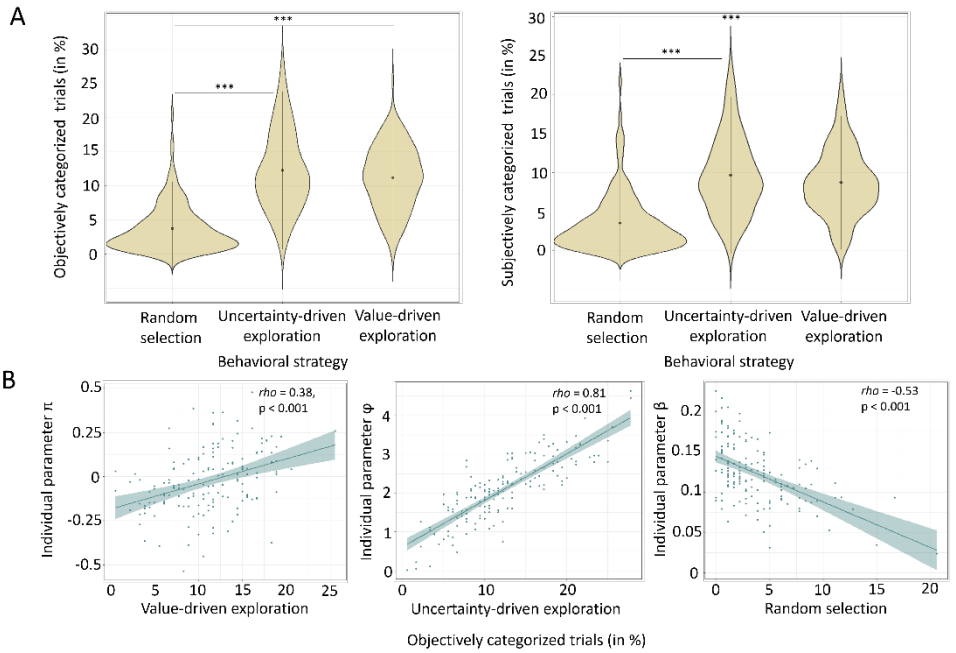


Figure 2. Participants employ multiple behavioral strategies. **A.** Categorization of trials based on the objective (i.e., model-free; left panel) and subjective (i.e., computed with a model; right panel) arm payoffs. Stars indicate a significant difference (two-sided t-test, $p < 0.001$, corrected for multiple comparisons with Bonferroni correction). **B.** Correlations between individual parameters from the winning model and model-free individual estimates of the frequency of use of corresponding explorative strategies.

To identify whether participants (N = 159, one participant excluded due to non-understanding of the task) used different forms of exploration, we fitted six models, each with a different set of parameters (Figure 2A). We computed 159 expected log-predictive densities leave-one-(choice)-out (ELPD-LOO) defined as a sum of pointwise predictive accuracies, where each data point corresponds to a given choice from an individual dataset. Then, we summed up the individual ELPD-LOO measures and defined the winning model as the one with the smallest sum. The winning model included the inverse temperature β, which was different from 0 and 1, suggesting that our participants exhibited some degree of random behavior without going into pure exploitation or purely random selection of arms. Participants' choices were best explained by the model with the three bonus-associated parameters ϕ, π, and ω, which indicates that our participants presented a perseveration bias and used both uncertainty-driven and value-driven forms of directed exploration. However, the winning model did not include uncertainty-dependent random exploration, which is consistent

with previous reports on dynamic multi-armed bandits (Chakroun et al. 2020).

We also categorized participant's choices on each trial as reflection of one of three behavioral strategies: uncertainty-driven exploration (selecting the arm with the highest uncertainty), value-driven exploration (selecting the arm with the highest payoff and smallest uncertainty among the two informative arms), and random behavior (selecting a non-informative arm with a non-best payoff). With objectively defined payoff (i.e., programmed arm payoffs), participants on average used all three strategies ($p < 0.001$, one-sample t-test against 0, Figure 2A, left panel). Moreover, participants used directed exploration more often than random selection. Categorization based on subjective payoffs confirmed these results (Figure 2A, right panel). Finally, we found that individual parameters capturing different types of behavioral strategies in the winning computational model were significantly correlated with corresponding estimates of those strategies derived based on objective values in a model-free way (Figure 2B).

4 Conclusions

Our study introduces a new version of the classic dynamic multi-armed bandit task, which separates reward and information provided to the agent on each trial to minimize the reward-information confound. We show that human participants use both directed exploration and random selection. Moreover, we provide evidence that human directed exploration may be guided not only by uncertainty but also by value. However, it should be noted that data collection for this project has been finished only recently and all results should be considered as preliminary.

5 References

Bornstein, Aaron M, Mel W Khaw, Daphna Shohamy, and Nathaniel D Daw. 2017. "Reminders of Past Choices Bias Decisions for Reward in Humans." *Nature Communications* 8 (1): 1–9.

Chakroun, Karima, David Mathar, Antonius Wiehler, Florian Ganzer, and Jan Peters. 2020. "Dopaminergic Modulation of the Exploration/Exploitation Trade-off in Human Decision-Making." *Elife* 9: e51260.

Daw, Nathaniel D, John P O'doherty, Peter Dayan, Ben Seymour, and Raymond J Dolan. 2006. "Cortical Substrates for Exploratory Decisions in Humans." *Nature* 441 (7095): 876–79.

Ellsberg, Daniel. 1961. "Risk, Ambiguity, and the Savage Axioms." *The Quarterly Journal of Economics*, 643–69.

Gershman, Samuel J. 2018. "Deconstructing the Human Algorithms for Exploration." *Cognition* 173: 34–42.

———. 2019. "Uncertainty and Exploration." *Decision* 6 (3): 277.

Payzan-LeNestour, Elise, and Peter Bossaerts. 2011. "Risk, Unexpected Uncertainty, and Estimation Uncertainty: Bayesian Learning in Unstable Settings." *PLoS Computational Biology* 7 (1): e1001048.

Speekenbrink, Maarten, and Emmanouil Konstantinidis. 2015. "Uncertainty and Exploration in a Restless Bandit Problem." *Topics in Cognitive Science* 7 (2): 351–67.

Wilson, Robert C, Elizabeth Bonawitz, Vincent D Costa, and R Becket Ebitz. 2021. "Balancing Exploration and Exploitation with Information and Randomization." *Current Opinion in Behavioral Sciences* 38: 49–56.

Wilson, Robert C, Andra Geana, John M White, Elliot A Ludvig, and Jonathan D Cohen. 2014. "Humans Use Directed and Random Exploration to Solve the Explore–Exploit Dilemma." *Journal of Experimental Psychology: General* 143 (6): 2074.

Inventing Relational State and Action Abstractions for Effective and Efficient Bilevel Planning

Tom Silver* MIT CSAIL tslvr@mit.edu

Rohan Chitnis* MIT CSAIL ronuchit@mit.edu

Nishanth Kumar MIT CSAIL njk@mit.edu

Willie McClinton MIT CSAIL wbm3@mit.edu

Tomás Lozano-Pérez MIT CSAIL tlp@mit.edu

Leslie Pack Kaelbling MIT CSAIL lpk@mit.edu

Joshua Tenenbaum MIT CSAIL jbt@mit.edu

Abstract

Effective and efficient planning in continuous state and action spaces is fundamentally hard, even when the transition model is deterministic and known. One way to alleviate this challenge is to perform bilevel planning with abstractions, where a high-level search for abstract plans is used to guide planning in the original transition space. In this paper, we develop a novel framework for learning state and action abstractions that are explicitly optimized for both effective (successful) and efficient (fast) bilevel planning. Given demonstrations of tasks in an environment, our data-efficient approach learns relational, neuro-symbolic abstractions that generalize over object identities and numbers. The symbolic components resemble the STRIPS operators found in AI planning, and the neural components refine the abstractions into actions that can be executed in the environment. Experimentally, we show across four robotic planning environments that our learned abstractions are able to quickly solve held-out tasks of longer horizons than were seen in the demonstrations, and outperform the efficiency of abstractions that we manually specified. We also find that as the planner configuration varies, the learned abstractions adapt accordingly, indicating that our abstraction learning method is both "task-aware" and "planner-aware."

Keywords: learning to plan, abstraction learning, relational learning, neuro-symbolic learning, hierarchical planning

Acknowledgements

We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; and from MIT-IBM Watson Lab. Tom and Nishanth are supported by NSF Graduate Research Fellowships. Willie is supported by an MIT Presidential Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors. We thank Michael Katz, Christian Muise, Aidan Curtis, Jiayuan Mao, Zhutian Yang, and Amber Li for helpful comments on an earlier draft.

*First two authors contributed equally.

1 Introduction

An autonomous agent should make *good* decisions *quickly*. These two considerations — effectiveness and efficiency — are especially important, and often competing, when an agent is *planning* in long-horizon, continuous-space tasks. *Abstractions* offer a mechanism to overcome this intractability [1, 2]. While state and action abstractions have a rich history in AI and robotics, a major limitation of early work is the downward refinability assumption [3]: that planning can be decomposed into first searching for an abstract plan, and then refining it into an actual plan. This is untenable in many applications, especially in robotics, where complex geometric constraints cannot be easily abstracted. To avoid this assumption, we consider *bilevel planning*, where reasoning in a high-level abstraction provides guidance for reasoning in a low-level task [4]. Another clear limitation of early work on abstractions is the reliance on manual specification, which requires understanding not only the environment, but also the interplay between the abstractions and the planner.

We identify three key desiderata of a system for planning with state and action abstractions: **(1)** These abstractions should be learned, not manually designed. **(2)** Planning with the learned abstractions should be tolerant to violations of the downward refinability assumption. **(3)** The abstractions should be trained to explicitly optimize both the effectiveness and the efficiency of planning. In this paper, we develop a framework for learning abstractions for planning that addresses all three desiderata. Specifically, *we learn state and action abstractions that are explicitly optimized for effective and efficient bilevel planning*. We consider learning from demonstrations in deterministic, fully observed, goal-based planning problems. The problems have object-centric continuous states and hybrid discrete-continuous actions, as are common in robotics [4]. The agent knows the transition model, but it is highly intractable to plan directly using this model, motivating the need to learn abstractions to guide planning. We learn *relational*, *neuro-symbolic* abstractions, where the symbolic components resemble STRIPS operators [5], and the neural components refine the abstractions into environment actions.

We conduct experiments in four robotic planning environments. We show that our framework is very data-efficient, and that the resulting relational abstractions allow for effective and efficient planning in held-out tasks involving different numbers of objects and longer horizons than were seen in the demonstrations. Interestingly, the learned abstractions sometimes outperform ones that we manually specified. We also find that as the planner configuration varies, the learned abstractions adapt accordingly, indicating that our learning method is both "task-aware" and "planner-aware."

2 Problem Setting

We consider learning from demonstrations in deterministic planning problems. These problems are goal-based and object-centric, with continuous states and hybrid discrete-continuous actions. An *environment* is defined by (1) a finite set of object *types*, each with an associated dimensionality for the feature vector of any object with that type; (2) a set of *controllers* with discrete typed parameters and a continuous real-valued vector of parameters; (3) a known, deterministic *transition model* f; and (4) a known set of *goal predicates* Ψ_G. An environment is associated with a distribution over *tasks*, where a task is defined by (1) a fixed set of typed *objects*; (2) an *initial state*; and (3) a *goal*, a set of ground atoms over Ψ_G and the objects. A *state* $x \in \mathcal{X}$ is a mapping from each object to a feature vector. An *action* $a \in \mathcal{A}$ is a controller with objects tied to the discrete parameters and a real-valued vector θ tied to the continuous parameters (e.g., `Pick(b1,`θ`)`).

A solution to a task is a *plan* π, a sequence of actions such that successive application of the transition model f starting from the initial state results in the goal holding. The agent is provided with a set of *training tasks* and *one demonstration per task*. We assume action costs are unitary and demonstrations are near-optimal. Each demonstration consists of a training task $\langle \mathcal{O}, x_0, g \rangle$ and a plan π^* that solves the task. The agent's objective is to learn to *efficiently* solve held-out tasks.

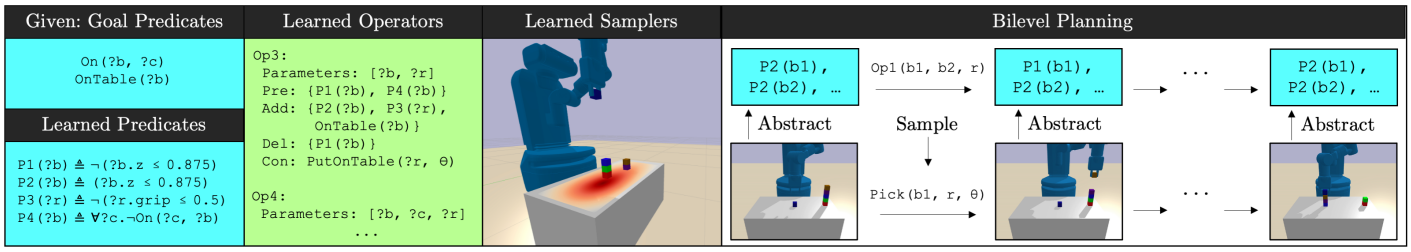
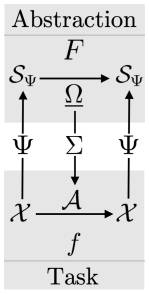


Figure 1: **Overview of our framework.** Given a small set of goal predicates (first panel, top), we use demonstration data to learn new predicates (first panel, bottom). In this Blocks example, the learned predicates `P1` – `P4` intuitively represent `Holding`, `NotHolding`, `HandEmpty`, and `NothingAbove` respectively. Collectively, the predicates define a state abstraction that maps continuous states x in the environment to abstract states s. After predicate invention, we learn abstractions of the continuous action space and transition model via planning operators (second panel). For each operator, we learn a sampler (third panel), a neural network that maps state features to action parameters. In this example, the sampler proposes different block placements. With these learned representations, we perform bilevel planning (fourth panel), with search in the abstract spaces guiding planning in the continuous spaces.

3 Learning Relational State and Action Abstractions for Bilevel Planning

Since the agent has access to the transition model f, one approach for optimizing the objective described in Section 2 is to forgo learning entirely, and plan directly in $\mathcal{X}$ and $\mathcal{A}$. However, searching for a solution directly in these large spaces is highly infeasible. Instead, we propose to *learn abstractions* using the provided demonstrations. We begin with an overview of the representations that we will learn from demonstrations and use to plan for held-out evaluation tasks.



Representations. Our representations are summarized in the figure on the right. A *predicate* represents a lifted relational binary state classifier (Figure 1, first panel). A predicate ground with objects induces a *ground atom*, which represents a binary state classifier. A set of predicates Ψ is used to transform a state $x \in \mathcal{X}$ into an *abstract state* $s \in S_\Psi$, which is the set of ground atoms that hold true in x. This predicate set includes the goal predicates $\Psi_g \subset \Psi$ and many other invented predicates (see "Learning" below). A set of *operators* Ω establishes the basis for *abstract actions*. These are STRIPS operators augmented with *abstract controllers*, where the discrete parameters of a controller are tied to the operator parameters (Figure 1, second panel). Given task objects, the set of *ground operators* $\underline{\Omega}$ defines the abstract action space, and induces an *abstract transition model* $F : \mathcal{S}_\Psi \times \underline{\Omega} \to \mathcal{S}_\Psi$, which is a partial function determined via add and delete effects, and defined when preconditions hold in the given state. Each operator is tied to a *sampler* $\sigma \in \Sigma$, which is lifted over the same parameters as the operator. When the operator is ground, the sampler σ is ground using the same objects, inducing a *ground sampler* $\underline{\sigma}$, which is a distribution over the continuous parameters of the operator's controller, conditioned on a state $x \in \mathcal{X}$. This ground sampler serves to stochastically *refine* the ground operator, suggesting $a \in \mathcal{A}$ that, when executed in x, should achieve the ground operator effects (Figure 1, third panel).

Planning. To use the components of an abstraction — predicates Ψ, operators Ω, and samplers Σ — for efficient planning, we build on *bilevel* planning techniques [4, 6]. We conduct an outer search over *abstract plans* using the predicates and operators, and an inner search over refinements of an abstract plan into a task solution π using the predicates and samplers. An *abstract plan* $\hat{\pi}$ for a task is a sequence of ground operators $\underline{\omega} \in \underline{\Omega}$ that achieves the task's goal under the abstract transition model F. Note that an abstract plan induces a sequence of abstract states, starting from the abstract initial state, which we call the *expected abstract state sequence*. Because downward refinability does not hold in our setting, an abstract plan is *not* guaranteed to be refinable into a solution π for the task, which necessitates bilevel planning.

For the outer search that finds abstract plans $\hat{\pi}$, we leverage the STRIPS-style operators and predicates [5] to automatically derive a domain-independent AI planning heuristic [7] such as hAdd or LMCut. We use this heuristic to run an A* search over the abstract state and action spaces. This A* search is used as a generator of abstract plans, outputting one at a time. Parameter n_{abstract} governs the maximum number of abstract plans that can be generated before the planner terminates with failure. For each abstract plan $\hat{\pi}$, we conduct an inner search that attempts to refine it into a solution π. While various implementations are possible [6, 8], we perform a backtracking search over the abstract actions. Recall that each $\underline{\omega}_i \in \underline{\Omega}$ includes a partially specified controller $C_i((o_1, \ldots, o_v)_i, \Theta_i)$ and has an associated ground sampler $\underline{\sigma}_i$. To begin the search, we initialize an indexing variable i to 1. On each step of search, we sample continuous parameters $\theta_i \sim \underline{\sigma}_i(x_{i-1})$, which fully specify an action $a_i = C_i((o_1, \ldots, o_v)_i, \theta_i)$. We then check whether $x_i = f(x_{i-1}, a_i)$ obeys the expected abstract state sequence. If so, we continue on to $i \leftarrow i + 1$. Otherwise, we repeat this step. Parameter n_{samples} governs the maximum number of times we invoke the sampler for a single value of i before backtracking to $i \leftarrow i - 1$. Refinement succeeds if the goal g holds when $i = |\hat{\pi}|$, and fails when i backtracks to 0. If refinement succeeds, the planner terminates with success and returns the plan $\pi = (a_1, \ldots, a_{|\hat{\pi}|})$. Crucially, if refinement fails, we continue the abstract search.

Learning. Our prior work [8, 9] has shown how to learn operators Ω and samplers Σ given a full set of predicates Ψ. That previous work learns STRIPS operators via inductive logic programming, and neural network samplers via supervised regression. Now, we focus on the problem of how to learn a good state abstraction Ψ from demonstrations $\mathcal{D}$.

Inspired by prior work [10, 11, 12], we approach the predicate invention problem from a program synthesis perspective. First, we define a compact representation of an infinite space of predicates in the form of a *grammar*. We then enumerate a large pool of *candidate predicates* from this grammar, with simpler candidates enumerated first. Next, we perform a *local search* over subsets of candidates, with the aim of identifying a good final subset to use as Ψ. The crucial question in this step is: what *objective function* $J(\Psi)$ should we use to guide the search over candidate predicate sets? Ultimately, we want to find a set of predicates that will lead to effective and efficient bilevel planning. However, planning is too expensive to be used in the objective function, which must be called many times during local search. There are two speed bottlenecks: neural sampler learning and the repeated attempts to refine abstract plans. Therefore, we propose to use a *proxy objective* that assumes an analytical model for refinement cost: $J_{\text{proxy}}(\Psi) \triangleq \sum_{(\mathcal{O}, x_0, g, \pi^*) \in \mathcal{D}} [\textsc{EstimatePlanTime}(x_0, g, \Psi, \Omega, \pi^*)]$.

To implement EstimatePlanTime, we perform the same abstract search as in planning, using the operators Ω learned from the candidate predicates Ψ. We keep track of the number of nodes created as a processor-independent proxy for the time spent on abstract search. For each abstract plan $\hat{\pi}$, instead of trying to refine it, we estimate how long refinement would take using two approximations: (1) that the probability of refining $\hat{\pi}$ is drawn from a geometric distribution with parameter equal to the absolute value difference between the lengths of $\hat{\pi}$ and the demonstration π^*, and (2) that the time spent trying to refine any candidate $\hat{\pi}$ is a constant, representing the time it takes to run exhaustive backtracking search.

	Ours		Manual		Down Eval		No Invent		Bisimulation		Branching		Boltzmann		GNN		Random	
Environment	Succ	Time	Succ	Time	Succ	Time	Succ	Time	Succ	Time	Succ	Time	Succ	Time	Succ	Time	Succ	Time
PickPlace1D	98.6	0.006	98.4	0.045	98.6	0.008	39.6	1.369	98.4	0.006	98.4	0.006	98.4	0.005	100.0	0.436	19.2	0.004
Blocks	98.4	0.296	98.6	0.251	98.2	0.318	3.2	1.235	19.0	0.158	98.4	0.284	64.8	0.954	27.8	0.138	0.6	0.006
Painting	100.0	0.470	99.6	0.464	98.8	0.208	0.0	–	0.0	–	20.2	4.186	88.6	0.600	59.2	2.077	0.0	–
Tools	96.8	0.457	100.0	0.491	42.8	0.060	0.0	–	26.2	0.699	75.8	0.109	64.2	0.247	25.6	0.311	0.0	–

Table 1: **Main results.** Percentage of 50 evaluation tasks solved under a 10-second timeout (Succ), and average planning time in seconds over successful tasks (Time). All numbers are averages over 10 seeds, which vary the training and evaluation tasks, random initializations during learning, and tiebreaking during planning. Our learned abstractions (Ours) perform extremely well compared to all baselines (rightmost four columns), and can even lead to more efficient planning than with the hand-designed abstraction (Manual).

We use J_{proxy} to run a local hill climbing search over $n_{\text{grammar}} = 200$ candidate predicates drawn from a grammar. For this grammar, we consider both the given goal predicates Ψ_G and *single-feature inequality classifiers*, which are less-than-or-equal-to expressions that compare a constant with an individual feature dimension, for some object type. We also allow negations and universal quantifications of predicates. See Figure 1 (first panel) for examples. Following prior work [12], we prune out candidate predicates if they are equivalent to any previously enumerated one, in terms of all groundings that hold in states in $\mathcal{D}$. This grammar associates each generated predicate with a cost, which we use to regularize J_{proxy}.

4 Experiments and Discussion

We evaluate nine methods across four robotic planning environments. All experiments are averaged over 10 random seeds. For each seed, in all four environments, we sample a set of 50 *evaluation tasks* from the task distribution $\mathcal{T}$, with hyperparameters chosen to involve more objects and harder goals than were seen at training. Our key measures of *effective and efficient planning* are (1) success rate and (2) wall-clock time. The timeout for planning is always 10 seconds.

Environments. We now briefly describe the environments. The first three environments were established in our prior work [9], but with manual state abstractions (we use the same state abstractions for our Manual baseline below).

- **PickPlace1D.** In this toy environment, a robot must pick blocks and place them onto target regions along a table surface. All pick and place poses are in a 1D line. Evaluation tasks require 1-4 actions to solve.
- **Blocks.** In this environment, a robot in 3D must interact with blocks on a table to assemble them into towers. This is a robotics adaptation of the blocks world domain in AI planning. Evaluation tasks require 2-20 actions to solve.
- **Painting.** In this challenging environment, a robot in 3D must pick, wash, dry, paint, and place widgets into either a box or a shelf, as specified by the goal. Evaluation tasks require 11-25 actions to solve.
- **Tools.** In this challenging environment, a robot operating on a 2D table surface must assemble contraptions by fastening screws, nails, and bolts, using a provided set of screwdrivers, hammers, and wrenches respectively. This environment has physical constraints outside the scope of our predicate grammar, and therefore tests the learner's ability to be robust to an insurmountable lack of downward refinability. Evaluation tasks require 7-20 actions to solve.

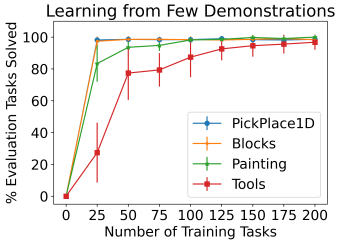
Methods. The methods we evaluate are: ours, a manually designed state abstraction, 2 ablations, and 5 baselines.

- **Ours.** Our main approach, which learns abstractions and uses them to guide planning on the evaluation tasks.
- **Manual.** An oracle approach that plans with manually designed state abstractions (predicates) for each environment.
- **Down Eval.** An ablation of Ours that uses $n_{\text{abstract}} = 1$ (Section 3, "Planning") during evaluation only.
- **No Invent.** An ablation of Ours that uses $\Psi = \Psi_G$, i.e., only goal predicates are used for the state abstraction.
- **Bisimulation.** A baseline that learns abstractions by approximately optimizing the *bisimulation criteria*, as in prior work [12]. Specifically, this baseline learns abstractions that minimize the number of transitions in the demonstrations where the abstract transition model F is applicable but makes a misprediction about the next abstract state.
- **Branching.** A baseline that learns abstractions by optimizing the *branching factor* of planning. Specifically, this baseline learns abstractions that aim to minimize the total number of applicable operators over states in the demonstrations.
- **Boltzmann.** A baseline that assumes the demonstrator is acting *noisily rationally*. Specifically, for any candidate abstraction in our search, we compute the probability of the demonstration under a Boltzmann policy, with the AI planning heuristic used as a proxy for the true cost-to-go; we seek to maximize this probability.
- **GNN.** A baseline that trains a graph neural network policy via behavior cloning. This GNN takes in the current state x, abstract state s, and goal g. It outputs an action a, via a one-hot vector encoding which controller to execute, one-hot vectors over all objects at each discrete argument position, and a vector of continuous arguments. At evaluation time, we sample trajectories by treating the outputted continuous arguments as the mean of a Gaussian with fixed variance. We use the transition model f to check if the goal is achieved, and repeat until the planning timeout.
- **Random.** A baseline that simply executes a random controller with random arguments on each step. No learning.

Additional details. All sampler neural networks are fully connected, with two hidden layers of size 32 each, and trained with the Adam optimizer for 10K epochs using learning rate 1e-3. Unless otherwise specified, the AI planning heuristic is LMCut. We use $n_{\text{abstract}} = 1000$ for Tools and 8 otherwise, and $n_{\text{samples}} = 1$ for Tools and 10 otherwise.

Results. Main results are in Table 1. Comparing Ours to the right five columns (baselines), we see that the proxy objective we use in our abstraction learning performs very well compared to more typical objectives such as bisimulation or optimizing rationality. Furthermore, compared to Manual, the learned abstractions are often *more efficient* from a planning time perspective, with very little reduction in success rate. Next, we consider ablations of our main approach (Table 1, middle two columns). Results for No Invent show that, as expected, using the goal predicates as a standalone state abstraction is completely insufficient for most tasks. Comparing Ours to Down Eval shows that assuming downward refinability at evaluation time works for PickPlace1D, Blocks, and Painting, but not for Tools. We were surprised by this result because the manually designed abstractions for PickPlace1D and Painting are *not* downward refinable [9].

We also assess the data efficiency of our system. In the figure on the right, each point shows a mean over 10 seeds, with standard deviations shown as vertical bars. Recall that there is one demonstration per training task. In most environments, the figure shows very good evaluation performance within just 50 demonstrations, and this performance typically improves smoothly as the number of demonstrations increases. Generally, providing more demonstrations helps the following aspects of our system: (1) inventing fewer extraneous, overfitted predicates; (2) learning more accurate operator preconditions; (3) allowing the sampler neural networks to be trained on a larger amount of data.



The table on the right shows an additional experiment we conducted in the Blocks environment, where we varied the AI planning heuristic used in predicate invention and evaluation. The Node column shows the number of nodes created during abstract search. While the gap in performance is limited with LMCut, our system shows a substantial improvement with hAdd. In the latter case, our approach invents four unary predicates with the intuitive meanings `Holding`, `NothingAbove`, `HandEmpty`, and `NotOnAnyBlock`, to supplement the given goal predicates `On` and `OnTable`. Comparing these to Manual, which has the same predicates and operators as those in the International Planning Competition (IPC), we see the following differences: `Clear` is omitted, and `NothingAbove` and `NotOnAnyBlock` are added. We observed that the latter are logical transformations of predicates used in the standard IPC blocks world representation, which motivated us to run a separate, symbolic-only experiment, where we collected IPC blocks world problems and transformed them to use these predicates and associated learned operators. We found that using A* and hAdd, planning with our learned representations is much faster than planning with the IPC representations. For example, using Fast Downward [7] on a problem from IPC 2000 with 36 blocks, planning succeeds with our representations in 12.5 seconds after approximately 7,000 expansions, whereas it fails within a 2 *hour* timeout with the standard encoding.

	Ours			Manual		
Heuristic	Succ	Time	Node	Succ	Time	Node
LMCut	98.4	0.296	2949	98.6	0.251	2941
hAdd	98.6	0.115	121.6	97.8	0.235	3883

Altogether, our experimental results affirm the promise of our approach for learning abstractions that are both "task-aware" and "planner-aware," leading to effective and efficient bilevel planning.

References

[1] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. *ISAIM*, 4(5):9, 2006.

[2] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[3] Bhaskara Marthi, Stuart J Russell, and Jason Andrew Wolfe. Angelic semantics for high-level actions. In *ICAPS*, 2007.

[4] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[5] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[6] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646. IEEE, 2014.

[7] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[8] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. *arXiv preprint arXiv:2105.14074*, 2021.

[9] Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *IROS*, 2021.

[10] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[11] João Loula, Tom Silver, Kelsey R Allen, and Josh Tenenbaum. Discovering a symbolic planning language from continuous experience. In *Annual Meeting of the Cognitive Science Society (CogSci)*, page 2193, 2019.

[12] Aidan Curtis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Discovering state and action abstractions for generalized task and motion planning. *AAAI*, 2022.

588

Emergent behavior and neural dynamics in artificial agents tracking turbulent plumes

Satpreet H Singh*
University of Washington
Seattle, WA, 98195
satsingh@uw.edu

Floris van Breugel
University of Nevada
Reno, NV 89557
fvanbreugel@unr.edu

Rajesh P. N. Rao
University of Washington
Seattle, WA 98195
rao@cs.washington.edu

Bingni W. Brunton
University of Washington
Seattle, WA 98195
bbrunton@uw.edu

Abstract

Tracking a turbulent plume to locate its source is a complex control problem requiring robust multi-sensory integration in the face of intermittent odors, changing wind direction, and variable plume shape. This task is routinely performed by flying insects, often over long distances, in pursuit of food or mates. Several aspects of this remarkable behavior have been studied in detail in many experimental studies. Here, we take a complementary *in silico* approach, using artificial agents trained with reinforcement learning to develop an integrated understanding of the behaviors and neural computations that support plume tracking. Specifically, we use Deep Reinforcement Learning (DRL) to train Recurrent Neural Network (RNN) based agents to locate the source of simulated turbulent plumes. Interestingly, the agents' emergent behaviors resemble those of flying insects, and the RNNs learn to represent task-relevant variables, such as head direction and time since last odor encounter. Our analyses suggest an intriguing experimentally testable hypothesis for tracking plumes in changing wind direction—that agents follow local plume shape rather than the current wind direction. While reflexive short-memory behaviors are sufficient for tracking plumes in constant wind, longer timescales of memory are essential for tracking plumes that switch direction. At the level of neural dynamics, the RNNs' population activity is low-dimensional and organized into distinct dynamical structures, with some correspondence to the uncovered behavioral modules. Our *in silico* approach provides key intuitions for turbulent plume tracking strategies and motivates future targeted experimental and theoretical developments.

Keywords: deep reinforcement learning, recurrent neural networks, neural dynamics, computational neuroscience, olfactory search, computational ethology, infotaxis

Acknowledgements

Funded by the AFRL award FA8651-20-1-0002 and FA9550-21-0122 (FvB); the NIH award P20GM103650 (FvB); the DARPA award HR001120C0021 (RPNR); the Templeton World Charity Foundation (RPNR); the NSF award EEC-1028725 (RPNR); the AFOSR awards FA9550-19-1-0386 and FA9550-18-1-0114 (BWB); and the Washington Research Foundation (BWB);

*Corresponding author; Manuscript, code, data, & animations: https://github.com/BruntonUWBio/plumetracknets

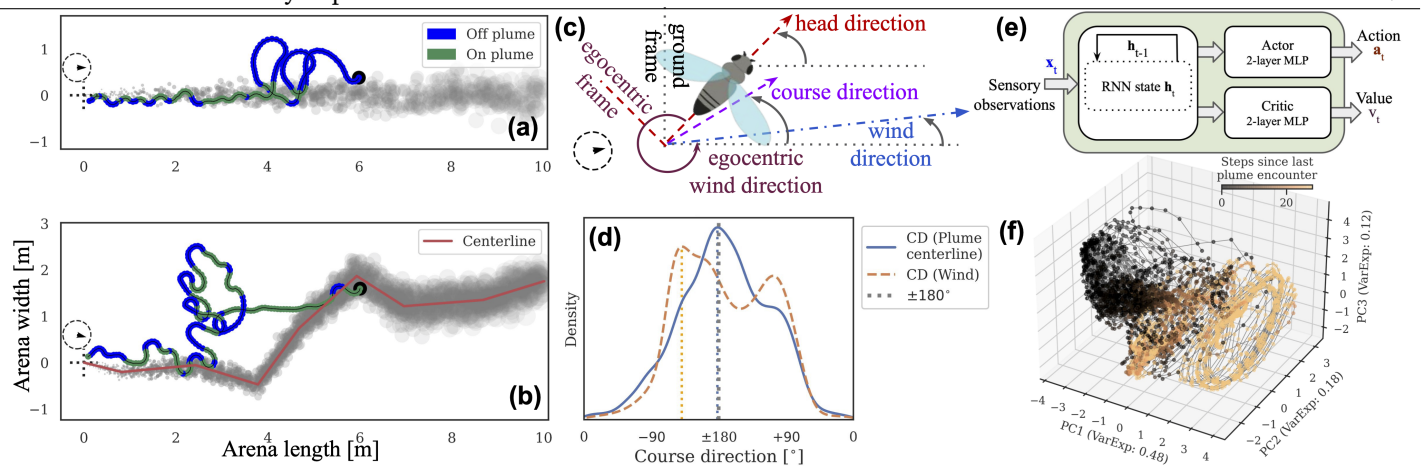


Figure 1: **(a-b)** An agent's plume tracking trajectories, each starting at a filled black circle and ending at dotted crosshairs indicating the plume source. Plume in dull grey color. Wind-direction shown by arrow in dotted circle. Successful tracking episodes in **(a)** a constant wind-direction plume, and **(b)** a plume that switches directions multiple times; **(c)** Agent course-direction (CD), measured counter-clockwise from ground x-axis, is the direction the agent actually moves taking into account *slip* due to wind-advection; **(d)** Empirical distribution of CD across multiple successful trajectories in plumes with switching wind-direction suggests that agents track with respect to local plume centerline rather than with respect to the wind; **(e)** An agent is a Recurrent Actor-Critic Network [1] trained using Deep Reinforcement Learning; **(f)** Neural activity across multiple episodes colored by time elapsed since agent last encountered plume (25 steps = 1.0s), shows how this task-relevant variable which has been previously reported in the Entomological literature [2], is clearly represented in the agent's neural data.

Detail: We implement a particle-based 2D plume simulation model that replicates the statistics of real-world turbulent plumes [3]. Our agents are actor-critic networks [4], each comprising a *vanilla* Recurrent Neural Network (RNN) followed by a pair of 2 consecutive feedforward layers corresponding to the Actor and Critic heads of the network (Fig. 1e). All layers are 64 units wide with $tanh$ nonlinearities. At each time step, agents receive real-valued 3-tuple inputs comprising egocentric wind velocities (x, y) and odor concentration at their current location, and output continuous valued turn and forward-movement actions matched to the capabilities of real flying insects [5, 6]. We train 14 agents with independently randomly initialized networks using policy gradient (PPO) [4] on a 2-stage *curriculum* of progressively complex plumes. We describe 1 of the top-5 best performing agents (measured by fraction of successful trajectories) here.

Results: Trained agents can track plumes across a variety of plume simulator configurations (Fig. 1a-b) using a set of emergent 3 behavior modules (Fig. 2a-b). These emergent behaviors are reminiscent of *upwind surging*, *zig-zagging*, *crosswind casting* and *U-turn* behaviors previously reported in the literature for flying insects [5, 7, 8]. Our analyses suggest an intriguing experimentally testable hypothesis for tracking plumes in changing wind direction—that agents follow local plume shape rather than the current wind direction. (Fig. 1c). Simultaneously, we find that the RNN learns to represent task-relevant variables such as the time elapsed since the last odor encounter (Fig. 1f). Neural activity is low-dimensional (Fig. 2f), and exhibits a signature structure for 2 of the 3 behavior modules (Fig. 2c-d), organized in overlapping but distinct regimes (Fig. 2e) at a macroscopic level. Using standard notation [9], we linearize the RNN update rule, $\mathbf{h}_t = F(\mathbf{h}_{t-1}, \mathbf{x}_t) = tanh(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + b)$, around an arbitrary expansion point $(\mathbf{h}^e\mathbf{x}^e)$ on the agent's trajectory, to get the approximate linear system [9, 10], $\mathbf{h}_t \approx F(\mathbf{h}^e, \mathbf{x}^e) + \mathbf{J}^{\text{rec}}\big|_{(\mathbf{h}^e,\mathbf{x}^e)} \Delta\mathbf{h}_{t-1} + \mathbf{J}^{\text{inp}}\big|_{(\mathbf{h}^e,\mathbf{x}^e)} \Delta\mathbf{x}_t$. Like in [10], we then use $\tau_i = |(1/\ln|\lambda_i|)|$ to obtain the stimulus integration timescales τ_i associated with stable eigenvalues λ_i of the RNN recurrence Jacobian $\mathbf{J}^{\text{rec}}\big|_{(\mathbf{h}^e,\mathbf{x}^e)}$ over multiple tracking trajectories. The bulk of these timescales are within ≈ 0.5s, suggesting that the plume tracking task predominantly needs short-timescale memories (Fig. 2g). Finally, DRL training appears to produce unstable eigenmodes in the RNN connectivity matrix $\mathbf{W}_h$ (Fig. 2h).

Discussion: Our normative RNN-based agent model enables an abstract yet interpretable understanding of the behaviors and neural computations that support turbulent plume tracking. Compared to the *Infotaxis* algorithm [11, 12], our learned policies produce trajectories with a stronger semblance to biology, react to changing wind conditions, and use a neural implementation that does not make any (potentially biologically implausible) assumptions about which variables are implemented and how inference is performed. Our findings motivate future neuroethological experiments and theoretical investigation of such DRL-trained RNNs.

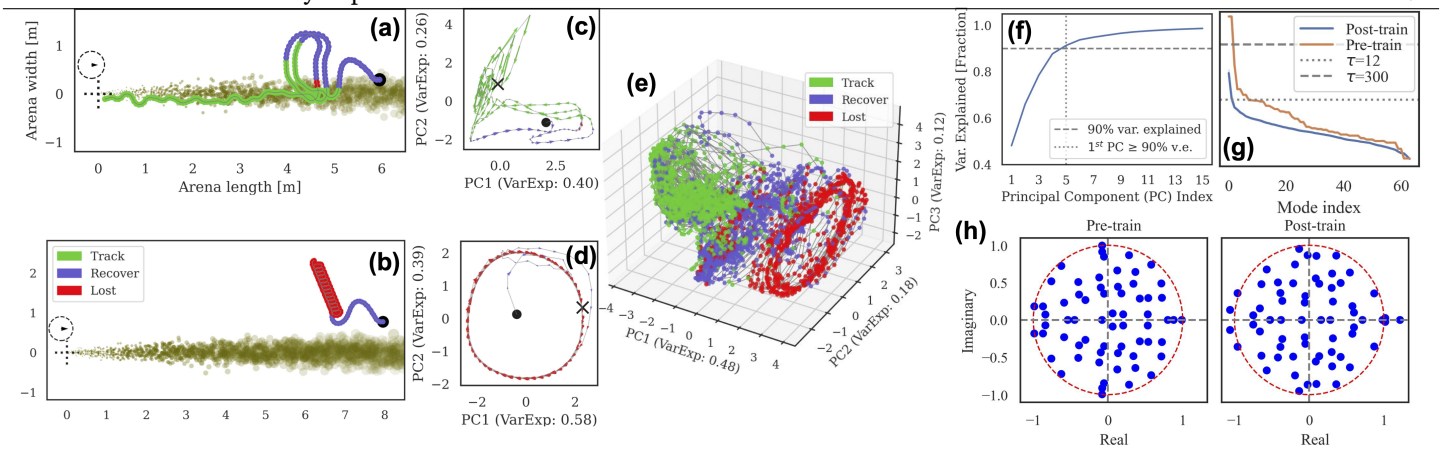


Figure 2: **(a-b)** Agent's plume tracking trajectories, colored by emergent behavior modules, namely: *track*, where agent stays inside, skims or zig-zags close to the boundary of the plume; *lost* where, after an extended duration of plume loss, it spirals in place (with slow drift); and *recover*, where, after brief periods of plume loss, it makes large (usually) cross-wind movements to find the plume. Trajectories start at filled circle (black) and plumes (dull grey) originate at dotted crosshairs. Wind-direction shown by arrow in dotted circle. **(c-d)** Neural activity corresponding to each row's trajectory, projected onto their respective top-2 Principal Components. Quivers indicate neural activity gradient. **(c)** A 'funnel' like structure (in green) associated with the *rack*, and a limit-cycle (in red) associated with the periodic *lost* behavior modules; **(e)** Neural dynamics from multiple trajectories shows distinct structures associated with *track* and *lost*, but an amorphous intermediate region for the *recover* behavior; **(f)** Like biological recordings, artificial neural activity is low dimensional, with 90% variance explained by top-5 PCs; **(g)** Time-averaged stimulus integration timescales τ associated with stable eigenmodes of recurrence Jacobian $\mathbf{J}^{\text{rec}}$ show a bulk of relatively short (within 12 timesteps, 25 steps = 1.0s) timescales. (Top 5 τs are 56.5, 13.0, 7.7, 6.8 & 5.8 timesteps). Before training, τs associated with $\mathbf{W_h}$'s eigenmodes can be large, even exceeding the length of the training/evaluation episodes (300 steps); **(h)** Eigenvalue spectra of the RNN recurrence matrix $\mathbf{W_h}$ before and after training show how training generates unstable eigenmodes.

References

[1] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, pp. 1008–1014, 2000.

[2] J. Kennedy, "Zigzagging and casting as a programmed response to wind-borne odour: a review," *Physiological Entomology*, vol. 8, no. 2, pp. 109–120, 1983.

[3] J. A. Farrell, J. Murlis, X. Long, W. Li, and R. T. Cardé, "Filament-based atmospheric dispersion model to achieve short time-scale structure of odor plumes," *Environmental Fluid Mechanics*, vol. 2, no. 1-2, pp. 143–169, 2002.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] F. van Breugel and M. H. Dickinson, "Plume-tracking behavior of flying drosophila emerges from a set of distinct sensory-motor reflexes," *Current Biology*, vol. 24, no. 3, pp. 274–286, 2014.

[6] F. van Breugel, W. Regan, and H. Lipson, "From insects to machines," *IEEE Robotics & Automation Magazine*, vol. 15, no. 4, pp. 68–74, 2008.

[7] K. L. Baker, M. Dickinson, T. M. Findley, D. H. Gire, M. Louis, M. P. Suver, J. V. Verhagen, K. I. Nagel, and M. C. Smear, "Algorithms for olfactory search across species," *Journal of Neuroscience*, vol. 38, no. 44, pp. 9383–9389, 2018.

[8] R. T. Cardé and M. A. Willis, "Navigational strategies used by insects to find distant, wind-borne sources of odor," *Journal of Chemical Ecology*, vol. 34, no. 7, pp. 854–866, 2008.

[9] D. Sussillo and O. Barak, "Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks," *Neural Computation*, vol. 25, no. 3, pp. 626–649, 2013.

[10] N. Maheswaranathan, A. Williams, M. Golub, S. Ganguli, and D. Sussillo, "Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics," in *Advances in Neural Information Processing Systems*, pp. 15696–15705, 2019.

[11] M. Vergassola, E. Villermaux, and B. I. Shraiman, "'infotaxis' as a strategy for searching without gradients," *Nature*, vol. 445, no. 7126, pp. 406–409, 2007.

[12] G. Reddy, V. N. Murthy, and M. Vergassola, "Olfactory sensing and navigation in turbulent environments," *Annual Review of Condensed Matter Physics*, vol. 13, no. 1, 2022591

Learning to be Process-Fair: Equitable Decision-Making using Contextual Multi-Armed Bandits

Arpita Singhal
Department of Computer Science
Stanford University
Stanford, CA 94305
arpitas@cs.stanford.edu

Henry Zhu
Stanford University
Stanford, CA 94305
hyzhu@cs.stanford.edu

Emma Brunskill
Stanford University
Stanford, CA 94305
ebrun@cs.stanford.edu

Abstract

Machine learning algorithms are increasingly being used to make decisions in critical situations, such as in banking and healthcare. To better understand these algorithms, there has been increased interest in the fairness, transparency and accountability of these algorithms. While there are a few online learning contextual bandit algorithms that address these concerns, most are computationally inefficient and do not learn policies that are fair during the course of learning. We devise a process-fair algorithm that aims to find an equitable policy at all time steps, and we validate our algorithm through a set of experiments, performed using a healthcare transport dataset.

Keywords: contextual multi-armed bandits, reinforcement learning, fairness

1 Introduction

Recently, a surge of automated statistical and machine learning techniques have been developed and used to inform decisions in high-stakes situations, such as in banking for loan decisions [3], hiring [5], and even criminal sentencing [2]. With the increased use of machine learning algorithms, comes an increased interest in fairness, transparency and accountability of these algorithms. Specifically, there has been an increased interest in understanding how these algorithms work and how to make them less biased.

However, most of the fairness work has been limited to supervised machine learning, focused on the predictions learned by the algorithm rather than how the algorithm is learning. The limited work that we have seen in the sequential decision-making space incorporate fairness but have a few limitations. We see work that deals with offline policy learning with fairness and safety constraints; however, they do not incorporate budget constraints and do not handle the exploration-exploitation tradeoff necessary for online learning [6]. We see a few online learning contextual bandit algorithms that have budget constraints. However, they are either computationally inefficient [1] or do not have a multi-objective optimization [7] which is important for equitable decision making. Also, we see the multi-objective optimization problem in Learning to be Fair which primarily focuses on trying to quickly learn a fair policy. [4].

In this work, we consider the extent to which fairness is compatible with learning in a multi-arm bandit setting, when we consider fairness constraints at every time step while the agent is learning. In this setting, the agent is a sequential decision-maker that must choose which decision to make from a finite set of k arms at each time step t. Based on the arm chosen, the agent observes a stochastic reward. Over time, the agent is optimizing to find the maximum total reward by learning the relationships between arms and rewards. Moreover, we are using contextual multi-arm bandits, in which the agent observes a context for a given individual, and the expected reward is some unknown function of the context.

Throughout this paper, we will be using the motivating application from the healthcare transport dataset, described in section 4.1.

2 Preliminaries/ Background

2.1 Contextual Bandits

We study the contextual multi-armed bandit setting, defined by a finite state space of domain $\mathcal{X}$ and a set of arms $[k] := 1, ...k$. An algorithm then chooses an arm, or action a_k, based on the input context and observes stochastic reward for the arm it chooses, $r^t_{a_k}$. Each action is then associated with a potential outcome, $Y_i(a_k)$, where taking action $\pi(X_i)$ results in the outcome $Y_i(\pi(X_i))$.

For example, in our motivating application $Y_i(a_2)$ may indicate whether the i^{th} individual would attend their appointment if offered a free ride, $Y_i(a_1)$ may indicate the outcome if offered a transit voucher, and $Y_i(a_0)$ may indicate the outcome if assistance were not provided.

2.2 States and Action Space

Each arm represents a different action that the agent can take. In our motivating application, we have three possible actions: a free ride, a free transit voucher, or no transportation assistance.

At each time step, the agent observes a vector of covariates X_i drawn from a distribution D_X, supported on $\mathcal{X}$. In our motivating application, X_i might encode an individual's demographics, medical status, transportation access, and history of appearance. For considerations of algorithmic fairness, we pay attention to groups defined by legally-protected characteristics, such as race and gender. We allow individuals to be associated with a set of identities $s(X_i) \subseteq G$, where for example, $s(X_i)$ can specify combinations of the legally-protected characteristics.

2.3 Learning to Be Fair [4]

In our paper, we compare process fairness with the formulation in [4]. In [4], they work on a multi-objective equitable decision-making task, where they balance between optimizing the reward and ensuring that the spending disparity between k groups is minimized, using a fairness term F, as described in Section 3.3. They formulate their policy, as follows:

$$\begin{aligned} \pi^*_{ltbf} = \arg\max_{\pi} \quad & \sum_x p_x \sum_a \pi_{xa} r_{xa} - \lambda F(\pi, r, c) & \\ \text{s.t.} \quad & 0 \le \pi_{xa} \le 1 & \forall x, a \\ & \sum_x p_x \sum_k \pi_{xa} c_{xa} \le b_k & \forall k \end{aligned} \tag{1}$$

3 Process Fairness

3.1 Optimization Problem

Building on the formulation from section 2.3, we define a policy that is process-fair, as follows:

$$\begin{aligned} \pi^*_{pf} = \arg\max_{\pi} \quad & \sum_x p_x \sum_a \pi_{xa} r_{xa} - \lambda F(\pi, r, c) \\ \text{s.t.} \quad & 0 \leq \pi_{xa} \leq 1 && \forall x, a \\ & \sum_x p_x \sum_a \pi_{xa} c_{xa} \leq b_k && \forall k \\ & F(\pi, r, c) \leq \gamma \end{aligned} \tag{2}$$

where π is the policy, r_{xa} is the reward for a particular context x and action a, c_{xa} is the associated cost, and budget parameter b_k is a parameter bounding the cost for each arm k, and parameter γ is a parameter bounding the disparity in fairness between groups. The auxiliary form of fairness $F(\pi, r, c)$ can take on different forms of fairness, such as allocation, budget, or outcome fairness. We define allocation fairness to be the splitting of material resources (e.g. transit vouchers) equally amongst groups. Budget fairness refers to the splitting of the budget (e.g. money) amongst groups. Outcome fairness refers to the final result, e.g. $Y_i(a_k)$ being equally observed between groups. In addition, the policy optimization can account for different forms of fairness by adding or modifying the constraints. We use both a regularization term and constraint on F to more easily compare with the formulation from [4].

3.2 Algorithm

We present our algorithm to learn a process-fair policy, . Note, the algorithm is very similar to the one presented in [4]. The differences are in the optimization problem, specifically in step 14 of Algorithm 1.

Algorithm 1 Process Fair Bandit Learning Algorithm

1: **Input:** Actions a_k, budget b, parity preferences λ, reward function r, fairness bound γ, covariate distribution $\mathbb{P}(X = x)$, group membership function s, bandit algorithm, ℓ
2: **Initialize:** Randomly first treat ℓ people.
3: **for** $t = 1$ **to** T **do**
4: Set $D_i := (X_j, A_j, Y_j)_{j=1}^{t-1}$ where X, A, and Y denote the covariates, actions, and outcomes of individuals seen at previous time steps.
5: Estimate $f(x, a)$ with a parametric family of functions $g(x, a; \theta)$ fit on D_i
6: **if** bandit algorithm == ϵ-greedy **then**
7: $\hat{f}(x, a) := g(x, a; \hat{\theta}_i)$ where $\hat{\theta}_i$ is the MLE
8: **else if** bandit algorithm == Thompson Sampling **then**
9: $\hat{f}(x, a) := g(x, a; \hat{\theta}^*_i)$ where $\hat{\theta}^*_i$ is drawn from the posterior of $\hat{\theta}_i$
10: **else if** bandit algorithm == UCB **then**
11: $\hat{f}(x, a) :=$ the α-percentile of the posterior of $g(x, a; \hat{\theta}_i)$
12: **end if**
13: Adjust $b^*_{k,i}$ using equation 3.
14: Find solution π^*_i of the LP in Equation 2 with input values $\hat{f}(x, a)$, s, λ, $b^*_{k,i}$, γ, and $\mathbb{P}(X = x)$.
15: **if** ϵ-greedy **and** Bernoulli(ϵ) $== 1$ **then**
16: Take random action A_i where $\mathbb{P}(A_i = a_k) \propto b^*_{k,i}$
17: **else**
18: Take action $A_i \sim \pi^*_i(X_i)$
19: **end if**
20: Observe outcome Y_i
21: **end for**

Similar to [4], we perform budget adjustment at each time step because the policy π^*_i evolves over time.

$$b^*_{k,i} = b_k \frac{\sum_{j=1}^{i-1} b^*_{k,j}}{\sum_{j=1}^{i-1} \mathbb{I}(A_j = a_k)} \tag{3}$$

where A_j is the action taken by the j^{th} individual and b_k is the target budget for action a_k.

3.3 Different Forms of Process Fairness

3.3.1 Treatment Allocation Fairness

Treatment allocation fairness corresponds to treatment parity across the population. For example, it may be advantageous to prefer policies in which a similar proportion of individuals receive the same types of treatments across neighborhoods. We would encode neighborhood membership into groups. More formally, we define allocation fairness as follows:

$$\begin{aligned} F(\pi, r, c) = F^{TA}(\pi) &= \sum_g ||\mathcal{D}(\pi(X)) - \mathcal{D}(\pi(X)|g \in s(X))||_1 \\ &= \sum_g \sum_a |Pr(\pi(X) = a) - Pr(\pi(X) = a|g \in s(X))| \\ &= \sum_g \sum_a |\sum_x p_x \pi_{xa} - \sum_{x|s(x)=g} \frac{p_x}{\sum_{x|s(x)=g} p_x} \pi_{xa}|. \end{aligned} \tag{4}$$

3.3.2 Budget Allocation Fairness

Budget allocation fairness corresponds to budget parity across the population. For example, given a budget for the entire population, we would want to ensure that the budget is equally split between groups, proportionally based on the number of individuals in the group. We define budget allocation fairness, as follows:

$$\begin{aligned} F(\pi, r, c) = F^{BA}(\pi, c) &= \sum_g \sum_k |\mathbb{E}_\pi[c(X, A)] - \mathbb{E}_\pi[c(x, A)|G = g]| \\ &= \sum_g \sum_a |\sum_x p_x \pi_{xa} c_{xa} - \sum_{x|s(x)=g} \frac{p_x}{\sum_{x|s(x)=g} p_x} \pi_{xa} c_{xa}|. \end{aligned} \tag{5}$$

3.4 Theorem

Theorem 3.4.1. *In order to validate whether process fairness will learn the optimal policy, the oracle $F(\pi, r, c)$ value $= \gamma$ given a specific λ value in Equation 1 is computed. Using γ as input into Equation 2, process fairness will learn the same optimal policy as learning to be fair. If $\gamma = F(\pi, r, c)^*$ then $U(\pi^*_{pf}) = U(\pi^*_{ltbf})$, where U is the utility function.*

Proof. Since the optimal policy still lies If $\pi^* = \arg\max(U(\pi)|\pi \in S)$ and $\pi^* \in A$ then $\arg\max(U(\pi)|\pi \in A) = \pi^*$ where $A \subseteq S$. S represents the set of policies which respect the budget constraint from Equation 1 and A is the set of policies which respect both the budget and fairness constraints from Equation 2. □

4 Experiments and Simulations

4.1 Healthcare Transport Dataset

To evaluate our learning approach, we conducted a simulation study, using a patient transportation dataset simulator. For this simulation study, patients can receive one of three mutually exclusive treatments: a free ride, a transit voucher, or no transportation assistance. The budgets for the transit vouchers is fixed at 20% of the population and for free rides at 5% of the population. The simulated population consists of two demographic groups.

4.2 Simulations

To evaluate our process fairness learning approach, we conducted a simulation study using the datasets described in Section 4.1. To efficiently learn decision policies in the real world, we integrate the LP formulation from Section 3.1 with the upper confidence bound (UCB), a common contextual bandit approach, as described in Algorithm 1. For the UCB approach, we compute the optimal policy π^*_i using an optimistic reward estimate of $r(x, k)$ (e.g. using the 95th percentile of the posterior of $\hat{r}(x, k)$).

In order to obtain γ for the process-fairness simulations, we first learned policies inputting specific λ_g values into the [4] algorithm. With these optimizations, we obtained oracle policies that balance between maximizing appearances and achieving parity in the distribution of transportation assistance across groups. These policies had an optimal fairness value that we used as input into our process-fairness simulations as γ. Using this approach, we could better understand how the process fairness constraint directly affects learning595

Additionally, we compare our approach against an oracle under the process fairness constraint that observes the true expected appearance probabilities.

5 Results and Discussion

5.1 Plots

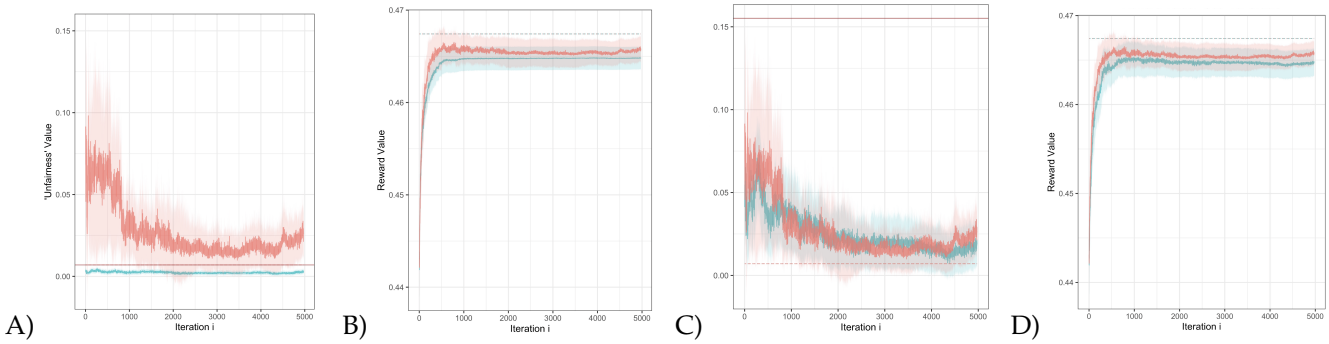


Figure 1: Transportation Allocation Fairness. This figure shows the comparison between learning to be fair (red) and process fairness (blue) at different fairness constraints. The dotted lines represent the fairness value of the oracle policies under the different constraints. B and D demonstrate the learned reward values of the policies at the corresponding fairness constraints from A and C.

5.2 Discussion

In Figure 5.1, the fairness value in plots A and C correspond to the plot of the second term of the objective, specifically $F(\pi, r, c)$, and the reward value in plots B and D correspond to the plot of the first term of the objective, specifically $\sum_x p_x \sum_a \pi_{xa} r_{xa}$. As we can see in both plots A and C, the fairness values from the process-fair policies (blue) are consistently below the threshold, indicated by the dashed lines, whereas for the learning to be fair policies the policies (red) do go above the threshold in some cases. The plots in A and B correspond to policies learned with a larger lambda and smaller γ threshold for process fairness, whereas the plots in C and D are the result of policies learned with smaller lambda values and larger γ threshold for process fairness. We see that the oracle policies under both process fairness and the learning to be fair constraints perform the same when the fairness bound constraint is looser or the same as the lambda penalty term on fairness, as demonstrated by the dashed lines in Figure 5.1. Moreover, when we compare process fairness with learning to be fair oracle performance, we mostly see that process fairness does worse than learning to be fair performance demonstrating that the process fairness constraint does cause a reduction in performance.

6 Future Work

In the future, we would like to better understand how process fairness affects other domains, where critical decisions need to be made. From the learning standpoint, we would like to further examine how process fairness impacts exploration when learning the optimal policy.

References

[1] S. Agrawal, N. R. Devanur, and L. Li. Contextual bandits with global constraints and objective. *CoRR*, abs/1506.03374, 2015. URL `http://arxiv.org/abs/1506.03374`.

[2] A. M. Barry-Jester, B. Casselman, and D. Goldstein. The new science of sentencing. *The Marshall Project*, 2015.

[3] N. Byrnes. Artificial intolerance. *MIT Technology Review*, 2016.

[4] A. Chohlas-Wood, M. Coots, H. Zhu, E. Brunskill, and S. Goel. Learning to be fair: A consequentialist approach to equitable decision-making, 2022.

[5] C. C. Miller. Can an algorithm hire better than a human? *The New York Times*, 2015.

[6] P. S. Thomas, B. C. da Silva, A. G. Barto, S. Giguere, Y. Brun, and E. Brunskill. http://people.cs.umass.edu/brun/pubs/pubs/Thomas19science.pdfPreventing Undesirable Behavior of Intelligent Machines. *Science*, 366(6468):999–1004, 22 November 2019. doi: 10.1126/science.aag3311.

[7] H. Wu, R. Srikant, X. Liu, and C. Jiang. Algorithms with logarithmic or sublinear regret for constrained contextual bandits, 2015.

An efficient code for predicting the time of future rewards

Margarida Sousa
Champalimaud Research
Champalimaud Foundation
Lisboa, 1400-038
margarida.sousa@research.fchampalimaud.org

Pawel Bujalski
Champalimaud Research
Champalimaud Foundation
Lisboa, 1400-038
pawel.bujalski@research.fchampalimaud.org

Bruno F. Cruz
Champalimaud Research
Champalimaud Foundation
Lisboa, 1400-038
bruno.cruz@neuro.fchampalimaud.org

Kenway Louie
Center for Neural Science
New York University
New York, NY 10003
klouie@cns.nyu.edu

Daniel McNamee
Champalimaud Research
Champalimaud Foundation
Lisboa, 1400-038
daniel.mcnamee@research.fchampalimaud.org

Joseph J. Paton
Champalimaud Research
Champalimaud Foundation
Lisboa, 1400-038
joe.paton@neuro.fchampalimaud.org

Abstract

In order to understand the structure of the world, agents must make causal inferences based on temporal relationships. However, standard value-based approaches in reinforcement learning learn estimates of temporally discounted average future reward, leading to ambiguity about reward timing and magnitude. To resolve such ambiguity, it has been proposed that a population of neurons corresponding to distinct parallel value channels that differ in reward and temporal sensitivity can facilitate the learning about distributions of reward amount over time. Here we present a population coding model that can optimally represent such information when faced with limited neural resources by adapting to the temporal statistics of experienced rewards. Furthermore, we derive a biologically plausible learning rule that converges to the information-theoretically optimal code. We then show that this code outperforms a non-adapting one and suggest how several features of animal behavior may be explained as resulting from adaptation to temporal reward statistics. Lastly, if the brain implements such a code, neural reward prediction errors should 1) express variable sensitivity to the timing of future reward and 2) adapt this sensitivity to changes in the temporal statistics of reward. We are testing these predictions by recording the responses of optogenetically identified midbrain dopamine neurons in mice to conditioned stimuli that predict rewards at varying delays. We present preliminary data consistent with the predictions of the theory.

Keywords: distributional reinforcement learning, efficient coding, time, reward, dopamine, basal ganglia

1 Introduction

Extracting temporal structure from the world forms the basis for learning to behave adaptively in complex, dynamic environments. Relatedly, knowing when behaviorally relevant events such as rewards will occur is often critical for survival. For example, crossing a desert to reach an oasis is only advisable if you won't perish before you get there. Recently it has been shown that an extension of temporal difference learning can be used to create a distributional map of future events [4]. This approach relies on a multiplicity of factors by which future rewards are temporally discounted. However, it is not specified how those factors should be chosen. Artificial and biological agents have implementational constraints, namely limited channel capacity, creating pressure to adapt coding mechanisms to the statistics of the environment so as to preserve information. Recently, an adaptive distributional code has been proposed for reward amounts [1]. At the behavior level, a swath of evidence strongly suggests that discounting behavior is plastic. For example, the rate at which humans discount the value of delayed rewards decreases with age and, in contexts in which a single exponential discounting rate is optimal, humans can learn to match that factor [5]. In the brain, there is evidence that a variety of discounting processes may underlie that plasticity in discounting behavior. Midbrain dopamine neurons, thought to encode a form of reward-prediction error, discount future reward heterogenously [3] and value-related signals are organized topologically in the ventral-dorsal axis of the striatum in a gradient of delay discounting [7]. Finally, there is some evidence that serotonin may modulate the dynamic nature of such discount factors. We propose an adaptive neural population code that optimally updates the prediction time range of rewards to match the statistics of the environment through adaptation of temporal discount factors. Moreover, we present a biologically plausible set of learning rules that might underlie its implementation. We confirm that the adaptive code outperforms the non-adaptive code suggested previously in predicting the temporal evolution of expected rewards and hypothesize that it can explain several features of animal behavior. Lastly, we present preliminary evidence that expressed discounting-related signals in midbrain dopamine neurons of mice qualitatively match the predictions of our theory. Specifically, the discount factors associated to the dopamine neurons adjust in response to changes in the temporal distribution of reward in the environment.

2 Multiple temporal discounts code

The value at time step t is the expected sum of discounted future rewards $V_t = \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j}\right]$ and $\gamma \in [0, 1)$ is the temporal discount factor that determines how much delayed reward is devalued relative to immediate reward. The value can be learnt using the temporal-difference (TD) algorithm that updates, at each time step, the current estimate using a reward prediction error (RPE) δ, weighted by a learning rate α, $V_t \leftarrow V_t + \alpha\delta$ with $\delta = r + \gamma V_{t'} - V_t$. In the brain, the basal ganglia (BG) are a candidate biological substrate for implementing reinforcement learning (RL) algorithms. Cortex provides information about the state of the world and available actions to the striatum, which is thought to learn their corresponding value. Such value is updated by a RPE conveyed by midbrain dopamine neurons. As mentioned before, evidence suggests that canonically identified parallel circuits within the BG might compute different temporally discounted values V_γ. In parallel, it has been shown that the value is the Laplace transform of the expected reward at future time steps with parameter $s = -\log\gamma$ [8], that is,

$$V_t = \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j}\right] = \sum_{j=0}^{\infty} \gamma^j \bar{r}(t+j) = \sum_{j=0}^{\infty} e^{-j(-\log\gamma)} \bar{r}(t+j) = F(-\log\gamma). \tag{1}$$

We represent the expected rewards at time t by $\bar{r}(t) = \mathbb{E}[r_t]$. There are well known methods for inverting the Laplace Transform that, given a set of temporal discounts $\gamma_1, \ldots, \gamma_N$ and values $V_{\gamma_1}, \ldots, V_{\gamma_N}$, reconstruct the expected reward at all future time steps $\{\bar{r}(t+1), \ldots, \bar{r}(t+N)\}$. At the circuit level, this suggests that by integrating the information of multiple parallel circuits in the BG it is possible to infer the temporal evolution of expected rewards. Specifically, Post's formula uses k-order derivatives over the s-space, that can be thought as center-surround receptive fields in neighborhood of points in the s-space, to estimate the expected future rewards at time steps t, $\hat{\bar{r}}(t) = \lim_{k\to\infty} s^{k+1} F^{(k)}(ks)$, where $F^{(k)}$ is the kth-derivative of $F(s)$. We assume each dopamine neuron encodes the RPE relative to a exponentially discounted value with a decay of s and a temporal discount of $\gamma = e^{-s}$. To decode the expected reward at time t the responses of neuron with $\gamma = e^{-1/t}$ and its k neighbors are used (Figure 1i). Values with associated low temporal discount factors, discriminate short reward times, those with associated high temporal discount factors discriminate longer times on a wider range (Figure 1a,e). Therefore the mapping from time to γ is monotonically increasing and biased to higher temporal discount factors (Figure 1i). k is an hyper-parameter that is tuned to trade off accuracy and system complexity, that linearly scales the population temporal discounts. Importantly, a network model implementation for this decoder has been proposed, highlighting a biologically plausible solution for its implementation.

3 Adaptive multiple temporal discounts code

We use an efficient population coding framework [2] to derive the distribution of tuning functions that optimally represent the temporal evolution of future rewards. Whereas in previous work, efficient coding has been typically deployed

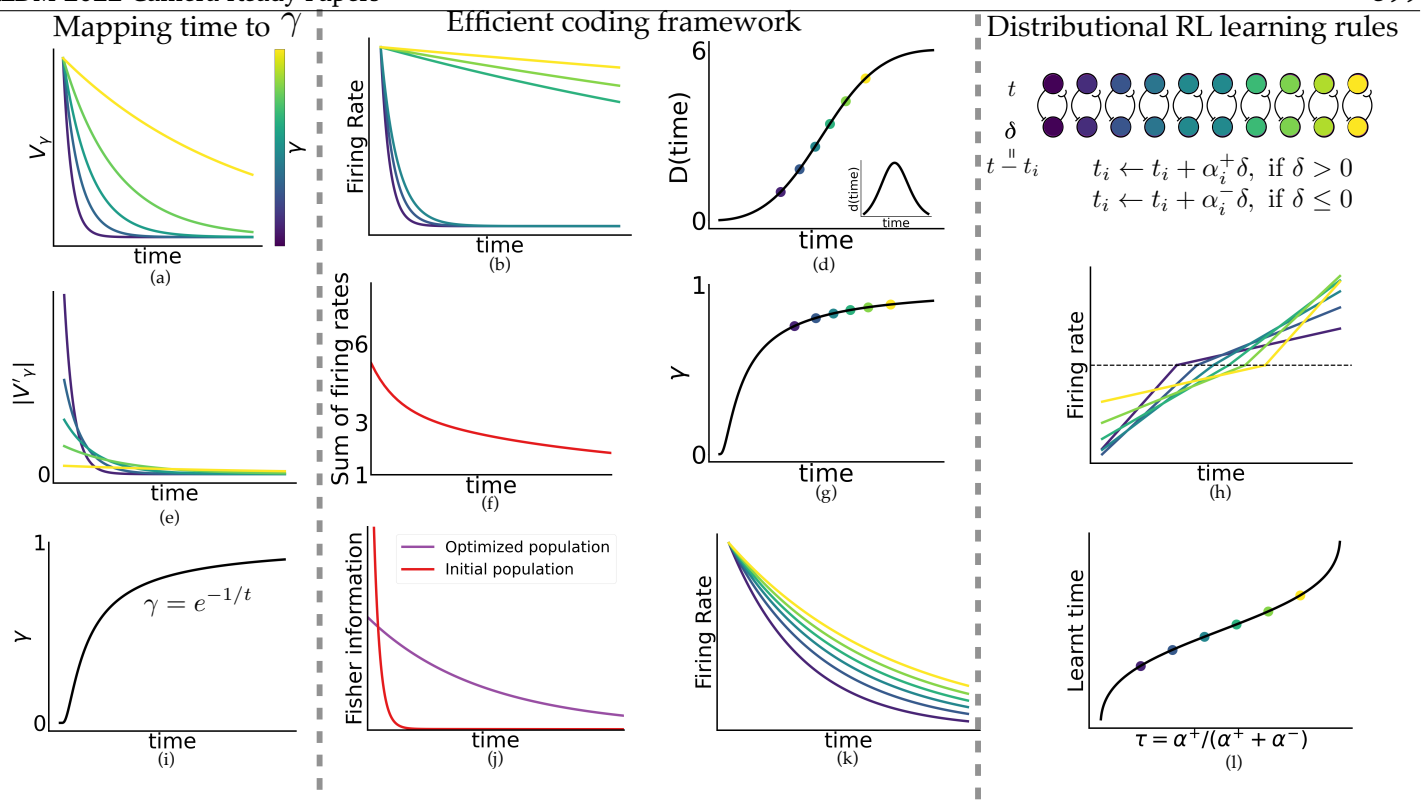


Figure 1: In the leftmost panel a set of temporal discounted values as a function of time to reward are shown (a). Low temporal discount factors discriminate short times to reward, high wider ranges, as shown by the derivatives in (e). The mapping between time to discount factors is represented in (i). In the middle panel, (b) shows the initial tuning functions and we confirm in (f) that they approximately tile the time space in the range of interest. In (d) the inset shows $d(\text{time})$, the cumulative integral $D(\text{time})$, changes the spacing between the times learnt t_i. These are then mapped to a set of temporal discount factots (g), and the optimal tuning functions are represented in (k). In (j) we compare the fisher information for the initial and optimized population, and confirm that initially it is approximately constant in the range of interest, and for the optimized population it is redistributed and higher for more frequent occurring reward times. In the rightmost panel, in (h) we represent the learning rules for t_i, and in (l) the distribution of t'_is learnt.

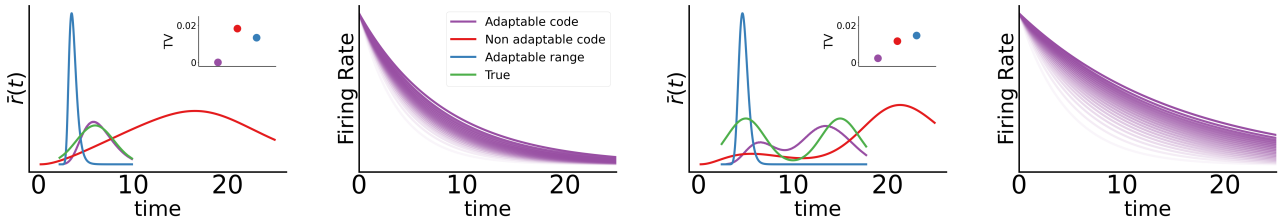
in order to embed natural stimulus statistics within the neural response functions for the purposes of accurate stimulus estimation, here our objective is to develop a neural population code optimized to make predictions of future rewards. In particular, we seek to maximize the *mutual information* $I(\bar{r}(t), \hat{\bar{r}}(t))$, between the true $\bar{r}(t)$ and that encoded from neural population responses, $\hat{\bar{r}}(t)$. We assume independent Poisson noise and consider a set of N tuning functions that approximately tile the future reward times (Figure 1f) and have constant Fisher information (Figure 1j), represented in Figure 1b. We parameterize the population with the density of tuning curves d as a function of time. Constraining on the number of neurons N, the solution that optimally represents $\bar{r}(t)$ is given by $d(t) \propto Np(\bar{r}(t))$ (Figure 1d). Using the mapping defined in the Laplace decoder (Figure 1g), the optimal tuning curves are represented in Figure 1k. Interestingly, the set of parameters we arrive at are quantiles of $\bar{r}(t)$, that are the *expectiles* of $r(t)$ [4]. If we assume the reward function is either 0 or 1, $p(r(t))$ collapses to the probability distribution of reward times, that we represent as $p(r|t)$. There is evidence that dopamine neurons signal the RPE relative to different expectiles of the probability distribution of reward amount, and TD-learning rules have been derived to learn these expectiles [1]. Similar learning rules can be used to learn the expectiles of the probability of rewards in time (Figure 1h,l).

4 Simulations

4.1 Classical conditioning task

We study a classical conditioning task, where a cue predicts reward with distinct delays sampled from a given probability distribution, and compare how well the adaptable and non-adaptable codes predict the temporal evolution of expected rewards at the cue. We consider a unimodal and a bimodal probability distribution and represent the optimized tuning functions and decoded temporal evolution of expected rewards at the cue in Figure 2. For the non-adaptable code, we

considered exponential decay parameters distributed uniformly in the interval $(0, 1/t_{max}]$ and also uniformly distributed in the range of the adaptable code. N=200 neurons were considered and derivatives of order $k = 20$ were used in the decoder for the unimodal distribution and $k = 40$ for the bimodal distribution. In both cases, the adaptable code outperforms the non-adaptable code.

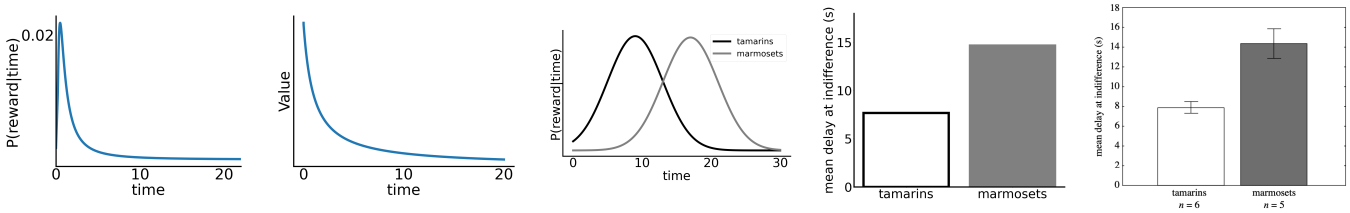


(a) (b) (c) (d)

Figure 2: In (a) and (c) the decoded expected future rewards are represented and in the insets the total variation distance (TV) between the decoded and true functions are shown. In (b) and (d) the tuning functions for the adaptable code are shown.

4.2 Intertemporal choice task

Intertemporal choice tasks are used to determine the influence of time delays on reward evaluation. There is evidence that pigeons, rodents, monkeys and humans discount hyperbolicaly delayed reward, i.e. the discounting of rewards in the future decreases constantly with time. We compute the distribution over reward delays that leads to a hyperbolic discounting (Figure 3a). We hypothesize that hyperbolic discounting can be a consequence of having this representation of probability of reward delays. Two closely related primates: marmosets and tamarins, were submitted to a intertemporal



(a) (b) (c) (d) (e)

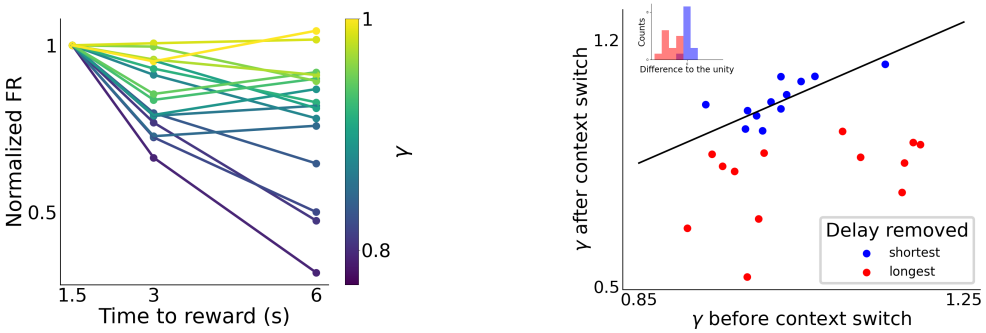
Figure 3: In (a) the distribution over reward delays that generates hyperbolic discounting is represented. In (b) we confirm that the generated value is hyperbolic. In (c) the distributions of reward delay in the environment of tamarins and marmosets are represented. The mean delay of indifference for these two species are represented in (e) and the one predicted by our model is in (d).

choice task. The marmosets waited significantly longer for food than tamarins ([6], Figure 3e). In their natural environment, marmosets rely on gum, a food product acquired by waiting for exudate to flow from trees, whereas tamarins feed on insects, a food requiring fast actions. We hypothesize that in an evolutionary perspective, the tamarins were submitted to shorter delays to get food, and therefore the set of temporal discount factors are biased to smaller values and the marmosests on the other hand are biased to larger. We consider two distributions of delay to reward (Figure 3c), shifted by a fixed amount, and show how this shift generates the observed difference in indifference values (Figure 3d).

5 Preliminary data

Currently, we are testing predictions of our model by recording the responses of optogenetically identified midbrain dopamine neurons in mice to conditioned stimuli that predict rewards at different delays: 1.5s, 3s and 6s. Critically, in the middle of each recording session there is a context switch and we remove either the condition associated with the longest reward delay (6s) or the shortest delay (1.5s). The theory predicts that when the longest delay is removed, neurons decrease their temporal discount factor and when the shortest delay is removed, neurons increase their temporal discount factor. Dopamine neurons show different sensitivities for the timing of rewards (Figure 4a), and a clear adaptation to lower discounts when the longest delay is removed (Figure 4b). However, when the shortest delay is removed, we do not observe an adaptation of the population discounts. We suggest that this temporal asymmetry in neural adaptation

may be due to a rational bias in the neural code towards ensuring that predictions for the very near future are accurately encoded regardless of the temporal reward distribution in the environment. The model can generate this asymmetry if a prior over time $p(t)$ biased to short delays is included such that the optimal distribution of tuning curves becomes $d(t) \propto Np(\bar{r}(t))p(t)$.



(a) (b)

Figure 4: In (a) we show the population of dopamine neurons' FR normalized to the shortest delay FR. In (b) the measured discount rate before and after context switches are plotted. In the inset we show the histogram of the difference to the unity for the two sets of points and confirm that the red set of points is biased to negative values and the blue has zero mean. We estimated these temporal discount factors using the FR at only two delays, and therefore do not rely on the absolute estimated values that may be subject to noise, and that in some cases are greater than one, but on the relative adaptation pattern.

6 Discussion

We propose a neural population code that optimally updates the prediction time range of rewards to match the statistics of the environment through adaptation of temporal discount factors. In simulations, we show that this code outperforms the non-adaptable code and that hyperbolic discounting and environment dependent temporal discounting can be explained by this theory. Lastly, we present preliminary dopamine data that qualitatively match the predictions of the theory. As future steps, we aim to derive an efficient code for jointly representing reward delay and amount.

References

[1] Will Dabney, Zeb Kurth-Nelson, Naoshige Uchida, Clara Kwon Starkweather, Demis Hassabis, Rémi Munos, and Matthew Botvinick. A distributional code for value in dopamine-based reinforcement learning. *Nature*, 577(7792):671–675, 2020.

[2] Deep Ganguli and Eero P Simoncelli. Efficient sensory encoding and bayesian inference with heterogeneous neural populations. *Neural computation*, 26(10):2103–2134, 2014.

[3] Shunsuke Kobayashi and Wolfram Schultz. Influence of reward delays on responses of dopamine neurons. *Journal of neuroscience*, 28(31):7837–7846, 2008.

[4] Mark Rowland, Robert Dadashi, Saurabh Kumar, Remi Munos, Marc Bellemare, and Will Dabney. Statistics and samples in distributional reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5528–5536, 2019.

[5] Nicolas Schweighofer, Mathieu Bertin, Kazuhiro Shishida, Yasumasa Okamoto, Saori C Tanaka, Shigeto Yamawaki, and Kenji Doya. Low-serotonin levels increase delayed reward discounting in humans. *Journal of Neuroscience*, 28(17):4528–4532, 2008.

[6] Jeffrey R Stevens, Elizabeth V Hallinan, and Marc D Hauser. The ecology and evolution of patience in two new world monkeys. *Biology letters*, 1(2):223–226, 2005.

[7] Saori C Tanaka, Kenji Doya, Go Okada, Kazutaka Ueda, Yasumasa Okamoto, and Shigeto Yamawaki. Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. In *Behavioral economics of preferences, choices, and happiness*, pages 593–616. Springer, 2016.

[8] Pablo Tano, Peter Dayan, and Alexandre Pouget. A local temporal difference code for distributional reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13662–13673. Curran Associates, Inc., 2020.

601

Designing Rewards for Fast Learning

Henry Sowerby
Department of Computer Science
Brown University
Providence, RI 02912
henry_sowerby@brown.edu

Zhiyuan Zhou
Department of Computer Science
Brown University
Providence, RI 02912
zhouzy@brown.edu

Michael L. Littman
Department of Computer Science
Brown University
Providence, RI 02912
mlittman@cs.brown.edu

Abstract

To convey desired behavior to a Reinforcement Learning (RL) agent, a designer must choose a reward function for the environment, arguably the most important knob designers have in interacting with RL agents. Although many reward functions induce the same optimal behavior (Ng et al., 1999), in practice, some of them result in faster learning than others. In this paper, we look at how reward-design choices impact learning speed and seek to identify principles of good reward design that quickly induce target behavior. This reward-identification problem is framed as an optimization problem: Firstly, we advocate choosing state-based rewards that maximize the action gap, making optimal actions easy to distinguish from suboptimal ones. Secondly, we propose minimizing a measure of the horizon, something we call the "subjective discount", over which rewards need to be optimized to encourage agents to make optimal decisions with less lookahead. To solve this optimization problem, we propose a linear-programming based algorithm that efficiently finds a reward function that maximizes action gap and minimizes subjective discount. We test the rewards generated with the algorithm in tabular environments with Q-Learning, and empirically show they lead to faster learning. Although we only focus on Q-Learning because it is perhaps the simplest and most well-understood RL algorithm, preliminary results with R-max (Brafman and Tennenholtz, 2000) suggest our results are much more general. Our experiments support three principles of reward design: 1) consistent with existing results, penalizing each step taken induces faster learning than rewarding the goal. 2) When rewarding subgoals along the target trajectory, rewards should gradually increase as the goal gets closer. 3) Dense reward that's nonzero on every state is only good if designed carefully.

Keywords: Reward design, Q-Learning, Machine Learning as Programming, Inverse Reinforcement Learning, Interactive Reinforcement Learning

Acknowledgements

We are grateful to David Abel and his reward research group at Google DeepMind for discussions and feedback, as well as NSF, DARPA, and ONR for funding support.

Problem Setting

We formulate the RL problem as a Markov Decision Process, modeling the environment in terms of states S, actions A, reward function $R(s)$, discount factor γ, and transition function $T(s, a, s')$. Reward functions are often defined on state–action pairs (Puterman, 1994), but we limit ourselves to the simpler case in which a reward function depends only on the states (Russell and Norvig, 1994). To further simplify reward specification, states are described in terms of a vector of features $F(s, i)$ and rewards are computed as a linear combination of these features (Ng and Russell, 2000). We seek to encourage, via rewards on state features, a target policy π^+ (Abel et al., 2021) mapping states to actions. An optimal policy π^*_R is one that maximizes the cumulative discounted expected reward from all states. We say a reward function R is *correct* if $\pi^*_R = \pi^+$.

To evaluate the ease of learning for a given correct reward function, we count the number of discrete timesteps where the learner's preference matches the target policy. We stray away from the usual benchmark of cumulative reward achieved because reward is misleading in our case; designing a reward function that has across-the-board higher reward is not necessarily better at the task of encouraging the target behavior. Specifically, we define

$$\text{correct actions} = \sum_{t=1}^{T} \mathbb{1}[\text{argmax}_a Q(s_t, a) = \pi^+(s_t)].$$

We start by examining the diversity of correct reward functions for a given target policy. Consider the grid world example from Russell and Norvig (1994), described in Figure 1. To assess desirable aspects of different reward functions, we sample a collection of random reward functions that set $R(s) \in [-1, 1]$ for each state $s \in S$. Of the approximately $10,000,000$ randomly-generated reward functions, $5,000$ (0.05%) were correct, and we ran Q-Learning on these reward functions for 10,000 steps. The right margin of Figure 2 shows the distribution of cumulative number of correct actions these rewards induce. The distribution shows that although each of these reward functions produce the target policy when fully optimized, they differ significantly in the speed with which Q-Learning aligns itself with this target policy.

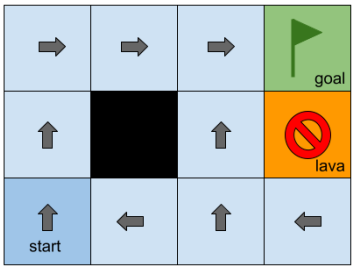


Figure 1: Russell/Norvig grid: The agent tries to get to the goal while avoiding the lava. At each step, the agent can choose to move in four cardinal directions, each of which has probability 0.8 to transition in that direction, and probability 0.1 to slip into the two orthogonal directions. The target policy is illustrated with arrows.

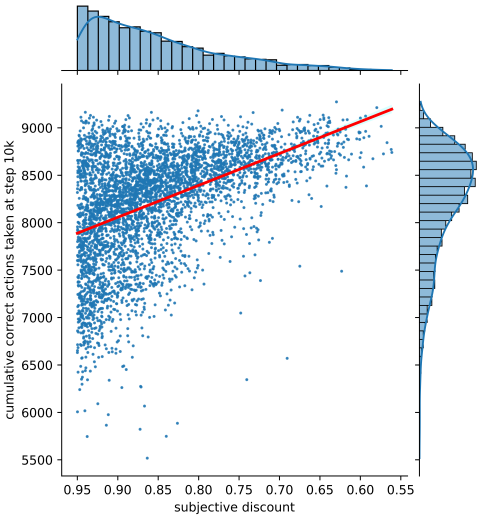


Figure 2: Cumulative correct actions of random reward functions vs. their subjective discount, Russell/Norvig grid, averaged over $1,000$ runs. Regression in red.

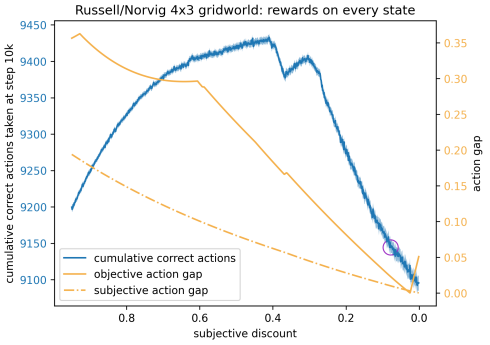


Figure 3: Learning performance (with 99% confidence interval averaged over 5,000 runs) of linear-program-designed rewards and their action gap. The objective action gap is computed at $\gamma = 0.95$, and the subjective action gap at $\tilde{\gamma}$. The purple circle marks the cumulative correct actions at threshold $\tilde{\delta} = 0.01$.

Subjective Discount

In trying to understand the properties of reward functions that lead to faster learning, we leverage the following insight: The convergence of algorithms like Q-Learning and R-Max and even value iteration depends on the discount factor. Smaller discount factors lead to faster convergence.

In our setting, we cannot simply adopt a smaller discount factor because we consider it to be part of the environment. As such, we adopt an idea from Jiang et al. (2015) and differentiate between two kinds of discount factors: the objective discount γ of the environment and a *subjective discount* $\tilde{\gamma}$, the latter of which is a property of the reward function. γ is used by the Q-Learning agent to estimate returns during learning, and $\tilde{\gamma}$ to design rewards.

Given a reward function R, we define π_R^γ to be the policy of an agent optimizing R in an environment with discount γ. The subjective discount for reward function R and target policy π^+ is the smallest $\tilde{\gamma}$ such that $\pi_R^{\gamma'} = \pi^+, \forall \gamma' \in [\tilde{\gamma}, \gamma]$. That is, we want to know how small the discount factor could get while still encouraging the target policy. In our implementation, we use a binary search to efficiently compute the subjective discount.

Even though the learning agent does not have direct access to $\tilde{\gamma}$, it still seems to be a useful marker for good reward functions. Returning to Figure 2, the x-axis plots the subjective discount against the cumulative number of correct actions that match our target policy. Although the two measures are not perfectly correlated, low subjective discounts are reliable estimators of fast learning.

Reward Construction via Linear Programming

To aid our exploration of the relationship between subjective discount and rewards for fast learning, we created an efficient algorithm for identifying correct reward functions with minimal subjective discounts. Whereas our random reward-function search was able to identify reward functions with subjective discounts of around 0.55 for the Russell/ Norvig grid, our linear-programming-based optimization process can push this value down to effectively zero. The generated reward for step cost, goal reward, and lava penalty, respectively, was $(-0.0223, +0.6119, -1)$, which outperforms the original rewards $(-0.04, +1, -1)$, with 88,364 vs. 83,666 correct actions, after 100k steps, averaged over 100 runs.

We make use of the discounted expected feature expectation (Abbeel and Ng, 2004) $D(s,i)$ and state–action feature expectations $D_a(s,i)$ for state s, action a, and feature i. They are defined using the feature value $F(s,i)$ and target policy π^+ at discount γ:

$$D(s,i) = F(s,i) + \gamma \sum_{s'} T(s, \pi^+(s), s') \cdot D(s', i)$$

$$D_a(s,i) = F(s,i) + \gamma \sum_{s'} T(s, a, s') \cdot D(s', i)$$

Similarly, we also define the same quantities $\widetilde{D}(i,s)$ and $\widetilde{D}_a(s,i)$ using subjective discount $\tilde{\gamma}$. It has been shown (Syed et al., 2008) that the state value function and Q-value of the target policy can be expressed as $V^{\pi^+}(s) = \sum_i D(s,i) \cdot R(i)$ and $Q^{\pi^+}(s,a) = \sum_i D_a(s,i) \cdot R(i)$, where $R(i)$ is the reward function based on the feature i.

The goal of our optimizer is to construct a reward function that induces the target policy π^+ at the objective discount *and* the lowest possible subjective discount. Our linear program is then formulated as:

$$\begin{aligned}
&\text{choose } \delta, R(i) \text{ to maximize } \delta \\
&\text{subject to} \sum_i D(s,i) \cdot R(i) \geq \sum_i D_a(s,i) \cdot R(i) + \delta \\
&\qquad \sum_i \widetilde{D}(s,i) \cdot R(i) \geq \sum_i \widetilde{D}_a(s,i) \cdot R(i) + \delta \\
&\qquad -1 \leq R(i) \leq 1 \\
&\qquad \forall\, i, s \in S, a \in A \setminus \{\pi^+(s)\}
\end{aligned}$$

In words, choose the reward function such that, for each state, the target policy's action in that state is at least δ better than any other action (for both the objective and subjective discount factors). The role of the δ variable here is to make sure that the target policy's action is truly better ($\delta > 0$) than the other actions.

Next, we more closely evaluate the rewards constructed. For a range of $\tilde{\gamma} \in [0, 1]$, we ran the linear program to construct a range of reward functions, one for each $\tilde{\gamma}$. Figure 3 plots the cumulative correct actions taken in $10,000$ steps for these rewards. The result shows that the best learning performance is achieved at an intermediate value of subjective discount, and we hypothesize that the reason is a kind of policy optimization regularization (Jiang et al., 2015).

Action Gap

Although the maximization objective δ was included as a mathematical convenience, it measures a property that has been recognized as significant in the literature. The difference between the value of the optimal action and the second best action is known as the *action gap* (Farahmand, 2011). It is commonly believed that a large action gap is beneficial for learning, mitigating effects of estimation errors (Lehnert et al., 2018), improving policies derived from Q-values (Bellemare et al., 2015), and achieving faster convergence rates (Farahmand, 2011). To hedge against action gap being optimized to 0 as subjective discount decreases, we add the constraint $\delta \geq 0.01$.

Principles of Good Reward Design

Advice: Penalize Steps

In a first-of-its kind result, Koenig and Simmons (1993) analyzed the time complexity of Q-Learning in a deterministic goal-based task, where they considered two two-featured correct reward functions: $(1, 0)$ and $(0, -1)$. The first, the *goal-reward* representation, provides the agent with a reward of $+1$ when the goal is reached and 0 while en route. The second, the *action-penalty* representation, penalizes the agent for each step taken, ending when the agent reaches the goal.

From a zero-initialized Q-function, they argued that the time to learn a good behavior in the goal-reward representation can grow exponentially with the size of the state space, while in the action-penalty representation $O(|A||S|^2)$ steps suffice. Essentially, the zero-initialization and the zero rewards means the agent has no guidance and explores randomly until the goal is encountered by accident. With action penalties, the agent tries to avoid actions it has already taken, resulting in much more directed exploration.

We replicate this finding experimentally in a simple 60-state chain MDP with objective discount 0.95. It is a one-dimensional environment where the agent starts at the leftmost state, and needs to take rightward actions to arrive at the goal state on the opposite side. The two available actions transition the agent deterministically left or right.

We compare the performance of the two reward functions just described with one that our linear program generates. For this environment, our linear program chooses $(1, -1)$, which we call "combo" as it combines the penalty and the reward.

At the objective discount of 0.95, the action gaps for the three reward functions are approximately 0.0024 (goal reward), 0.0485 (action penalty), and 0.0509 (combo). Since we use an action-gap threshold of 0.01, goal reward doesn't have a well defined subjective discount, while action penalty achieves a subjective discount of 0.9249, and combo 0.9238, approximately. The subjective discounts suggest that combo and action penalty will be similar, and both better than goal reward. Figure 4 confirms this hypothesis.

Advice: Reward Subgoals

In the context of hierarchical reinforcement learning (Parr and Russell, 1998), long-horizon tasks are broken into smaller tasks. Commonly, policies for the subtasks have their own pseudorewards that encourage subtask completion. In the context of a long-horizon goal, how should rewards be assigned to subtasks to encourage fast and accurate learning?

In naturally occurring problems, subtasks can break up goal-seeking sequences into heterogeneous chunks. In the simplified setting we study here, subtask completion states are spaced evenly along the 60-state chain and each has its own feature (and therefore can receive its own reward value).

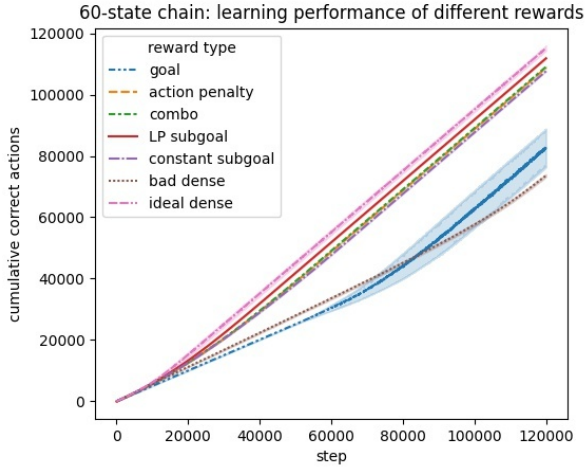


Figure 4: Comparison of reward functions on the 60-state chain.

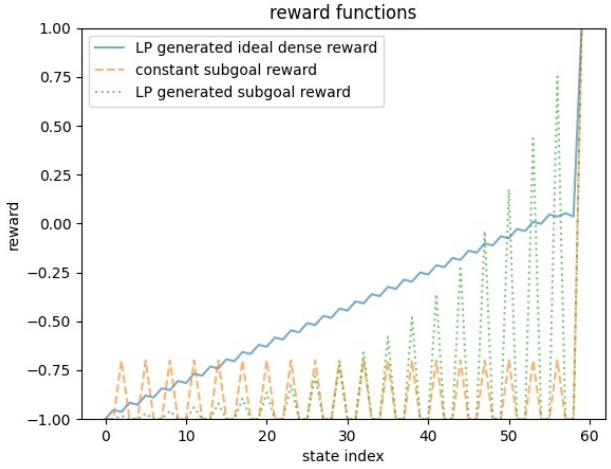


Figure 5: Two reward functions that place rewards on intermediate subgoals

We consider subgoal states placed every three states along the chain. The natural design choice would be to reward each of the resulting 19 subgoals the same reward value r, the goal $+1$, and all other states a step penalty -1, as suggested by the last section. However, it is not immediately obvious how to arrive at a value for r such that the more frequently

spaced rewards induce the target policy. One example that we have found is $r = -0.7$, a small advantage over the -1 rewarded non-subgoal states.

We compared the constant subgoal reward above to a subgoal reward profile generated by our linear program. The reward functions are shown in Figure 5. The constant subgoal reward has a subjective discount of 0.9510, while the subgoal profile has a much lower subjective discount of 0.8232. The learning results are in Figure 4.

From Figure 4, we observe significant differences in learning time; the subgoal profile reward learns significantly faster than the constant subgoal reward, and, the goal-only "combo" reward from the last section is included, which actually learns faster than the constant subgoal reward. This result suggests that only appropriate subgoal rewards speed learning, while inappropriate ones may actually lead to slower learning.

We find that, when defining subgoal rewards, it helps to gradually increase rewards as the agent gets closer to the goal state. This design helps counteract the effect of discounting, but also continually spurs the agent forward, much like an annual salary raise is considered to be a good motivator in the commercial sector.

Advice: Use Dense Rewards

It is widely recognized that the kind of sparse rewards that come from rewarding only the goal is inferior to "dense" rewards like the subgoal rewards discussed in the previous section (Smart and Kaelbling, 2002).

Here, we look at the limiting case where every state has its own reward value. Figure 5 shows the reward function found by the linear program. It has a subjective discount of 0.2128. Organizing rewards in a jagged shape lets the reward function have the largest possible state-by-state increase while still remaining in the required -1 to $+1$ range. Notice that the intermediate rewards are all negative in value—that's to prevent the agent from finding the intermediate rewards more attractive than reaching the goal.

Figure 4 compares the Q-Learning performance on this LP-produced dense reward to the spare rewards from the previous section. We also included a second fully dense reward function selected to have a low action gap of near 0, which performs much worse. An important lesson here is that sparse rewards are indeed difficult to learn from, but not all dense rewards are equally good.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 1.

Abel, D., Dabney, W., Harutyunyan, A., Ho, M. K., Littman, M. L., Precup, D., and Singh, S. (2021). On the expressivity of Markov reward. arXiv preprint arXiv:2111.00876.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., and Munos, R. (2015). Increasing the action gap: New operators for reinforcement learning.

Brafman, R. I. and Tennenholtz, M. (2000). A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence*, 121(1–2):31–47.

Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 172–180.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of AAMAS*, pages 1181–1189.

Koenig, S. and Simmons, R. G. (1993). Complexity analysis of real-time reinforcement learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 99–105, Menlo Park, CA. AAAI Press/MIT Press.

Lehnert, L., Laroche, R., and van Seijen, H. (2018). On value function representation of long horizon problems. In *AAAI Conference on Artificial Intelligence*.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670.

Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference*.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

Russell, S. J. and Norvig, P. (1994). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation)*, volume 4, pages 3404–3410. IEEE.

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1032–1039, New York, NY, USA. Association for Computing Machinery.

Trust-Region-Free Policy Optimization for Stochastic Policies

Mingfei Sun
University of Oxford
mingfei.sun@cs.ox.ac.uk

Benjamin Ellis
University of Oxford
benjamin.ellis@keble.ox.ac.uk

Anuj Mahajan
University of Oxford
anuj.mahajan@cs.ox.ac.uk

Sam Devlin
Microsoft Research
sam.devlin@microsoft.com

Katja Hofmann
Microsoft Research
katja.hofmann@microsoft.com

Shimon Whiteson
University of Oxford
shimon.whiteson@cs.ox.ac.uk

Abstract

Trust Region Policy Optimization (TRPO) is an iterative method that simultaneously maximizes a *surrogate objective* and enforces a trust region constraint over consecutive policies in each iteration. The combination of the surrogate objective maximization and the trust region enforcement has been shown to be crucial to guarantee a monotonic policy improvement. However, solving a trust-region-constrained optimization problem can be computationally intensive as it requires many steps of conjugate gradient and a large number of on-policy samples. In this paper, we show that the trust region constraint over policies can be safely substituted by a *trust-region-free* constraint without compromising the underlying monotonic improvement guarantee. The key idea is to generalize the surrogate objective used in TRPO in a way that a monotonic improvement guarantee still emerges as a result of constraining the maximum advantage-weighted ratio between policies. This new constraint outlines a conservative mechanism for iterative policy optimization and sheds light on practical ways to optimize the generalized surrogate objective. We show that the new constraint can be effectively enforced by being conservative when optimizing the generalized objective function in practice. We call the resulting algorithm Trust-REgion-Free Policy Optimization (TREFree) as it is free of any explicit trust region constraints. Empirical results show that TREFree outperforms TRPO and Proximal Policy Optimization (PPO) in terms of policy performance and sample efficiency.

Keywords: policy optimization; deep reinforcement learning

Acknowledgements

Mingfei Sun is also affiliated with Microsoft Research. He is partially supported by funding from Microsoft Research. The experiments were made possible by a generous equipment grant from NVIDIA.

1 Introduction

Trust Region Policy Optimization (TRPO) [9] is an iterative method that optimizes stochastic policies with a trust region constraint. One of the key ideas in TRPO is to simultaneously optimize a *surrogate objective* and enforce a trust region constraint over consecutive policies at each iteration. The use of surrogate objectives stems from the seminal work of [6], which modifies the policy gradient (PG) objective [12] by substituting the on-policy state distribution with a distribution induced by the policy from the preceding iteration. [9] show that, despite the mismatch between what the policy update should optimize, i.e. the PG objective, and what is optimized in practice, i.e. the surrogate objective, a monotonic improvement guarantee for policy performance can still emerge from constraining the policy update at each iteration. The resulting TRPO algorithm thus strictly enforces a Kullback-Leibler (KL) divergence constraint between consecutive policies, and seeks to solve a KL-constrained surrogate objective optimization at each iteration. This combination of the surrogate objective maximization and the trust region enforcement has also been shown to be crucial for policy improvement in practice [9, 2].

However, solving a KL-constrained optimization problem can be computationally intensive [10]. In particular, TRPO use a quadratic approximation of the KL that augments natural policy gradients [7] with a line-search step that critically ensures KL enforcement [9]. This procedure requires many steps of conjugate gradient and a large number of on-policy samples making it both computationally intensive and sample inefficient [13]. Many follow-up studies attempt to improve TRPO, for example by leveraging Kronecker-factored approximated curvature to approximate the trust region [13], solving the KL-regularized optimization analytically via Expectation-Maximization [1, 5], transforming TRPO into an unconstrained optimization by policy space projections [3] or integrating the constraint into differentiable layers [8].

In this paper, we propose simplifying policy optimization by completely removing the trust region constraint, without compromising the underlying monotonic improvement guarantee. Specifically, instead of the framework of surrogate objective optimization [6, 9], we generalize the surrogate objective used in TRPO in a way that a monotonic improvement guarantee still emerges as a result of constraining the maximum advantage-weighted ratio between policies. This new constraint is different from the trust region constraint in TRPO in that it does not seek to impose any divergence constraint over consecutive policies. Instead, it outlines a conservative mechanism to bound the maximum advantage-weighted ratios in each iteration, and sheds light on practical ways to directly optimize the generalized surrogate objective. We show that the new constraint can be simply enforced by being conservative when optimizing the generalized objective function in practice. Furthermore, we present Trust-REgion-Free Policy Optimization (TREFree), a practical policy optimization method for optimizing stochastic policies. Empirical results show that TREFree is effective in optimizing policies, outperforming TRPO and PPO in both performance and sample efficiency.

2 Preliminaries

Markov decision process (MDP). Single-agent RL can be modelled as an infinite-horizon discounted Markov decision process (MDP) $\{\mathcal{S}, \mathcal{A}, P, r, d_0, \gamma\}$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $d_0 : \mathcal{S} \to \mathbb{R}$ is the initial state distribution and $\gamma \in [0, 1)$ is the discount factor. Let π denote a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, the performance for a stochastic policy $\pi(a|s)$ is defined as: $J(\pi) \triangleq \mathbb{E}_{s_0 \sim d_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)} \big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\big]$ The action-value function Q_π and value function V_π are defined as: $Q_\pi(s_t, a_t) \triangleq \mathbb{E}_t \Big[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})\Big]$, $V_\pi(s_t) \triangleq \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} \Big[Q_\pi(s_t, a_t)\Big]$. Accordingly, the advantage function is defined as $A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s)$.

TRPO. Define the discounted state distribution as: $d_\pi(s) \triangleq \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi, d_0)$. The following equation is useful [6]:

$$J(\tilde{\pi}) = J(\pi) + \sum_s d_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a). \tag{1}$$

The complex dependency of $d_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes the right hand side (RHS) difficult to optimize directly. [9] proposed to consider the following surrogate objective:

$$L_\pi(\tilde{\pi}) \triangleq J(\pi) + \sum_s d_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a), \tag{2}$$

where $d_{\tilde{\pi}}$ is replaced with d_π. TRPO introduces the idea of bounding the distribution change via the policy divergence. Specifically, define $D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) \triangleq \max_s D_{\mathrm{TV}}\big(\pi(\cdot|s), \tilde{\pi}(\cdot|s)\big)$, where D_{TV} is the total variation (TV) divergence.

Theorem 2.1. *([9]) Let $\alpha \triangleq D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi})$, then the following bound holds: $J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$, where $\epsilon = \max_{s,a} |A_\pi(s, a)|$.*

Since the TV divergence and the Kullback-Leibler (KL) divergence are related as follows: $D_{\mathrm{TV}}^2(\pi, \tilde{\pi}) \leq \frac{1}{2} D_{\mathrm{KL}}(\pi, \tilde{\pi})$, we then have the following $J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{2\epsilon\gamma}{(1-\gamma)^2} D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi})$, where $D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) \triangleq \max_s D_{\mathrm{KL}}(\pi, \tilde{\pi})$. This forms the

foundation of many policy optimization methods, including TRPO [9] and Proximal Policy Optimization (PPO) [10]. The KL divergence imposed over the consecutive policies, π and $\tilde{\pi}$, is also called the trust region. In practice, TRPO adopts a robust way to take large update steps by using a constraint (rather than a penalty) on the KL divergence, and also considers using the expected KL divergence, instead of the maximum over all states:

$$\max_{\tilde{\pi}} \quad \mathbb{E}_{(s,a)\sim d_\pi}\Big[\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A_\pi(s,a)\Big], \quad \text{s.t.} \quad \mathbb{E}_s\big[D_{\mathrm{KL}}(\pi(\cdot|s),\tilde{\pi}(\cdot|s))\big] \le \delta, \tag{3}$$

where δ is a hyperparameter to specify the trust region. PPO with ratio clipping further simplifies such trust region constraint by leverage ratio clipping and considers the following optimization problem:

$$\max_{\tilde{\pi}} \mathbb{E}_{d_\pi}\big[\min\big(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A_\pi, \mathrm{clip}(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}, 1-\epsilon, 1+\epsilon)A_\pi\big)\big], \quad \text{where } \epsilon \text{ is the clipping hyperparameter.} \tag{4}$$

3 Conservative policy optimization

We show in this section that the trust region constraint over policies can be safely substituted by a *trust-region-free* constraint when we consider a generalized form of the surrogate objective function. We also present a monotonic improvement guarantee for stochastic policies with the generalized surrogate objective and the trust-region-free constraint.

3.1 Optimization of stochastic policies

Definition 3.1. Define state-action function: $A(s,a) \triangleq r(s,a) + \mathbb{E}_{s'\sim P(\cdot|s,a)}[f(s')] - f(s)$, where f is a function $f : \mathcal{S} \to \mathbb{R}$.

Consider updating a stochastic policy from π to $\tilde{\pi}$ via policy gradients. The following proposition from [2] is useful.

Proposition 3.2. *For any stochastic policies $\tilde{\pi}$, π, and the state-action function defined above,*

$$J(\tilde{\pi}) - J(\pi) = \mathbb{E}_{s\sim d_{\tilde{\pi}}(s), a\sim\tilde{\pi}}[A(s,a)] - \mathbb{E}_{s\sim d_\pi(s), a\sim\pi}[A(s,a)]. \tag{5}$$

This proposition generalizes (1) to a broader family of functions $A(s,a)$. One can easily verify that the advantage function of π, i.e., $A_\pi(s,a)$, satisfies the definition with f function as the value function. In this case, (5) is equivalent to (1). Furthermore, this proposition implies that the performance difference between any two policies can be described by their state-action distribution shift, i.e., $d_{\tilde{\pi}}(s)\tilde{\pi}(a) - d_\pi(s)\pi(a)$, weighted by a function $A(s,a)$. In practice, it would be very unlikely to have access to $d_{\tilde{\pi}}(s)$. We thus leverage the same trick used in TRPO to substitute $d_{\tilde{\pi}}(s)$ with the state distribution induced by policy π, i.e., $d_\pi(s)$, and consider the following objective:

$$G_\pi(\tilde{\pi}) \triangleq \mathbb{E}_{s\sim d_\pi(s), a\sim\pi(\cdot|s)}\big[\big(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} - 1\big)A(s,a)\big]. \tag{6}$$

This new objective generalizes the surrogate objective in (3) to any function defined in 3.1. We have the following bound,

Theorem 3.3. *For any two stochastic policies $\tilde{\pi}$ and π, the following bound holds:*

$$J(\tilde{\pi}) - J(\pi) \ge G_\pi(\tilde{\pi}) - \frac{2\gamma}{1-\gamma}(\delta+\epsilon), \quad \textit{where } \delta = \max_{s,a}\left|\big(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} - 1\big)A(s,a)\right| \textit{ and } \epsilon = \left|\sum_a \pi(a|s)A(s,a)\right|.$$

To simplify further analysis, we call $\big(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} - 1\big)$ the *ratio deviation*. This theorem states that the policy improvement gap can be effectively bounded by the maximum product of the ratio deviation $\big(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} - 1\big)$ and the state-action function $A(s,a)$. Moreover, as this product also appears in the definition of $G_\pi(\tilde{\pi})$ in (6), one can thus consider constraining it when optimizing $G_\pi(\tilde{\pi})$. This is what we call the *conservative policy optimization*. We discuss how this conservative policy update can be implemented in practice in the next section.

Theorem 3.3 differs from Theorem 2.1 in two respects. First, it presents a lower bound for the performance improvement with respect to a state-action function defined in Definition 3.1. This $A(s,a)$ does not necessarily need to be the advantage function. Second, instead of imposing the TV constraint over the policies as in Theorem 2.1, the above theorem considers the maximum product of the ratio deviation and the state-action function. According to [11], the TV constraint between any two policies can be equivalently translated into a constraint over ratio deviations. In this sense, TRPO is essentially a special case of Theorem 3.3 by leveraging the TV to bound the ratio deviations under the assumption that the advantage function should be small. Namely, TRPO optimizes the policy regardless of how the magnitude of the advantage might change throughout optimization. Consequently, TRPO may fail to optimize the policy when the advantage function is large in magnitude at some state-action sample even though the TV divergence is well bounded at one iteration.

3.2 Practical policy optimization methods

We now present the practical policy optimization methods for stochastic policies. We first offer intuitions to understand the underlying idea of conservative policy optimization in the above theorems.

Theorem 3.3 is closely related to some existing policy optimization methods. For example, optimizing $G_\pi(\tilde{\pi})$ without any conservative constraint is equivalent to the policy gradient method [12]. Also, there are two ways to impose such conservative constraints: the ratio-conservative and the objective-conservative, which refer to removing the incentive of increasing the ratio deviations (corresponding to PPO [10]) or the objective, respectively, when optimizing the objective function.

Algorithm 1 TREFree algorithm

for iterations $i = 1, 2, ...$ **do**
  for actor $= 1, 2, ..., N$ **do**
    Run policy π in environment
    Compute advantage estimates $\hat{A}_\pi$
  end for
  for epoch $= 1, 2, ..., K$ **do**
    Sample M samples $\{(s, a)\}$ from previous rollouts.
    Compute $\mathcal{L}(\theta) \triangleq \frac{1}{M} \sum_{s,a} \min\Big(\big(\frac{\tilde{\pi}_\theta(a|s)}{\pi(a|s)} - 1\big)\hat{A}_\pi(s, a), \delta\Big)$.
    Maximize $\mathcal{L}(\theta)$ w.r.t θ via gradient descent.
  end for
  $\pi \leftarrow \tilde{\pi}_\theta$.
end for

Non-conservative With no consideration of the conservative policy update principle, one can directly optimize $G_\pi(\tilde{\pi})$ with the state-action function chosen as the advantage function of π. Such policy optimization is performed by *policy gradient methods* [12]: $\max_\theta \quad \mathbb{E}_{(s,a)\sim d_\pi}\big[\frac{\tilde{\pi}_\theta(s,a)}{\pi(s,a)} A_\pi(s, a)\big]$. Theorem 3.3 implies that optimizing the above objective with the same set of sampled data for multiple times (i.e., multi-epoch optimization as in [10]), could incur a significant degradation in policy performance, since the product between the ratio deviation and the advantage can be large. Thus, applying policy gradients for multiple epoch optimization does not guarantee policy improvement [6].

Ratio-conservative One can take into account the conservative policy update rule by constraining the ratio deviations. Specifically, when optimizing $G_\pi(\tilde{\pi})$ with the state-action function as the advantage, one can clip the ratio deviations to remove the incentive of inducing unexpected large deviations (i.e., *ratio-conservative*), as follows $\max_\theta \quad \mathbb{E}_{(s,a)\sim d_\pi}\big[\mathrm{clip}\big(\frac{\tilde{\pi}_\theta(s,a)}{\pi(s,a)} - 1, -\lambda, \lambda\big)A_\pi(s, a)\big]$, where $\lambda > 0$ is a hyper-parameter for ratio deviation clipping. This new objective resembles the ratio clipping objective (4) used in *PPO* [10], which has been shown to be effective in practice, especially with a normalized advantage function. However, such ratio clipping scheme ignores the potential effect of the advantage function on the ratio deviation, and thus can fail to monotonically improve the policy performance when the advantage is large at some state-action point and dominates $\big(\frac{\tilde{\pi}_\theta(s,a)}{\pi(s,a)} - 1\big)A_\pi(s, a)$ for a small ratio deviation. Furthermore, solely bounding the divergence between policies, as used in trust region methods for policy optimization [9, 2], may not be a good option in practice, as it is not sufficient for policy improvement when the advantage function fluctuates greatly across state-action samples.

Objective-conservative We can instead apply the clipping scheme to the objective. Namely, we can clip the objective directly to achieve this conservative update principle, as follows:

$$\max_\theta \quad \mathbb{E}_{(s,a)\sim d_\pi}\Big[\min\Big(\big(\frac{\tilde{\pi}_\theta(s, a)}{\pi(s, a)} - 1\big)A_\pi(s, a), \delta\Big)\Big],$$

where $\delta > 0$ is a hyper-parameter to control the conservativeness when optimizing the objective. The δ operates as a threshold beyond which the objective quantity have no contribution to the optimization. We call it *objective conservative*, and the resulting algorithm *Trust-REgion-Free Policy Optimization (TREFree)* as it is free of any explicit trust region constraint. TREFree is detailed in Algorithm 1.

4 Experiments

In this section, we compare TREFree with TRPO and PPO across the Mujoco continuous control tasks. We adopt the same strategy as in TRPO [9] and PPO [10] to normalize the observations, rewards, and advantages. Specifically, the observations, rewards and advantages are normalized to zero mean and unit variance using a running mean and standard deviation. Both observations and rewards are normalized using a running mean and standard deviation per-timestep over the whole training process, while the advantages are normalized only within a training batch. Moreover, we leverage the actor-critic framework by parameterizing the actor and critic with 2-layered perceptrons, each of which has 64 hidden units and is activated with $\tanh$. Also, the actor and critic share parameters by reusing the first layer of their neural networks, which has been reported to stabilize the training and improve performance [9, 10, 4]. The policy is modeled as a Gaussian distribution, with mean and variance parameterized by the actor neural network.

We now compare TREFree with TRPO and PPO across the Mujoco continuous control tasks with different control complexity [4], We used the publicly available and widely used repository (https://github.com/openai/baselines) as the baseline implementation. For TRPO and PPO, we use the default hyper-parameters given in [9, 10]. We also sweep over

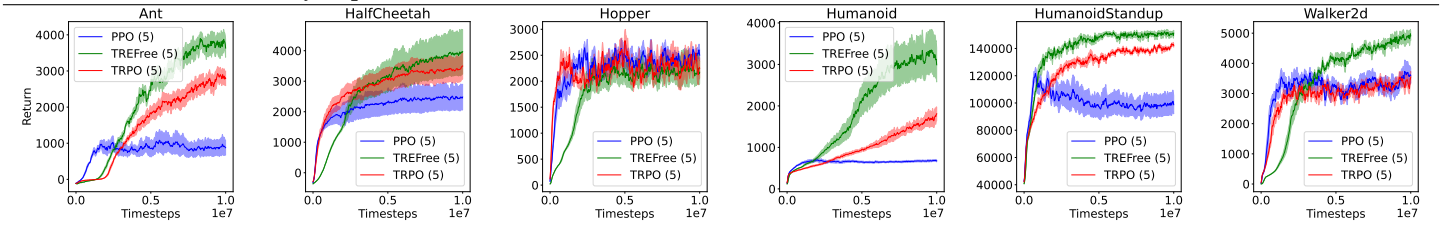


Figure 1: Contrasting TREFree with TRPO and PPO on Mujoco benchmark tasks; training with a margin size 0.01.

the clipping range for PPO and use the best performing value as the baseline. Furthermore, we heuristically decay the learning rate of TREFree and PPO linearly from 0.0003 to 0, as we found this annealing strategy stabilizes training for both PPO and TREFree. We heuristically set δ in TREFree to 0.01 as it is found to perform well across all tasks.

The performance comparison on the Mujoco continuous control tasks is presented in Figure 1. Overall, TREFree performs better than the baselines on all the tasks except Hopper. TREFree outperforms TRPO and PPO by a large margin in terms of the final policy performance on tasks Ant, HalfCheetah, Humanoid, HumanoidStandup and Walker2d. These five Mujoco environments are more complicated than Hopper. The training curves in Figure 1 also show how TREFree often outpaces other baselines in improving policy performance. Though TREFree is outperformed by PPO and TRPO on Hopper, the performance curves of all these methods in this specific environment fluctuates greatly over time, and overlap each other.

We also report the ratio ranges of different methods in Figure 2 to show the underlying differences between TREFree and the baseline methods. The probability ratios $\frac{\tilde{\pi}(a|s)}{\pi(a|s)}$ are an important indicator in TRPO and PPO training as they are closely related to the total variation divergence [10]. Figure 2 shows that the ratios in both TRPO and TREFree are better bounded than in PPO, where the ratios grow without bound. However, TREFree constrains ratios in a dramatically different way from TRPO: TRPO bounds ratios between $[-2, 1]$ (log-scale) in a symmetrical way, while TREFree bounds ratios between $[0, 1]$, which implies that the policy is most often updated to increase the probability at empirical samples. This contrast in ratio ranges suggests that TREFree is fundamentally different from trust region methods.

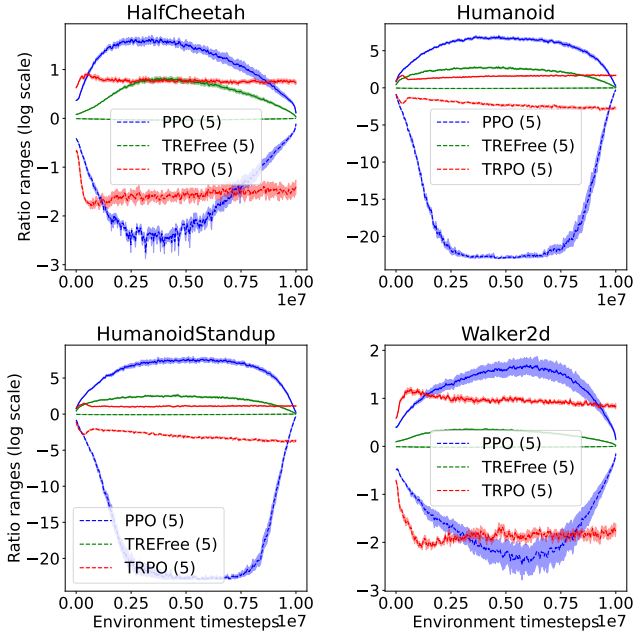


Figure 2: Contrasting ratio ranges.

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[3] Riad Akrour, Joni Pajarinen, Jan Peters, and Gerhard Neumann. Projections for approximate policy iteration algorithms. In *International Conference on Machine Learning*, pages 181–190. PMLR, 2019.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Matteo Hessel, Ivo Danihelka, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, and Hado van Hasselt. Muesli: Combining improvements in policy optimization. *arXiv preprint arXiv:2104.06159*, 2021.

[6] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

[7] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

[8] Fabian Otto, Philipp Becker, Ngo Anh Vien, Hanna Carolin Ziesche, and Gerhard Neumann. Differentiable trust region layers for deep reinforcement learning. *arXiv preprint arXiv:2101.09207*, 2021.

[9] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[11] Mingfei Sun, Vitaly Kurin, Guoqing Liu, Sam Devlin, Tao Qin, Katja Hofmann, and Shimon Whiteson. You may not need ratio clipping in ppo. *arXiv preprint arXiv:2202.00079*, 2022.

[12] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[13] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems*, 30:5279–5288, 2017.

Semi-Supervised Data Generation for Offline Reinforcement Learning via the Occupancy Information Ratio

Wesley A. Suttle
U.S. Army Research Laboratory
Stony Brook University
wesley.suttle@stonybrook.edu

Garrett Warnell
U.S. Army Research Laboratory
garrett.a.warnell.civ@army.mil

Alec Koppel
Amazon
aekoppel@amazon.com

Ji Liu
Stony Brook University
ji.liu@stonybrook.edu

Abstract

Offline reinforcement learning (ORL) methods have shown impressive performance on a range of benchmark tasks, but the problem of dataset generation in ORL is still relatively understudied. While recent work in this area has focused on the use of task-agnostic unsupervised reinforcement learning to generate data for ORL, in many potential applications (e.g., robot navigation), we have *a priori* knowledge of the kinds of downstream tasks the learning agent will have to solve. In this paper, we seek to exploit that knowledge by exploring the application of a recent class of reinforcement learning methods based on the occupancy information ratio (OIR) [11] for ORL dataset generation. Specifically, we hypothesize that OIR can be brought to bear on ORL by providing a new method for obtaining data generation policies anywhere on the spectrum between purely unsupervised (e.g., exploratory) policies and completely task-oriented policies. We experimentally validate our hypothesis by first showing how OIR policies can be used to explicitly control the ratio of task learning to exploration, then comparing the robustness of several offline RL algorithms to downstream task changes on datasets generated using OIR. When used in this way, OIR provides a novel method for training a class of *semi-supervised* data generation policies for ORL that strike a *user-specified* balance between exploration and solving downstream tasks. This provides a useful new tool for researchers interested in studying the best ways to do data generation for ORL.

Keywords: reinforcement learning, unsupervised reinforcement learning, offline reinforcement learning

Acknowledgements

The research of W. Suttle was supported by the U.S. Army Research Laboratory and was accomplished under Cooperative Agreements W911NF-21-2-0127 and W911NF-22-2-0003. The research of J. Liu was supported in part by Army Research Laboratory Cooperative Agreement W911NF-21-2-0098. A. Koppel contributed to this work in his previous role with the Army Research Laboratory in Adelphi, MD 20783.

1 Introduction

Offline reinforcement learning (ORL) – a machine learning paradigm in which reinforcement learning (RL) is used on a set of pre-collected data – has recently enjoyed a great deal of interest within the research community. Broadly speaking, ORL seeks to remove the cost of agent-environment interaction for learning new tasks by leveraging the advances in computational power and data efficiency that have led to successes in the supervised learning community. This is accomplished by first generating and storing a large, static dataset of interactions with a given environment, then labeling each individual transition with a scalar reward, and finally applying ORL algorithms to the relabeled dataset in an attempt to learn an optimal policy. In a robotic exploration task, for example, the data generation phase could consist of logging the position, velocity, and sensor data of a robot as it explores a building. Then, after the data has been collected, an objective such as "go to the southeast corner of the building" is specified, and a scalar reward corresponding to that objective is applied to the data. Finally, an ORL algorithm is applied to the relabelled data to learn a policy to achieve the objective. A large number of ORL techniques have been proposed (see [10] for a comprehensive survey), and many have shown good performance when applied to popular ORL benchmark datasets [3, 5]. However, up until recently these datasets have been generated by using an *ad hoc* variety of data generation policies, such as random policies, optimal policies for some underlying task, expert demonstrations, and others.

This *ad hoc* approach to data collection in offline RL naturally raises the question: what is the best way to generate data for subsequent ORL? As demonstrated in [13], effective data generation policies can be trained by using methods from the field of unsupervised reinforcement learning (URL) [9]. URL comprises a range of techniques for learning policies that maximize *intrinsic* rewards, rather than the *extrinsic* rewards commonly associated with a Markov decision process (MDP). By learning to maximize a certain intrinsic reward, the hope is that the resulting policy will provide a good "warm start" with which to initialize a standard RL algorithm before resuming online training on a downstream task. A prototypical example of an intrinsic reward is the entropy of the state occupancy measure, or state marginal distribution, induced by the current policy, which measures how well the policy covers the state space. The idea is that, when state entropy is maximized, the policy does a good job exploring the environment; this results in a policy that understands how to navigate the environment, which is a valuable precursor skill to have when subsequently learning to solve downstream tasks. The recent work [13] addresses the question of data generation for ORL by leveraging techniques from the URL literature with the Exploratory Data for Offline RL (ExORL) scheme, which generates data for ORL using policies learned using URL. Extensive experiments are provided illustrating the results of using several of the best-known URL algorithms in combination with several prominent ORL algorithms.

Though URL is clearly effective when generating data for ORL, URL methods typically focus solely on intrinsic rewards, i.e., without any downstream tasks in mind. This raises a further question: what data generation methods are best when we have some prior over the type of downstream tasks that we might need to solve using ORL? In the robotic exploration task above, for example, it would be useful if we could incorporate *a priori* knowledge about likely goal locations or dangerous areas that the robot should avoid into the data generation process. This interesting problem is almost unstudied in the literature, though its potential importance is mentioned in the recent [13], where, in addition to purely URL data generation, the authors discuss the utility of "semi-supervised" data generation for subsequent offline training. In this situation, the data generation policy learns using some mixture of extrinsic and intrinsic rewards, potentially improving task transfer on problems where there exists some prior over the class of possible downstream reward functions. Importantly, this type of semi-supervised data generation may also result in datasets that are more robust to perturbations of the downstream reward function, as discussed in the experiments section below.

Orthogonal to the ORL and URL literatures, the recent work [11] developed a new tool, called the *occupancy information ratio* (OIR), for addressing the trade-off between exploration and task learning in RL. In the present paper, we hypothesize that the inherently semi-supervised OIR policy search techniques proposed in [11] provide a promising new framework for training data generation policies that allow the user to implicitly specify the desired balance between exploration and task prioritization. OIR is semi-supervised in the sense that it explicitly balances the desire to solve a representative *extrinsic* task drawn from some class of possible downstream tasks with the *intrinsic* goal of maximizing exploration. In addition to enjoying strong theoretical guarantees and promising empirical performance, we aim to show that policies trained using OIR may be used to obtain datasets for ORL with a user-controlled level of robustness to changes in the downstream reward function. [1]

In this paper, we explore the application of the OIR and entropy maximization methods developed in [11] to data generation for ORL. First, we describe the problem setting underlying the OIR optimization problem. Second, we present the entropy gradient and OIR policy gradient theorems for this setting, then provide an actor-critic algorithm for minimizing the OIR, called Information-Directed Actor-Critic (IDAC). Next, we describe how one might use OIR methods to learn semi-supervised data generation policies for ORL that strike a user-specified balance between exploration and

[1] As an aside, OIR methods rely on the entropy gradient theorem (Theorem 3.1) given below; as discussed in [11], this theorem provides a simple expression for the gradient of the state entropy of a policy that can be used to develop straightforward policy search algorithms for entropy maximization. This is likely of independent interest to the URL and ORL communities, since it provides additional tools for developing unsupervised and semi-supervised data generation methods.

task prioritization. Finally, we provide experiments that (i) illustrate how OIR policies can be used to explicitly control the ratio of task learning to exploration, and (ii) compare the effect of varying this ratio during OIR data generation on the robustness of several offline RL algorithms to downstream task changes. Our results illustrate that OIR methods provide a new class of semi-supervised data generation policies on the spectrum between purely unsupervised policies and completely task-oriented policies, yielding a flexible new tool for ORL.

2 Problem Formulation

In this section we describe our problem setting and formulate the OIR objective. We first define an underlying MDP, then formulate the OIR as an objective to be optimized over it.

Markov Decision Processes. Consider an average-cost MDP described by the tuple $(\mathcal{S}, \mathcal{A}, p, c)$, where $\mathcal{S}$ is the finite state space, $\mathcal{A}$ is the finite action space, $p : \mathcal{S} \times \mathcal{A} \to \mathcal{D}(\mathcal{S})$ is the transition probability kernel mapping state-action pairs to distributions over the state space, and $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ is the cost function mapping state-action pairs to positive scalars. In this setting, at time-step t, the agent is in state s_t, chooses an action a_t according to a policy $\pi : \mathcal{S} \to \mathcal{D}(\mathcal{A})$ mapping states to distributions over $\mathcal{A}$, incurs cost $c(s_t, a_t)$, and then the system transitions into a new state $s_{t+1} \sim p(\cdot|s_t, a_t)$. Since we are primarily interested in policy gradient methods, we give the following definitions with respect to a parameterized family $\{\pi_\theta : \mathcal{S} \to \mathcal{D}(\mathcal{A})\}_{\theta\in\Theta}$ of policies, where $\Theta \subset \mathbb{R}^d$ is some set of permissible policy parameters. Note that analogous definitions apply to any policy π. For any $\theta \in \Theta$, let $d_\theta(s) = \lim_{t\to\infty} P(s_t = s \mid \pi_\theta)$ denote the steady-state occupancy measure over $\mathcal{S}$ induced by π_θ, which we assume to be independent of the initial start-state. Furthermore, let $J(\theta) = \sum_s d_\theta(s) \sum_a \pi_\theta(a|s)c(s, a)$ denote the long-run average cost of using policy π_θ. Finally, given θ, define the entropy of the state occupancy measure induced by π_θ to be $H(d_\theta) = -\sum_s d_\theta(s) \log d_\theta(s)$. This quantity is a measurement of how well π_θ covers the state space $\mathcal{S}$ in the long run.

Occupancy Information Ratio. In this paper we consider the OIR objective $\rho(\theta) = \frac{J(\theta)}{\kappa+H(d_\theta)}$, where $\kappa > -\min_\theta H(d_\theta)$ is user-specified constant that, at an intuitive level, allows the user to control the ratio of task learning to exploration (see Remark 1). When π_θ and the system dynamics are such that $H(d_\theta) = 0$ is possible, restricting $\kappa > 0$ ensures that $\rho(\theta)$ is well-defined. Under suitable conditions, however, $H(d_\theta) > 0$ is guaranteed, so $\kappa = -\min_\theta H(d_\theta)$ is allowed. Given an MDP $(\mathcal{S}, \mathcal{A}, p, c)$, our goal is to find a policy parameter θ^* such that π_{θ^*} minimizes $\rho(\theta)$ *over the MDP*, i.e., subject to its costs and dynamics.

Remark 1. Since κ scales the relative importance of $H(d_\theta)$ in $\rho(\theta)$, it can be viewed (and used) as a regularizer. When minimizing a function $f(x)$, one frequently considers a regularized objective function $f(x) + \kappa \|x\|$, where κ is some positive scalar. Here, the larger κ becomes, the more important the regularization term becomes with respect to the objective function. In contrast, for $\rho(\theta)$, the relative importance of the entropy term actually *diminishes* as κ becomes larger: when κ is small, even minor changes in the value of $H(d_\theta)$ can have a large effect on the value of $\rho(\theta)$; when κ is large, on the other hand, even significant perturbations of the value of $H(d_\theta)$ have little effect on the value of $\rho(\theta)$. As we will see in Figure 1, for the problem of using OIR for subsequent ORL, κ controls the extent to which the learner cares about learning an optimal policy versus finding maximally-diverse data.

Remark 2. Though we stipulated that $\kappa > -\min_\theta H(d_\theta)$ in the definition of the OIR above, letting $\kappa < -\max_\theta H(d_\theta)$ has a very important interpretation as well. When $\kappa < -\max_\theta H(d_\theta)$ and $J(\theta) \geq 0$, for all $\theta \in \Theta$, clearly the OIR $\rho(\theta)$ will always be non-positive. Because of this, minimizing the OIR will in fact minimize the ratio of $-J(\theta)$ to the absolute value $|\kappa + H(d_\theta)|$. This means that the expected cost $J(\theta)$ of the underlying MDP is instead treated as an expected reward to be maximized, and any algorithm for minimizing the OIR will therefore balance maximizing the reward $J(\theta)$ with maximizing the shifted entropy $|\kappa + H(d_\theta)|$. This allows the OIR framework to accommodate rewards by simply replacing the cost function c in the MDP with a reward function r, and choosing $\kappa < -\max_\theta H(d_\theta)$.

3 Actor-Critic for the OIR

In this section we present the Information-Directed Actor-Critic (IDAC) algorithm, which minimizes the OIR, $\rho(\theta)$. IDAC will be the OIR-based tool that we use in what follows for ORL data-generation. Throughout this section, we will assume that an average-cost MDP $(\mathcal{S}, \mathcal{A}, p, c)$ is fixed, though we note that the average-reward setting can be accommodated with minor changes by Remark 2. We start by first presenting the policy gradient results from [11] upon which IDAC is based.

Theorem 3.1. *Let an MDP $(\mathcal{S}, \mathcal{A}, p, c)$ and a differentiable parametrized policy class $\{\pi_\theta\}_{\theta\in\Theta}$ be given. Fix a policy parameter iterate θ_t at time-step t. The gradient $\nabla H(d_\theta)|_{\theta=\theta_t}$ with respect to the policy parameters θ of the state occupancy measure entropy $H(d_\theta)$, evaluated at $\theta = \theta_t$, satisfies $\nabla H(d_\theta)\big|_{\theta=\theta_t} = \mathbb{E}_{\pi_{\theta_t}}[-\log d_{\theta_t}(s) - H(d_{\theta_t})\nabla \log \pi_{\theta_t}(a|s)]$.*

With Theorem 3.1 in hand, we have the following OIR policy gradient theorem:

Theorem 3.2. *Let an MDP $(\mathcal{S}, \mathcal{A}, p, c)$, a differentiable parametrized policy class $\{\pi_\theta\}_{\theta\in\Theta}$, and a constant $\kappa \geq 0$ be given. Fix a policy parameter iterate θ_t at time-step t. The gradient $\nabla\rho(\theta_t)$ with respect to the policy parameters θ of the OIR $\rho(\theta)$, evaluated at*

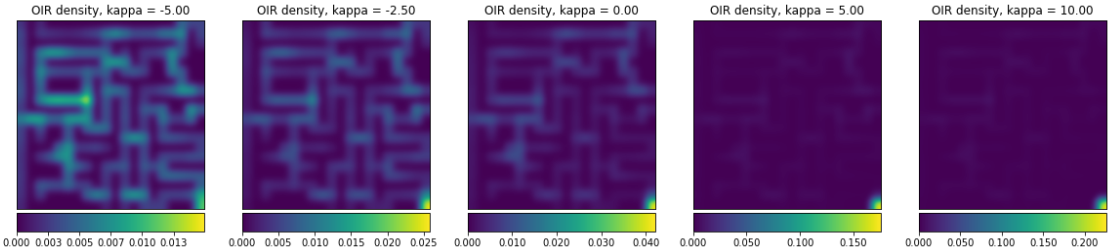


Figure 1: The state occupancy measures of optimal OIR policies for various values of κ on a maze environment. This is a cost-based environment, so values $\kappa > -\min_\theta H(d_\theta)$ were chosen. The state occupancy measure represents the frequency with which the agent visits the various regions of the state space. Larger numerical values represent higher visitation frequencies. The start state is in the upper left-hand corner, while the goal state is in the bottom right. Episodes are fixed-length at 1000 timesteps. Agent can move up, down, left, right, or stay. Agent receives costs of 1 for occupying the goal state, 10 for occupying a non-goal state, and 100 for choosing an action that runs into a wall. The figure illustrates how the ratio of task completion (i.e., reaching the goal state) to exploration of OIR policies changes as a function of κ in the cost setting: for small values of κ, OIR policies tend to be highly exploratory, paying less attention to reaching the goal state; as κ increases, OIR policies explore less and concentrate more on reaching the goal state.

$\theta = \theta_t$, satisfies $\nabla\rho(\theta_t) = \mathbb{E}_{\pi_{\theta_t}}\left[\frac{\delta_t^J\left(\kappa + H(d_{\theta_t})\right) - J(\theta_t)\delta_t^H}{\left[\kappa + H(d_{\theta_t})\right]^2}\psi_t\right]$, where we define the temporal value difference $\delta_t^J = c(s,a) - J(\theta_t)$, the temporal entropy difference $\delta_t^H = -\log d_{\theta_t}(s) - H(d_{\theta_t})$, and $\psi_t = \nabla \log \pi_{\theta_t}(a|s)$.

Information-Directed Actor-Critic. IDAC is an OIR-based method providing a new semi-supervised data-generation technique for ORL. The algorithm is a variant of the classic actor-critic scheme [7, 2] with two critics: the standard critic corresponding to average cost $J(\theta)$, and an entropy critic corresponding to the *shadow MDPs* $(\mathcal{S}, \mathcal{A}, p, r_t)$, $t \geq 0$, where $r_t(s,a) = -\log d_{\theta_t}(s)$. The role of the shadow MDPs is to enable us to estimate $H(d_{\theta_t})$ and its gradient, at each timestep t. We assume access to an oracle, DENSITYESTIMATOR, which returns the state occupancy measure $d_\theta = \text{DENSITYESTIMATOR}(\theta)$ induced by the policy π_θ, for any $\theta \in \Theta$. In our experiments we used a counting-based density estimator, though this can be replaced with kernel density estimation or other techniques in practice. At timestep t, the algorithm computes two different TD errors: one corresponds to the critic for the MDP $(\mathcal{S}, \mathcal{A}, p, c)$, while the other corresponds to the critic for the shadow MDP $(\mathcal{S}, \mathcal{A}, p, r_t)$. Note that the entropy critic TD error computation requires a call to DENSITYESTIMATOR. Next, the cost and entropy critic TD errors are used to update their respective critics, which are in turn combined to perform the actor update. Pseudocode for IDAC is provided in Algorithm 1.

Algorithm 1 IDAC

Initialization: Set rollout length K, stepsizes $\{\alpha_t\}, \{\beta_t\}, \{\tau_t\}$, policy class $\{\pi_\theta\}_{\theta\in\Theta}$, critic class $\{v_\omega\}_{\omega\in\Omega}$, constant $\kappa \geq 0$. Sample $s_0, \theta_0, \omega_0^J, \omega_0^H$, select μ_{-1}^H, $\mu_{-1}^J > 0$, and set $t \leftarrow 0$.
repeat
  Generate trajectory $\{(s_i, a_i)\}_{i=1,\ldots,K}$ using π_{θ_t}
  $\mu_t^J = (1-\tau)\mu_{t-1}^J + \tau\frac{1}{K}\sum_{i=1}^K c(s_i, a_i)$
  $d_{\theta_t} = \text{DENSITYESTIMATOR}(\theta_t)$
  $\mu_t^H = (1-\tau)\mu_{t-1}^H + \tau\frac{1}{K}\sum_{i=1}^K \left(-\log d_{\theta_t}(s_i)\right)$
  for $i = 1, \ldots, K$ **do**
    Set $v_{\omega_t^J}(s_{K+1}) = v_{\omega_t^H}(s_{K+1}) = 0$
    $\delta_i^J = c(s_i, a_i) - \mu_t^J + v_{\omega_t^J}(s_{i+1}) - v_{\omega_t^J}(s_i)$
    $\delta_i^H = -\log d_{\theta_t}(s_i) - \mu_t^H + v_{\omega_t^H}(s_{i+1}) - v_{\omega_t^H}(s_i)$
    $\psi_i = \nabla \log \pi_{\theta_t}(a_i|s_i)$
  end for
  $\omega_{t+1}^J = \omega_t^J + \alpha\frac{1}{K}\sum_{i=1}^K \delta_i^J \nabla v_{\omega_t^J}(s_i)$
  $\omega_{t+1}^H = \omega_t^H + \alpha\frac{1}{K}\sum_{i=1}^K \delta_i^H \nabla v_{\omega_t^H}(s_i)$
  $\widehat{\nabla\rho(\theta_t)} = \frac{1}{\left[\kappa+\mu_t^H\right]^2}\frac{1}{K}\sum_{i=1}^K \left[\delta_i^J\left(\kappa + \mu_t^H\right) - \mu_t^J\delta_i^H\right]\psi_i$
  $\theta_{t+1} = \theta_t - \beta\widehat{\nabla\rho(\theta_t)}$
  $t \leftarrow t+1$
until convergence

4 Semi-Supervised OIR Data Generation for Offline Reinforcement Learning

In this section we explore the application of OIR to data generation for ORL. We first describe how OIR methods can be used to obtain semi-supervised data generation policies for ORL that lie on the spectrum between purely unsupervised and completely task-oriented policies. We illustrate this point with Figures 1 and 2, which show how the OIR parameter κ can be used to explicitly control the ratio of task learning to exploration. Second, we experimentally evaluate the effect of varying this ratio during data generation on the robustness of offline algorithms to downstream task changes. Figure 3 illustrates that, as we vary the downstream task over OIR-generated offline datasets, the performance of ORL algorithms on these datasets depends greatly on the value of κ, since it controls the ratio of task learning to exploration. For ease of experimentation and since our focus was on data-generation instead of IDAC performance, the global optimal OIR policies in this section were obtained using the concave programming techniques described in [11] instead of IDAC. For an in-depth discussion of how IDAC has been used to solve these and more complicated problems, see [11].

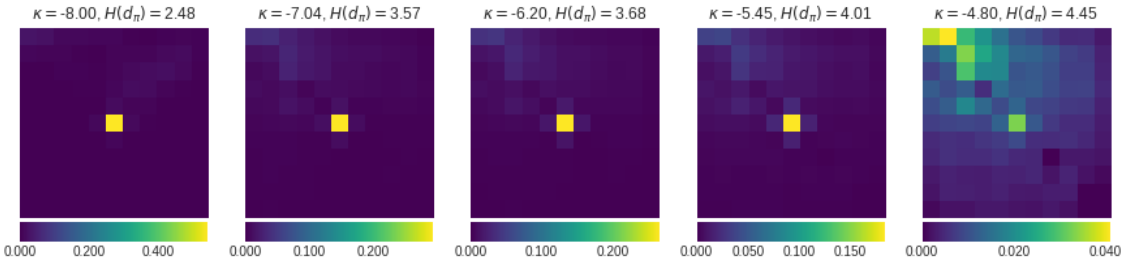


Figure 2: The state occupancy measures of optimal OIR policies for various values of κ on a gridworld environment. This is a reward-based environment, so values $\kappa < -\max_\theta H(d_\theta)$ were chosen. The start state is in the upper left-hand corner, while the goal state is in the center. Agents receive a large reward for reaching the goal state, a small bonus for choosing an action that moves it closer to the goal state, and 0 otherwise. The episode ends once the agent enters the goal state. The figure illustrates how the ratio of task completion to exploration of OIR policies changes as a function of κ in the reward setting: for very negative values of κ, OIR policies tend to be highly task-oriented, performing significantly less exploration; as κ increases, OIR policies become increasingly exploration-oriented, focusing less on task completion.

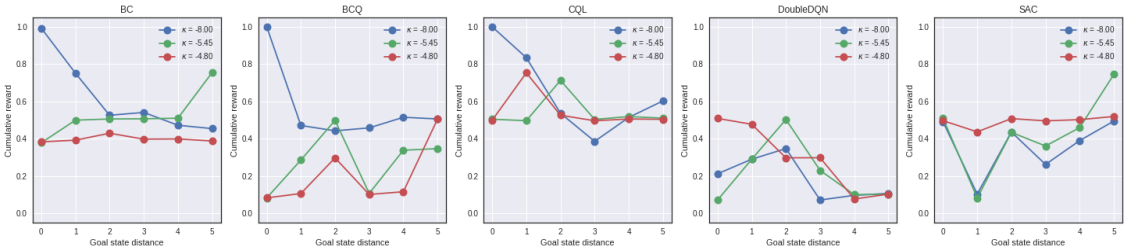


Figure 3: Performance of several offline algorithms on data generated using optimal OIR policies for different values of κ and different downstream tasks on the gridworld environment from Figure 2. Plots are of normalized cumulative reward as a function of distance of the downstream goal state from the original goal state. The OIR data generation policies were obtained for the original problem depicted in Figure 2 with goal state in the center, then used to general offline datasets. We obtained new offline datasets from these by varying the distance of the downstream goal state from the original goal state. We subsequently trained behavioral cloning (BC) as a baseline, then compared the offline RL algorithms batch-constrained deep Q-learning (BCQ) [4] and conservative Q-learning (CQL) [8], as well as offline versions of the off-policy double deep Q-learning (DoubleDQN) [12] and soft actor-critic (SAC) [6] algorithms. For offline training we used the `d3rlpy` library [1]. The figure illustrates how performance of the offline algorithms changes as the ratio of task completion to exploration – controlled by choice of κ – varies. When $\kappa = -8.0$, the OIR policy is primarily focused on reaching the central square. BC and the ORL algorithms BCQ and CQL consequently perform well when the downstream goal state is in the center, while performance decreases as distance increases. When $\kappa = -4.8$, the OIR policy is primarily exploratory. Here BC, CQL, and SAC perform roughly the same at all goal state distances. When $\kappa = -5.45$, the OIR policy is more evenly balancing reaching the goal with exploration. As a result, the corresponding ORL policies appear to exhibit behavior somewhere between these two extremes. It is interesting to note that: BCQ does poorly on the more exploratory datasets, while CQL tends to do better; DQN's performance degrades with increasing distance, while SAC's actually improves. This suggests that fixed OIR datasets may effect offline algorithms differently and merits further study.

References

[1] d3rlpy. https://d3rlpy.readthedocs.io. Accessed: 2022-05-10.

[2] Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[3] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[4] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[5] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. RL unplugged: Benchmarks for offline reinforcement learning. 2020.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[7] Vijaymohan Konda. *Actor-Critic Algorithms*. PhD thesis, MIT, 2002.

[8] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[9] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. URLB: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*, 2021.

[10] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[11] Wesley A Suttle, Alec Koppel, and Ji Liu. Occupancy information ratio: Infinite-horizon, information-directed, parameterized policy search. *arXiv preprint arXiv:2201.08832*, 2022.

[12] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[13] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

The Quest for a Common Model of the Intelligent Decision Maker

Richard S. Sutton
DeepMind and the University of Alberta
Edmonton, Alberta, Canada
rich@richsutton.com

Abstract

The premise of the *Multi-disciplinary Conference on Reinforcement Learning and Decision Making* is that multiple disciplines share an interest in goal-directed decision making over time. The idea of this paper is to sharpen and deepen this premise by proposing a perspective on the decision maker that is substantive and widely held across psychology, artificial intelligence, economics, control theory, and neuroscience, which I call the *common model of the intelligent agent*. The common model does not include anything specific to any organism, world, or application domain. The common model does include aspects of the decision maker's interaction with its world (there must be input and output, and a goal) and internal components of the decision maker (for perception, decision-making, internal evaluation, and a world model). I identify these aspects and components, note that they are given different names in different disciplines but refer essentially to the same ideas, and discuss the challenges and benefits of devising a neutral terminology that can be used across disciplines. It is time to recognize and build on the convergence of multiple diverse disciplines on a substantive common model of the intelligent agent.

Keywords: decision making, multi-disciplinary, common model, intelligence

Acknowledgements

The author gratefully acknowledges that his perspectives on these topics have been immeasurably improved by discussions with Joseph Modayil, Adam White, Marlos Machado, Andy Barto, Satinder Singh, David Silver, Doina Precup, Ben Van Roy, Patrick Pilarski, Alex Kearney, Elliot Ludvig, and James Kehoe. For any remaining naïvete or hyperbole, the author alone is responsible. This work was supported by funding from the Alberta Machine Intelligence Institute (Amii), DeepMind, and CIFAR.

This paper appeared as an extended abstract at the fifth *Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, held in Providence, Rhode Island, on June 8-11, 2022.

1 The Quest

The premise of the Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM) is that there is value in all the disciplines interested in "learning and decision making over time to achieve a goal" coming together and sharing perspectives. The natural sciences of psychology, neuroscience, and ethology, the engineering sciences of artificial intelligence, optimal control theory, and operations research, and the social sciences of economics and anthropology—all focus in part on intelligent decision makers. The perspectives of the various disciplines are different, but they have common elements. One cross-disciplinary goal is to identify the common core—those aspects of the decision maker that are common to all or many of disciplines. To the extent that such a common model of the decision maker can be established, the exchange of ideas and results may be facilitated, progress may be faster, and the understanding gained may be more fundamental and longer lasting.

The quest for a common model of the decision maker is not new. One measure of its current vitality is the success of cross-disciplinary meetings such as RLDM and the Conference on Neural Information Processing Systems (NeurIPS), and journals such as Neural Computation, Biological Cybernetics, and Adaptive Behavior. There have been many scientific insights gained from cross-disciplinary interactions, such as the now-widespread use of Bayesian methods in psychology, the reward-prediction error interpretation of dopamine in neuroscience (Schultz, Dayan & Montague 1997), and the long-standing use of the neural-network metaphor in machine learning. Although significant relationships between many of these disciplines are as old the disciplines themselves, they are still far from settled. To find commonalities among disciplines, or really even within one discipline, one must overlook many disagreements. We must be selective. We must look for the big picture without expectation that there will be no exceptions.

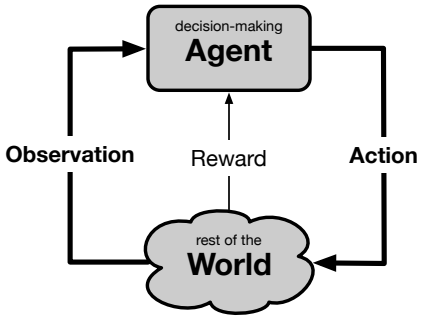
In this short paper I hope to advance the quest for a model of the intelligent decision-maker that resonates across disciplines in the following small ways. First, I explicitly identify the quest as distinct from fruitful cross-disciplinary interaction. Second, I highlight the formulation of goals as the maximization of a cumulative numerical signal as highly inter-disciplinary. Third, I highlight a particular internal structure of the decision maker—as four principal components interacting in a specific way—as already being common to multiple disciplines. Finally, I highlight terminological differences that obscure the commonalities between fields and offer terms that instead encourage the multi-disciplinary mindset.

2 Interface Terminology

The decision-maker makes its decisions over time, which may be divided into discrete steps at each of which new information is received and a decision is made that may affect the information that is received later. That is, there is an interaction over time with signals exchanged. What terminology shall we use for the signals and for the entities exchanging them? In psychology, the decision maker is the "organism," receiving "stimuli" and sending "responses" to its "environment." In control theory, the decision-maker is termed the "controller," receiving "state" and sending "control signals" to the "plant." Other fields use still other terms, but these illustrate the challenge—to find terms that do not prejiduce the reader towards one field or the other, but rather facilitate thinking that crosses disciplinary boundaries.

A good way to start in establishing terminology is to clarify the ideas that the words are meant to convey—and not convey. The latter is particularly significant for us, as we do not want our terms to evoke intuitions that are specific to any particular discipline. For example, calling the decision-maker an "organism" would interfere with thinking of it as a machine, as we would in artificial intelligence. The essence of a decision maker is that it acts with some autonomy, is sensitive to its input, and has an purposeful effect on its future input. A good word for this is *agent*, for which my dictionary offers the following definition: "a person or thing that takes an active role or produces a specified effect." The word is commonly used this way within artificial intelligence for decision-makers that could be either machines or people. The term "agent" is also preferable to "decision maker" because it connotes autonomy and purposiveness.

What then does the decision-making agent interact with? One answer is that it interacts with everything that is not the agent, which might be termed its "environment" or "world." Either term is good for our purposes—not strongly linked to a specific discipline—but let's go with *world* for this paper just because it is simpler and shorter while also being evocative in a way not linked to any specific discipline. To complete the picture of the agent–world interaction (right) we must give names to the signals passing in each direction. Without belaboring the points, it is natural to say that the agent takes *actions*, and receives "sensations" or "observations." Let us use *observations* because it is an established term for this purpose and avoids metaphysical discussions about whether a machine can have "sensation." In its standard usage, "observations" means information about the state of the world that is potentially incomplete. Reward will be discussed shortly.



3 Ground Rules

The preceding discussion exemplifies the ground rules and the steps that I attempt to follow regarding terminology: 1) identify the discipline-independent idea that the word is meant to connote, and 2) find a commonsense word that captures that meaning without being unduly biased toward one discipline or another. And finally: 3) repeat the first two steps until consensus has been found across disciplines.

A second kind of ground rule that I follow is not about terminology, but about content. As we attempt to develop a common model of decision making, what aspects should we include and which exclude? The rule I attempt to follow is to include the intersection of the fields rather than the union. That is, in order for a aspect to be included it is not enough for it arise in just one of the fields. There must be at least be an argument that it is pertinent to many if not all of them. The aspects of the common model must be general to all decision making over time to achieve a goal.

Moreover, there should be nothing in the common model that is specific to our world—for example, nothing about vision, objects, 3D space, other agents, or language. Simple examples of what we exclude are all the things that make people special and distinct from other animals, or all the special knowledge that animals have built into them by evolution to fit to their ecological niches. These things are vitally important topics in anthropology and ethology, and genuinely improve our understanding of naturally intelligent systems, but have no place in the common model. Similarly, we exclude all the domain knowledge that is build into artificial intelligent systems by their human designers in order to make a more useful application that requires less extensive training. All of these things are important within their discipline separately, but are not relevant to a common model intended to apply broadly across disciplines.

A common model of decision making could have utility beyond facilitating cross-disciplinary interaction. It is relatively easy to see the benefits within each each discipline, when they occur, of a common model because the existing disciplines and their value are already established. Understanding natural systems has clear scientific value. Creating more useful engineered artifacts has clear utilitarian value. But there is also scientific value in understanding the process of intelligent decision making independent of its relationship to natural decision making, and independent of the practical utility of its artifacts. I think so. It is not a currently established science, but one day perhaps there will be established a science of decision making independent of biology and of its engineering applications.

4 Additive Rewards

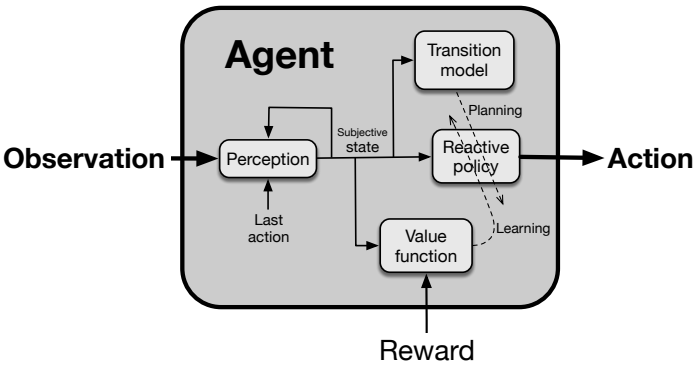
We turn now to the decision-making agent's goal. Today most disciplines formulate the agent's goal in terms of a scalar signal generated outside the agent's direct control, and thus we place its generation, formally, in the world. In the general case this signal arrives on every time step and the goal is to maximize its sum. Such *additive rewards* may be used to formulate the goal as a delay-discounted sum, as a sum over a finite horizon, or in terms of average reward per time step. Many other names have been used for reward, such as "payoff," "gain," "utility," and, when minimized rather than maximized, "costs." Formulation in terms of "costs" and minimization is formally equivalent if the costs are allowed to be negative, but just a little more complicated. A simpler but still popular notion of goal is as a state of the world to be reached. The goal-state formulation is sometimes adequate, but less general than additive rewards. For example, it cannot handle goals of maintenance, or specify how time-to-goal and uncertainty are traded off, all of which are easily handled with additive frameworks.

Additive rewards have a long inter-disciplinary history. In psychology "reward" was used primarily for external objects or events that were pleasing to the animal, and even if that pleasing-ness was derived from an association of the object with something that was rewarding in a more basic way—a "primary reinforcer." Today's usage of "reward" in operations research, economics, and artificial intelligence is restricted to that more primary signal, and is also more explicitly a received signal rather than being tied to external objects or events. That usage appears to have been established with the development of Markov decision processes within optimal control and operations research in the 1960s. It is now standard in an impressively wide range of disciplines, including economics, reinforcement learning, neuroscience, psychology, operations research, and multiple subfields of artificial intelligence.

5 Standard Components of the Decision-Making Agent

We turn now to the internal structure of the agent. This may be the topic on which it most difficult to obtain multi-disciplinary consensus. I have opted to include in the agent only the most essential elements for which there is widespread (albeit not universal) agreement within and across disciplines, and to describe them only in general terms. Even so, I hope that my selections and descriptions may be useful for communication across disciplines and that even those who disagree with them will see my choices as representative of a plurality of researchers in the field. Even if a view turns out to be wrong, it can still be a contribution to make it clear.

The proposed common model of the internal structure of the agent has four principal components—*perception*, *reactive policy*, *value function*, and *transition model*—shown in the figure below as boxes interconnected by a central signal, the "subjective state." Although the four components are each common to many disciplines, it is rare for all to be present in fully developed form within a single agent, and of course specific agents may have additional components beyond these four. Let us consider each component of the common model in turn.



The *perception* component processes the stream of observations and actions to produce the *subjective state*, a summary of the agent–world interaction so far that is useful for selecting action (the reactive policy), for predicting future reward (the value function), and for predicting future subjective states (the transition model). The state is subjective in that it is relative to the agent's observations and actions and may not correspond to the actual internal workings of the world. Often the construction of subjective state is a fixed pre-processing step, in which case the agent is assumed to receive the subjective state directly as an observation. For example, in Atari game playing, the subjective state may be the last four video frames together with the most recent action. In Bayesian approaches the subjective state *does* have a relationship to the internal workings of the world: the subjective state is intended to approximate the probability distribution over the latent states that the world uses internally (as in Partially Observable Markov Decision Processes, or Kalman filters). In predictive state methods (e.g., Littman et al. 2002) the subjective state is a set of predictions. In deep learning, the subjective state is typically the transient activity of a recurrent artificial neural network (e.g., Hochreiter & Schmidhuber, 1997). In control theory, the computations of the perception component are often referred to as 'state identification," or "state estimation." In general (and in many of these examples) the perception component should have a *recursive form* allowing the subjective state to be computed efficiently from the preceding subjective state, the most recent observation, and the most recent action, without revisiting the lengthy history of prior observations and actions. Processing in the perception component needs to be *fast*, that is, completed well within the time interval between successive time steps of the agent–world interaction.

The perception component is not just about short-term memory, but also about representation. For example, it includes any domain-dependent feature construction processes such as are commonly used in classical pattern recognition systems. More generally, when a person observes a complex visual image and re-represents it in terms of objects and relationships, or observes a chess position and then re-represents it in terms of threats and pawn structure—these things too are done in the perception component. The perception component converts the stream of observation–action events into a summary representation that is relevant to the current moment. Such examples suggest the ways in which the term "perception" is appropriate here: one dictionary definition of perception is "the neurophysiological processes, including memory, by which an organism becomes aware of and interprets external stimuli." Generalizing that beyond biology, perception is an entirely appropriate name for the recursive updating of the agent's state representation.

The *reactive policy* component of the common model maps the subjective state to an action. Like perception, the reactive policy must be fast; the speed of perception and the reactive policy together determine the overall reaction time of the agent. Sometimes perception and the reactive policy are treated together, as in end-to-end learning, but still both functions are generally identifiable. Separating overall action generation into these two parts—perception and policy—is common in many disciplines. In engineering it is common to assume that perception is given, not learned, and not even part of the agent. Engineering clearly has the idea of a reactive policy, usually computed or derived analytically. Artificial intelligence systems more often assume that substantial processing is possible before acting (e.g., chess-playing programs). Classical psychology has its Stimulus-Response associations. In psychology it is common to view perception as some that supports but precedes action and can be studied independently of its effects on particular actions.

The *value function* component of the common model maps the subjective state (or state–action pair) to a scalar assessment of its desirability, operationally defined as the expected cumulative reward that follows it. This assessment is fast and independent of its justification (like an intuition) but may be based on long experience (or even on expert design or generations of evolution) or from extensive computations that are effectively stored or cached. In either way, the assessments are such that can be called up quickly to support processes that alter the reactive policy.

Value functions have a very broad multi-disciplinary history. They were first extensively developed, in optimal control, operations research, and dynamic programming, as "cost-to-go" functions satisfying Bellman equations and, in continuous time, Hamilton-Jacobi-Bellman equations. In economics they are called "utility functions" and appeared initially as inputs to economic calculations capturing consumer preferences. In psychology, they are related to old ideas of secondary reinforcers and to newer ideas of reward prediction. The terminology of value functions came initially from dynamic programming and was then taken up within reinforcement learning, where value functions are used extensively as critical components of the theory and of almost all learning methods. Artificial intelligence and operations

research also have a long history of search methods that operate on approximate cost-to-go functions. In neuroscience, the error in the value function, or "reward-prediction error," has been hypothesized as an interpretation of the phasic signal of the neurotransmitter dopamine (Schultz, Dayan & Montague 1997). The theory has explained a wide range of puzzling empirical results and has been extremely influential (e.g., see Glimcher 2011).

The fourth and last component of the common model of the agent, the *transition model*, takes in states and predicts what next states would result if various actions were taken. The transition model could be termed a "world model," but this would overstate its role, as much of what constitutes a model of the world is actually in the perception component and its state representations. The transition model is used to simulate the effects of various actions and, with the help of the value function, evaluate the possible outcomes and change the reactive policy to favor actions with predicted good outcomes and disfavor actions with predicted poor outcomes. When this process is done without actually taking the actions, possibly even without visiting the states, then it is appropriate to call it "planning," or even "reason."

Transition models play important roles in many disciplines. In psychology, internal models of the world such as provided by the transition model together with perception have been prominent models of thought since the work of Kenneth Craik (1943) and Edward Tolman (1948). Tolman showed that the results of many rat experiments could be interpreted as the rat forming and using a "cognitive map" (i.e., transition model) of its world. In neuroscience, the hippocampus is now widely seen as providing such a map, and theorists including Karl Friston and Jeff Hawkins have extensively developed brain theories based on this idea. More recently, in psychology, Daniel Kahneman (2011) has proposed the idea of two mental systems, System 1 and System 2. The first three components of the common model of the agent fit squarely in System 1, and model-based prediction and planning are naturally interpretted as important parts of System 2. In control theory and operations research it has always been common to use transition models of many forms, including differential equation models, difference equations, and Markov models. Models of the world are a common assumption in classical artificial intelligence, such as in single-agent and game-tree search. In reinforcement learning, model-based approaches in which the model is learned have long been proposed and are now beginning to be effective in large applications (e.g., Schrittwieser et al. 2020). In modern deep learning, prominent researchers such as Yoshua Bengio, Yann LeCun, and Jürgen Schmidhuber all place predictive models of the world at the center of their theories of the mind.

6 Limitations and Assessment

This has been a short treatment of the quest for a common model of the intelligent agent. All the points briefly made here deserve elaboration and a deeper treatment of the history. Nevertheless, the main points seem clear. We have presented a prominent candidate for the common model. Its external interface—in terms of agent, world, action, observation, and reward—is general, natural, and widely assumed in both natural sciences and engineering. The four internal components of the agent also each have a long and broad multi-disciplinary tradition.

The proposed common model could be criticized because of what it leaves out. For example, there is no explicit role in it for predictions of observations other than reward, and there is no treatment of exploration, curiosity, or intrinsic motivation. And all of the the four components must involve learning, but here we have described learning only in the reactive policy, and that only in general terms. Many readers are no doubt disappointed that the common model fails to include their favorite feature, whose importance is underappreciated. I, for example, feel that auxiliary sub-tasks posed by the agent for itself (as in Sutton et al. 2022) are an important and under-appreciated means by which the agent developes abstract cognitive structure. Yet, precisely because auxiliary subtasks are not widely appreciated, they should not appear in the common model of the agent; they are not sufficiently agreed upon across disciplines.

The ambition of the common model of the intelligent agent proposed here is not to be the latest and best agent, but to be a point of departure. It seeks to be a straightforward design that is well and broadly understood in many disciplines. Whenever a researcher introduces a novel agent design, the common model is meant to be useful as a standard that can be pointed to in explaining how the new design differs from or extends the common model.

References

Craik, K. J. W. (1943). *The Nature of Explanation*. Cambridge University Press, Cambridge.

Glimcher, P. W. (2011). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences, 108*(Supplement 3), 15647–15654.

Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735–1780.

Kahneman, D. (2011). *Thinking, Fast and Slow*. Macmillan.

Littman, M. L., Sutton, R. S., Singh, S. (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pp. 1555–1561. MIT Press, Cambridge, MA.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature, 588*(7839), 604–609.

Schultz, W., Dayan, P., Montague, P. R. (1997). A neural substrate of prediction and reward. *Science, 275*(5306):1593–1598.

Sutton, R. S., Machado, M. C., Holland, G. Z., Timbers, D. S. F., Tanner, B., White, A. (2022). Reward-respecting subtasks for model-based reinforcement learning. arXiv:2202.03466.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review, 55*(4), 189.

Reward prediction error modulates sustained attention

Juliana E. Trach
Department of Psychology
Yale University
New Haven, CT, 06520
juliana.trach@yale.edu

Jed Burde
Department of Psychology
Yale University
New Haven, CT, 06520
jed.burde@yale.edu

Megan T. deBettencourt
Department of Psychology
University of Chicago
Chicago, IL,60637
debetten@uchicago.edu

Angela Radulescu
Center for Data Science
New York University
New York, NY, 10011
angela.radulescu@nyu.edu

Samuel D. McDougle
Department of Psychology
Yale University
New Haven, CT, 06520
samuel.mcdougle@yale.edu

Abstract

Attention and reinforcement learning (RL) are intertwined. While previous work has primarily focused on how your attentional state impacts and shapes RL, how the dynamics of learning might impact your attentional state on a moment-to-moment basis is an open question. Here, we leverage reinforcement learning theory to investigate the moment-to-moment influence of rewards and reward prediction errors on sustained attention. Specifically, we ask how trial-by-trial reward prediction errors might affect ongoing attentional vigilance. Using a task that simultaneously queried people's sustained attention and RL performance, we demonstrate that attentional state is influenced by the magnitude and valence (positive or negative) of recent reward prediction errors. This finding highlights the influence of RL computations on one's attentional state, and provides preliminary evidence for a potential role of the dopaminergic system in meditating the relationship between learning and attentional control.

Keywords: vigilance, sustained attention, reward prediction error

Acknowledgements

We would like to thank the members of the ACT lab at Yale university for productive discussions about this project. J.E.T was funded by the NSF's Graduate Research Fellowship Program.

1 Introduction

The interplay between attention and learning is a crucial aspect of human behavior. Consider learning to play a new song on piano: If you want to play the new piece well, you must be able to sustain your attention as you read and practice the notes. However, inevitably, your attentional state will naturally fluctuate over time [1], waxing and waning between higher and lower attentional states. One possibility is that sustaining a vigilant attention state better will improve learning. But is this interaction a one-way street? That is, can the moment-to-moment dynamics of learning also affect the dynamics of sustained attention?

Previous work indicates that reward and sustained attention do interact such that offering rewards or incentives to participants can substantially boost performance on a sustained attention task [2–5]. For example, Esterman and colleagues [6] found that participants who were incentivized with money or a shortened task duration were more accurate and consistent in their performance on a continuous performance task that taxed sustained attention. While work in this vein broadly suggests that reward can influence sustained attention, it compares aggregate performance between rewarded versus unrewarded blocks of trials, pointing to general motivation as the key factor. This leaves open critical computational questions concerning the interaction of RL and sustained attention in real time: How might trial-by-trial learning outcomes (rewards, prediction errors) modulate sustained attention on a moment-to-moment basis? Do the dynamics of RL computations have a lawful relationship to ongoing attentional vigilance?

In the following study, we integrate a continuous performance task with a probabilistic RL task. The design of our experiment was chosen to assess the impact of core constructs of RL – reward and reward prediction error (RPE) – on moment-to-moment sustained attention. We used RL models to explore how computations in the brain's RL system may influence sustained attention. We tested two potential lawful relations between RPE and sustained attention. One possibility is that especially large unsigned RPEs boost sustained attention performance. In other words, surprise – unexpected rewards (positive RPEs) or unexpected omissions of rewards (negative RPEs) – could boost sustained attention, consistent with the role of surprise in boosting memory encoding [7]. Alternatively, the magnitude and valence (positive or negative) of RPEs could jointly affect sustained attention, such that positive RPEs increase sustained attention relative to negative RPEs. Some straightforward null hypotheses are that RPEs and sustained attention do not dynamically interact, or that RPEs act to distract subjects, perhaps leading to an inverted relationship between (signed or unsigned) RPEs and sustained attention. Our computational approach allows us to test all of the above hypotheses. Behavioral evidence of interactions between RPEs and sustained attention can inspire future investigations into the neural mechanisms that mediate the relationship between RL processes and attention.

2 Hybrid sustained attention/RL task

Thirty participants (N = 19 female; mean age = 20, range = 18-21) were recruited through the subject pool at Yale University and took part in the study for course credit (1 credit/hour).

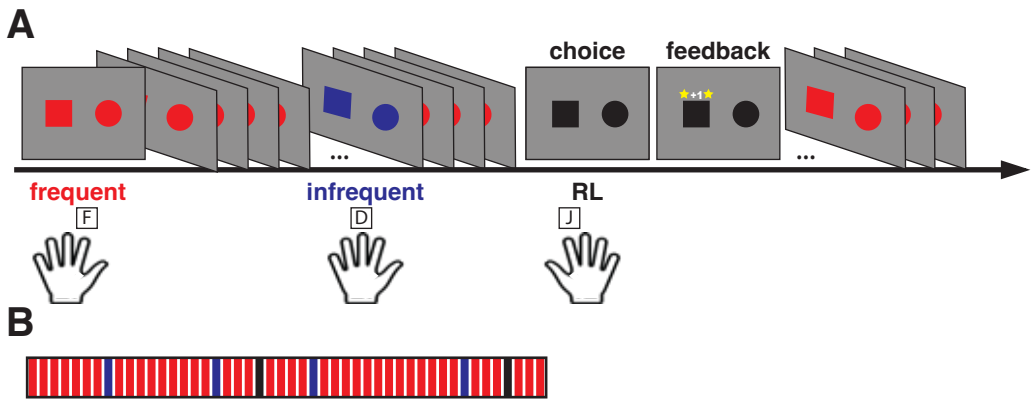


Figure 1: A) Task schematic, depicting frequent and infrequent sustained attention trials, as well as RL trials (choice and feedback). The left hand was used for attention trials and the right hand for RL trials. B) Schematic of trial sequence. Frequent sustained attention trials in red, infrequent sustained attention trials in blue, and RL trials in black.

The task was a combination of a sustained attention task [8] and a probabilistic RL task (to elicit prediction errors; Figure 1A). During a sustained attention task trial, participants saw two adjacent shapes on the screen that were either both orange or both blue. Participants used their left hand to indicate the color of the shapes and were instructed which key corresponded to orange and which to blue at the onset of the task (this assignment was counterbalanced across participants). Each trial was 800ms in duration, regardless of whether the participant responded sooner than that. For this task, on 90% of the trials the shapes were one of the two colors ("frequent" trials) and on the remaining 10% of trials they were the other color ("infrequent" trials). Thus, participants were responding with one button press on the vast majority of trials but occasionally had to inhibit this frequent response to respond correctly on infrequent trials. Participants were not explicitly told about the imbalance

in the color frequencies. Following previous work [8, 9], we used accuracy on the infrequent trials to operationalize sustained attention.

Sustained attention trials occurred in pseudorandomized blocks of 17-27 consecutive trials, and the length of each block was not predictable. At the end of a given block, participants would be presented with an RL trial (a probabilistic two-arm bandit task; Figure 1). Here, the shapes would turn black and participants used their right hand to choose one of the two shapes (Figure 1A). There was no cue dissociating the trial types other than the change of stimulus color to black. On RL trials, participants had 1.5 s to make a choice. After they made their response, they received feedback on whether or not they received a reward on that trial (+1 or +0). One shape was associated with an 80% chance of reward and the other was associated with a 20% chance of reward. There were 100 RL trials in total (and thus 100 blocks of attention trials), and the reward probabilities associated with each shape were reversed three times, after 25, 50, and 75 trials. We inserted these reversals so that participants had to continually update the value of the two shapes throughout the task. The task started with a short practice block for each individual task, and then two short blocks of the two tasks combined. Participants then began the main task, which lasted approximately 50 minutes.

3 Computational modeling

We were interested in examining the moment-to-moment relationship between sustained attention and RL computations. To that end, we fit a simple RL model to participants' choices in the probabilistic learning task [10]. We used a standard RL model:

$$Q_{t+1}(s) = Q_t(s) + \alpha\delta_t \tag{1}$$

$$p(s) = \frac{e^{Q(s)\beta}}{\Sigma_i e^{Q(s_i)\beta}} \tag{2}$$

Where Q represents the value of stimulus s on trial t, δ represents the reward prediction error ($reward_t$–$Q_t(s)$), α represents the learning rate, and β the softmax temperature. During fitting, α was constrained on [0,1] and beta on [0,50], and a Gamma (2,3) prior distribution was used to discourage extreme values of beta (following [11]). We fit two variants of this model, one where a single alpha was used for all trials, and another that allowed asymmetric learning rates for non-rewarded versus rewarded trials. We used the MATLAB function *fmincon* to find parameter values that maximized the log posterior probability of participants' choice data given the model. Fitting runs were conducted 100 times for each model to avoid local minima during optimization, using different randomized starting parameter values over each iteration. The resulting best fit model was used in all further analyses. Model fit quality was evaluated using the Akaike information criteria [12].

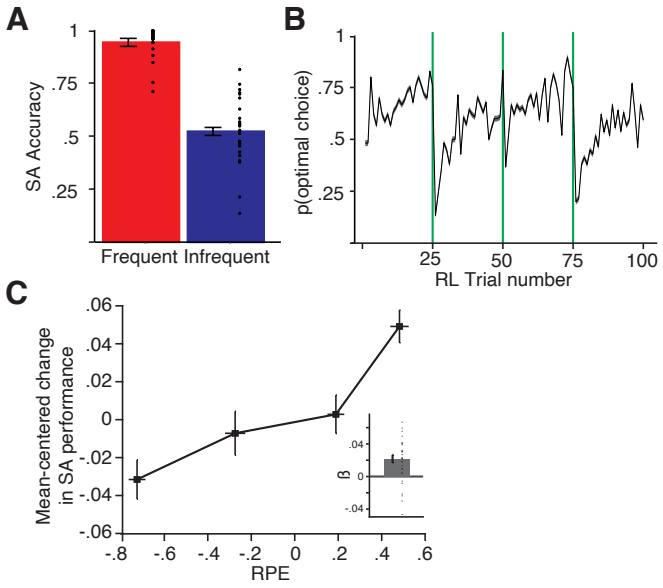


Figure 2: A) Accuracy on frequent and infrequent sustained attention trials. B) learning curve for RL task. Green lines indicate value reversal points. C) Binned RPE quantiles and mean-centered sustained attention performance (i.e., changes in average accuracy on infrequent sustained attention trials following RPEs of various sizes). Inset: regression betas reflecting the impact of RPEs on subsequent sustained attention accuracy. Error bars = 1 SEM.

4 Results

Participants exhibited typical performance on the sustained attention task, being significantly more accurate on frequent trials (M = 94%, SD = 7.6%) as compared to infrequent trials (M = 52%, SD = 15%; $t(29) = 15.71$, $p < .001$; Figure 2A; [8, 9]). In addition, on RL trials, participants learned to select the stimulus most likely to reward them throughout the task (Figure 2B).

Our primary interest was assessing whether RPE modulated sustained attention performance. To do this, we first fit an RL model to each participant's choice data and obtained participant-specific learning rates and temperature parameters (see Methods). We could then extract trial-by-trial RPEs, yielding 100 such values for each subject. Both model variants we tested fit the data well, though we observed a slight advantage for the variant with asymmetric learning rates (summed AIC for single learning rate model = 3407, summed AIC for two learning rate model = 3360; t-test on difference

in AIC values; $t(29) = -2.12$, $p = .043$). We note that the key results described below were comparable when using the worse-fitting single-rate model.

We performed a linear regression analysis that included all modeled trial-by-trial RPEs as a predictor variable and the mean performance on infrequent sustained attention trials in the subsequent blocks as the dependent variable. Across subjects, we observed a significant effect of RPE on sustained attention (Figure 2C inset; $B = 0.026$, one sample t-test: $t(29) = 4.80$, $p < .001$), providing preliminary evidence that signed RPE modulates sustained attention.

To further examine the relationship between RPE and sustained attention, and dissociate it from more basic effects of reward, we binned RPEs into four quantiles (large, negative RPEs, small negative RPEs, small positive RPEs, large positive RPEs) for each subject and compared average sustained attention changes (i.e., relative to the mean sustained attention performance) on the attention task blocks that followed those RPEs. We performed paired t-tests on sustained attention performance between bins of RPEs (Bonferroni correction: $\alpha = .05/6 = .0083$). The results of this analysis are depicted in Figure 2C. Overall, we found that participants were significantly more accurate on the sustained attention task after large, positive RPEs (ts$(29) > 3.01$, ps $< .0055$), as compared to small positive or negative RPEs. While we also found that participants were more accurate on the attention task after small positive RPEs versus large negative RPEs ($t(28) = 2.86$, $p = .0079$). Contrasts between the two middle bins ($t(28) = 0.47$, $p = .64$) or the two negative bins ($t(29) = 1.3$, $p = .20$) were not significant. We note here that infrequent sustained attention trials never occurred immediately following RL outcomes – at least three frequent attention trials intervened after RL trials before any given infrequent trial within a block. This precludes a localized post-error slowing interpretation of our results. Taken together, these results provide evidence for the hypothesis that signed RPEs influence sustained attention performance, particularly for positive RPEs.

To investigate a more global relationship between RL and sustained attention, we also correlated modeled RL learning rates with average sustained attention performance. We found that sustained attention performance was significantly correlated with learning rates for negative RPE trials (Spearman correlation: $\rho = .47$, $p = .010$), and marginally correlated with learning rates for positive RPE trials (Spearman correlation: $\rho = .35$, $p = .055$).

Finally, exploratory hypotheses initially included additional predictions of bidirectional effects of sustained attention on RL. That is, we also hypothesized that sustained attention might influence ongoing RL, whereby elevated sustained attention predicts better choice and learning behavior on RL trials. We tested this hypothesis with a logistic regression analysis with accuracy on infrequent attention trials as the predictor variable and accuracy on the subsequent RL trials (operationalized as choosing the shape associated with 80% chance of reward) as the dependent variable. We did not observe a significant effect in this direction ($B = .12$, $t(29) = 0.98$, $p = .333$). Thus, at least in the context of our task, the influence of RL on sustained attention was robust while effects in the opposite direction were not detected.

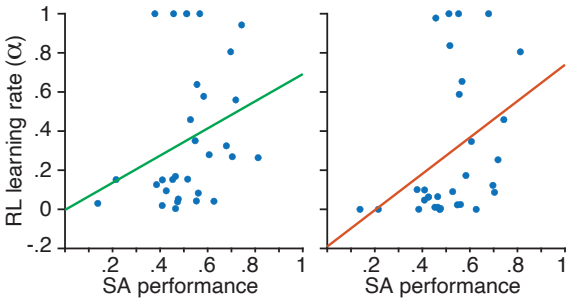


Figure 3: Correlation of learning rate and sustained attention performance. Left: learning rate for rewarded trials, Right: learning rate for unrewarded trials.

5 Discussion and Conclusions

In this study, we sought to investigate the interplay of RL and sustained attention. Overall, we found evidence that signed RPE modulates sustained attention performance in a near-linear manner, such that increasing signed RPEs led to concomitant increases in the agent's attentional state.

This result builds on previous work suggesting that performance-based incentives (rewards) improve sustained attention performance (e.g., [6]). This previous work primarily contributes to debates about theoretical causes of sustained attention lapses. Specifically, the fact that reward can reduce lapses in sustained attention appears to provide evidence against pure "resource" theories of sustained attention (i.e., that lapses in sustained attention are caused by the depletion of limited attentional resources), and is more aligned with "underload" models of sustained attention (i.e., that boredom or lack of motivation causes lapses in sustained attention [6]). We build on this work, showing how reward might boost sustained attention on a moment-to-moment timescale. Critically, our result makes a useful advance by linking sustained attention to reward prediction error, not just reward, providing a more tractable computational explanation.

The fact that signed RPE, rather than unsigned RPE, was related to sustained attention provides initial clues concerning the neural processes that might mediate this relationship. In the context of RPE and memory, effects that correlate with unsigned RPE are thought to be modulated by norepinephrine [13], whereas effects related to signed RPE are thought to be modulated by dopamine [14]. Our results therefore indirectly suggest that the increased phasic dopamine underlying RPEs could boost sustained attention. Of course, the current results do not preclude the involvement of other neuro-

transmitters in linking attention and RL, such as norepinephrine and acetylcholine. Future work using pharmacological tools is necessary to clarify how each of these systems might interact.

While this work contributes to a mechanistic understanding of the impact of reward on sustained attention, there are a number of limitations to be addressed in the future. First, our results suggest that positive RPE can cause a global boost in sustained attention, considering that the sustained attention and RL tasks used here were unrelated. That is, the RPEs that appeared to impact sustained attention were not directly relevant to the actual sustained attention task. Future work should address whether boosts in sustained attention after RPEs influence any task a subject is performing, or if these effects are more constrained. Second, we curiously did not detect an effect of sustained attention performance on RL. This result is somewhat surprising given the context of previous work on RL and selective attention demonstrating that selective attention significantly impacts both choice and updating processes in a similar RL task [11, 15, 16]. It is possible that the structure of our task, or the context changes between sustained attention and RL trials, attenuated effects of sustained attention on RL. Alternatively, the RL system may be robust to fluctuations in sustained attention.

Overall, our findings demonstrate a tight link between reinforcement learning and moment-to-moment attentional vigilance. This link might be mediated by dopamine [14, 17], the currency of the RL system. How exactly changes in phasic dopamine could induce more tonic effects on sustained attention, and the neural and computational processes supporting this interaction, make for exciting future directions.

References

1. Esterman, M. & Rothlein, D. Models of sustained attention. *Current Opinion in Psychology. Attention & Perception* **29,** 174–180 (2019).
2. Esterman, M. *et al.* Anticipation of Monetary Reward Can Attenuate the Vigilance Decrement. *PLOS ONE* **11.** Publisher: Public Library of Science, e0159741 (2016).
3. Robison, M. K., Unsworth, N. & Brewer, G. A. Examining the effects of goal-setting, feedback, and incentives on sustained attention. *Journal of Experimental Psychology: Human Perception and Performance* **47,** 869–891 (2021).
4. Esterman, M., Poole, V., Liu, G. & DeGutis, J. Modulating Reward Induces Differential Neurocognitive Approaches to Sustained Attention. *Cerebral Cortex* **27,** 4022–4032 (2017).
5. Esterman, M., Reagan, A., Liu, G., Turner, C. & DeGutis, J. Reward Reveals Dissociable Aspects of Sustained Attention. *Journal of experimental psychology. General* **143** (2014).
6. Esterman, M., Rosenberg, M. D. & Noonan, S. K. Intrinsic Fluctuations in Sustained Attention and Distractor Processing. *Journal of Neuroscience* **34,** 1724–1730 (2014).
7. Rouhani, N., Norman, K. A. & Niv, Y. Dissociable effects of surprising rewards on learning and memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **44,** 1430–1443 (2018).
8. deBettencourt, M. T., Keene, P. A., Awh, E. & Vogel, E. K. Real-time triggering reveals concurrent lapses of attention and working memory. *Nature Human Behaviour* **3.** Publisher: Springer US (2019).
9. deBettencourt, M. T., Norman, K. A. & Turk-Browne, N. B. Forgetting from lapses of sustained attention. *Psychonomic Bulletin & Review* **25,** 605–611 (2018).
10. Rescorla, R. & Wagner, A. A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory,* 64–99 (1972).
11. Leong, Y. C., Radulescu, A., Daniel, R., DeWoskin, V. & Niv, Y. Dynamic Interaction between Reinforcement Learning and Attention in Multidimensional Environments. *Neuron* **93.** Publisher: Cell Press, 451–463 (2017).
12. Akaike, H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control* **19.** Conference Name: IEEE Transactions on Automatic Control, 716–723 (1974).
13. Sara, S. J. The locus coeruleus and noradrenergic modulation of cognition. *Nature Reviews Neuroscience* **10.** Number: 3 Publisher: Nature Publishing Group, 211–223 (2009).
14. Schultz, W., Dayan, P. & Montague, P. R. A Neural Substrate of Prediction and Reward. *Science* **275,** 8 (1997).
15. Daniel, R., Radulescu, A. & Niv, X. Y. Intact Reinforcement Learning But Impaired Attentional Control During Multidimensional Probabilistic Learning in Older Adults (2020).
16. Niv, Y. *et al.* Reinforcement learning in multidimensional environments relies on attention mechanisms. *Journal of Neuroscience* **35,** 8145–8157 (2015).
17. Nieoullon, A. Dopamine and the regulation of cognition and attention. *Progress in Neurobiology,* 31 (2002).

Achieving Zero-Shot Task Generalization with Formal Language Instructions[†]

Pashootan Vaezipoor[*]
Department of Computer Science
University of Toronto & Vector Institute
Toronto, Ontario, Canada
pashootan@cs.toronto.edu

Andrew C. Li[*]
Department of Computer Science
University of Toronto & Vector Institute
Toronto, Ontario, Canada
andrewli@cs.toronto.edu

Rodrigo Toro Icarte
Department of Computer Science
Universidad Católica de Chile & Vector Institute
Santiago, Región Metropolitana, Chile
rodrigo.toro@ing.puc.cl

Sheila A. McIlraith
Department of Computer Science
University of Toronto & Vector Institute
Schwartz Reisman Institute
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract

We address the problem of generalization of deep RL agents to very large sets of compositional instructions. We employ a well-known formal language – linear temporal logic (LTL) – to specify instructions and propose a novel learning approach that exploits the compositional syntax and semantics of LTL. Leveraging the expressive power of LTL, our agents are tasked to perform diverse and complex temporally extended behaviours including conditionals and alternative realizations. Experiments on discrete and continuous domains demonstrate the strength of our approach in a zero-shot setting, allowing us to tackle unseen instructions up to 3x larger than observed in training.

Keywords: Compositional Generalization, Zero-Shot Generalization, Formal Language, Linear Temporal Logic

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. The third author acknowledges funding from ANID (Becas Chile). This work was done while he was a graduate student at the University of Toronto.

[†]The full paper appeared in the Proceedings of the 38th International Conference on Machine Learning, (ICML 2021) [5]
[*]Equal Contribution

1 Introduction

A long-standing aspiration of artificial intelligence is to build agents that can understand and follow human instructions to solve problems. Recent advances in RL have made it possible to learn a policy that decides the next action conditioned on the current observation and a natural language instruction. Such approaches, while demonstrating some degree of generalization to unseen instructions, require manually building a large training set comprised of natural language instructions with their corresponding reward functions. Instructing RL agents using structured or formal languages, on the other hand, allows RL practitioners to (automatically) generate massive training data, thanks to the unambiguous semantics and compact compositional syntax that these languages offer.

In this work, we focus on solving complex, temporally-extended tasks with compositional instructions expressed in a formal language. Many previous works have proposed decomposing such tasks into smaller subtasks that are solved independently, an approach which we refer to as *myopic*. This can lead to individual policies that are optimal on subtasks, but suboptimal in aggregate. We use *linear temporal logic (LTL)*[*] over domain-specific vocabulary to instruct RL agents to learn policies that generalize well to unseen instructions without compromising optimality guarantees. Our learning algorithm exploits a semantics-preserving rewriting operation, called *LTL progression*, that allows the agent to identify aspects of the original instructions that remain to be addressed in the context of an evolving experience. This enables learning policies in a non-myopic manner, all the while preserving optimality guarantees and supporting generalization.

Our contributions are as follows:

- We propose a novel approach for teaching deep RL agents to follow LTL instructions with theoretical advantages over existing methods [3, 4]. This leads to better generalization when extrapolating to unseen instructions with more complex structure across both continuous and discrete action-space domains.
- We show that encoding LTL instructions via a neural architecture, viz. GRU, LSTM and GNN, equipped with LTL progression yielded higher reward policies relative to myopic approaches.
- We propose an environment-agnostic LTL pretraining scheme to improve sample efficiency on downstream tasks.

2 Multitask Learning with LTL

In order to instruct RL agents using language, the first step is to agree upon a common vocabulary between us and the agent. Here, we use a set of propositional symbols $\mathcal{P}$ as the vocabulary, representing high-level events or properties whose occurrences in the environment can be detected by the agent. For instance, in a smart home environment, $\mathcal{P}$ could include events such as opening the living room window, activating the fan, turning on/off the stove, or entering the living room. Then, we use LTL to compose temporally-extended occurrences of these events into instructions. Two possible instructions that can be expressed in LTL (but described here in plain English) are (1) "Open the living room window and activate the fan in any order, then turn on the stove" and (2) "Open the living room window but don't enter the living room until the stove is turned off".

Linear Temporal Logic (LTL): LTL extends propositional logic with two temporal operators: $\bigcirc$ (*next*) and U (*until*), from which additional operators $\Box$ (*always*), $\Diamond$ (*eventually*) are defined and often included for convenience. Given a finite set of propositional symbols $\mathcal{P}$, the syntax of an LTL formula is defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \bigcirc \varphi \mid \varphi \,\mathsf{U}\, \psi \mid \Box\varphi \mid \Diamond\varphi \quad \text{where } p \in \mathcal{P}$$

In contrast to propositional logic, LTL formulas are evaluated over sequences of observations (i.e., *truth assignments* to the propositional symbols in $\mathcal{P}$). For brevity we omit the formal definition of satisfiability, but intuitively, the formula $\bigcirc\varphi$ (*next* φ) holds if φ holds at the next time step; $\varphi \,\mathsf{U}\, \psi$ (φ *until* ψ) holds if φ holds until ψ holds; $\Box\varphi$ if φ holds at every time step in the sequence; $\Diamond\varphi$ holds if φ holds at some point in the future.

Example: Consider the MiniGrid environment in Figure 2. There are two rooms – one with a blue and a red square, and one with a blue and a green square. The agent, represented by a red triangle, can rotate left and right, and move forward. Suppose that the set of propositions $\mathcal{P}$ includes R, G, and B, which are true iff the agent is standing on a red/green/blue square (respectively) in the current time step. Then, we can define a wide variety of tasks using LTL:

- Single goal: $\Diamond\mathrm{R}$ (reach a red square).
- Goal sequences: $\Diamond(\mathrm{R} \wedge \Diamond\mathrm{G})$ (reach red and then green).
- Disjunctive goals: $\Diamond\mathrm{R} \vee \Diamond\mathrm{G}$ (reach red or green).
- Conjunctive goals: $\Diamond\mathrm{R} \wedge \Diamond\mathrm{G}$ (reach red and green in any order).
- Safety constraints: $\Box\neg\mathrm{B}$ (do not touch a blue square).

[*]LTL is an expressive formal language that combines temporal modalities such as *eventually*, *until*, and *always* with binary predicates that establish the truth or falsity of an event or property (e.g., *have-coffee*), composed via logical connectives to support specification of goal sequences, partial-order tasks, safety constraints, etc.

We can also combine these tasks to define new tasks, e.g., "*go to a red square and then a green square but do not touch a blue square*" can be expressed as $\Diamond(\mathrm{R} \wedge \Diamond \mathrm{G}) \wedge \Box \neg \mathrm{B}$.

From LTL Instructions to Rewards: So far, our discussion about LTL instructions has been environment-agnostic (the syntax and semantics of LTL are independent of the environment). Now, we show how to reward an RL agent for realizing LTL instructions via an MDP. We accomplish this by using a *labelling function* $L : S \times A \to 2^{\mathcal{P}}$ that assigns truth values to the propositions in $\mathcal{P}$ given the current state $s \in S$ of the environment and the action $a \in A$ selected by the agent. One may think of the labelling function as having a collection of *event detectors* that fire when the propositions in $\mathcal{P}$ hold in the environment. In our running example, $\mathrm{R} \in L(s, a)$ iff the agent is on top of the red square.

Given a labelling function, the agent automatically evaluates whether an LTL instruction has been satisfied or falsified. Note that the satisfaction of many LTL formulas can be ensured after a finite number of steps — e.g., $\Diamond \mathrm{R}$ (*eventually red*) is ensured once R is true. If the instruction is satisfied (i.e., completed) we give the agent a reward of 1 and if the instruction is falsified (e.g., the agent breaks a safety constraint) we penalize the agent with a reward of -1. The episode ends as soon as the instruction is ensured to be satisfied or falsified. Note that this reward function might be non-Markovian, as it depends on sequences of states and actions, making the overall learning problem partially observable. We discuss how to deal with this issue below.

Instructing RL Agents using LTL: We now formalize the problem of learning a policy that can follow LTL instructions. Hereafter, LTL instruction/task may be used interchangeably. Given an MDP without a reward function $\mathcal{M}_e = \langle S, T, A, p, \gamma, \mu \rangle$, a finite set of propositional symbols $\mathcal{P}$, a labelling function $L : S \times A \to 2^{\mathcal{P}}$, a finite (but potentially large) set of LTL formulas Φ, and a probability distribution τ over those formulas $\varphi \in \Phi$, our goal is to learn an optimal policy $\pi^*(a_t | s_1, a_1, ..., s_t, \varphi)$ w.r.t. $R_\varphi(s_1, a_1, ..., s_t, a_t)$ for all $\varphi \in \Phi$. To learn this policy, the agent samples a new LTL task φ from τ every episode and is rewarded according to R_φ. The episode ends when the task is completed, falsified, or a terminal state is reached.

Since the reward is non-Markovian, an optimal policy $\pi^*(a_t | s_1, a_1, ..., s_t, \varphi)$ has to consider the whole history of states and actions. To handle this issue, Kuo et al. [3] proposed to encode the policy using a recurrent neural network. However, we show that we can overcome this complexity by exploiting a procedure known as LTL progression [2].

LTL progression is a semantics-preserving rewriting procedure $\mathrm{prog}(\sigma, \varphi)$ that takes an LTL formula φ and current labelled state $\sigma \in 2^{\mathcal{P}}$ as input and returns a formula that identifies aspects of the original instructions that remain to be addressed. Progress towards completion of the task is reflected in diminished remaining instructions. For instance, the task $\Diamond(\mathrm{R} \wedge \Diamond \mathrm{G})$ (*go to red and then to green*) will progress to $\Diamond \mathrm{G}$ (*go to green*) as soon as R holds in the environment. We use LTL progression to make the reward function R_φ Markovian. We achieve this by (1) augmenting the MDP state with the current LTL task φ that the agent is solving, (2) progressing φ after each step given by the agent in the environment, and (3) rewarding the agent when φ progresses to true ($+1$) or false (-1). We theoretically show that this procedure preserves the optimal policy for the original non-Markovian reward function.

Similar Works: Two recent works have explored how to teach RL agents to follow unseen instructions using temporal logic [3, 4]. Here we discuss the theoretical advantages of our approach over theirs. Kuo et al. [3] proposed to learn a recurrent policy $\pi^*(a_t | s_1, a_1, ..., s_t, \varphi)$ using a recurrent neural network, by solving a partially observable problem (i.e., a POMDP). In contrast, we propose to learn a stationary policy $\pi^*(a_t | s_t, \varphi)$. Since solving MDPs is easier than solving POMDPs (MDPs can be solved in polynomial time whereas POMDPs are undecidable), this gives our approach a theoretical advantage which results in better empirical performance (as shown in Section 4).

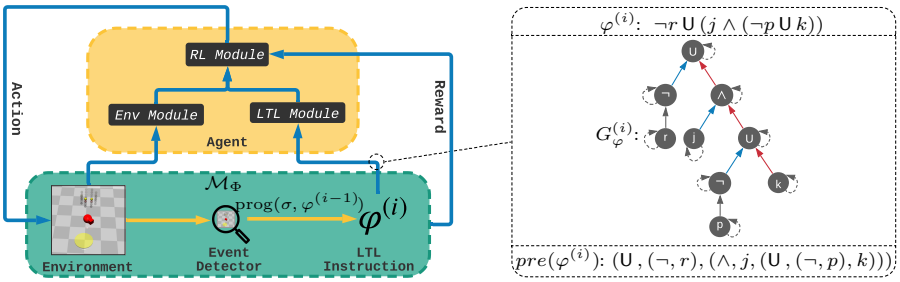
Leon et al. [4] instructed agents using a fragment of LTL and defined a reasoning module which decides the next proposition to satisfy. Thus, the agent only needs to learn a policy $\pi(a | s, p)$ conditioned on the state s and a proposition $p \in \mathcal{P}$. However, this approach is myopic – it optimizes for solving the next subtask without considering what the agent must do after and, as a result, might converge to suboptimal solutions. This is a common weakness across recent approaches that instruct RL agents. As an example, consider the MiniGrid from Figure 2. The two red doors at the entrance to each room automatically lock upon entry so the agent cannot visit both rooms. Suppose the agent has to solve two LTL tasks, uniformly sampled at the beginning of each episode: $\Diamond(\mathrm{B} \wedge \Diamond \mathrm{G})$ (*go to a blue square and then to a green square*) or $\Diamond(\mathrm{B} \wedge \Diamond \mathrm{R})$ (*go to a blue square and then to a red square*). For both tasks, a myopic approach will tell the agent to first achieve $\Diamond \mathrm{B}$ (*go to blue*), but doing so without considering where the agent must go after might lead to a dead end (due to the locking doors). In contrast, an approach that learns an optimal policy $\pi^*(a | s, \varphi)$ can consistently solve these two tasks (Figure 2(b)).

3 Model Architecture

In this section, we build on the RL framework with LTL instructions from Section 2 and explain a way to realize this approach in complex environments using deep RL.

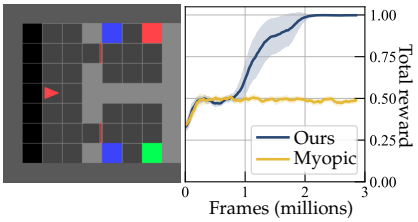
In each episode, a new LTL task φ is sampled. At every step, the environment returns an observation, the event detectors return a truth assignment σ of all propositions in $\mathcal{P}$ and the LTL task is automatically progressed to $\varphi' := \mathrm{prog}(\sigma, \varphi)$. The

Figure 1: (a) Our RL framework for solving LTL tasks (b) Possible representations of an LTL formula. *Top*: standard LTL syntax. *Middle*: abstract syntax tree. *Bottom*: prefix notation.



(a) (b)

Figure 2: (left) A toy minigrid environment where doors lock upon entry. The task is equally likely to be either *go to blue then red* or *go to blue then green*. (right) A Myopic policy only succeeds in 50% of tasks while ours obtains the maximum reward.



agent then receives both the environment observation and the progressed formula and emits an action via a modular architecture consisting of three trainable components (see Figure 1): **(1) Env Module:** an environment-dependent model that preprocesses the observations (e.g., a convolutional or a fully-connected network). **(2) LTL Module:** a neural encoder for the LTL instructions (discussed below). **(3) RL Module:** a module which decides actions to take in the environment, based on observations encoded by the Env Module and the current (progressed) task encoded by the LTL module.

LTL Module: LTL formulas can be encoded through different means, the simplest of which is to apply a sequence model (e.g., LSTM) to the input formula. However, given the tree-structured nature of these formulas, *Graph Neural Networks* (GNNs) may provide a better inductive bias. We represent an LTL formula φ as a directed graph G_φ, as shown in Figure 1(b). This is done by first creating φ's parse tree, where each subformula is connected to its parent operator via a directed edge, and adding self-loops for all the nodes. The GNN performs T message passing steps over G_φ. Due to the direction of the edges in G_φ, the messages flow in a bottom-up manner and after T steps we regard the embedding of the root node of G_φ as the embedding of φ.

We experimented with both sequence models and GNN. In each case we first create one-hot encodings of the formula tokens (operators and propositions). For sequence models we convert the formula to its prefix notation $pre(\varphi)$ and then replace the tokens with their one-hot encodings. For GNN, these encodings serve as input node features $\boldsymbol{x}_v^{(0)}$.

Pretraining the LTL Module: Simultaneous training of the LTL Module with the rest of the model can be a strenuous task. We propose to pretrain the LTL module, taking advantage of the environment-agnostic nature of LTL semantics and the agent's modular architecture. Specifically, the pretraining task is to progress formulas $\varphi \sim \tau(\Phi)$ to true in as few steps as possible by setting a single proposition to true at each step. Hence our scheme is: (1) pretrain the LTL module with formula set Φ and task distribution τ; (2) use the learned LTL Module in the downstream task. While many pretraining schemes are possible, this one involves a simple task which can be viewed as an abstracted version of the downstream task. Since pretraining does not require interaction with a physical environment, it is more wall-clock efficient than training the full model on the downstream environment. In Section 4 we demonstrate the empirical benefits of pretraining the LTL Module.

4 Experiments

We designed our experiments to investigate whether RL agents can learn to solve complex, temporally-extended tasks specified in LTL. We answer the following questions: **(1) Performance:** How does our approach fare against baselines that do not utilize LTL progression or are myopic? **(2) Pretraining:** Does pretraining the LTL Module result in more rapid convergence in novel downstream environments? **(3) Upward Generalization:** Can the RL agent trained using our approach generalize to larger instructions than those seen in training? **Environments:** We experiment with a discrete 7×7 grid environment, similar to [1], with objects appearing on some of the squares. Furthermore, we experimented on a similar environment with continuous actions based on MuJoCo.

Tasks: Every episode, the agent faces some i.i.d-sampled LTL task via procedural generation from a large set of tasks. We consider two LTL task spaces: *Partially-Ordered Tasks*, where multiple sequences of goals can be solved in parallel, however, the goals within each sequence must be satisfied in order ($\sim 5 \times 10^{39}$ possible tasks); *Avoidance Tasks*, similar to *P.O. tasks* but also includes objects which must be avoided ($\sim$970 million possible tasks).

Our Method: Our method exploits LTL progression and uses PPO for policy optimization. We experimented with three architectures for the LTL Module: GNN, GRU and LSTM. We refer to these as GNN_{prog}, GRU_{prog} and $\text{LSTM}_{\text{prog}}$.

Baselines: We compared our method against three baselines: **(1) No LTL:** ignores the LTL instructions, but learns a non-stationary LSTM-based policy; **(2) GRU:** based on Kuo et al. [3], this approach learns a non-stationary policy that

Figure 3: Our approaches using LTL progression (marked by $\bullet_{\text{prog}}$) outperformed other baselines. We report *discounted return* over the duration of training (averaged over 30 seeds, with 90% confidence intervals).

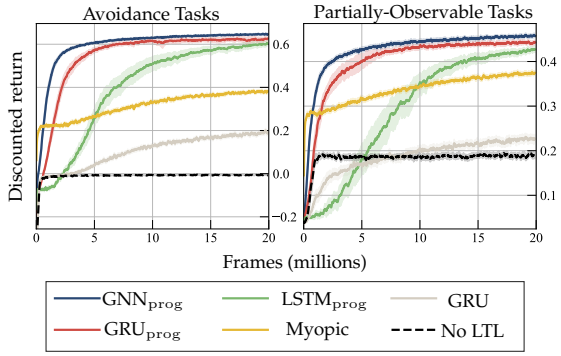


Table 1: Trained RL agents are evaluated on the training distribution of tasks, as well as out-of-distribution tasks with increased depth of sequences, and increased number of conjuncts. In each entry, we report *total reward* and *discounted return* (in parentheses) averaged over 30 seeds and 100 episodes per seed.

	I.I.D	↑ Depth	↑ Conjuncts
	(a) Avoidance Tasks		
GNN$_{\text{prog}}$ (ours)	**0.98(0.66)**	**0.98(0.33)**	0.57(0.14)
GRU$_{\text{prog}}$ (ours)	0.89(0.63)	0.42(0.07)	**0.58(0.25)**
GRU	0.32(0.22)	-0.03(-0.01)	-0.35(-0.16)
Myopic	0.88(0.50)	0.71(0.07)	0.58(0.11)
	(b) Partially-Ordered Tasks		
GNN$_{\text{prog}}$ (ours)	**1.0(0.47)**	**0.97(0.0074)**	0.98(**0.0340**)
GRU$_{\text{prog}}$ (ours)	**1.0**(0.46)	0.29(0.0005)	**0.99**(0.0252)
GRU	0.87(0.24)	0.03(0.0000)	0.54(0.0075)
Myopic	**1.0**(0.40)	0.94(0.0042)	**0.99**(0.0221)

encodes the LTL instructions with a GRU, but does not progress the formula over time; **(3) Myopic:** inspired by Leon et al. [4], uses a reasoning technique to tell the agent which propositions to achieve next in order solve the LTL task. Specifically, the agent observes whether making a particular proposition true would (a) progress the current formula, (b) have no effect, or (c) make it unsatisfiable. Given these observations, the agent then learns a stationary policy. Note that this approach might converge to suboptimal solutions as discussed at the end of Section 2.

4.1 Results

Performance: Figure 3 shows the grid environment results for Partially-Ordered and Avoidance Tasks: (1) LTL progression significantly improved i.i.d. generalization; (2) A compositional architecture such as GNN learned to encode LTL formulas better than sequential ones like GRU and LSTM; (3) The myopic baseline initially learned quickly, but was eventually outperformed by all our methods. Similar results were observed for the continuous MuJoCo environment.

Pretraining: We investigated the effects of pretraining the LTL Module. We observed that pretraining the LTL Module converged to better policies, often several times faster on both tasks (results omitted due to space limitations).

Upward (Out-of-Distribution) Generalization: We evaluated trained agents on Partially-Ordered and Avoidance Tasks with: (a) several times longer sequences, and (b) many times more conjuncts (i.e., more tasks to be completed in parallel) than seen in training. Table 1 shows that our approaches using LTL progression generalized well to more complex formulas, with GNN generally outperforming GRU.

5 Conclusion

Creating learning agents that understand and follow open-ended human instructions is a challenging problem with significant real-world applicability. Part of the difficulty stems from the need for large training sets of instructions and associated rewards. In this work, we trained an RL agent to follow temporally extended instructions specified in the formal language, LTL. The compositional syntax of LTL allowed us to generate massive training data, automatically. We proposed a novel approach to multitask RL which exploits LTL progression, allowing us to learn Markovian policies while preserving optimality guarantees. We realized our approach via deep RL, exploring several neural methods to encode LTL formulas and developing an environment-agnostic pretraining scheme which accelerated learning. Our experiments on discrete and continuous domains demonstrated robust generalization across different tasks, while outperforming a prevalent myopic approach.

References

[1] Jacob Andreas et. al., *"Modular Multitask Reinforcement Learning with Policy Sketches,"* ICML 2017.

[2] Fahiem Bacchus et. al., *"Using Temporal Logics to Express Search Control Knowledge for Planning,"* AI 2000.

[3] Yen-Ling Kuo et. al., *"Encoding Formulas as Deep Networks: Reinforcement Learning for Zero-Shot Execution of LTL Formulas,"* arXiv preprint arXiv:2006.01110, 2020.

[4] Borja Leon et. al., *"Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning,"* arXiv preprint arXiv:2006.08767, 2020.

[5] Pashootan Vaezipoor et. al., *"LTL2Action: Generalizing LTL Instructions for Multi-Task RL,"* ICML 2021.

LaVa: Latent Variable Models for Sample Efficient Multi-Agent Reinforcement Learning

Aravind Venugopal
Robert Bosch Center for Data Science and AI
Indian Institute of Technology
Madras, India
aravindvenugopal19@gmail.com

Elizabeth Bondi
School of Engineering and Applied Sciences
Harvard University
Massachusetts, USA
ebondi@g.harvard.edu

Fei Fang
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
feifang@cmu.edu

Balaraman Ravindran
Dept. of CSE; Robert Bosch Center for Data Science and AI
Indian Institute of Technology
Madras, India
ravi@cse.iitm.ac.in

Abstract

Multi-agent reinforcement learning (MARL) has widespread potential applications in real-world cooperative scenarios such as multi-robot coordination, smart grid optimization and autonomous driving, to name a few. Since each agent's policy changes while learning, multi-agent environments are non-stationary with respect to each agent. Challenges arising from non-stationarity make learning difficult in environments where only a portion of the true state is visible to the agents (partially observable). High-dimensional inputs further complicate learning, as learning effective policies requires first learning good compact representations of the inputs. Thus, MARL tasks are associated with very high sample complexity. Despite recent advances in MARL, ensuring sample-efficient policy optimization through efficient representation learning remains a challenging question, rendering model-free MARL algorithms sample-inefficient. This limits their applicability in scenarios where it is costly to collect real-world data.

We propose **La**tent **Va**riable Models for Multi-Agent Reinforcement Learning (LaVa), a novel, sample-efficient approach that utilizes an explicitly and efficiently learned model of environment dynamics to perform policy optimization using latent state representations. Efficient learning of dynamics is ensured using an exploration scheme that operates in the latent space to seek out expected future novelty of states. By separating representation learning from policy optimization, LaVa reduces environment interactions and accelerates learning. We perform empirical evaluation on complex, continuous control multi-agent tasks showing that our algorithm outperforms state-of-the-art model-free and model-based baselines in sample efficiency and final performance. Furthermore, our approach can be used with any multi-agent reinforcement learning algorithm.

Keywords: Reinforcement Learning; Multi-agent Systems; Representation Learning

Acknowledgements

Co-author Fei Fang is supported in part by NSF IIS-2046640 (CAREER).

1 Introduction

Multi-agent reinforcement learning (MARL) provides a flexible and adaptive learning framework for modeling complex multi-agent problems such as social dilemmas, multi-robot control and smart grids using RL. We deal with MARL tasks where agents have to learn cooperative behaviors to achieve a common goal in an environment where only a part of the environment state is visible (partially observable). The underlying environment dynamics are non-stationary with respect to each agent since the behaviors of other agents keep changing [5].

Further, these tasks involve learning low dimensional, generalizable representations from high-dimensional input spaces which become significantly larger as the number of agents increases, as the inputs are joint observations, i.e., the concatenated observations of all agents. Recent model-free MARL approaches have attempted to address these issues by using centralized critics [5] and factored critics [7] that have access to joint observations and the true global state, respectively. However, the challenge remains that a great deal of data is needed, or in other words, these approaches suffer from high sample complexity. Similar to humans learning representations [6] for tasks, learning effective policies requires first learning good representations of high-dimensional inputs. Model-free MARL algorithms perform representation learning and policy optimization simultaneously due to their end-to-end training procedure. As a result, a considerable amount of data is required for training [9]. This issue limits the applicability and widespread acceptance of current MARL approaches to solve real-world problems where data is not widely available and data collection is costly.

In this paper, we make two major contributions. **First**, to address the challenges associated with sample efficiency and representation learning in MARL, we separate representation learning from policy optimization. For doing this, we propose **La**tent **Va**riable Models for Multi-Agent Reinforcement Learning (LaVa), a novel MARL approach that learns environment dynamics using a sequential latent variable model, and then generates multi-agent trajectories of compact latent state representations with this model. LaVa then trains an MARL algorithm purely using these trajectories, thus combining a learnt dynamics model with a model-free MARL algorithm. An intuitive analogue to the model would be a device that allows multiple humans share a "dream" to learn a coordinated task.

This allows efficient policy optimization, as it is separated from representation learning and as data generated for policy optimization now consists of compact, already information-rich, latent states. Furthermore, generating data also reduces interactions with the environment, which speeds up training and improves sample efficiency. We use an exploration scheme which explicitly seeks out latent states with high expected future novelty. As a result, agents visit states with highest model uncertainty in predictions during an initial exploration phase of training. This ensures efficient model learning and also ensures good quality representations for policy optimization.

Second, focusing on fully cooperative Markov games [4], we perform empirical analysis comparing LaVa against state-of-the-art model-free and model-based MARL algorithms. We show that **(i)** LaVa outperforms all baselines in sample efficiency and final performance; **(ii)** our exploration scheme leads to much better exploration of the joint observation space; and **(iii)** our latent variable model is able to accurately learn environment dynamics in complex multi-agent tasks.

2 Preliminaries

We consider MARL in partially observable, fully cooperative settings where a team of agents interacts with the environment to achieve a common goal with a team reward. Our problem setting can be modeled as a fully cooperative Markov game consisting of a tuple $G = \langle N, S, A, P, R, \{\mathcal{O}^i\}, \{O^i\}, \gamma \rangle$. $N \equiv \{1, ..., n\}$ is the set of agents. $s \in S$ represents the true global state. $A \equiv \Pi_i A^i$ is the joint action space, where A^i is the action space for player i.

At each step, each agent $i \in N$ receives a private observation $o^i \in \mathcal{O}^i$ according to the observation function $O^i(s) : S \rightarrow \mathcal{O}^i$. It then chooses an action $a^i \in A^i$. The environment then transitions to the next state according to the state transition function $P(s'|s, \mathbf{a}) : S \times A \times S \rightarrow [0, 1]$, where $\mathbf{a}$ is the joint action. All agents receive the same reward and share the reward function $R : S \times A \rightarrow \mathcal{R}$. The discount factor is $\gamma \in [0, 1]$. Each agent's policy $\pi^i(a^i|\tau^i)$ is conditioned on its action-observation history $\tau^i \in H^i \equiv (A^i \times \mathcal{O}^i)^*$. The joint stochastic policy is represented by π and induces the joint action-value function $Q^\pi(s_t, \mathbf{a}_t) = \mathbf{E}_{s_{t+1:T}, \mathbf{a}_{t+1:T}} [\sum_{j=0}^{T} \gamma^j (r_{t+j}|s_t, \mathbf{a}_t)]$ where r_t is the agent's reward at step t.

We follow the Centralized Training Decentralized Execution (CTDE) [5] paradigm where each agent can access the global state and joint action while training, but can access only its individual observation-action history during execution.

3 Proposed Architecture

First, we explain how LaVa efficiently learns environment dynamics in a fully cooperative Markov game. We then explain in detail, our proposed exploration scheme and describe our training framework.

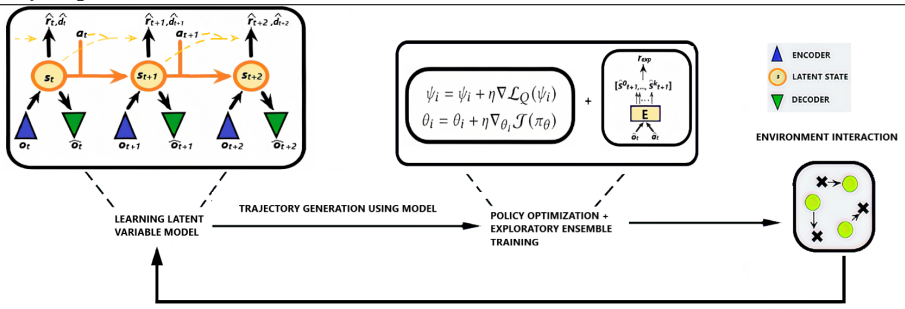


Figure 1: LaVa algorithm pipeline

Learning Environment Dynamics: We use a sequential latent variable model to learn environment dynamics by maximizing the marginal likelihood $p(\mathbf{o}_{1:\tau+1}|\mathbf{a}_{1:\tau})$ where $\mathbf{o}_{1:\tau+1}$ and $\mathbf{a}_{1:\tau}$ represent the joint observations and joint actions over $\tau+1$ timesteps, respectively. The prior distribution over the latent variables $z_{1:\tau+1}$ can be factorized as $p(z_{1:\tau+1}|\mathbf{a}_{1:\tau}) = \prod_{t=0}^{\tau} p(z_{t+1}|z_t, \mathbf{a}_t)$. The posterior distribution can be factorized as $q(z_{1:\tau+1}|\mathbf{o}_{1:\tau+1}, \mathbf{a}_{1:\tau}) = \prod_{t=0}^{\tau} q(z_{t+1}|\mathbf{o}_{t+1}, z_t, \mathbf{a}_t)$. Using this posterior, we obtain the following Evidence Lower Bound (ELBO) [2] (D_{KL} is the KL-Divergence):

$$\log p(\mathbf{o}_{1:\tau+1}|\mathbf{a}_{1:\tau}) \geq \mathbf{E}_{z_{1:\tau+1}\sim q}\left[\sum_{t=0}^{\tau} \log p(\mathbf{o}_{t+1}|z_{t+1}) - D_{KL}(q(z_{t+1}|\mathbf{o}_{t+1}, z_t, \mathbf{a}_t)\|p(z_{t+1}|z_t, \mathbf{a}_t))\right] \tag{1}$$

We learn a reward predictor as part of our model since learning to predict rewards that correlate with observations and vice-versa, is likely to improve the model. To predict terminal states d_t (Boolean), we also learn a termination function predictor. We use a neural network with parameters ϕ to represent the model which maximizes the objective $\mathcal{L}_M(\phi)$:

$$\begin{aligned}\mathcal{L}_M(\phi) = \mathbf{E}_{z_{1:\tau+1}\sim q_\phi}\Bigg[\sum_{t=0}^{\tau}\Big[\log p_\phi(\mathbf{o}_{t+1}|z_{t+1}) + \log p_\phi(r_{t+1}|z_{t+1}, z_t, \mathbf{a}_t) + \log p_\phi(d_{t+1}|z_{t+1}, a_{t+1}) \\ - D_{KL}(q_\phi(z_{t+1}|\mathbf{o}_{t+1}, z_t, \mathbf{a}_t)\|p_\phi(z_{t+1}|z_t, \mathbf{a}_t))\Big]\Bigg]\end{aligned} \tag{2}$$

Exploration scheme: To learn the dynamics model efficiently, and to prevent overfitting of the model to certain parts of the state space early on in the training process, we require an exploration policy that seeks out such states as to bring about greatest improvement to the model. To perform such exploration is to seek out states with greatest uncertainty in model predictions. To quantify uncertainty, we use a measure of ensemble disagreement, which has been empirically shown to be effective [8]. We use an ensemble $\mathbf{E}$ of k smaller models $e_j(z_{t+1}|\mathbf{o}_t, \mathbf{a}_t), j \in [1:k]$, that take as input the joint observation and action to predict $\hat{s}_{t+1}$, the latent model's representation of the next global state. We compute the variance over the means of $\hat{s}_{t+1}$ predicted by the ensemble. This variance is used as the reward r_{exp} for training the exploration policy. Once the uncertain states have been visited by the exploration policy, r_{exp} for these states decreases.

$$e_j(z_{t+1}|\mathbf{o}_t, \mathbf{a}_t) = \mathcal{N}(\mu_j, 1)\,;\, r_{exp} = Var(\mu_j|j \in [1:k]) \tag{3}$$

LaVa: As shown in Fig. 1, LaVa consists of the following steps performed iteratively:

1. The model learns environment dynamics by encoding the joint observation-actions to compact latent states.
2. The model generates multi-agent trajectories starting from real joint observations drawn from a replay buffer shared by all agents. These trajectories are used to train the MARL algorithm and exploration ensemble.
3. Individual agent policies interact with the environment and real trajectories are stored in the replay buffer.

Any model-free MARL algorithm can be used with LaVa. We use Multi-Agent Soft Actor Critic (MASAC) [1] for policy optimization. The critic loss for each agent i is calculated using temporal difference learning as follows:

$$\mathcal{L}_Q(\psi_i) = \sum_{i=0}^{N_b} \mathbf{E}_{\sim D}\left[(Q_i^{\psi_i}(z_t, \mathbf{a}_t) - y_i)^2\right]\,;\, y_i = \beta r_{exp} + \kappa r_t + \gamma \mathbf{E}_{a_{t+1}\sim\bar{\pi}}\left[(Q_i^{\bar{\psi}_i}(z_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi^i_{\bar{\theta}_i}(a_t^i|o_t^i))\right] \tag{4}$$

where θ_i refers to the parameters of the actor i, ψ_i refers to the parameters of the critic i. $\bar{\theta}_i$ and $\bar{\psi}_i$ are the target actor and critic parameters. The s used for training is the global state predicted by the model and D is the replay buffer. $\beta, \kappa \in [0, 1]$ are used to balance model learning and policy learning. N_b is batch size. Each actor is then updated as follows:

$$\nabla_{\theta_i}\mathcal{J}(\pi^i_{\theta_i}) = \mathbf{E}_{o\sim D, a\sim\pi}(\nabla_{\theta_i}\log(\pi^i_{\theta_i}(a^i_t|o^i_t))\big(Q_i^{\psi_i}(z_t, \mathbf{a}_t) - \alpha\log(\pi^i_{\theta_i}(a^i_t|o^i_t))\big) \tag{5}$$

Episodes:	1250	2500	3750	5000
MADDPG	-181.3	-165.5	-151.2	-143.1
MASAC	-165.1	-143.6	-132.9	-127.7
MASAC-MIX	-168.5	-155.6	-146.5	-141.4
Attn-MIX	-164.1	-150.4	-142.0	-136.4
MAMBPO	-156.6	-132.5	-127.5	**-124.2**
LaVa-NE-1-step	-156.6	-131.2	-127.5	-125.3
LaVa-1-step	-148.8	-130.2	-127.1	-125.1
LaVa-10-step	**-140.0**	**-129.1**	**-126.9**	**-124.2**

Episodes:	1250	2500	3750	5000
MADDPG	10.1	18.9	62.3	109.1
MASAC	14.1	22.4	32.3	51.8
MASAC-MIX	**16.8**	36.4	68.4	111.9
Attn-MIX	11.5	34.3	82.7	122.0
MAMBPO	15.6	**112.6**	120.5	115.7
LaVa-NE-1-step	12.33	27.82	113.6	127.1
LaVa-1-step	8.83	32.5	110.2	153.5
LaVa-10-step	13.5	57.5	**131.7**	**154.3**

Table 1: Average returns on Cooperative Navigation (**left**) and Predator Prey (**right**)

The input to each actor is o_t^i. We do not condition policies on latent states to prevent over-optimistic behavior due to learning Q-values for policies with full access to latent states [3]. This would also violate CTDE.

4 Experiments and Results

We tested performance on the fully cooperative continuous control tasks **Cooperative Navigation** (3 agents and 3 landmarks) and **Predator Prey** (3 predators, 2 landmarks and 1 prey) from the Multi-agent Particle Environment (MPE) [5]. As baselines, we used **(i) MASAC**[1]: To provide a direct comparison without a dynamics model **(ii) MADDPG**[5]: A state-of-the-art, model-free, deterministic, off-policy MARL algorithm **(iii) MASAC-MIX**: MASAC with a factored critic [7] for monotonic value function decomposition **(iv) Attention(Attn)-MIX**[1]: We added the attention module used in [1] to MASAC-MIX, **(v) MAMBPO**[10]: MAMBPO uses a non-Markovian dynamics model to generate synthetic samples in the real observation space, using MASAC for MARL, and **(vi) LaVa-NE**: LaVa without the exploration ensemble.

We used a Variational Auto Encoder (VAE) [2] for implementing our latent variable dynamics model. We set β to 1 and κ to 0 for the first 3000 timesteps. During this phase, the joint policy is essentially a joint exploration policy, receiving a team reward of r_{exp}. After this, we decreased β to 0, while increasing κ to 1 so that the dynamics model adjusted to the MARL task. Each task was run for 5000 episodes, with each episode running for T = 25 timesteps.

For a fair comparison with MAMBPO, we ensured that both LaVa and MAMBPO used MASAC for policy optimization. As with our model-free baselines, for each environment step, we ran a version of LaVa that performed 1 policy update (LaVa-1-step). MAMBPO performed 10 policy updates per environment step. For comparison, we also ran LaVa-10-step that performed 10 policy updates per environment step. Each algorithm was trained across 5 independent runs.

Performance of LaVa: As seen in Table 1, **LaVa outperformed all baselines in terms of final returns and sample efficiency**. LaVa-10-step was about **2x faster** on Cooperative Navigation and over **3x faster** on Predator Prey than MASAC, its corresponding model-free MARL algorithm. Addressing representation learning separately using a dynamics model did indeed accelerate sample efficiency and performance, as evidenced by LaVa-1-step coming second-best. LaVa-10-step perfoming best showed that training using more synthetic data improved performance even further.

Effectiveness of Exploration Scheme: We collected joint observations from states visited during environment interaction by LaVa and LaVa-NE over 350 episodes on both tasks. For LaVa, β, κ = 1,0 and for LaVa-NE, β, κ = 0,1. The data was visualized in Fig. 2 **(left)**, where the orange region represents joint observations from states visited by LaVa-NE while the blue region represents joint observations from states visited by LaVa. **With our exploration scheme, we visited a much larger region of the joint observation space in both tasks**.

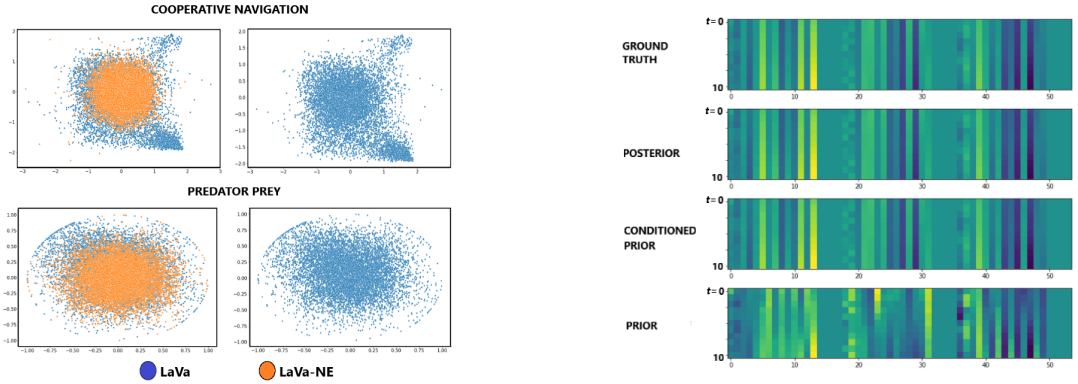


Figure 2: (**Left**) Joint observations visited by LaVa(**blue**) vs LaVa-NE(**orange**); (**Right**) A reconstructed sample trajectory. The columns represent joint observations while rows show timesteps

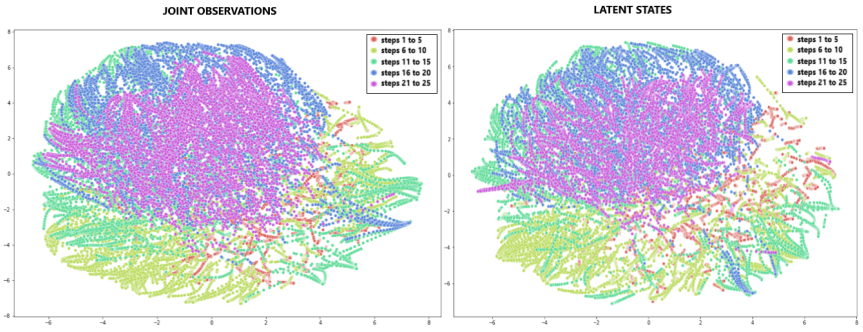


Figure 3: t-SNE plots of joint observations (**left**) and latent states (**right**) over 1000 episodes on Cooperative Navigation

Analysis of Dynamics Model: Figure 2 **(right)** shows a Cooperative Navigation trajectory of 10 timesteps and visualizes the ground truth joint observation, decoder reconstruction of samples from the posterior, conditioned prior (posterior is used in the first timestep) and prior distributions. **LaVa generated highly realistic samples using all the distributions**. We used the conditioned prior for trajectory generation in our experiments. We see from Fig. 3 **left and right** that **LaVa learnt latent states in such a way that similar latent states also corresponded to similar joint observations**, essentially learning environment dynamics in such a way that it could generate fairly accurate "latent versions" or **"shared dreams"** of the real world. Each color corresponds to 5 timesteps of an episode.

5 Conclusion

We presented a novel algorithm, LaVa, that used a latent variable dynamics model in conjunction with a CTDE exploration scheme and an MARL algorithm. We trained a dynamics model that learnt accurate models of environments in complex multi-agent tasks. We showed that our exploration scheme led to improved exploration and better model learning. Using LaVa, we were able to improve sample efficiency by 2x and 3x on our tasks and outperform state-of-the-art baselines. **We also find it interesting that LaVa learnt cooperative behaviors by training purely using latent trajectories generated by the model.** An interesting future direction would be to perform detailed empirical analysis on visual tasks with large numbers of agents, which will lead to extremely high-dimensional image inputs.

References

[1] Iqbal, S., Sha, F.: Actor-attention-critic for multi-agent reinforcement learning. In: International Conference on Machine Learning, pp. 2961–2970. PMLR (2019)

[2] Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)

[3] Lee, A., Nagabandi, A., Abbeel, P., Levine, S.: Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. Advances in Neural Information Processing Systems **33** (2020)

[4] Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Machine learning proceedings 1994, pp. 157–163. Elsevier (1994)

[5] Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Pieter Abbeel, O., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in neural information processing systems **30** (2017)

[6] Radulescu, A., Shin, Y.S., Niv, Y.: Human representation learning. Annual Review of Neuroscience **44**, 253–273 (2021)

[7] Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S.: Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International Conference on Machine Learning, pp. 4295–4304. PMLR (2018)

[8] Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., Pathak, D.: Planning to explore via self-supervised world models. In: International Conference on Machine Learning, pp. 8583–8592. PMLR (2020)

[9] Shelhamer, E., Mahmoudieh, P., Argus, M., Darrell, T.: Loss is its own reward: Self-supervision for reinforcement learning. arXiv preprint arXiv:1612.07307 (2016)

[10] Willemsen, D., Coppola, M., de Croon, G.C.: Mambpo: Sample-efficient multi-robot reinforcement learning using learned world models. arXiv preprint arXiv:2103.03662 (2021)

Equivariant Reinforcement Learning for Robotic Manipulation

Dian Wang
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115, USA
wang.dian@northeastern.edu

Robin Walters
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115, USA
r.walters@northeastern.edu

Mingxi Jia
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115, USA
jia.ming@northeastern.edu

Xupeng Zhu
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115, USA
zhu.xup@northeastern.edu

Robert Platt
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115, USA
rplatt@ccs.neu.edu

Abstract

Equivariant neural networks enforce symmetry within the structure of their convolutional layers, resulting in a substantial improvement in sample efficiency when learning an equivariant or invariant function. Such models are applicable to robotic manipulation learning which can often be formulated as a rotationally symmetric problem. This paper studies equivariant model architectures in the context of actor-critic reinforcement learning. We identify equivariant and invariant characteristics of the optimal Q-function and the optimal policy and propose Equivariant SAC algorithm that leverages this structure. We present experiments that demonstrate that our Equivariant SAC can be significantly more sample efficient than competing algorithms on an important class of robotic manipulation problems.

Keywords: Reinforcement learning, Equivariance, Robotic Manipulation

Acknowledgements

This work is supported in part by NSF 1724257, NSF 1724191, NSF 1763878, NSF 1750649, and NASA 80NSSC19K1474. R. Walters is supported by the Roux Institute and the Harold Alfond Foundation and NSF grants 2107256 and 2134178.

1 Introduction

A key challenge in reinforcement learning is to improve sample efficiency – that is to reduce the amount of environmental interactions that an agent must take in order to learn a good policy. This is particularly important in robotics applications where gaining experience potentially means interacting with a physical environment. One way of improving sample efficiency is to create "artificial" experiences through data augmentation [1, 2, 3]. These approaches implicitly assume that the transition and reward dynamics of the environment are invariant to affine transformations of the visual state.

Recent work in geometric deep learning suggests that it may be possible to learn transformation-invariant policies and value functions in a different way, using equivariant neural networks [4]. The key idea is to structure the model architecture such that it is constrained only to represent functions with the desired invariance properties. In principle, both this approach and the data augmentation approaches aim at improving sample efficiency by introducing an inductive bias. However, the equivariance approach achieves this more directly by modifying the model architecture rather than by modifying the training data. Since with data augmentation, the model must learn equivariance in addition to the task itself, more training time and greater model capacity are often required. While equivariant architectures have recently been applied to reinforcement learning, this has been done only in toy settings (grid worlds, etc.) where the model is equivariant over small finite groups, and the advantages of this approach over standard methods is less clear.

Our work [5] explores the application of equivariant methods to more realistic problems in robotics such as object manipulation. We make several contributions. First, we define and analyze an important class of MDPs that we call *group invariant MDPs*. Second, we introduce an equivariant variation of SAC [6]. Finally, we show that our methods convincingly outperform recent competitive data augmentation approaches [1, 2, 3]. Our Equivariant SAC method, in particular, outperforms these baselines so dramatically (Figure 5) that it could make reinforcement learning feasible for a much larger class of robotics problems than is currently the case. Our on-robot learning experiments demonstrate that Equivariant SAC can learn several non-trivial manipulation tasks from scratch in less than an hour of wall clock time

2 Background

C_n actions: The group C_n is the discrete subgroup of $\mathrm{SO}(2)$: $C_n = \{\mathrm{Rot}_\theta : \theta \in \{\frac{2\pi i}{n} | 0 \leq i < n\}\}$. We are interested in *actions* of C_n which formalize how vectors or feature maps transform under rotation. The group C_n acts in three ways that concern us:

1. $\mathbb{R}$ through the *trivial representation* ρ_0. Let $g \in C_n$ and $x \in \mathbb{R}$. Then $\rho_0(g)x = x$. For example, the trivial representation describes how pixel color/depth values change when an image is rotated, i.e. they do not change (Figure 1 left).

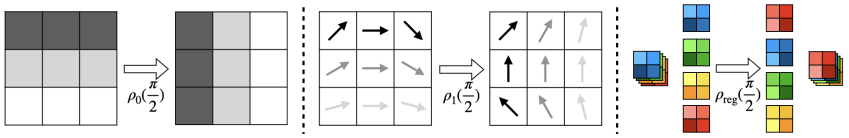


Figure 1: Illustration of how an element $g \in C_n$ acts on the feature map by rotating the pixels and transforming the channel space through ρ_0, ρ_1, and ρ_{reg}. Left: C_n acts on the channel space of a 1-channel feature map by identical mapping. Middle: C_n acts on the channel space of a vector field by rotating the vector at each pixel through ρ_1. Right: C_n acts on the channel space of a 4-channel feature map by permuting the order of the channels by ρ_{reg}.

2. $\mathbb{R}^2$ through the *standard representation* ρ_1. Let $g \in C_n$ and $v \in \mathbb{R}^2$. Then $\rho_1(g)v = \begin{pmatrix} \cos g & -\sin g \\ \sin g & \cos g \end{pmatrix} v$. This describes how elements of a vector field change when rotated (Figure 1 middle).

3. $\mathbb{R}^n$ through the *regular representation* ρ_{reg}. Let $g = r^m \in C_n = \{1, r, r^2, \ldots, r^{n-1}\}$ and $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. Then $\rho_{reg}(g)x = (x_{n-m+1}, \ldots, x_n, x_1, x_2, \ldots, x_{n-m})$ cyclically permutes the coordinates of $\mathbb{R}^n$ (Figure 1 right).

C_n actions on vectors and feature maps: C_n acts on vectors and feature maps differently depending upon their semantics. We formalize these different ways of acting as follows. Let $\mathcal{F}\colon \mathbb{R}^2 \to \mathbb{R}^m$ be an m-channel feature map and let $V \in \mathbb{R}^{m \times 1 \times 1} = \mathbb{R}^m$ be a vector represented as a special case of a feature map with 1×1 spatial dimensions. Then g is defined to act on $\mathcal{F}$ by

$$(g\mathcal{F})(x, y) = \rho_j(g)\mathcal{F}(\rho_1(g)^{-1}(x, y)). \tag{1}$$

For a vector V (considered to be at $(x, y) = (0, 0)$), this becomes: $gV = \rho_j(g)V$. In the above, $\rho_1(g)$ rotates pixel location and $\rho_j(g)$ transforms the pixel feature vector using the trivial representation ($\rho_j = \rho_0$), the standard representation ($\rho_j = \rho_1$), the regular representation ($\rho_j = \rho_{\mathrm{reg}}$), or some combination thereof.

Equivariant convolutional layer: A C_n-equivariant layer is a function h whose output is constrained to transform in a defined way when the input feature map is transformed by a group action. Consider an equivariant layer h with an input $\mathcal{F}_{\mathrm{in}} : \mathbb{R}^2 \to \mathbb{R}^{|\rho_{\mathrm{in}}|}$ and an output $\mathcal{F}_{\mathrm{out}} : \mathbb{R}^2 \to \mathbb{R}^{|\rho_{\mathrm{out}}|}$, where ρ_{in} and ρ_{out} denote the group representations associated with $\mathcal{F}_{\mathrm{in}}$ and $\mathcal{F}_{\mathrm{out}}$, respectively. When the input is transformed, this layer is constrained to output a transformed version of the same output feature map:

$$h(g\mathcal{F}_{\mathrm{in}}) = g(h(\mathcal{F}_{\mathrm{in}})) = g\mathcal{F}_{\mathrm{out}}. \tag{2}$$

where $g \in C_n$ acts on $\mathcal{F}_{\mathrm{in}}$ or $\mathcal{F}_{\mathrm{out}}$ through Equation 1, i.e., this constraint equation can be applied to arbitrary feature maps $\mathcal{F}$ or vectors V. A linear convolutional layer h satisfies Equation 2 with respect to the group C_n if the convolutional kernel $K\colon \mathbb{R}^2 \to$

$\mathbb{R}^{|\rho_{\text{out}}|\times|\rho_{\text{in}}|}$ has the following form: $K(\rho_1(g)v) = \rho_{\text{out}}^{-1}(g)K(v)\rho_{\text{in}}(g)$. Since the composition of equivariant maps is equivariant, a fully convolutional equivariant network can be constructed by stacking equivariant convolutional layers that satisfy the constraint of Equation 2 and together with equivariant non-linearities.

3 Problem Statement

3.1 Group-invariant MDPs

In a group-invariant MDP, the transition and reward functions are invariant to group elements $g \in G$ acting on the state and action space. For state $s \in S$, action $a \in A$, and $g \in G$, let $gs \in S$ denote the action of g on s and $ga \in A$ denote the action of g on a.

Definition 3.1 (G-invariant MDP)**.** *A G-invariant MDP $\mathcal{M}_G = (S, A, T, R, G)$ is an MDP $\mathcal{M} = (S, A, T, R)$ that satisfies the following conditions:*

1. Reward Invariance: *The reward function is invariant to the action of the group element $g \in G$, $R(s, a) = R(gs, ga)$.*

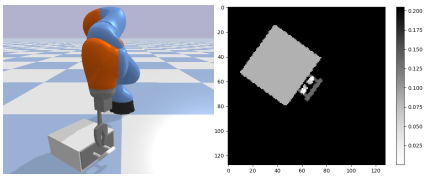
2. Transition Invariance: *The transition function is invariant to the action of the group element $g \in G$, $T(s, a, s') = T(gs, ga, gs')$.*

A key feature of a G-invariant MDP is that its optimal solution is also G-invariant (proof in [5]):

Proposition 3.1. *Let $\mathcal{M}_G$ be a group-invariant MDP. Then its optimal Q-function is group invariant, $Q^*(s, a) = Q^*(gs, ga)$, and its optimal policy is group-equivariant, $\pi^*(gs) = g\pi^*(s)$, for any $g \in G$.*

3.2 $\mathrm{SO}(2)$-invariant MDPs in Robotic Manipulation

We focus exclusively on an important class of $\mathrm{SO}(2)$-invariant MDPs in robotic manipulation. We approximate $\mathrm{SO}(2)$ by its subgroup C_n. We express the state as a depth image $\mathcal{F}_s : \mathbb{R}^2 \to \mathbb{R}$ centered on the gripper position where depth is defined relative to the gripper. The orientation of this image is relative to the base reference frame – not the gripper frame. We require the fingers of the gripper and objects grasped by the gripper to be visible in the image. Figure 2 shows an illustration. The group operator $g \in C_n$ acts on this image as defined in Equation 1 where we set $\rho_j = \rho_0$: $g\mathcal{F}_s(x, y) = \rho_0(g)\mathcal{F}_s(\rho_1(g)^{-1}(x, y))$, i.e., by rotating the pixels but leaving the depth values unchanged. The action is a tuple, $a = (a_\lambda, a_{xy}, a_z, a_\theta) \in A \subset \mathbb{R}^5$, where $a_\lambda \in A_\lambda$ denotes the commanded gripper aperture, $a_{xy} \in A_{xy}$ denotes the commanded change in gripper xy position, $a_z \in A_z$ denotes the commanded change in gripper height, and $a_\theta \in A_\theta$ denotes the commanded change in gripper orientation. Here, the xy action is equivariant with $g \in C_n$, $a_{\text{equiv}} \in A_{\text{equiv}} = A_{xy}$, and the rest of the action variables are invariant, $a_{\text{inv}} \in A_{\text{inv}} = A_\lambda \times A_z \times A_\theta$. Therefore, the rotation operator $g \in C_n$ acts on $a \in A$ via $ga = (\rho_1(g)a_{\text{equiv}}, a_{\text{inv}})$ where $a_{\text{inv}} \in A_{\text{inv}}$ and $a_{\text{equiv}} \in A_{\text{equiv}}$. Notice that the transition dynamics are C_n-invariant (i.e. $T(s, a, s') = T(gs, ga, gs')$) because the Newtonian physics of the interaction are invariant to the choice of reference frame. If we constrain the reward function to be C_n-invariant as well, then the resulting MDP is C_n-invariant.



(a) (b)

Figure 2: The manipulation scene (a) and the visual state space (b).

4 Equivariant SAC

In SAC [6], we learn the parameters for two networks: a policy network Π (the actor) and an action-value network Q (the critic). The critic $Q : S \times A \to \mathbb{R}$ approximates Q values in the typical way. However, the actor $\Pi : S \to A \times A_\sigma$ estimates both the mean and standard deviation of action for a given state. Here, we define $A_\sigma = \mathbb{R}^k$ to be the domain of the standard deviation variables over the k-dimensional action space defined in Section 3.2. Since Proposition 3.1 tells us that the optimal Q is invariant and the optimal policy is equivariant, we must model Q as an invariant network and Π as an equivariant network.

Policy network: First, consider the equivariant constraint of the policy network. The state is encoded by the image $\mathcal{F}_s$. However, we must express the action as a vector over $\bar{A} = A \times A_\sigma$. Factoring A into its equivariant and invariant components, we have $\bar{A} = A_{\text{equiv}} \times A_{\text{inv}} \times A_\sigma$. In order to identify the equivariance relation for $\bar{A}$, we must define how the group operator $g \in G$ acts on $a_\sigma \in A_\sigma$. Here, we make the simplifying assumption that a_σ is invariant to the group operator. This choice makes sense in robotics domains where we would expect the variance of our policy to be invariant to the choice of reference frame. As a result, we have that the group element $g \in G$ acts on $\bar{a} \in \bar{A}$ via:

$$g\bar{a} = g(a_{\text{equiv}}, a_{\text{inv}}, a_\sigma) = (\rho_{\text{equiv}}(g)a_{\text{equiv}}, a_{\text{inv}}, a_\sigma). \tag{3}$$

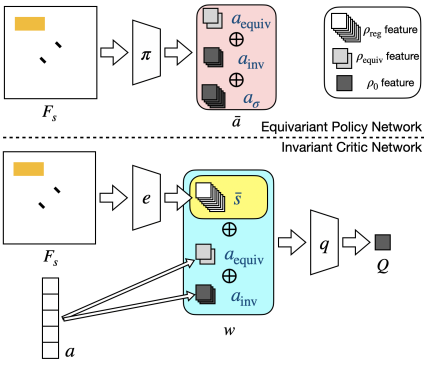
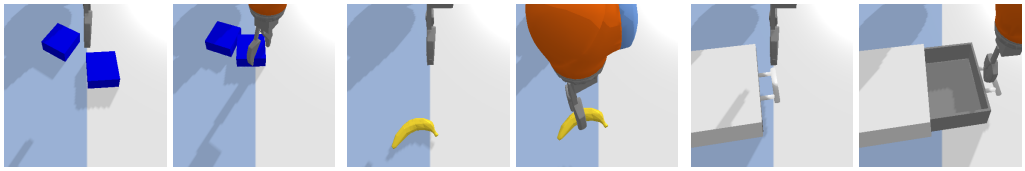
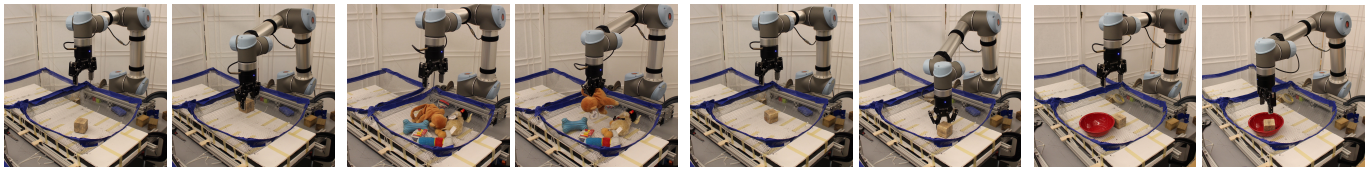


Figure 3: Illustration of the equivariant actor network (top) and the invariant critic network (bottom).



(a) Block Pulling (b) Object Picking (c) Drawer Opening



(d) Block Picking (e) Clutter Grasping (f) Block Pushing (g) Block in Bowl

Figure 4: (a)-(c): The simulated experimental environments implemented in PyBullet simulator. (d)-(g): Our on-robot learning environments. The left image in each sub figure shows an initial state of the environment; the right image shows the goal state.

We can now define the actor network π to be a mapping $\mathcal{F}_s \mapsto \bar{a}$ (Figure 3 top) that satisfies the following equivariance constraint (Equation 2):

$$\pi(g\mathcal{F}_s) = g(\pi(\mathcal{F}_s)) = g\bar{a}. \tag{4}$$

Critic network: The critic network takes both state and action as input and maps onto a real value. We define two equivariant networks: a state encoder e and a Q network q. The equivariant state encoder, e, maps the input state $\mathcal{F}_s$ onto a regular representation $\bar{s} \in (\mathbb{R}^n)^\alpha$ where each of n group elements is associated with an α-vector. Since $\bar{s}$ has a regular representation, we have $g\bar{s} = \rho_{\text{reg}}(g)\bar{s}$. Writing the equivariance constraint of Equation 2 for e, we have that e must satisfy $e(g\mathcal{F}_s) = ge(\mathcal{F}_s) = g\bar{s}$. The output state representation $\bar{s}$ is concatenated with the action $a \in A$, producing $w = (\bar{s}, a)$. The action of the group operator is now $gw = (g\bar{s}, ga)$ where $ga = (\rho_{\text{equiv}}(g)a_{\text{equiv}}, a_{\text{inv}})$. Finally, the q network maps from w onto $\mathbb{R}$, a real-valued estimate of the Q value for w. Based on proposition 3.1, this network must be invariant to the group action: $q(gw) = q(w)$. All together, the critic satisfies the following invariance equation:

$$q(e(g\mathcal{F}_s), ga) = q(e(\mathcal{F}_s), a). \tag{5}$$

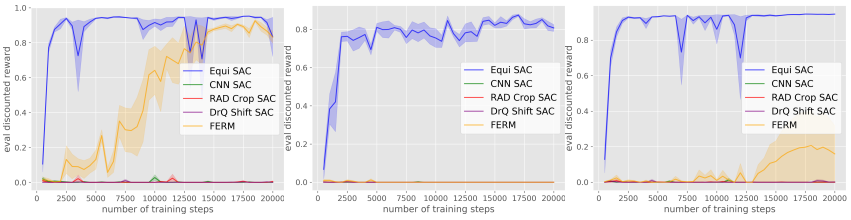
This network is illustrated at the bottom of Figure 3. For a robotic manipulation domain in Section 3.2, we have $A_{\text{equiv}} = A_{xy}$ and $A_{\text{inv}} = A_\lambda \times A_z \times A_\theta$ and $\rho_{\text{equiv}} = \rho_1$.

5 Experiments

We evaluate Equivariant SAC in the manipulation tasks shown in Figure 4. These tasks can be formulated as $\text{SO}(2)$-invariant MDPs. All environments have sparse rewards (+1 when reaching the goal and 0 otherwise).

5.1 Simulation Experiment

In this experiment, we evaluate the performance of Equivariant SAC in the simulation experiments shown in Figure 4 (a)-(c). We compare against the following baselines: 1) CNN SAC: SAC with conventional CNN rather than equivariant networks. 2) RAD Crop SAC [1]: same model architecture as CNN SAC with random crop data augmentation when sampling transitions. 3) DrQ Shift SAC [2]: same model architecture as CNN SAC with random shift data augmentation when calculating the Q-target and the loss. 4) FERM [3]: a combination of SAC, contrastive learning, and random crop augmentation. All methods use a $\text{SO}(2)$ data augmentation buffer, where every time a new transition is added, we generate 4 more augmented transitions by applying random continuous rotations to the transition. Prior to each training run, we pre-load the replay buffer with 20 episodes of expert demonstration.



(a) Block Pulling (b) Object Picking (c) Drawer Opening

Figure 5: Comparison of Equivariant SAC (blue) with baselines. The plots show the evaluation performance of the greedy policy in terms of the discounted reward. The evaluation is performed every 500 training steps. Results are averaged over four runs. Shading denotes standard error.

Task	Block Picking	Clutter Grasping	Block Pushing	Block in Bowl
Number of training steps	2000	2000	2000	4000
Approximate time for training	45 minutes	45 minutes	1 hour	2 hours 40 minutes
Evaluation success rate	100% (50/50)	96% (48/50)	92% (46/50)	92% (46/50)

Table 1: The number of training steps, approximate time for training, and the revaluation success rate of the trained policy of our on-robot learning.

Figure 5 shows the comparison among the various methods. Notice that Equivariant SAC outperforms the other methods significantly. Without the equivariant approach, Object Picking and Drawer Opening appear to be infeasible for the baseline methods. In Block Pulling, FERM is the only other method able to solve the task.

5.2 On-robot Experiment

This section evaluates the Equivariant SAC in on-robot learning. Our experimental setup is shown in Figure 6. Two bins in front of the robot make up the workspace. We experiment with four tasks (Block Picking, Clutter Grasping, Block Pushing and Block in Bowl in Figure 4 (d)-(g)). In tasks that involve a single object (Block Picking and Block Pushing), the robot will only use one bin as the workspace. In tasks involving multiple objects (Clutter Grasping and Block in Bowl), the robot will iteratively use one of the two bins as the active workspace and use the other bin to reset the environment.

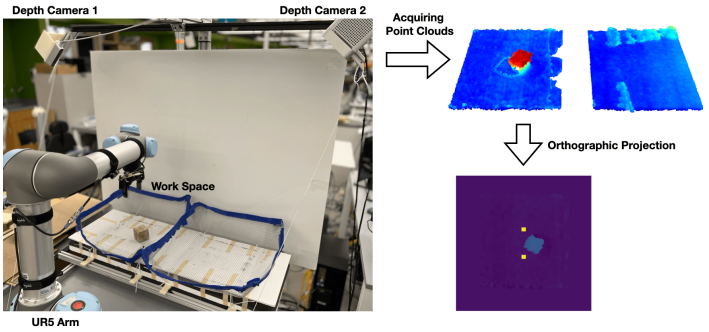


Figure 6: Our experimental set up for on-robot learning. The observation (bottom right) is generated by first acquiring point clouds from two depth cameras above the workspace then creating an orthographic projection at the gripper's position. The gripper is drawn at the center of the observation (in yellow) with its current aperture and orientation.

Table 1 shows the statistics of our on-robot learning. In Block Picking, Clutter Grasping, and Block Pushing, the Equivariant SAC only requires 2000 steps for learning the policy, which is approximately 45 minutes in total, including the time required for resetting the environment. In Block in Bowl, Equivariant SAC takes 4000 steps to converge, which is 2 hours and 40 minutes. We evaluate our trained policy for each task for 50 episodes. In Block Picking, the robot succeeds in all trails. In Clutter Grasping, the robot reaches a 96% success rate. In Block Pushing and Block in Bowl, the robot demonstrates a 92% test success rate.

6 Discussion

We define a class of group-invariant MDPs and identify the invariance and equivariance characteristics of their optimal solutions. We further propose Equivariant SAC that encodes the equivariance in the structure of the actor and critic networks. We show experimentally in the robotic manipulation domains that our proposal substantially surpasses the performance of competitive baselines. A key limitation of this work is that our definition of G-invariant MDPs requires the MDP to have an invariant reward function and invariant transition function. Though such restrictions are often applicable in robotics, they limit the potential of the proposed methods in other domains like some ATARI games. Furthermore, if the observation is from a non-top-down perspective, or there are non-equivariant structures in the observation (e.g., the robot arm), the invariant assumptions of a G-invariant MDP will not be directly satisfied.

References

[1] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *arXiv preprint arXiv:2004.14990*, 2020.

[2] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," *arXiv preprint arXiv:2004.13649*, 2020.

[3] A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "A framework for efficient robotic manipulation," *arXiv preprint arXiv:2012.07975*, 2020.

[4] T. S. Cohen and M. Welling, "Steerable cnns," *arXiv preprint arXiv:1612.08498*, 2016.

[5] D. Wang, R. Walters, and R. Platt, "SO(2)-equivariant reinforcement learning," in *International Conference on Learning Representations*, 2022.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

Enforcing Delayed-Impact Fairness Guarantees

Aline Weber*
Manning College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01002
alineweber@cs.umass.edu

Blossom Metevier*
Manning College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01002
bmetevier@cs.umass.edu

Yuriy Brun
Manning College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01002
brun@cs.umass.edu

Philip S. Thomas
Manning College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01002
pthomas@cs.umass.edu

Bruno Castro da Silva
Manning College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01002
bsilva@cs.umass.edu

Abstract

Recent research has shown that seemingly fair machine learning models, when used to inform decisions that have an impact on peoples' lives or well-being (e.g., applications involving education, employment, and lending), can inadvertently increase social inequality in the long term. This is because prior fairness-aware algorithms only consider static fairness constraints, such as equal opportunity or demographic parity. However, enforcing constraints of this type may result in models that have negative delayed impact on disadvantaged individuals and communities. We introduce ELF (Enforcing Long-term Fairness), the first algorithm that provides high-confidence fairness guarantees in terms of delayed impact, using importance sampling techniques similar to those in the offline reinforcement learning literature. We prove that ELF will not return an unfair solution with probability greater than a user-specified tolerance. Furthermore, we show (under mild assumptions) that given sufficient training data, ELF is able to find and return a fair solution if one exists. We show experimentally that ELF can successfully mitigate long-term unfairness.

Keywords: Fair machine learning, delayed impact, fair classification

*Equal contribution.

1 Introduction

The use of machine learning for high-stakes applications such as lending, hiring, and criminal sentencing has the potential to harm historically disadvantaged communities [5, 3, 2]. For example, software meant to guide bank decisions in lending has been shown to exhibit racial bias [2]. Consequently, extensive research has been devoted to designing algorithms that promote fairness and ameliorate concerns of bias and discrimination for socially impactful applications. The bulk of this research has focused on the classification setting, in which *static* fairness definitions, i.e., fairness definitions that rely on statistical metrics such as true and false positive rates, are studied. However, it has been shown that model decisions that appear fair with respect to static fairness measures can nevertheless negatively affect the community they aim to protect in the long-term. For example, consider a bank lending setting, where the repayment predictions influence lending decisions and can change the future financial stability of different groups (e.g. by not getting a loan, the financial stability of a person may decay drastically). When a subset of the population is disadvantaged, instead of maximizing profit, the bank may want (or be required by law) to maximize profit subject to a fairness constraint that considers the delayed impact of model predictions in terms of the borrowers' future financial stability. Work that enforces long-term, or *delayed-impact* (DI) constraints when the relationship between predictions and DI is known has been proposed [7], but designing algorithms that mitigate negative delayed impact when the relationship between predictions and DI is not known has remained an open problem.

In this paper, we develop the first classification algorithm that can ensure with high probability that the classifiers it learns are fair with respect to delayed impact when the relationship between predictions and DI is not known *a priori*. To accomplish this, we simultaneously formulate the fair classification problem as both a classification and reinforcement learning problem—classification for optimizing the primary objective (a measure of classification loss) and reinforcement learning when considering DI. Specifically, we use importance sampling techniques similar to those in the offline reinforcement learning literature [8], and make use of confidence intervals for the mean [9] to derive a method for computing high probability bounds on DI fairness.

2 Problem Statement

We now formalize the problem of classification with delayed-impact fairness guarantees. As in the standard classification setting, a dataset consists of n data points, the i^{th} of which contains X_i, a feature vector describing a person, and a label Y_i. Each data point also contains a set of *sensitive attributes*, such as race and gender. Though our algorithm works with an arbitrary number of such attributes, for brevity our notation uses a single attribute, T_i. We assume that each data point also contains a prediction $\widehat{Y}_i^\beta$ made by a stochastic model β. We call β the *behavior model*, defined as $\beta(x, \hat{y}) \coloneqq \Pr(\widehat{Y}_i^\beta{=}\hat{y}|X_i{=}x)$. The predictions made by a model deployed in the real-world can have long-term, or delayed, impact. For example, by influencing who gets a loan, a model's predictions can affect applicants' long-term net worth. Formally, let I_i^β be a measure of the *delayed impact* resulting from deploying β for the person described by the i^{th} data point. We assume that larger values of I_i^β correspond to better delayed impact. We append I_i^β to each data point, and thus define the dataset to be a sequence of n independent and identically distributed (i.i.d.) data points $D \coloneqq \{(X_i, Y_i, T_i, \widehat{Y}_i^\beta, I_i^\beta)\}_{i=1}^n$. For notational clarity, when referring to an arbitrary data point, we write $X, Y, T, \widehat{Y}^\beta$ and I^β without subscripts to denote $X_i, Y_i, T_i, \widehat{Y}_i^\beta$ and I_i^β, respectively.

Given a dataset D, the goal is to construct a classification algorithm that takes as input D and outputs a new model π_θ that is as accurate as possible while enforcing constraints on delayed impact. This new model π_θ is of the form $\pi_\theta(x, \hat{y}) \coloneqq \Pr(\widehat{Y}^{\pi_\theta}{=}\hat{y}|X{=}x)$, where π_θ is parameterized by a vector $\theta \in \Theta$, for some feasible set Θ, and where $\widehat{Y}^{\pi_\theta}$ is the prediction made by π_θ given X. Like I_i^β, let $I_i^{\pi_\theta}$ be the delayed impact if the model outputs the prediction $\widehat{Y}_i^{\pi_\theta}$.

We model the setting in which a classification model's prediction depends only on the feature vector X, formalized by the assumption that for all x, t, y, and $\hat{y}$, $\Pr(\widehat{Y}^{\pi_\theta}{=}\hat{y}|X{=}x, Y{=}y, T{=}t) = \Pr(\widehat{Y}^{\pi_\theta}{=}\hat{y}|X{=}x)$. We also assume that regardless of the model used to make predictions, the distribution of delayed impact given a prediction remains the same: $\forall x, y, t, \hat{y}, i$, $\Pr(I^\beta{=}i|X{=}x, Y{=}y, T{=}t, \widehat{Y}^\beta{=}\hat{y}) = \Pr(I^{\pi_\theta}{=}i|X{=}x, Y{=}y, T{=}t, \widehat{Y}^{\pi_\theta}{=}\hat{y})$.

Our problem setting can alternatively be described from the reinforcement learning perspective, where feature vectors are the states of a Markov decision process (specifically, a contextual bandit), predictions are the actions taken by an agent, and DI is the reward received after the agent takes an action (makes a prediction) given a state (feature vector). From this perspective, the latter assumption asserts that regardless of the strategy (model) used to choose actions, the distribution of rewards given an action remains the same.

We consider k *delayed-impact objectives* $g_j : \Theta \to \mathbb{R}, j \in \{1, ..., k\}$ that take as input a parameterized model θ and return a real-valued measurement of fairness in terms of delayed impact. We adopt the convention that $g_j(\theta) \leq 0$ iff θ causes behavior that is fair with respect to delayed impact, and $g_j(\theta) > 0$ otherwise. To simplify notation, we assume there

exists only a single DI objective (i.e., $k = 1$). We focus on the case in which each DI objective is based on a conditional expected value, having the form $g(\theta) \coloneqq \tau - \mathbf{E}[I^{\pi_\theta}|c(X,Y,T)]$, where $\tau \in \mathbb{R}$ is a tolerance, and $c(X,Y,T)$ is a Boolean conditional relevant to defining the objective. Consider an example in which a bank determines whether to provide a loan to an applicant. We assume that each individual in the population of loan applicants is associated with type A or B, e.g., A and B can represent different genders. The bank is interested in enforcing a fairness definition that protects type A applicants. Specifically, the bank would like to ensure that, for a model π_θ being considered, the future financial status of applicants of type A impacted by π_θ's lending predictions does not decline relative to those induced by the previously deployed model, β. In this case, I^{π_θ} is the financial status of an applicant t months after the loan application and $c(X,Y,T)$ is the Boolean event that an applicant is of type A. Lastly, τ could represent a threshold on which the bank would like to improve, e.g., the average financial status of applicants in the disadvantaged group given the historical data collected using β. The bank is therefore interested in enforcing the following DI objective: $\mathbf{E}[I^{\pi_\theta}|T = A] \geq \tau$. Then, defining $g(\theta) = \tau - \mathbf{E}[I^{\pi_\theta}|T = A]$ ensures that $g(\theta) \leq 0$ iff the new model π_θ satisfies the DI objective. Note that an additional constraint of the same form can be added to protect group B.

Algorithmic properties of interest. We would like to ensure that $g(\theta) \leq 0$, where θ is the model returned by a classification algorithm. However this is often not possible, as it requires highly accurate assumptions of prior knowledge about how predictions influence delayed impact. Instead, we aim to create an algorithm that uses data to reason about its confidence that $g(\theta) \leq 0$. That is, we desire a classification algorithm, a, where $a(D) \in \Theta$ is the solution provided by the algorithm when given dataset D as input, that satisfies DI constraints of the form

$$\Pr(g(a(D)) \leq 0) \geq 1 - \delta, \tag{1}$$

where $\delta \in (0,1)$ limits the admissible probability that the algorithm returns a model that is unfair with respect to the DI objective. Algorithms that satisfy (1) are called Seldonian [10]. In practice, there might be constraints that are impossible to enforce [6] or the amount of data may be insufficient to ensure fairness with high confidence. In such cases, instead of returning a solution the algorithm does not trust, the algorithm should return "No Solution Found" (NSF). Let $\texttt{NSF} \in \Theta$ and $g(\texttt{NSF}) = 0$, indicating that it is always fair for the algorithm to say "I'm unable to ensure fairness with the required confidence."

3 Methods for Enforcing Delayed Impact

The distribution of delayed impacts in D is a result of using the model β to make predictions. However, we are interested in evaluating the DI of a different model, π_θ. This presents a challenging problem: given data that includes the DI when a model β was used to make predictions, how can we estimate the DI if π_θ were used instead?

We solve this problem using techniques from the reinforcement learning literature called *off-policy evaluation* methods—methods that use data from running one *policy* (decision-making model) to predict what would happen (in the long-term) if a different policy were used to make decisions. Specifically, we use an off-policy evaluation method called *importance sampling* [8] to obtain a new random variable $\hat{I}^{\pi_\theta}$, constructed using data from β, such that $\mathbf{E}[\hat{I}^{\pi_\theta}|c(X,Y,T)] = \mathbf{E}[I^{\pi_\theta}|c(X,Y,T)]$. For each data point, the importance sampling estimator, $\hat{I}^{\pi_\theta}$, weights the observed delayed impacts I^β based on how likely the prediction $\widehat{Y}^\beta$ is under π_θ. If π_θ would make the label $\widehat{Y}^\beta$ more likely, then I^β is given a larger weight (at least one), and if π_θ would make $\widehat{Y}^\beta$ less likely, then I^{π_θ} is given a smaller weight (positive, but less than one). Formally, the importance sampling estimator is $\hat{I}^{\pi_\theta} = \pi_\theta(X,\widehat{Y}^\beta)\big(\beta(X,\widehat{Y}^\beta)\big)^{-1} I^\beta$, where the term $\pi_\theta(X,\widehat{Y}^\beta)/\beta(X,\widehat{Y}^\beta)$ is called the *importance weight*. This particular weighting scheme is chosen to ensure that $\hat{I}^{\pi_\theta}$ is an unbiased estimator of I^{π_θ}, which can be proven under the assumption that the model π_θ can only select labels for which there is *some* probability of the behavior model selecting.

Bounds on delayed impact. Given unbiased estimates of I^{π_θ}, computed according to the scheme discussed above, we can construct unbiased estimates of $g(\theta)$ by subtracting each estimate of I^{π_θ} from τ, the user-defined tolerance. We now discuss how to use these estimates of $g(\theta)$, along with confidence intervals for the mean, to derive high confidence upper bounds on $g(\theta)$. Given a vector of m i.i.d. samples $(Z_i)_{i=1}^m$ of a random variable Z, let $\bar{Z} = \frac{1}{m}\sum_{i=1}^m Z_i$ be the sample mean, let $\sigma(Z_1,...,Z_m) = \sqrt{\frac{1}{m-1}\sum_{i=1}^m (Z_i - \bar{Z})^2}$ be the sample standard deviation (with Bessel's correction), and let $\delta \in (0,1)$ be a confidence level. From Student [9], we have the property that if $\sum_{i=1}^m Z_i$ is normally distributed, then $\Pr\left(\mathbf{E}[Z_i] \geq \bar{Z} - \frac{\sigma(Z_1,\ldots,Z_m)}{\sqrt{m}} t_{1-\delta,m-1}\right) \geq 1 - \delta$, where $t_{1-\delta,m-1}$ is the $1-\delta$ quantile of the Student's t distribution with $m-1$ degrees of freedom. We can use this property to obtain a high-confidence upper bound for the mean of Z : $U_{\texttt{ttest}}(Z_1,\ldots,Z_m) = \bar{Z} + \frac{\sigma(Z_1,\ldots,Z_m)}{\sqrt{m}} t_{1-\delta,m-1}$.

Let $\hat{g}$ be a vector of i.i.d. and unbiased estimates of $g(\theta)$ such that the sample mean of $\hat{g}$ is normally distributed. These estimates can be provided to $U_{\texttt{ttest}}$ to derive a high-confidence upper bound on $g(\theta)$: $\Pr(\tau - \mathbf{E}[\hat{I}^{\pi_\theta}|c(X,Y,T)] \leq U_{\texttt{ttest}}(\hat{g})) \geq 1 - \delta$.

Complete algorithm. Our algorithm (Algorithm 1) has three main steps. In the first step, the dataset D is divided into two datasets, D_c and D_f (line 1). In the second step (line 3), which we refer to as *candidate selection*, D_c is used to find and train a model, called the *candidate solution*, θ_c. The last step (lines 4–9) is the *fairness test*, in which D_f is used to compute a $(1-\delta)$-confidence upper bound on $g(\theta_c)$ and determine whether `NSF` or the candidate solution should be returned.

In particular, in the fairness test, unbiased estimates of $g(\theta_c)$ are calculated using the importance sampling method described previously (lines 4–7). These estimates are used to calculate a high-confidence upper bound, U, on $g(\theta_c)$ using Student's t-test (line 9). Finally, if U is below 0, then the solution θ_c is returned. If not, the algorithm returns `NSF`. In candidate selection, a similar strategy is used to calculate the cost of a potential solution θ. Again, unbiased estimates of $g(\theta)$ are calculated (Algorithm 2 lines 2–6), this time using D_c. Instead of calculating a high confidence upper bound on $g(\theta)$ using Student's t-test, we calculate an *inflated* upper bound (Algorithm 2 lines 7–8). Our choice to inflate the confidence interval is empirically driven and was first proposed for other Seldonian algorithms [10]. Finally, if the inflated upper bound is higher than a small negative constant $(-\xi/4)$, the cost associated with the loss of θ, $\hat{\ell}(\theta, D_c)$, is returned. Otherwise, the cost of θ is defined as the sum of the inflated upper bound and the maximum loss that can be obtained using D_c (Algorithm 2 lines 9–10). This discourages candidate selection from returning models unlikely to pass the fairness test.

Algorithm 1 `ELF`$(D, c, \delta, \tau, \beta)$

1: $D_c, D_f \leftarrow$ `partition`(D)
2: $n_{D_f} =$ `length`(D_f); $\quad \hat{g} \leftarrow \langle\,\rangle$
3: $\theta_c \leftarrow \arg\min_{\theta\in\Theta}$ `cost`$(\theta, D_c, c, \delta, \tau, \beta, n_{D_f})$
4: **for** $i \in \{1, ..., n\}$ **do**
5: **if** $c(X_i, Y_i, T_i)$ is `True` **then**
6: $\hat{g}$.append$\Big(\tau - \frac{\pi_{\theta_c}(X_i, \widehat{Y}_i^{\beta})}{\beta(X_i, \widehat{Y}_i^{\beta})} I_i^{\beta}\Big)$
7: **end if**
8: **end for**
9: **if** $U_{\texttt{ttest}}(\hat{g}) \geq 0$ **then return** `NSF` **else return** θ_c

4 Empirical Evaluation

To empirically evaluate our method, we consider a classifier tasked with making predictions about people in the United States foster care system; for example, whether youth currently in foster care are likely to get a job in the near future. These predictions may have a delayed impact on the person's life if, for instance, they influence whether that person receives additional financial aid. Here the goal is to ensure that a trained classifier is fair with respect to delayed impact when considering race. Our experiments use two data sources from the National Data Archive on Child Abuse and Neglect [4]: *(i)* the Adopting and Foster Care Analysis and Reporting System—a dataset containing demographic and foster care-related information about youth; and *(ii)* the National Youth in Transition Database (Services and Outcomes)—a dataset containing information about the well-being, financial, and educational status of youth over time and during their transition from foster care to independent adulthood. We wish to guarantee with high probability that the DI caused by a new classifier, π_θ, is better than the DI resulting from the currently-deployed classifier, β. This guarantee should hold simultaneously for both races: White (instances where $T = 0$) and Black (instances where $T = 1$). In the following experiments, the confidence levels δ_0 and δ_1, associated with these objectives, are both set to 0.1.

Preventing Delayed-Impact Unfairness. We first evaluate whether `ELF` can prevent DI unfairness with high probability, and whether existing algorithms fail. We compare `ELF` with a fairness-unaware algorithm (logistic regression (LR)) and three state-of-the-art fairness-aware algorithms: *(i)* Fairlearn [1], *(ii)* Fairness Constraints [11], and *(iii)* quasi-Seldonian algorithms (QSA) [10] designed to enforce static fairness constraints. We consider five static fairness constraints: demographic parity (DP), equalized odds (EqOdds), disparate impact (DisImp), equal opportunity (EqOpp), and predictive equality (PE).

Algorithm 2 `cost`$(\theta, D_c, c, \delta, \tau, \beta, n_{D_f})$

1: $\hat{g} \leftarrow \langle\,\rangle$
2: **for** $i \in \{1, ..., m\}$ **do**
3: **if** $c(X_i, Y_i, T_i)$ is `True` **then**
4: $\hat{g}$.append$\Big(\tau - \frac{\pi_{\theta}(X_i, \widehat{Y}_i^{\beta})}{\beta(X_i, \widehat{Y}_i^{\beta})} I_i^{\beta}\Big)$
5: **end if**
6: **end for**
7: Let $\lambda = 2$; $\quad n_{\hat{g}} =$ `length`$(\hat{g})$
8: $U^+ = \frac{1}{n_{\hat{g}}}\left(\sum_{\iota=1}^{n_{\hat{g}}} \hat{g}_\iota\right) + \lambda \frac{\sigma(\hat{g})}{\sqrt{n_{D_f}}} t_{1-\delta, n_{D_f}-1}$
9: $\ell_{\max} = \max_{\theta'\in\Theta} \hat{\ell}(\theta', D_c)$
10: **if** $U^+ \leq -\frac{\xi}{4}$ **return** $\hat{\ell}(\theta, D_c)$ **else return** $(\ell_{\max} + U^+)$

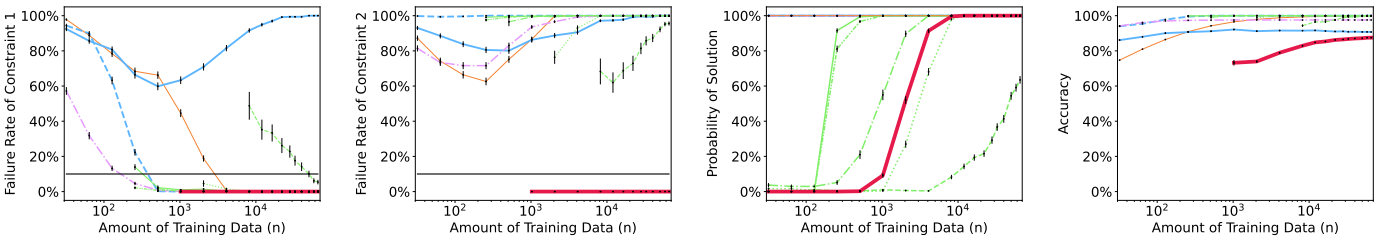
In this comparison, we investigate how often each fairness-aware algorithm returns an unfair model (with respect to the DI constraints) as a function of the amount of training data. We refer to the probability that an algorithm returns an unfair model as its *failure rate*. To measure the failure rate, we compute how often the classifiers returned by each algorithm are unfair when evaluated on a significantly larger dataset, to which the algorithms do not have access during training time. Figures 1a and 1b present the failure rate of each algorithm as a function of the amount of available training data. We computed all failure rates and corresponding standard errors over 500 trials. Notice that the solutions returned by `ELF` are *always fair* with respect to the DI constraints.[1] Existing methods that enforce static fairness criteria, by contrast, either *(i) always fail* to satisfy both

[1] `ELF` does not return solutions if trained with $n < 1{,}000$ data points because it cannot ensure DI fairness with high confidence.

DI constraints, independently of the amount of training data; or *(ii) always fail* to satisfy one of the DI constraints—the one related to delayed impact on Black youth.

The Cost of Ensuring Delayed-Impact Fairness. The previous analyses show that ELF is capable of satisfying DI constraints with high probability, but this often comes at a cost. First, there may be a trade-off between the amount of training data and the confidence that a fair solution has been identified. Recall that some algorithms (including ours) may not return a solution if they cannot ensure fairness with high confidence. Therefore, we study how often each algorithm identifies and returns a candidate solution as a function of n. Figure 1c shows that as the amount of training data increases, the probability of ELF returning solutions increases rapidly. Although some competing techniques always return solutions, or may require less training data than ELF, these solutions never satisfy both DI constraints.

Secondly, there may be a trade-off between satisfying fairness constraints and optimizing accuracy. Figure 1d presents the accuracy of classifiers returned by different algorithms as a function of n. Even though there is an accuracy gap of approximately 10% in the limit, ELF *always* returns fair solutions, while other methods fail to satisfy at least one DI constraint. *While there is a cost to enforcing DI constraints, ELF succeeds in its main objectives: to ensure DI fairness with high probability, without requiring unreasonable amounts of data, and with no significant loss of accuracy.*



(a) Failure rate 1. (b) Failure rate 2. (c) Prob. of returning solution. (d) Accuracy.

Figure 1: Algorithms' failure rates with respect to the DI constraints associated with White people *(a)* and Black people *(b)*, as a function of n. The black horizontal lines indicate the maximum admissible probability of unfairness, $\delta_0 = \delta_1 = 10\%$. In *(c)* we show the probability that algorithms (subject to different fairness constraints) return a solution as a function of n. Finally, in *(d)* we show the accuracy of the solutions returned by algorithms (subject to different fairness constraints) as a function of n. All plots use the following legend: ELF LR QSA with DP QSA with EqOdds QSA with EqOpp QSA with PE QSA with DisImp Fairlearn with DP Fairlearn with EqOdds Fairness Constraints.

References

[1] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69. PMLR, 2018.

[2] Robert Bartlett, Adair Morse, Richard Stanton, and Nancy Wallace. Consumer-lending discrimination in the fintech era. *Journal of Financial Economics*, 2021.

[3] Joseph Blass. Algorithmic advertising discrimination. *Northwestern University Law Review*, 114:415–468, 2019.

[4] Children's Bureau, Administration on Children, Youth and Families. National Data Archive on Child Abuse and Neglect (NDACAN). 2021.

[5] Alexandre Flage. Ethnic and gender discrimination in the rental housing market: Evidence from a meta-analysis of correspondence tests, 2006–2017. *Journal of Housing Economics*, 41:251–273, 2018.

[6] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

[7] Lydia T Liu, Sarah Dean, Esther Rolf, Max Simchowitz, and Moritz Hardt. Delayed impact of fair machine learning. In *International Conference on Machine Learning*, pages 3150–3158. PMLR, 2018.

[8] D. Precup, R. S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning*, pages 417–424, 2001.

[9] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[10] Philip S Thomas, Bruno Castro da Silva, Andrew G Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

[11] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*, pages 962–970. PMLR, 2017.

Analyzing and Overcoming Degradation in Warm-Start Off-Policy Reinforcement Learning

Benjamin Wexler
Department of Computer Science
Bar-Ilan University
Ramat-Gan, Israel
benwex93@gmail.com

Elad Sarafian
Department of Computer Science
Bar-Ilan University
Ramat-Gan, Israel
elad.sarafian@gmail.com

Sarit Kraus
Department of Computer Science
Bar-Ilan University
Ramat-Gan, Israel
sarit@cs.biu.ac.il

Abstract

Reinforcement Learning (RL) can benefit from a warm-start where the agent is initialized with a pretrained behavioral policy. However, when transitioning to RL updates, degradation in performance can occur, which may compromise the agent's safety. This degradation, which constitutes an inability to properly utilize the pretrained policy, is attributed to extrapolation error in the value function, a result of high values being assigned to Out-Of-Distribution actions not present in the behavioral policy's data. We investigate why the magnitude of degradation varies across policies and why the policy fails to quickly return to behavioral performance. We present visual confirmation of our analysis and draw comparisons to the Offline RL setting which suffers from similar difficulties. We propose a novel method, Confidence Constrained Learning (CCL) for Warm-Start RL, that reduces degradation by balancing between the policy gradient and constrained learning according to a confidence measure of the Q-values. For the constrained learning component we propose a novel objective, Positive Q-value Distance (CCL-PQD). We investigate a variety of constraint-based methods that aim to overcome the degradation, and find they constitute solutions for a multi-objective optimization problem between maximimal performance and miniminal degradation. Our results demonstrate that hyperparameter tuning for CCL-PQD produces solutions on the Pareto Front of this multi-objective problem, allowing the user to balance between performance and tolerable compromises to the agent's safety.

Keywords: Reinforcement Learning, Warm Start, Degradation, Extrapolation Error, Offline RL

Behavioural signatures of hierarchical task representation during sequential decision making

Sven Wientjes*
Department of Experimental Psychology
Ghent University
Henri Dunantlaan 2, 9000 Ghent
wientjes.s@gmail.com

Clay B. Holroyd
Department of Experimental Psychology
Ghent University
Henri Dunantlaan 2, 9000 Ghent
clay.holroyd@ugent.be

Abstract

Humans have the ability to craft abstract, temporally extended and hierarchically organized plans. For instance, when considering how to make a pasta dish for dinner, we typically concern ourselves with useful 'subgoals' in the task, such as cutting onions, boiling pasta, and cooking a sauce, rather than particulars such as how many cuts to make to the onion, or exactly which muscles to contract. A core question is how such decomposition of a more abstract task into logical subtasks happens in the first place.

Previous research has shown how neural responses and reaction times can be sensitive to hierarchical structure in the environment. It remains to be seen how such learned structure can be put toward goal-directed behavior. To investigate this, we developed a novel goal-directed navigation task in a hierarchical environment. Goal locations vary throughout the environment, so participants had to learn its structure. Participants had agency over the general direction they would move in, but the actual progression through the environment was still partially random. Participants were never given an overview of the environment, so they had to learn through observation and plan their moves using an internal model. Using Bayesian model comparison, we found that participants are sensitive to the hierarchical organization of the environment, and that the Successor Representation can explain their behavior better than perfect model-based agents or explicitly hierarchically structured internal models.

These results open up the possibility to use this novel task to investigate hierarchically structured prediction errors and representations in future neuroimaging work.

Keywords: Sequential decision-making, Behavioral modeling, Successor representation, Community structure, Statistical learning

*web: https://users.ugent.be/~swientje/
twitter: https://twitter.com/SvenWientjes

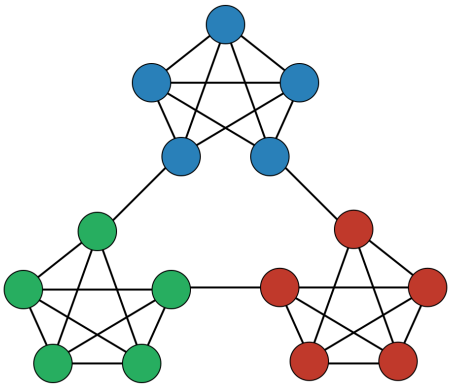
1 Introduction

Temporally extended tasks often consist of several subtasks that need to be completed in succession. Previous research has shown that participants are sensitive to the subtask-structure of certain problems [5] [8] [10]. One question this left open, is how exactly this structure is learned. From the perspective of statistical learning, decomposing an entire task into several subtasks corresponds to finding abstract structure within a large and complex space of possible actions and sensory observations. Learning such structure can be formalized as 'higher-order graph learning' [3]. Innovative research has shown certain parts of the brain are sensitive to exactly such higher-order learning [7], as well as its influence on reaction times in simple motor sequencing tasks [4]. However, to our knowledge, no single study so far has linked the learning signals required for higher-order graph learning directly to its behavioural relevance for task execution. We designed a novel sequential decision-making task where the behavioural strategy can be informed differently by a lower-level or a higher-level task representation. The outcomes are not fully deterministic, so we can investigate prediction-error like effects while participants are learning and executing the task.

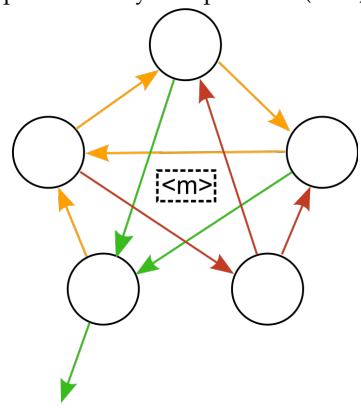
2 Methods

2.1 Task Design

We developed a novel hierarchical goal-directed task where the participant has to explore and navigate through a virtual museum consisting of multiple rooms. Each room contained a single unique painting. Because some of the rooms are clustered together into 'wings' of the museum, subjects can learn there is a hierarchical structure. Participants were given a cover story telling them they were a tour guide in this museum and had to guide visitors to specific paintings they requested to see. In total there were 15 rooms in the museum, following the layout of the graph used by [7] shown in Figure 1a. The experiment was divided into 'miniblocks' where the participant had to find a specific painting for each miniblock. The timeline of a typical miniblock is shown in Figure 2. The participant could navigate through the rooms by pressing the <z> or <m> key to indicate in which direction to move next. Once the participant reached the current goal painting, they had to indicate this by pressing <space>. If they did this correctly, they would receive a reward of £0.15. If they did this incorrectly, either by pressing <space> before they reached the goal, or by pressing <z> or <m> when they were at the goal, they would lose £0.02. The miniblock would always and only end when the current goal was reached, regardless of the keys that were pressed. The end of a miniblock also served as the start of the next miniblock, so the participants experienced one continuous walk throughout the museum over the course of the entire experiment.



(a) Layout of the museum, following the graph as used by Schapiro et al (2013) [7]



(b) Illustration of possible outcomes when pressing <m>. The outcomes for <z> will be the mirror image of this.

Figure 1: Illustrations of task environment and action-outcome mapping.

Each of the two keys mapped to two edges per node, and the mapping is symmetrical across the 3 different clusters. The mapping for a single cluster is shown in Figure 1b. When the participant selects a key, the next state is sampled with 50% chance by following one of the two edges mapped to that key. Critically, the mapping was designed to afford a hierarchical choice policy, as each key only allows for wing-transitions in one direction (the green arrows). This allows for directed 'rotation' through the wings of the museum, but without direct control over which precise rooms would show up. Always selecting the key that points in the correct rotation is much better than following a random policy, but an optimal model-based agent with perfect knowledge would follow a policy of higher complexity and reach the goal faster. This allows us to distinguish contributions of lower-order and higher-order graph learning at the level of choices. Main predictions rest however in the domain of reaction times: Specifically, we expected participants to slow down upon between-wing transitions compared to within-wing transitions. Also, we expected the proximity to the current goal to slow down participants reaction times, so as not to miss the goal and forget to press <space>.

Data of 27 participants were collected online through Prolific. Participants first went through a 'training phase' of 75 miniblocks, where we exactly balanced the presentation of each possible start-goal combination, so no model-free preferences would arise. Then, participants were given a 'budget' of 1000 transitions to find as many goals as possible, where they were always allowed to finish the 'final' miniblock on which they exceeded the 1000 steps. During the testing phase, goals were sampled randomly from any of the 10 rooms outside the wing the participant was currently in. Only data from the testing phase was used in further analyses. After the testing phase, the participants were shown 'free-sort' trial.

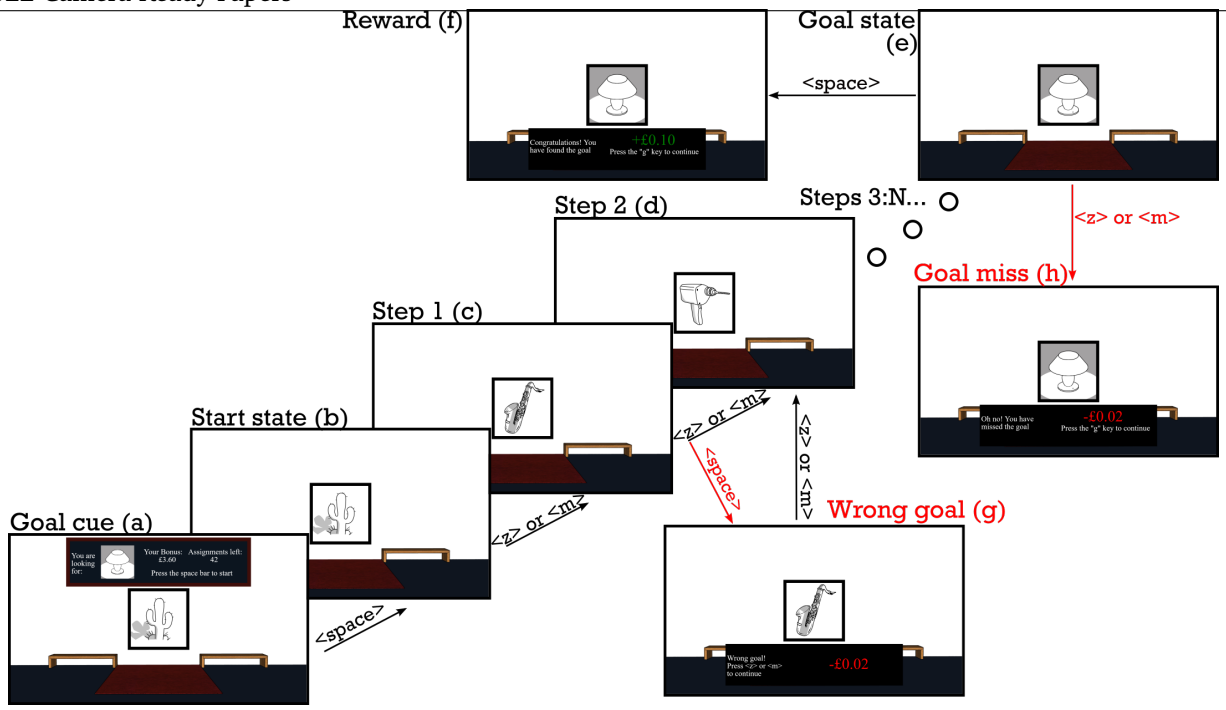


Figure 2: Illustration of a single miniblock. The first image is the goal cue (a), showing the participant is looking for the 'lamp'. They initiate the miniblock by pressing <space>, and then start in the same room as where the cue was given (b). From there on, they must select either the <z> or the <m> key to move on (c, d), and use <space> to indicate that they have reached the goal (e), after which they will receive a reward (f). As illustrated, pressing <space> in a non-goal room leads to a 'wrong goal', giving a small punishment (g). Similarly, pressing <z> or <m> in the true goal room leads to a 'goal miss' and a small punishment (h). The miniblock always ends with a 'goal miss' (h) or a 'reward' (f) screen.

An empty grid and the 15 paintings that hung in each room were shown. Participants were asked to place the paintings on the grid, so that they would resemble what they thought was the spatial layout of the museum.

2.2 Cognitive Models

We fitted and compared 4 cognitive models to the reaction time (RT) data. These models correspond to a *null model*, only capturing 'nuisance' regressors which are not of theoretical interest, a *model-based* model assuming perfect knowledge of the layout of the museum, an *explicit* model which only knows about the wing-layout of the museum, and a Successor Representation (*SR*) model which learns the structure of the museum associatively. Models were fitted in Stan. The approximate leave-one-out cross-validation score was computed for each model for each participant using Pareto Smoothed Importance Sampling, and relative model evidence was computed as Pseudo-BMA+ weights. Participant-level model evidences were then submitted to Group Bayesian Model Selection (GroupBMS; [6]). This allowed us to compute a Bayesian Omnibus Risk (BOR) which indicates whether any model among a set of models is more likely than the others. It also allowed us to compute a Protected Exceedance Probability (pxp) for each model, which is an estimate of how likely that model is to best describe participants' behavior.

The SR learns a matrix $\mathbf{M}$ of expected future (discounted) state visits. A separate matrix can be learned for each available action. If there are A different actions and S different states in the environment, $\mathbf{M} \in \mathbb{R}^{A \times S \times S}$. Each entry $\mathbf{M}_{a,s,s'} = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}_{s_t=s'} | s_0 = s, a_0 = a]$ where a and s are the previous action and state and s' is the next state. This can be learned with the SARSA update rule. Starting $\mathbf{M}$ out as an identity matrix for each action, updates follow $\mathbf{M}[a_t, s_t, :] \leftarrow \mathbf{M}[a_t, s_t, :] + \lambda(\mathbb{I}_{s_t} + \gamma \mathbf{M}[s_{t+1}, a_{t+1}, :] - \mathbf{M}[a_t, s_t, :])$. The SR introduces two parameters to estimate: The discount factor γ and the learning rate λ. We fix $\lambda = 0.1$ in our model fitting, while the discount factor $\gamma \sim Beta(1, 1)$ is always given a uniform prior.

We fit a regression onto the mean parameter of a lognormal distribution in order to model reaction times, specified as $(y_{ij} - ndt_i) \sim lognormal(\alpha_{im} + \boldsymbol{\beta}_{im}\mathbf{x}_{ijm}, \sigma_{im})$, where i indexes the relevant participant, j the relevant trial, and m the relevant cognitive model. For each participant we will fit a separate non-decision time ndt_i which has a uniform prior between 0 and the minimum reaction time for that participant. The intercept $a_{im} \sim \mathcal{N}(6, 1.5)$ is set to match a realistic range of typical reaction times. The regression slopes $\boldsymbol{\beta}_{im} \sim \mathcal{N}(0, 0.1)$ will capture small deviations in response to specific task events captured by the independent variables $\mathbf{x}_{ijm}$ which differ between the different cognitive models

and are described below. Each participant is also fitted with their own variance parameter σ_{im}, as some might be more variable in their RTs than others.

We excluded trials with $RT > 5s$, as these were considered outliers. There were three nuisance regressors, which fully specify the *null* model, but were also included in all other models. The first is the (log-transformed) number of trials that the current miniblock has been active, as we hypothesized participants might speed up as they get frustrated with not progressing. The second is the (log-transformed) recency of encountering the current room, simply counting how many trials ago the current room was last visited. The third was a binary regressor indicating if the current response was different from the last response, as response switching might lead to slower responses.

For the *Model-Based* agent, we computed 3 specific regressors. EVs were calculated by iterating the Bellman equation until convergence, assuming a completely accurate transition model of the environment and reward vector $\mathbf{r}$ of size S where every entry was a small cost value (-0.08) except for the current goal, which was set to 1. Secondly, we computed reward prediction error (RPE) as the difference between the EV of the current trial minus the EV of the previous trial. Thirdly, we computed conflict as the absolute difference between the EV of the two currently available actions, which we call EV_{diff}.

For the *explicit* hierarchical agent, we computed 3 specific regressors. Firstly, we included a binary regressor that would indicate whether the current room allowed for a transition to an adjacent wing (a 'boundary' room) to model increased conflict between the two possible actions. Secondly, we included a binary regressor that indicated whether a transition between two wings was just made, to capture slowing based on surprise. Thirdly, we included a binary regressor that indicated whether the current room was in the same wing as the goal room, to model proximity to the goal.

For the *SR* agent, we computed 4 specific regressors. Firstly, we computed the SR prediction error (SRPE) for trial j as the angular distance between $\mathbf{M}[a_j, s_j, :]$ and $\mathbf{M}[a_{j-1}, s_{j-1}, :]$ to model surprise. Secondly, we computed EV as $\mathbf{M}[a_j, s_j, :] \cdot \mathbf{r}$ to model proximity to the goal. Thirdly, we computed the RPE by subtracting the EV of the previous trial from the EV of the current trial. Fourthly, we computed the EV_{diff} between the two possible actions for each trial.

For the choices we only analyzed the trials outside the goal wing, as there is no correct 'rotation' inside a goal wing. We fit logistic regressions for the 4 different models. The priors for the intercept and regression slopes were set to $\mathcal{N}(0, 2)$. The null model only included an intercept, capturing any bias participants might have to prefer responding <z> or <m>. The three other models all contained one additional regressor. The model-based agent computed the expected value (EV) of <m> minus the EV of <z>. For the explicit hierarchical agent we computed a binary regressor which was 1 when <m> was the current 'correct rotation', meaning it allowed for the direct between-wing transition into the goal wing and 0 otherwise. For the SR model, we compute $\mathbf{M}$ based on the posterior mean of the discount factor from the fitted RT model for each participant. We then computed a similar EV difference as for the model-based agent.

In order to compute the Bayes factor evidence in favour of a relationship between a regressor and a variable of interest (i.e. RT or choice), we always compare models that freely estimate population-level parameters and include varying effects for each participant ($H1$) to models that fix the population-level parameters to zero, but still include varying effects for each participant ($H0$) [11]. Marginal likelihoods will be computed with the R package $bridgesampling$ [1]. In the case of multiple regressors we will compute the marginal likelihoods of all possible combinations of models that do and do not estimate each population parameter, so if there are O regressors, we will fit 2^O models. Let $\mu_o \in \mathcal{M}$ indicate that that the population effect was included in a model, and $\mu_o \notin \mathcal{M}$ indicate that the population effect of variable o was fixed to 0. We can then compute an inclusion Bayes factor [2] for each parameter as $BF_{10}^{inclusion} = \frac{\sum_{\mu_o \in \mathcal{M}} p(y|\mathcal{M})}{\sum_{\mu_o \notin \mathcal{M}} p(y|\mathcal{M})}$.

3 Results

All results presented here come from an exploratory data analysis. With respect to the RTs, GroupBMS including all 4 models shows a significant preference for the SR model ($BOR = 0.001$, $pxp_{SR} = 0.999$). We show the posterior model probabilities in Figure 3A, the participant-level posterior distributions of the regression slopes for the SR model in Figure 3B with below them in Figure 3C the population level posterior distributions. We compute inclusion Bayes factors BF_{10}^{incl} for each regressor and show these in Table 1. With respect to the choices. Data and posterior predictions incl 95% Highest Density Interval (HDI) for the mean between-wing RT minus the mean within-wing RT are shown in Figure 4.

With respect to the choices, GroupBMS shows a significant preference for the model of explicit structural hierarchy ($BOR = 0.001$, $pxp_{explicit} = 0.992$). We find strong evidence that participants are more likely to select the key that rotates in the direction of the current goal wing ($BF_{10} = 30.028$, $HDI_{95\%} = [0.582, 1.679]$).

Additionally, we asked whether the distances between the final locations of the paintings during the free-sort trial were smaller for paintings from the same wings than from different wings. A linear regression was set up with $y_{ij}^{free-sort}$ as the Euclidean distance between two sorted paintings standardized within participant. We find strong evidence that participants prefer to group paintings from the same wing together ($BF_{10}^{inclusion} = 14.735$, $HDI_{95\%} = [-0.557, -0.142]$). Our data are inconclusive on whether participants prefer to place paintings that connect between wings (boundary rooms)

Figure 3: Results from the RT cognitive modeling. **A**: Posterior model probabilities for each participant. All participants favour the SR model except for participant 8, who favours the Null model. **B**: Posteriors of the regression slopes of the SR model for each participant. Density below 0 is colored red and density above 0 is colored blue. **C**: Posterior of the population level effects for each regression slope in the SR model. Note that the scale of the x-axis is different than in B. Single asterisks indicate a $BF_{10}^{incl} \geq 3$ or $\leq \frac{1}{3}$. Double asterisks indicate a $BF_{10}^{incl} \geq 10$ or $\leq \frac{1}{10}$.

Regressor	BF_{10}^{incl}	$HDI_{95\%}$
Steps	0.142	[-0.026,0.030]
Hand switch	0.906	[-0.007,0.077]
Recency	2.516	[0.004,0.022]
SRPE	10.228	[0.008,0.021]
EV	11.192	[0.016,0.034]
RPE	10.786	[-0.058,-0.026]
EV_{diff}	0.185	[-0.002,0.021]

Table 1: Inclusion Bayes factor of population effects for each regressor in the SR RT model, posterior distributions shown in Figure 3C.

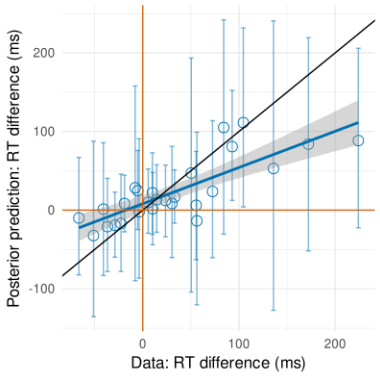


Figure 4: Data and SR model posterior predictions of mean RT for between-wing transitions minus within-wing transitions.

closer together ($BF_{10}^{inclusion} = 0.617$, $HDI_{95\%} = [-0.548, 0.040]$). Our data are also inconclusive whether participants place unconnected paintings from the same wing further apart ($BF_{10}^{inclusion} = 0.335$, $HDI_{95\%} = [-0.055, 0.429]$).

4 Conclusion

The results from cognitive modeling yield evidence for associative learning of task structure in accordance with an SR model. Sensibly, participants seem to slow down relatively to how close they are to the goal. Most interestingly, participants seem to slow down specifically when their SR prediction error is higher, which is most pronounced for transitions between different wings. While reaction times show evidence for sensitivity to SR effects, choices seem to reflect an explicit understanding of the hierarchical 'rotational' structure of our task. It has been shown before that explicit hierarchical knowledge can be derived from a learned SR representation by taking its' eigendecomposition [9]. Our data confirm task representation at both levels of abstraction, with evidence for explicit hierarchical knowledge triangulated with the results from our free-sort trial, where participants seem to group paintings from the same wing closer together.

References

[1] Q. F. Gronau, H. Singmann, and E.-J. Wagenmakers. bridgesampling: An r package for estimating normalizing constants. *arXiv preprint arXiv:1710.08162*, 2017.
[2] M. Hinne, Q. F. Gronau, D. van den Bergh, and E.-J. Wagenmakers. A conceptual introduction to bayesian model averaging. *Advances in Methods and Practices in Psychological Science*, 3(2):200–215, 2020.
[3] C. W. Lynn and D. S. Bassett. How humans learn and represent networks. *Proceedings of the National Academy of Sciences*, 117(47):29407–29415, 2020.
[4] C. W. Lynn, A. E. Kahn, N. Nyema, and D. S. Bassett. Abstract representations of events arise from mental errors in learning and memory. *Nature communications*, 11(1):1–12, 2020.
[5] S. Mark, R. Moran, T. Parr, S. W. Kennerley, and T. E. Behrens. Transferring structural knowledge across cognitive maps in humans and models. *Nature communications*, 11(1):1–12, 2020.
[6] L. Rigoux, K. E. Stephan, K. J. Friston, and J. Daunizeau. Bayesian model selection for group studies—revisited. *Neuroimage*, 84:971–985, 2014.
[7] A. C. Schapiro, T. T. Rogers, N. I. Cordova, N. B. Turk-Browne, and M. M. Botvinick. Neural representations of events arise from temporal community structure. *Nature neuroscience*, 16(4):486–492, 2013.
[8] A. Solway, C. Diuk, N. Córdova, D. Yee, A. G. Barto, Y. Niv, and M. M. Botvinick. Optimal behavioral hierarchy. *PLoS computational biology*, 10(8):e1003779, 2014.
[9] K. L. Stachenfeld, M. M. Botvinick, and S. J. Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653, 2017.
[10] M. S. Tomov, S. Yagati, A. Kumar, W. Yang, and S. J. Gershman. Discovery of hierarchical representations for efficient planning. *PLoS computational biology*, 16(4):e1007594, 2020.
[11] J. van Doorn, F. Aust, J. M. Haaf, A. M. Stefan, and E.-J. Wagenmakers. Bayes factors for mixed models. *Computational Brain & Behavior*, pages 1–13, 2021.

Adaptive patch foraging in deep reinforcement learning agents

Nathan J. Wispinski*
University of Alberta
Edmonton, Alberta, Canada
nathan3@ualberta.ca

Andrew Butcher
DeepMind
Edmonton, Alberta, Canada

Craig S. Chapman
University of Alberta
Edmonton, Alberta, Canada

Matthew M. Botvinick
DeepMind
London, UK

Patrick M. Pilarski
DeepMind & University of Alberta
Edmonton, Alberta, Canada

Abstract

When to explore and when to exploit is a fundamental decision problem that all biological agents must face. One ecological explore-exploit problem, patch foraging, provides a touchstone for artificial intelligence where biological intelligence is successful, adaptive, and sometimes optimal. Here we show deep reinforcement learning agents that can successfully and adaptively forage in a patchy three-dimensional environment. Agents learn to tradeoff exploration and the exploitation of patches, and strike this balance differently in scarce and plentiful environments similar to biological foragers. However, these agents tend to overstay in patches relative to the optimal solution from the marginal value theorem in behavioural ecology, suggesting potential key differences in how artificial and biological agents make tradeoffs during foraging.

Keywords: Foraging
Animal behaviour
Neuroscience
Reinforcement learning
Deep learning

Acknowledgements

We are deeply indebted to our DeepMind colleagues Leslie Acker, Andrew Bolt, Michael Bowling, Dylan Brenneis, Adrian Collister, Elnaz Davoodi, Richard Everett, Arne Olav Hallingstad, Nik Hemmings, Edward Hughes, Michael Johanson, Marlos Machado, Kory Mathewson, Drew Purves, Kimberly Stachenfeld, Richard Sutton, Jane Wang, and Alexander Zacherl for their support, suggestions, and insight regarding this work.

*Corresponding author. This work was conducted at DeepMind with collaboration from the University of Alberta.

1 Foraging in biological and artificial agents

When to explore and when to exploit is a fundamental decision problem that all biological agents must face. Foraging at its core is one such explore-exploit problem, where biological agents must tradeoff the exploitation of resources they currently have access to with the exploration for more resources. In patch foraging theory, spatial patches are frequently modeled as exponentially decaying in resources, with areas outside of patches as having no resources (Charnov, 1976). Agents are faced with a fundamental decision about when to cease foraging in a depleting patch in order to begin travelling some distance to a richer patch. Research has shown that many animals are adaptive patch foragers in this context, intelligently staying in patches for longer when the environment is resource scarce, and staying a shorter time in patches when the environment is resource rich (Cowie, 1977; Hayden, Pearson, & Platt, 2011; Krebs, Ryan, & Charnov, 1974).

Foraging is so important to the survival of biological agents that theorists argue that the foraging behaviour of animals should not only be adaptive, but should approach optimal in natural environments because of strong selective pressures (Stephens & Krebs, 2019). In patch foraging, the marginal value theorem (MVT) offers a solution to optimal patch foraging behaviour (Charnov, 1976). In short, the MVT states that the optimal solution is to cease foraging within a patch and search for a new patch when the reward rate of the current patch drops below the average reward rate of the environment. Many animals, including humans, have been shown to behave optimally in patch foraging tasks in the wild and in the laboratory. For example, human mushroom foragers (Pacheco-Cobos et al., 2019), non-human primates (Hayden et al., 2011), and birds, fish, and bees (Cowie, 1977; Krebs et al., 1974; Stephens & Krebs, 2019) have all shown to behave consistent with the MVT solution of optimal patch foraging.

Many computational models of patch foraging are agent-based models with fixed decision rules (Tang & Bennett, 2010), although recent work has involved the use of tabular reinforcement learning models (Constantino & Daw, 2015; Goldshtein et al., 2020; Miller, Ringelman, Eadie, & Schank, 2017; Morimoto, 2019). Neural networks have also displayed foraging behaviour in ecological tasks such as patch selection (Coleman, Brown, Levine, & Mellgren, 2005; Montague, Dayan, Person, & Sejnowski, 1995; Niv, Joel, Meilijson, & Ruppin, 2002). Foraging behaviour has also been shown in deep reinforcement learning agents searching environments for rewarding collectibles like apples while avoiding obstacles and/or enemies (Lin, 1991; Platanios, Saparov, & Mitchell, 2020). However, these deep reinforcement learning environments often significantly differ from those in theoretical and experimental ecological research.

Here, we first ask if deep reinforcement learning agents can learn to forage in a 3D patch foraging environment inspired by experiments from behavioural ecology. Next we ask whether these agents forage intelligently—adapting their behaviour to the environment in which they find themselves. Finally, we investigate if agent foraging behaviour in these environments approaches the gold standard—the optimal solution determined by the marginal value theorem (Charnov, 1976). This paper provides the first investigation of deep reinforcement learning agents in an ecological patch foraging task, and adds to a literature suggesting that current reinforcement learning methods may fall short of foraging solutions in biological agents (Constantino & Daw, 2015; Miller et al., 2017). These experiments are described not as a performance benchmark for artificial agents, but rather as an empirical investigation into the emergence of complex patch foraging behaviour, and the potential limits of discounted reinforcement learning algorithms in ecological environments.

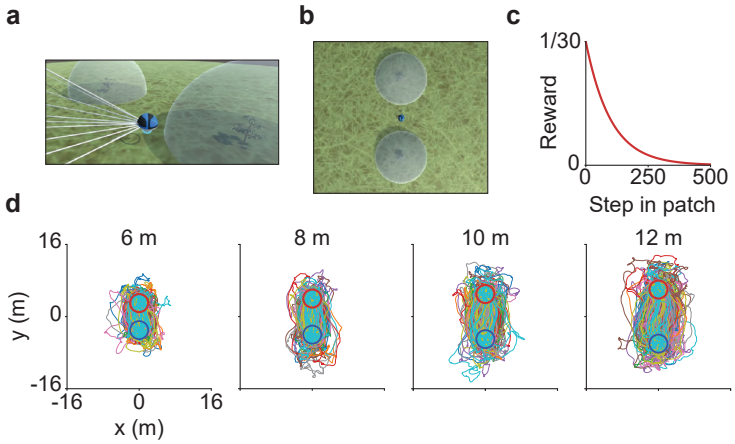


Figure 1: Task. **a)** Mock-up of the 3D foraging environment and agent with LIDAR rays. **b)** Overhead view. An agent starts each episode between two equidistant patches. **c)** The agent receives exponentially decreasing reward on every step it is within a patch. When the agent enters one patch, the opposite patch is refreshed to its starting reward state. **d)** Overhead spatial trajectories of a representative trained agent in each evaluation environment.

2 Experiments

Environment. A continuous 2D environment was selected to approximate the rich sensorimotor experience involved in ecological foraging experiments, as well as for future extensions into multi-patch foraging. The environment consisted of a 32 x 32 m flat world with two patches (i.e., half spheres) equidistant from the center of the world (Figure 1a; see Cultural General Intelligence Team et al., 2022). Patches always had a diameter of 4 m. Agents started each episode at the middle of the world, facing perpendicular to the direction of the patches. Each episode terminated after 3600 steps. Agents received a reward of zero on each step they were outside of both patches. When an agent was within a patch, it received reward according to the exponentially decaying function, $r(n) = N_0 e^{-\lambda n}$, where n is the number of non-consecutive steps the agent has been inside a patch without being inside the alternative patch. In this way, as soon as an agent entered a patch, the alternative patch was refreshed to its initial reward state (i.e., $n = 0$). As such, agents are faced with a decision about how long to deplete the current patch before traveling toward a newly-refreshed patch. For all experiments, the initial patch reward, N_0, was set to $1/30$, and the patch reward decay rate, λ, was set to 0.01 (Figure 1c). The surface color of each patch changed proportional to the reward state of the patch in RGB space. Patches changed color from white (i.e., [1, 1, 1]) to black (i.e., [0, 0, 0]) following the function, $r(n)/N_0$. In this way, agents had access to the instantaneous reward rate of the patch through patch color, rather than having to estimate patch reward rate by estimating the decay function and keeping track of steps spent within a patch.

Agents. Agents had a LIDAR-based observation space—a common sensory modality for physical robots (Malavazi, Guyonneau, Fasquel, Lagrange, & Mercier, 2018), and agents in other simulated environments (e.g., Baker et al., 2019; Cultural General Intelligence Team et al., 2022). The current agents had eight horizontal LIDAR rays evenly spaced throughout 90°, and 3 vertical rays evenly spaced throughout 60°, extending from the front center of the agent. Each LIDAR ray coded for distance (max distance 128 m; normalized), one-hot encoded object type, and RGB colour of the first object it intersected with. LIDAR inputs were convolved (24 output channels, 2x2 kernel shape), before they were concatenated with the reward and action taken on the previous step. These values were then passed through a MLP (3 layers of 128, 256, and 256 units) and a LSTM layer (256 units). Finally, LSTM outputs were passed to an actor and a critic network head. Agents were given a continuous action space that included strafing forward/backward and left/right, looking up/down and left/right, jump/crouch, and a grasp and use action that were not relevant for this environment. Actions were taken by sampling from Gaussians parameterized by the policy network head output for each action dimension. Agents were trained in a distributed manner, each interacting with 16 environments in parallel. Experience was saved in a buffer and agent parameter updates were accomplished via the MPO learning algorithm (Abdolmaleki et al., 2018). A neural network was chosen over tabular methods because of the complexity of the environment, and the potential for future comparisons of internal network dynamics to neural recordings in biological agents.

Three agents were trained in each of four discount rate treatments ($N = 12$), selected on the basis of MVT simulations (Figure 3d). Agents were each initialized with a different random seed, and trained for $12e^7$ steps using the Adam optimizer (Kingma & Ba, 2014) and a learning rate of $3e^{-4}$. On each training episode, patch distance was drawn from a random uniform distribution between 5 m and 12 m, and held constant for each episode. Trained agents were evaluated on 50 episodes of each evaluation patch distance (i.e., 6, 8, 10, and 12 m). If an agent was within a patch at the end of an evaluation episode, this final patch encounter was rejected from all analyses, as no distinct patch leave behaviour could be verified to determine the total steps in this patch.

3 Results

Trained agents displayed behaviour consistent with successful patch foraging—agents learned to leave patches before they were fully depleted of reward (mean leaving step = 121.7), and traveled for several steps without reward in order to reach a refreshed patch (mean travel steps between patches = 57.7). Agents also achieved a higher score on episodes where patches were closer together ($b = -5.82 \pm 0.21$, $p = 3.96x10^{-166}$).

In patch foraging, it is not only important to balance the exploitation of patches with exploration to find new patches, but also to intelligently adapt this balance when in more plentiful or more scarce environments. This adaptive behaviour is

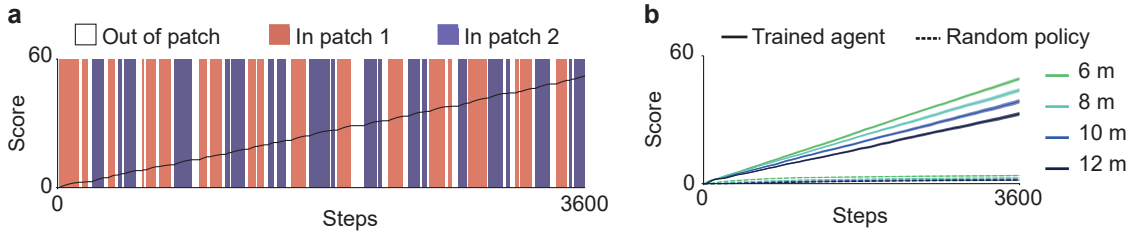


Figure 2: Performance. **a)** Agent behaviour from a representative evaluation episode. Shaded regions define when the agent is outside of any patch (white), inside patch 1 (red), or inside patch 2 (blue). **b)** Episode score for a trained agent (solid lines), and a random agent (dashed lines) in each evaluation environment. Shaded regions denote standard errors.

present in the current agents; trained agents adapted their patch leaving times to the environment, leaving patches later when travel distance is higher ($b = 9.60 \pm 0.87$, $p = 4.03x10^{-28}$; Figure 3a).

Above we show that trained agents are able to successfully forage, and intelligently adapt their foraging behaviour to the environment in accordance with patch foraging theory (Charnov, 1976; Stephens & Krebs, 2019), and animal behaviour (e.g., Cowie, 1977). However, do these agents adapt optimally according to the marginal value theorem (MVT), like many animals (Stephens & Krebs, 2019)? As stated above, the MVT provides a simple rule for when to leave patches optimally (Charnov, 1976). That is, an agent should leave a patch when the reward rate of the patch drops below the average reward rate of the environment. For each agent and evaluation environment (e.g., 6 m), we can estimate the average reward rate of the environment by calculating the average reward per step for each evaluation episode. Over all evaluation episodes, this provides an estimate of the optimal patch leaving step (Figure 3c). Comparing the difference between average observed and optimal patch residence times, agents tend to overstay in patches relative to the optimal solution, $t(11) = 5.60$, $p = 1.60x10^{-4}$ (Figure 3e). Bonferroni-corrected t-tests show that agents significantly overstayed relative to the MVT solution in all evaluation environments ($ps < 0.0015$), except for 12 m ($p = 0.047$).

The current agents however use temporal discounting methods, which exponentially diminish rewards in the future. Given that agents are effectively asked to compare the values between the current patch reward on the next step relative to a refreshed patch reward after several travel steps, temporal discounting encourages longer patch residence times. The difference between observed and optimal behaviour is modulated by the temporal discounting rate, where agents trained with higher temporal discounting rates tend to behave closer to optimal ($b = -2784.01 \pm 992.19$, $p = 0.019$; Figure 3f). Are agents then optimal after accounting for temporal discounting rates in the MVT solution? We accounted for the temporal discounting rate by simulating individual stay and leave decisions at many patch residence steps. Agents could either stay for an additional step of reward before leaving a patch, or immediately leave the patch, where the subsequent 5000 steps were simulated as alternating between a fixed number of steps in a patch and a fixed number of steps traveling between patches. Over a grid of fixed subsequent patch and travel steps, the difference in the discounted return (sum of discounted rewards) between each stay/leave decision provided an indifference curve, where the 5000-step discounted return was equal for staying relative to leaving. Where this stay/leave indifference step matched the fixed patch steps provided an approximation of an average patch time where the value of leaving is about to exceed the value of staying. After accounting for each agent's temporal discounting rate in the MVT (Figure 3d), agent patch residence times were closer to the optimal solution (Figure 3h). Bonferroni-corrected t-tests show that agents significantly overstayed in the 6 and 8 m evaluation environments ($ps < 0.0067$), understayed in the 12 m ($p = 0.0023$), and were not significantly different from optimal in the 10 m environment ($p = 0.30$).

4 Discussion

Here we tested deep reinforcement learning agents in a foundational decision problem facing biological agents—patch foraging. We find that these agents successfully learn to forage in a 3D patch foraging environment. Further, these agents intelligently adapt their foraging behaviour to the resource richness of the environment in a pattern similar to many biological agents (Stephens & Krebs, 2019).

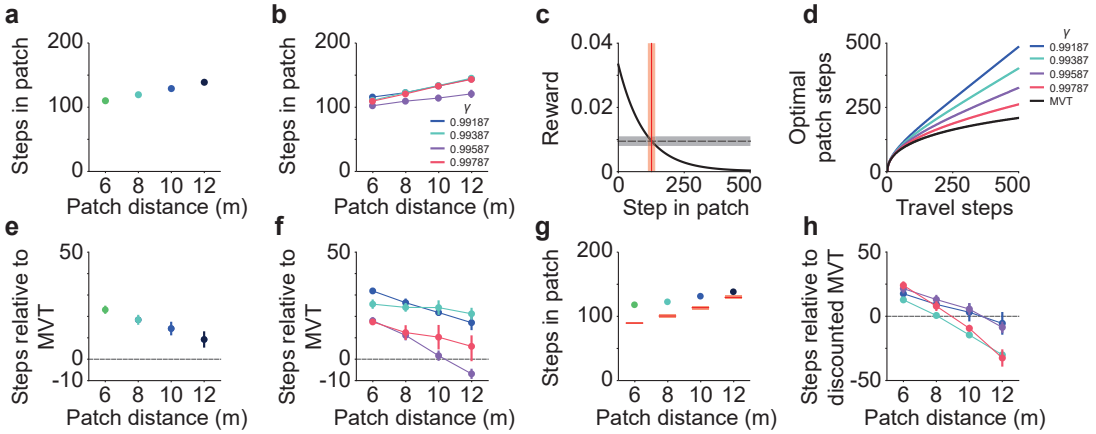


Figure 3: Patch residence times. **a)** Average of all agents. **b)** Average of agents grouped by discount rate. **c)** Graphical model of the MVT solution. Where the patch reward rate (solid black line) intersects the observed average reward rate of the environment as determined by the agent's behaviour (dashed horizontal black line), determines the MVT optimal average patch leaving time (solid vertical red line). **d)** Patch leaving times prescribed by the MVT (black), and simulation results for the MVT considering discount rates. **e)** Mean difference between the observed and optimal patch residence time for all agents, and **f)** agents grouped by discount rate. **g)** Representative single trained agent patch residence times (dots) against the MVT solution (red lines). **h)** Mean difference between the observed and discounted MVT patch residence time for agents grouped by discount rate. All vertical lines and/or shaded regions denote standard errors.

Many animals, including humans in the wild (Pacheco-Cobos et al., 2019), have been shown to be optimal patch foragers. The deep reinforcement learning agents investigated here are adaptive, yet sub-optimal foragers. These results are similar to those from humans in computerized patch foraging tasks, where they tend to overstay relative to the MVT solution (Constantino & Daw, 2015). We find that agents trained with higher temporal discounting rates tend to display patch leaving times closer to the MVT solution (Figure 3f). Further, in a human behavioural study, Constantino and Daw (2015) found that reinforcement learning methods that estimate cumulative long-term discounted rewards are a poor fit for human foraging behaviour in a lab setting, relative to methods which estimate the average reward per step. Overall, these findings suggest potential key differences in how many artificial and biological agents make tradeoffs during foraging, which may be reconciled with further work on average reward reinforcement learning (Sutton & Barto, 2018).

Biologists and behavioural ecologists may benefit from reinforcement learning approaches to agent-based modelling (Frankenhuis, Panchanathan, & Barto, 2019). Further, tasks from behavioural ecology, such as patch foraging, provide insights into fundamental decision problems facing intelligent biological agents. At its core, foraging is an explore-exploit tradeoff, and taking cues from how biological agents (often optimally) solve this dilemma may provide novel methods for artificial agents to do the same.

References

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., & Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., & Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*.

Charnov, E. L. (1976). Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, *9*(2), 129–136.

Coleman, S. L., Brown, V. R., Levine, D. S., & Mellgren, R. L. (2005). A neural network model of foraging decisions made under predation risk. *Cognitive, Affective, & Behavioral Neuroscience*, *5*(4), 434–451.

Constantino, S. M., & Daw, N. D. (2015). Learning the opportunity cost of time in a patch-foraging task. *Cognitive, Affective, & Behavioral Neuroscience*, *15*(4), 837–853.

Cowie, R. J. (1977). Optimal foraging in great tits (Parus major). *Nature*, *268*(5616), 137–139.

Cultural General Intelligence Team, Bhoopchand, A., Brownfield, B., Collister, A., Lago, A. D., Edwards, A., … Zhang, L. M. (2022). Learning robust real-time cultural transmission without human data. *arXiv preprint arxiv.2203.00715*.

Frankenhuis, W. E., Panchanathan, K., & Barto, A. G. (2019). Enriching behavioral ecology with reinforcement learning methods. *Behavioural Processes*, *161*, 94–100.

Goldshtein, A., Handel, M., Eitan, O., Bonstein, A., Shaler, T., Collet, S., … others (2020). Reinforcement learning enables resource partitioning in foraging bats. *Current Biology*, *30*(20), 4096–4102.

Hayden, B. Y., Pearson, J. M., & Platt, M. L. (2011). Neuronal basis of sequential foraging decisions in a patchy environment. *Nature Neuroscience*, *14*(7), 933–939.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krebs, J. R., Ryan, J. C., & Charnov, E. L. (1974). Hunting by expectation or optimal foraging? A study of patch use by chickadees. *Animal Behaviour*, *22*, 953–IN3.

Lin, L. J. (1991). Self-improvement based on reinforcement learning, planning and teaching. In *Proceedings of the Eighth International Conference on Machine Learning* (p. 323–327).

Malavazi, F. B., Guyonneau, R., Fasquel, J.-B., Lagrange, S., & Mercier, F. (2018). Lidar-only based navigation algorithm for an autonomous agricultural robot. *Computers and Electronics in Agriculture*, *154*, 71–79.

Miller, M. L., Ringelman, K. M., Eadie, J. M., & Schank, J. C. (2017). Time to fly: A comparison of marginal value theorem approximations in an agent-based model of foraging waterfowl. *Ecological Modelling*, *351*, 77–86.

Montague, P. R., Dayan, P., Person, C., & Sejnowski, T. J. (1995). Bee foraging in uncertain environments using predictive hebbian learning. *Nature*, *377*(6551), 725–728.

Morimoto, J. (2019). Foraging decisions as multi-armed bandit problems: Applying reinforcement learning algorithms to foraging data. *Journal of Theoretical Biology*, *467*, 48–56.

Niv, Y., Joel, D., Meilijson, I., & Ruppin, E. (2002). Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*.

Pacheco-Cobos, L., Winterhalder, B., Cuatianquiz-Lima, C., Rosetti, M. F., Hudson, R., & Ross, C. T. (2019). Nahua mushroom gatherers use area-restricted search strategies that conform to marginal value theorem predictions. *Proceedings of the National Academy of Sciences*, *116*(21), 10339–10347.

Platanios, E. A., Saparov, A., & Mitchell, T. (2020). Jelly bean world: A testbed for never-ending learning. *arXiv preprint arXiv:2002.06306*.

Stephens, D. W., & Krebs, J. R. (2019). *Foraging theory*. Princeton University Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tang, W., & Bennett, D. A. (2010). Agent-based modeling of animal movement: A review. *Geography Compass*, *4*(7), 682–700.

Learning to Navigate in Unseen Environments Using 2-D Rough Maps

Chengguang XU
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
xu.cheng@northeastern.edu

Lawson L.S. Wong
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
lsw@ccs.neu.edu

Christopher Amato
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
c.amato@northeastern.edu

Abstract

Can robots navigate in unseen environments using rough maps (e.g., a coarse floor plan or a hand-drawing floor draft)? In robot navigation, it is always important for mobile robots to perform quick generalization to unseen environments. Traditional simultaneously localization and mapping (SLAM) methods are impractical for such task because obtaining accurate occupancy maps is computationally expensive. Recent progress in *map-free* methods show promising results, requiring billions of data to train a good policy. For other *map-based* methods, mapping errors and local optima issues also affect the robot to reach the long-distance goals, revealing the importance of having correct global information as guidance. This study exams the usage of 2-D rough maps and proposes a framework that achieves robust generalization in unseen environments. The framework consists of a graph-structured 2-D rough map, a deep local map predictor, a noisy Bayesian filter and a heuristic planner. Unlike *map-based* methods, our framework neither requires a global map identical to the real environment nor builds an occupancy map from scratch. Unlike *map-free* methods, our framework uses less training data and achieves robust navigation performance. We compare our framework against one of the state-of-the-art *map-based* methods in Habitat, a photo-realistic house environment. The preliminary results show that the *rough-map-based* framework achieves a mean success 93.7% in seen environments and 91.0% in the unseen environments.

Keywords: Robot navigation, Generalization, Rough maps, Planning

1 Introduction

Mobile robots have become ubiquitous in our society and frequently encounter novel scenarios, suggesting the quick generalization to novel cases is critical. Traditional SLAM methods [3] are widely used. However, building accurate metric maps of unseen environments is computationally expensive and time intensive, especially if the robot only has to reach a goal location once in a novel environment.

Instead of building such maps, deep learning-based *map-free* navigation algorithms [9, 11] are proposed and achieve appealing generalization performance. However, those methods require billions of training data and months of GPU hours to learn a good policy, causing the training prohibitive on computers with limited resources. Besides, during long-distance navigation tasks, the local optima issues still affect the performance of map-free methods because of the absence of global information. Unlike the *map-free* approaches, *map-based* methods [4, 2, 6], aim to learn the SLAM behaviors using deep neural networks. In particular, these methods learn to build 2-D occupancy maps from onboard sensor inputs (e.g., cameras, odometry). Given the progressively built map, these methods plan actions towards the goal, achieving more stable navigation performance. Compared with traditional SLAM benchmarks that build global occupancy maps first and use them for planning later, *map-based* methods plan the action based on the latest occupancy map, capturing only partial information at the beginning of the navigation. Planning on the partial map might be problematic because the

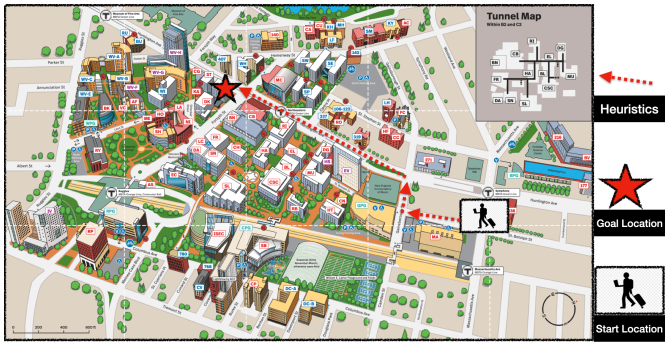


Figure 1: Example of using rough maps. When a new student first arrives at Northeastern University and wants to go to the target building marked at the red star. Given a campus map, he can plan a path, marked as dash lines, from his start location to the target building. Although the campus map is rough and animated, the student can still find the correct direction (i.e., the heuristic vector in our case) to go after he estimates his location on the map. Then, he simply takes the action that results in heading in the same direction as the map indicated.

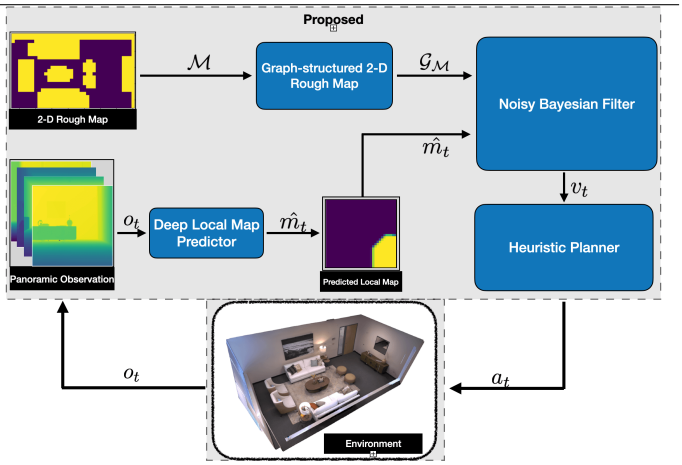


Figure 2: The 2-D rough map $\mathcal{M}$ is first converted to the graph-structured 2-D rough map $\mathcal{G}_{\mathcal{M}}$. At time step t, the agent receives a panoramic observation o_t and predicts the local rough map $\hat{m}_t$. The noisy Bayesian Filter takes in the $\mathcal{G}_{\mathcal{M}}$ and $\hat{m}_t$ and estimates the agent's location. A heuristic vector v_t is computed based on the estimation. The heuristics planner uses v_t and finds the optimal a_t. The agent executes a_t and the environment outputs an observation o_{t+1}.

planned action might be locally optimal, causing agents stuck in some local regions or explore unrelated areas, especially for long-distance goals.

The limitations of *map-free* and *map-based* methods motivate the usage of the global information during decision-making processes, especially for reaching long-distance goals. Having the global information helps to avoid the local optima issue and unnecessary exploration behaviors. Although obtaining the accurate global information (e.g., the 2-D occupancy maps) in unseen environments is impractical, it is much easier to obtain rough maps (e.g., a rough floor plan, a hand-drawing floor daft etc.). Such rough maps might be coarse in many aspects. For example, they might have different scales or contain partial information. However, rough maps capture the global geometry and provide useful global information to guide navigation. For example, a new student can easily find the target building at Northeastern University as long as the student has a campus map, which only contains rough information (See Figure 1 for example).

In this work, we exam the usage of the 2-D rough maps for the visual PointGoal navigation task in unseen environments and propose a framework. We compare the proposed framework against a SLAM-based baseline in a photo-realistic house environment [7]. Empirical results show that our method outperforms the SLAM-based baseline in both *seen* and *unseen* cases and achieves a robust navigation generalization performance, demonstrating the effectiveness of using the 2-D rough maps with the proposed framework.

2 Problem Statement

We exam the PointGoal navigation problem as described in [1, 7]. In particular, the agent is initialized randomly in an unseen environment and asked to navigate to random goal coordinates specified relative to the agent location. The agent navigates using its onboard sensors - in our case the Depth camera, Odometry sensor (use orientation only), and a 2-D rough map (see Figure 2). When one episode begins, we assume only the rough goal location on the 2-D rough map is known.

We first formulate the PointGoal navigation in a single environment problem as a partially observable Markov decision process (POMDP)[5], represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, where $\mathcal{S}$, $\mathcal{A}$, and Ω are finite sets of states, actions, and observations respectively, $\mathcal{T}$ and $\mathcal{O}$ specify transition and observation probabilities, and $\mathcal{R}$ is the reward function. In PointGoal navigation, states s consist of 2-D location and orientation (x, y, θ) relative to the start location, whereas observations o are Depth images corresponding to the panoramic view at the given state s. To specify PointGoal navigation in one environment with arbitrary distance, we define ρ_0 and ρ_g as the distributions for start states $s_0 \in \mathcal{S}$ and goal states $g \in \mathcal{G}$, where $\mathcal{S}$ and $\mathcal{G}$ is identical. Furthermore, we define the reward function as $r(s, a, g) = 0$ if $s = g$ and $r(s, a, g) = -1$ otherwise. Given the start $s \sim \rho_0$ and goal $g \sim \rho_g$ locations, the objective is to find a policy

Figure 3: An example for the training environment in Replica 15

Figure 4: An example for the testing environment in Gibson 45

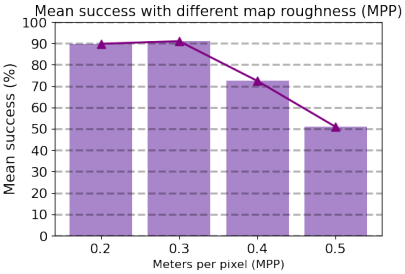


Figure 5: Effect of map roughness. Our method achieves the mean success rate (*unseen*) $\geq$ 70% (**Neural-SLAM**: 77.6%) with different map roughness.

$\pi : \mathcal{O}_{\mathcal{S}}^* \times \mathcal{S}_{\mathcal{G}} \rightarrow \mathcal{A}$ that maximizes the expected accumulative rewards for all the start and goal pairs $E_{s_0 \sim \rho_0, g \sim \rho_g}\left[V(s_0, g)\right]$, where $V(s_0, g) = E\left[\sum_{t=0}^{T-1} r(s_t, a_t, g) | s_0, g\right]$.

Similarly, we can formulate the PointGoal navigation in multiple unseen environments using 2-D rough maps as finding a policy $\pi : \mathcal{O}_{\mathcal{S}}^* \times \mathcal{S}_{\mathcal{G}} \times \mathcal{M} \rightarrow \mathcal{A}$ that maximizes the expected accumulative rewards for all the start and goal pairs in all unseen environments $E_{s_0 \sim \rho_0, g \sim \rho_g, e \sim ENVs, m \sim \mathcal{M}}\left[V(s_0, g, e, m)\right]$, where $V(s_0, g, e, m) = E\left[\sum_{t=0}^{T-1} r(s_t, a_t, g) | s_0, g, e, m\right]$, e is one unseen environment, and m is the corresponding 2-D rough map.

3 Method

We propose a *rough-map-based* method to solve the PointGoal navigation tasks. Figure 2 shows a diagram of the proposed framework. In a nutshell, the framework consists of four essential components as follows:

The graph-structured 2-D rough maps Instead of learning a global rough 2-D map $\mathcal{M}$ interpreter, we propose to predict the local rough patches, which are more likely to be shared among the environments. Specifically, we decompose a 2-D rough map into local rough patches and build a graph $\mathcal{G}_{\mathcal{M}}$. There are two advantages: 1) The graph decomposes the global 2-D rough map into shareable local rough patches, allowing for solutions for perception models with better generalization performance. 2) The graph still preserves the global information by maintaining the connectivity between two adjacent local patches with edges. Similar to the human's common sense of using maps, we use the graph-structured 2-D rough map to compute a rough direction towards the goal location. The rough direction is represented as a vector v_t, called the heuristic direction vector, and is generated for every time step t.

The deep local map predictor To compute the heuristic direction vector on the graph-structured 2-D rough map, we first need to estimate the location of the agent on the map. Since the initial agent's location on the map is unknown, we use a Bayesian filter to estimate the location of agent. However, using the Bayesian filtering on a 2-D *rough* map raises two key issues: 1) The agent observes images in the real environment while the observations on the 2-D rough map should be 2-D local patches; 2) The 2-D map is not identical to the real environment. In other words, there exists a mismatch between the local details the maps and the environments. For example, when the agent moves forward in the real environment, this action can change the agent's location in the real environment. But, on the 2-D rough map, the agent might still ends up within the same map location. The **local map predictor** is proposed to map the observations from image-based space to local occupancy map space (See Figure 2). In other words, the model predicts the 2-D local occupancy patch $\hat{m}_t$ from panoramic observations o_t.

The noisy Bayesian Filter We propose a noisy Bayesian filter to deal with the mismatch between the maps and the real environments. In particular, we assume the local map predictor model is informative and is powerful to correct the errors caused by the first Bayesian update (i.e., computing the predictive belief). Therefore, instead of updating the predictive belief using the ground truth transition probability, we sample a random noise from a uniform distribution between $(0, 1)$. Then, in the observation update, we hope the observation probability will correct those errors. Empirical results in Figure 5 show such design works for rough maps within a range of map roughness.

The heuristic planner Given the heuristic direction vector v_t, we propose a simple heuristic planner to find the best action to execute. In particular, v_t indicates the global direction heading to the final goal. Therefore, the planner only has

to find the action resulting in the same direction as v_t. Note that, our planner does not require the ground truth of agent's location. Instead, we design a function to generate the Δ change of the location after taking an action. The function can be easily programmed in a few code lines or modeled as a simple neural network learned from a small number of experiences. In the current version, we implement a tree-structured planner to find the action based on v_t.

4 Experiment

We compare our method against a strong SLAM-based baseline in **Habitat** [7], a photo-realistic simulator. To evaluate the performance, we randomly sample N navigation trials in each testing environment and report the mean success rate, defined as $\text{SR} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{S}_i$, where $\mathcal{S}_i = 1$ for a successful navigation trial, and 0 otherwise; N is the total number of trials.

In Habitat, we use the Replica dataset [8], containing 15 houses, for training and use the Gibson dataset [10], containing 45 houses, for testing. We compare our method against **Neural-SLAM** [2], a state-of-the-art SLAM-based method. Preliminary results in Table 1 show that our method achieves better performance than Neural-SLAM but uses much less data for training, which highlights the importance of using the 2-D rough maps and also demonstrates the effectiveness of our method in visual realistic domains.

Table 1: Main results for the ***seen*** Replica 15 environments and ***unseen*** Gibson 45 environments (%)

Mean success rate	**Seen**	**Unseen**
Neural-SLAM	92.4	77.6
Our	93.7	91.0

5 Discussion

In this work, we present a *rough-map-based* framework for visual PointGoal navigation in unseen environments. Unlike other work, we propose to use rough 2-D maps instead of 2-D maps identical to the real environments. The proposed framework consists of a graph-structured rough 2-D map, a deep local map predictor, a noisy Bayesian filter, and a heuristic planner, achieving robust generalization performance in unseen environments. Empirical results in Habitat show that our method can be effectively applied to realistic environments using rough 2-D maps.

References

[1] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[2] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020.

[3] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[4] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[5] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[6] Peter Karkus, Shaojun Cai, and David Hsu. Differentiable slam-net: Learning particle slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2815–2825, 2021.

[7] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019.

[8] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[9] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

[10] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[11] Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning pointgoal navigation. *arXiv preprint arXiv:2007.04561*, 2020.

Two-stage task with increased state space complexity to assess online planning

Jungsun Yoo
Department of Cognitive Science
University of California, Irvine
Irvine, CA 92697
jungsun.yoo@uci.edu

Aaron M. Bornstein
Department of Cognitive Science
University of California, Irvine
Irvine, CA 92697
aaron.bornstein@uci.edu

Abstract

Humans use an internal model to predict and navigate through a series of decisions in reinforcement learning (RL) tasks, referred to as planning. Studies show that planning predicts performance in such tasks, but they only provide a partial description because they do not separate local deliberation (i.e., online planning) from using plans made ahead of time (i.e., offline planning). To address this gap, we introduce a variant of the canonical two-stage task (TST), called the *multinomial* TST, that discourages planning beforehand by increasing state-space complexity (SSC). Here, we report behavioral results from three versions of multinomial TST with increasing SSC, in addition to the canonical TST (total $N = 418$). Consistent with the hypothesis that increasing SSC would lead to an increase in online planning, we found that increasing SSC induced longer response time (RT) during first-stage (but not second-stage) choices. We next decomposed RT into separable components of pre-trial and on-demand evaluation by fitting a novel variant of a reinforcement learning diffusion decision model (RLDDM). Model fits revealed that first-stage drift rate and starting point both showed influence of model-based values, but that, as SSC increased, this influence was stronger in drift rate and weaker in starting point. Further, we used a timeseries analysis to observe that the model-based contribution to starting point and drift rate were negatively correlated within each subject, suggesting that experience within each task, as well as task complexity, mediates the relative contribution of online and offline planning. Taken together, these results suggest that while planning without decision-time deliberation (offline planning) suffices for tasks with low SSC, online planning becomes more necessary with increasing SSC, and that our task and model could be a framework for further investigation of human online planning.

Keywords: Reinforcement learning, Planning, Model-based decision making, Online planning

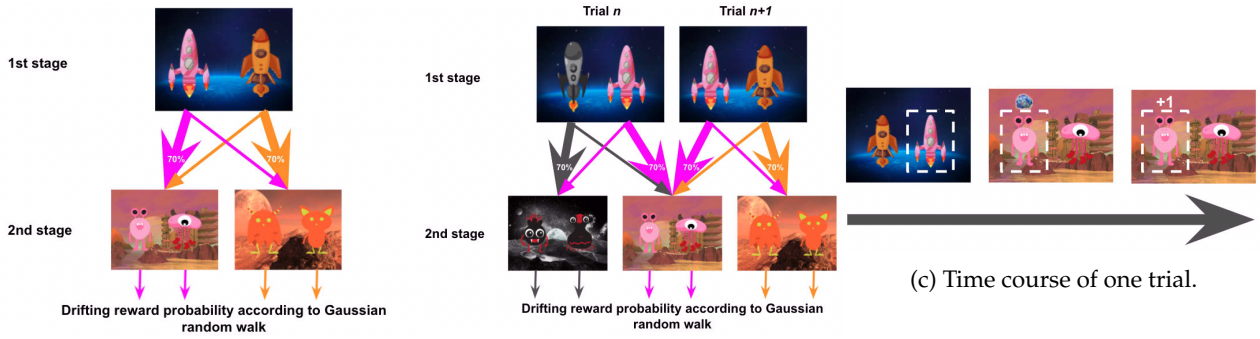
Acknowledgements

This work was supported by NIMH P50MH096889 and NIA R21AG072673 to AMB.

1 Introduction

Planning is one of the most fundamental functions for not only natural but also artificial agents' learning and decision-making. In the realm of reinforcement learning (RL), planning refers to deciding with internal models - a transition function and a reward function - of the environment, thereby being interchangeably referred to as model-based (MB) RL. Utilizing the knowledge of the model to look forward avails efficiency and flexibility of making decisions but comes at a computational cost. Model-free (MF) RL could be seen as a system that is opposite to planning in that it is computationally cheap but inflexible to adapting changes.

MB and MF systems have been found to complement each other in RL. The standard task for measuring this relationship is the two-stage task (TST) that consists of two consecutive decisions, where first-stage decisions stochastically lead to the second-stage decisions [1, 2]. Since the reward is given in the second-stage decision, the best decision to take in the first stage involves using the information of the second stage - in other words, the model. In this framework, a person's tendency to plan can be captured into a parameter w that directly weighs the tendency to use MB vs. MF value, defined by the following formula, $Q = wQ_{MB} + (1-w)Q_{MF}$, where $w = [0, 1]$. Here, MB Q-values (Q_{MB}) are derived by multiplying the transition probability and second-stage values, and MF Q-values (Q_{MF}) are updated via temporal difference (TD) learning upon direct experience. However, a distinction left uninterrogated by this approach is whether plans are constructed *offline* – that is, prior to the trial start – or *online*, after the presentation of options in the moment.



(a) Canonical TST (SSC=2). (b) Multinomial TST (SSC 3). (c) Time course of one trial.

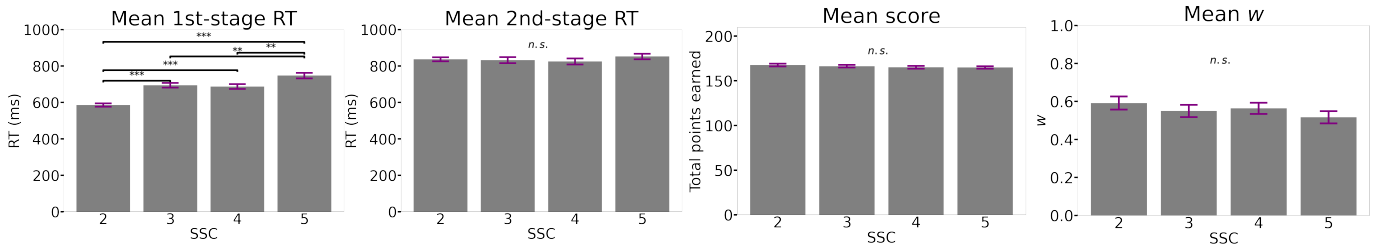
Figure 1: Experimental paradigm. (Figures 1a-1b) Each experiment consists of 300 trials and has 5 catch trials at random sequence. (Figure 1a) Experimental design of the canonical TST (equivalent to SSC-2). Note that the same first-stage state appeared on every trial, which allows for precomputation of plans in the intertrial interval (ITI). (Figure 1b) Experimental design of our proposed multinomial TST. The number of SSC corresponds to the number of rockets in the 1st stage, where each rocket mainly leads to its associated 2nd stage (planet). Therefore, each 1st-stage state varies according to the combination of the rockets. (Figure 1c) Temporal order of TST. Both the first and second stage decisions have a 2-second response window, followed by a feedback phase of 1 second. ITI lasted for 2 seconds with a fixation cross.

This distinction is materially relevant in multiple domains. Recently, it has been shown that artificial MB RL agents rely heavily on offline planning vs. online planning, likely because they can benefit from very large numbers of training examples that can be acquired without direct experience [4]. However, it remains an open question the degree to which humans can similarly eschew online planning, given that they tend to make decisions based on fewer experiences. Eye- and mouse-tracking studies using TST suggest that subjects plan before the trial onset [5, 6], thereby making the direct comparison between online vs. offline planning elusive within the TST paradigm. Here, we propose a novel variant of the TST, the *multinomial* TST, to delineate behavioral patterns of online planning by time-locking the availability of decision-relevant information to the first-stage stimulus onset [3]. This is achieved by increasing the number of possible first-stage options, and selecting combinations of them randomly at each trial, thereby making it difficult for agents to predict and plan the first-stage decisions beforehand. We increase the state-space complexity (SSC) in three levels (3, 4, and 5 first-stage options; Figure 1b), and compare it with the canonical TST which has 2 first-stage options (Figure 1a). Like the original TST, two options are presented on first- and second-stage decisions and the first-stage option stochastically leads to the second-stage state associated with the unchosen first-stage option. We recruited 110 participants between age 18-40 each for every variant - 2-, 3-, 4-, and 5-SSC tasks - via Amazon Mechanical Turk. We excluded 8, 4, 5, and 5 subjects from each experiment, due to one of the following criteria: responding with the same key on more than 95% of the trials, responding implausibly fast (RT below 150 ms) on more than 10% of the trials [9], choosing more than 90% to either option, failing to respond within the response window on more than 20% of the trials, below chance-level model fit, pressing certain button or choosing certain option consecutively for more than 10% of the trials, responding with RT of 0 for more than 5 trials (this indicates that the button was pressed before the onset of the trial), and scored less than 2 out of 5 catch trials. We hypothesized that increasing SSC will evoke online planning; behaviorally, we expected this to

be reflected in an increase in participants' response times (RT) in the first-stage decision, since precomputing the plans before trial onset would be more difficult, and an increased dependency in using trial-specific MB-information on each decision. Our main measures of interest to compare behavioral patterns across SSC come from two models - w from the choice model used by previous TST studies [1, 2] and the non-decision time (NDT; t), drift rate (v), and starting-point bias (z) parameters in a reinforcement learning diffusion decision model (RLDDM; [8]), here extended to incorporate the multiple value timeseries (MB and MF) available in the TST. Specifically, we measure online planning with the MB evidence accumulation rate (v_{MB}), since the drift rate stands for evidence accumulated at each given timepoint. In contrast, we hypothesize that the MB evidence already accumulated before the start of the trial (z_{MB}) represents offline planning.

2 Results

First, we tested the hypothesis that complex environments recruit online decision evaluation by analyzing the response time of first-stage states as a function of SSC (Figure 2). The first-stage RT indeed increased as a function of SSC; a one-way ANOVA yielded a significant difference between SSC ($F_{(3,412)} = 26, p < .001$), and a post-hoc t-test revealed that the RT of SSC-2 are significantly lower than other conditions (SSC-2 vs. SSC-3: $t_{(203)} = -6.69, p < .001$, SSC-2 vs. SSC-4: $t_{(201)} = -6.17, p < .001$, SSC-2 vs. SSC-5: $t_{(205)} = -9.01, p < .001$), and 1st-stage RT in SSC-5 were also significantly slower than other conditions (SSC-3 vs. SSC-5: $t_{(210)} = -2.66, p = .009$, SSC-4 vs. SSC-5: $t_{(208)} = -2.98, p = .003$; Figure 2a). It is also notable that neither the mean score (Figure 2c) nor mean w (Figure 2d) are different across conditions (all $p > 0.3$). This suggests that the differences in first-stage RT cannot be explained by the difficulty of the task *per se* or the degree of model use in each task. Further evidence in favor of this point is that second-stage RT, which also involves binary choice but not planning into the future, was invariant among conditions ($F_{(3,412)} = 1.86, p > .13$; Figure 2b).



(a) Mean first-stage RT (b) Mean second-stage RT (c) Mean score (d) Mean w

Figure 2: Behavioral analyses across conditions. Error bars indicate standard error. Significant results of pairwise t-tests are indicated with asterisks: $* = p < .05, ** = p < .01, *** = p < .001$) (a-b) Mean first- and second-stage RT across different SSC. Prior to statistical tests such as one-way ANOVA and t-tests, each participant's mean RTs for 1st- and 2nd-stage decisions were log-transformed and normalized. (a) First-stage RT of SSC-2 was significantly shorter than the rest conditions, while the opposite was true for SSC-5. There was no significant difference between 1st-stage RT of SSC-3 and SSC-4 ($t_{(206)} = .38, p > .7$). (b-d) A one-way ANOVA indicated no significant difference between groups in mean second-stage RT, mean score, and mean w across groups (all $p > .13$).

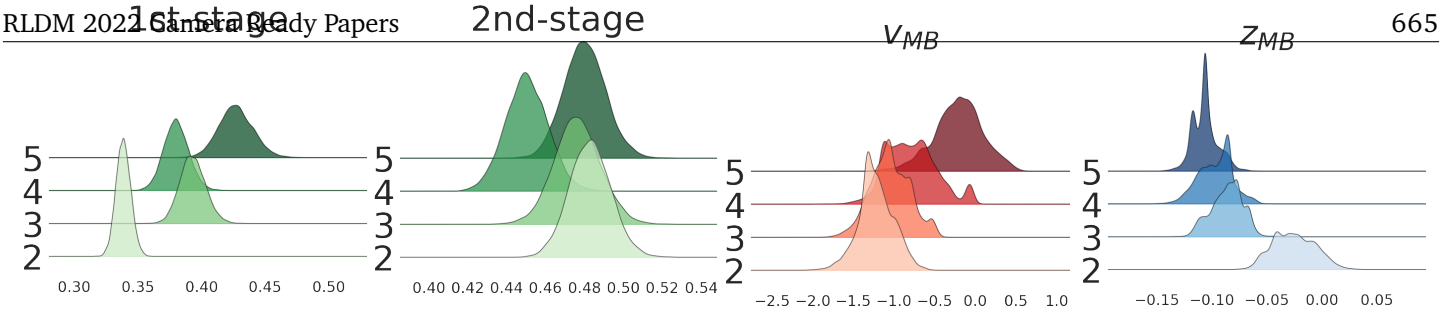
Next, we analyzed the data using an RLDDM to identify the within-trial pattern of planning. Our model inherited features of the original RLDDM [8], but augmented to model the TST. Specifically, in order to delineate the use of the MB vs. MF values at each 1st-stage decision, we added additional parameters to the original drift rate and the starting-point bias reflecting the trial-by-trial variation in these quantities, according to the following specification:

$$v = v_0 + v_{MB}\delta(Q_{MB}) + v_{MF}\delta(Q_{MF}) + v_{int}\delta(Q_{MB})\delta(Q_{MF}) \quad (1)$$

$$z = z_0 + z_{MB}\delta(Q_{MB}) + z_{MF}\delta(Q_{MF}) + z_{int}\delta(Q_{MB})\delta(Q_{MF}) \quad (2)$$

, where $\delta(Q_{MB})$ and $\delta(Q_{MF})$ stand for the MB or MF Q-value differences of the two first-stage options, respectively. In our RLDDM, the decision threshold (a) was fixed to 1 for a straightforward interpretation of v and z, and a sigmoid function was applied to z so that the starting point was bounded to $[0, 1]$. The model was fitted to the data via the HDDM package [10], where each SSC was estimated by 5 chains of 15,000 samples with 2,000 burn-in samples.

In line with the overall RT, the 1st-stage NDT significantly increased as a function of SSC ($F_{(3,412)} = 14.03, p < .001$; mean $t_{SSC-2} = .34$ s, mean $t_{SSC-3} = .4$ s, mean $t_{SSC-4} = .39$ s, mean $t_{SSC-5} = .43$ s; Figure 3a), while the 2nd-stage NDT did not differ among conditions (Figure 3b). This may reflect that in order to plan online in higher SSC environments, participants must spend a longer time identifying the options present on the screen (ostensibly to establish the current



(a) Posterior distribution of first-stage NDT (t)

(b) Posterior distribution of second-stage NDT (t)

(c) Posterior distribution of v_{MB}

(d) Posterior distribution of z_{MB}

Figure 3: Posterior distribution of parameters estimated from RLDDM. For each SSC, the RLDDM model was estimated via HDDM on 5 chains that iterated for 15,000 samples, of which 2,000 were burn-in samples. The results show distribution of samples concatenated from all 5 chains. (a-b) The unit for NDT (t) is seconds.

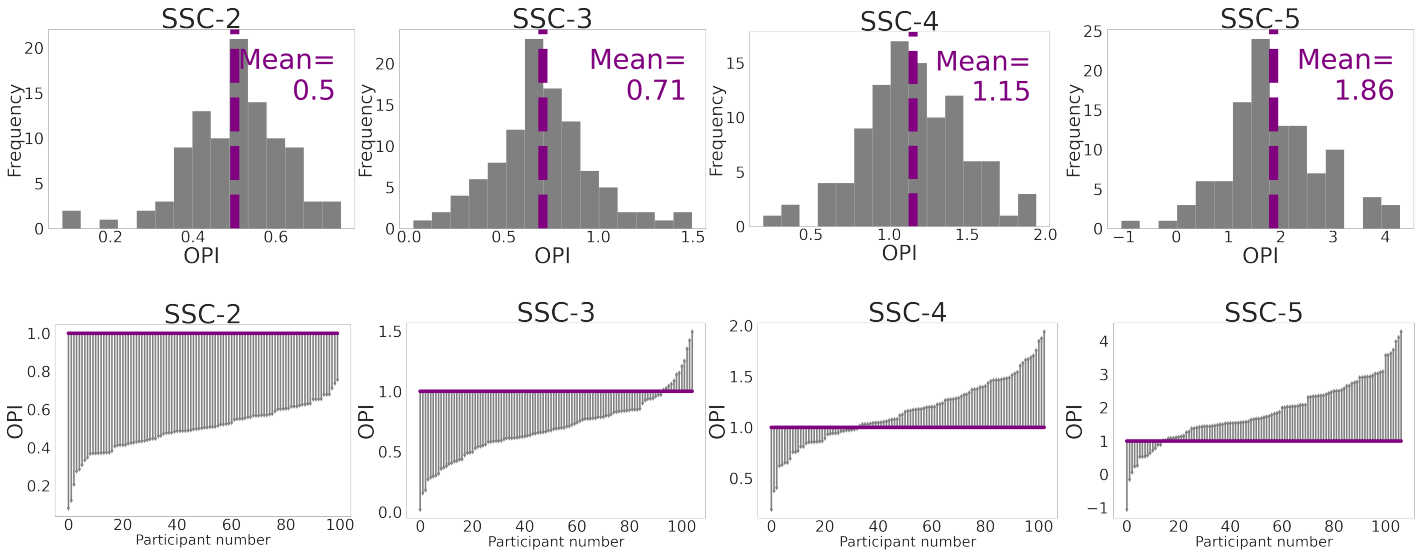


Figure 4: The distribution of the online planning index (OPI), defined by the ratio of v_{MB} to z_{MB}. (First row) The histogram of OPI for each condition. (Second row) A stem plot of OPI for each condition, with OPI=1 as the baseline.

decision tree) at each trial, which would have been unnecessary for the original TST. Importantly, after accounting for this condition-wise shift in RT using the NDT parameter, we examined how trial-varying MB and MF values affected both the starting point (z) and drift rate (v) differentially in each condition. Supporting the hypothesis that participants' dependency on using online, relative to offline, planning should increase as a function of SSC, we found a significant increase of v_{MB} and decrease of z_{MB} as a function of SSC through one-way ANOVA ($F_{(3,412)} = 2373.11, p < .001$; Figure 3c; $F_{(3,412)} = 55.18, p < .001$; Figure 3d). Post-hoc pairwise t-tests revealed that while all distributions of v_{MB} were significantly different among SSC (all $p < .001$), all pairs but SSC-4 vs. SSC-5 were significantly different for z_{MB} (rest $p < .001$).

Next, we examined whether online and offline planning traded off within each subject. Specifically, we calculated an *online planning index* (OPI), given by $OPI = (1 - z_{MB})/(1 - v_{MB})$. The first and second row of Figure 4 show that as SSC increases, so does the proportion of subjects adopting online planning ($F_{(3,412)} = 140.61, p < .001$), where subjects in SSC-2 and SSC-3 heavily rely on offline planning while the majority of participants in SSC-4 and SSC-5 uses online planning ($OPI = 1$ indicates equal use of online and online planning). To determine the extent to which general planning could be explained by online planning, we correlated w from the choice model and OPI from the RLDDM. Again supporting the hypothesis that online planning becomes more relevant as SSC increases, only SSC-5 showed a significant correlation between the two parameters (Pearson's $r = .25, p = .009$), indicating that planning in SSC-5 is heavily driven by online planning. Finally, to examine the tradeoff between online and offline planning over the course of task experience, we performed a sliding-window RLDDM with a window size of 50 trials to obtain the timeseries of v_{MB} and z_{MB}. We then correlated the timeseries of v_{MB} and z_{MB} for each subject, and (Figure 5) shows that all SSC yields significant, medium to strong negative correlations between v_{MB} and z_{MB}. Entering the z-transformed correlation coefficients into pairwise t-tests against zero yielded significant differences for all conditions (all $p < .001$).

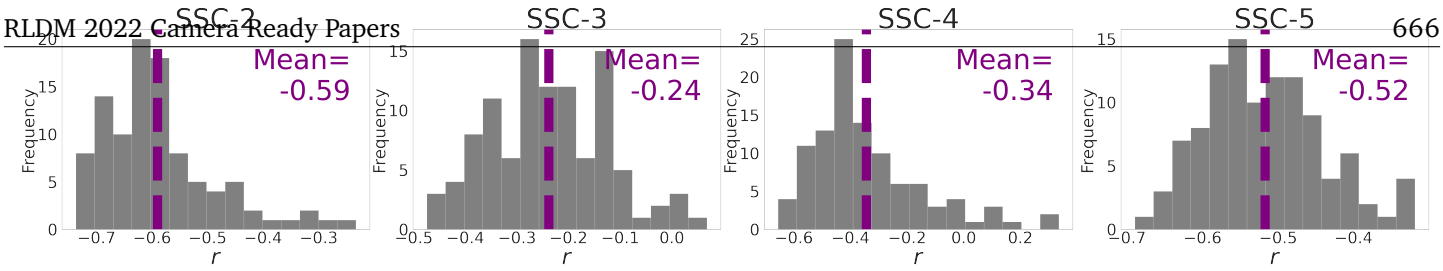


Figure 5: The distribution of the correlation coefficient (Pearson's r) between the timeseries of v_{MB} and z_{MB} for each subject. The timeseries is generated by applying the RLDDM model to a window of 50 trials, and sliding the start of the window from the first trial to the 250th trial.

3 Conclusion

The TST framework has been an important method for investigating planning, but a key limitation of this framework is that it prevents evaluating the online deliberation process directly in behavior. Here, we introduce a multinomial TST which sets aside this limitation, and across four experiments, demonstrate that response time patterns reflect a sensitivity to both each task's differing state complexity and also trial-wise learned values. The preliminary results reported here suggest that our novel multinomial TST can provide a useful foundation for investigating online planning patterns in humans. Importantly, we have shown that although plans could be made in advance in simplistic environments, this is gradually hampered with complexity and thus recruits decision-time planning. This trade-off relationship between online and offline planning is observed across both SSC and experience within the task. Together with this trade-off, the correlation between OPI and w in the most complex environment may provide an explanation for the invariant general planning (w) across contexts, also shown in previous studies [7]. Also, OPI, which directly compares the relationship between the two planning mechanisms, could be used as a new individual-difference measure with potential clinical relevance. We also anticipate that bringing this decision-time activity under behavioral control and experimental manipulation will allow us to interrogate the algorithmic structure of model-based decisions, to understand the balance between online and offline control in humans performing novel tasks, and will allow for evaluating online deliberation activity using neuroimaging measures.

References

[1] Nathaniel D Daw, Samuel J Gershman, Ben Seymour, Peter Dayan, and Raymond J Dolan. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 69(6):1204–1215, 2011.

[2] Johannes H Decker, A Ross Otto, Nathaniel D Daw, and Catherine A Hartley. From creatures of habit to goal-directed learners: Tracking the developmental emergence of model-based reinforcement learning. *Psychological science*, 27(6):848–858, 2016.

[3] Laura Fontanesi, Sebastian Gluth, Mikhail S Spektor, and Jörg Rieskamp. A reinforcement learning diffusion decision model for value-based decisions. *Psychonomic bulletin & review*, 26(4):1099–1121, 2019.

[4] Jessica B Hamrick, Abram L Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Buesing, Petar Veličković, and Théophane Weber. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020.

[5] Arkady Konovalov and Ian Krajbich. Gaze data reveal distinct choice processes underlying model-based and model-free reinforcement learning. *Nature communications*, 7(1):1–11, 2016.

[6] Arkady Konovalov and Ian Krajbich. Mouse tracking reveals structure knowledge in the absence of model-based choice. *Nature communications*, 11(1):1–9, 2020.

[7] Wouter Kool, Samuel J Gershman, and Fiery A Cushman. Planning complexity registers as a cost in metacontrol. *Journal of cognitive neuroscience*, 30(10):1391–1404, 2018.

[8] Mads L Pedersen and Michael J Frank. Simultaneous hierarchical bayesian parameter estimation for reinforcement learning and drift diffusion models: a tutorial and links to neural data. *Computational Brain & Behavior*, 3(4):458–471, 2020.

[9] Nitzan Shahar, Tobias U Hauser, Michael Moutoussis, Rani Moran, Mehdi Keramati, Nspn Consortium, and Raymond J Dolan. Improving the reliability of model-based decision-making estimates in the two-stage decision task with reaction-times and drift-diffusion modeling. *PLoS computational biology*, 15(2):e1006803, 2019.

[10] Thomas V Wiecki, Imri Sofer, and Michael J Frank. Hddm: Hierarchical bayesian estimation of the drift-diffusion model in python. *Frontiers in neuroinformatics*, page 14, 2013.

Hierarchical Reinforcement Learning of Locomotion Policies in Response to Approaching Objects: A Preliminary Study

Shangqun Yu
Department of Computer Science
Brown University
Providence, RI 02906
syu68@cs.brown.edu

Sreehari Rammohan
Department of Computer Science
Brown University
Providence, RI 02906
sreehari@brown.edu

Kaiyu Zheng
Department of Computer Science
Brown University
Providence, RI 02906
kzheng10@cs.brown.edu

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02906
gdk@cs.brown.edu

Abstract

Animals such as rabbits and birds can instantly generate locomotion behavior in reaction to a dynamic, approaching object, such as a person or a rock, despite having possibly never seen the object before and having limited perception of the object's properties. Recently, deep reinforcement learning has enabled complex kinematic systems such as humanoid robots to successfully move from point A to point B. Inspired by the observation of the innate reactive behavior of animals in nature, we hope to extend this progress in robot locomotion to settings where external, dynamic objects are involved whose properties are partially observable to the robot. As a first step toward this goal, we build a simulation environment in MuJoCo where a legged robot must avoid getting hit by a ball moving toward it. We explore whether prior locomotion experiences that animals typically possess benefit the learning of a reactive control policy under a proposed hierarchical reinforcement learning framework. Preliminary results support the claim that the learning becomes more efficient using this hierarchical reinforcement learning method, even when partial observability (radius-based object visibility) is taken into account.

Keywords: locomotion, reactive control, hierarchical reinforcement learning

1 Introduction

Animals have the ability to command their body—a complex kinematic system—quickly in response to environment stimuli [Pavlov, 2010]. When being approached by a person from an unexpected direction, a rabbit can immediately run away, even if the rabbit has never seen a person before. When being thrown a rock, a bird can very quickly fly away to avoid being hit, even if the rock is moving very fast and the bird does not notice until near collision.

Our work draws a connection to both this observation of nature and the progress of deep reinforcement learning (RL), where we noticed remarkable progress in using deep RL for locomotion [Peng et al., 2016, 2020; Tassa et al., 2018]. Many of the existing works in deep RL for locomotion control only consider controlling the agent from point to point. Works that do consider controlling an external object (e.g. dribbling) assume full observability of the object's properties permitted for the application of animation [Peng et al., 2017]. Other works present empirical evidence for the viability of training end-to-end locomotion policies directly from pixel input [Tassa et al., 2018], but they typically use third-person images with high sample complexity, which creates challenges for practical implementation. In this work, we focus on learning a reactive locomotion policy, which contrasts the settings in prior works where the agent is typically trained to proactively complete some task.

Our goal is to extend the success of deep reinforcement learning for locomotion control to settings where the agent must react to external objects under partial observability. As a first step, we consider a task where the agent must react to avoid being hit by an approaching ball (the "Dodge Ball Task"). The only reward signal the agent gets is a negative reward when being hit by the ball. Although reinforcement learning has achieved great improvements in domains such as games [Mnih et al., 2015] and continuous control for robotics [Gu et al., 2017], learning a policy in the continuous control setting with sparse rewards is still a major challenge in RL [Li et al., 2019]. Hierarchical Reinforcement Learning (HRL), with its structured policy and decomposition of problems into smaller subproblems [Levy et al., 2019], not only has shown strength under these challenging settings, but also provides a solution to reuse low-level skill modules on different tasks [Li et al., 2019]. Therefore, we investigate the benefits of having prior locomotion experience for this task under an HRL framework. Our preliminary results have shown that a two level feudal hierarchical agent with pre-trained low-level controller can solve the task with high sample efficiency while the end to end agent completely fails to learn with even five times the training samples.

2 Hierarchical Reinforcement Learning for Legged Reactive Control

A legged reactive control task can be formulated as a partially observable sequential decision-making problem. The environment state s can be broken into two parts, that is, $s = [s_o, s_h]$, where s_o represents the fully observable internal state of the agent, and s_h represents the state of the external object hidden to the agent. At each time step, the agent executes an action a to control its joint velocities, and it receives an observation, denoted $z = [s_o, \phi(s_h)]$, as a function of the state as a result of the action. The task is specified via a reward function $R(s)$. We are interested in the standard reinforcement learning objective, where the agent needs to maximize the cumulative reward as it interacts with the environment.

In our preliminary investigation of the reactive control setting, the objective is to minimize the expected number of collisions $\mathbb{E}[n]$ by developing an optimal policy $\pi^*(a|z)$ for an agent in a world with a single object, which takes in the current state, and outputs joint velocities for the robot. To this end, we develop the Dodge Ball Task, shown in the center of Figure 1, implemented in MuJoCo. The agent (an 8 degree of freedom ant) must learn to dodge a ball which continuously re-spawns from different locations before being shot at the agent along a linear path. In the environment, we define a sparse reward function for our task based on whether a collision happens. Negative reward is only assigned when the robot is hit by the projectile. One episode lasts 1000 steps and a new ball is spawned randomly in a position that is 5 meters away from the agent and fired at the agent at a speed of 2 meters/second every 100 steps. An agent following the optimal policy will be able to dodge all balls and thus will have a final cumulative reward close to 0.

$$R(s) = \begin{cases} \text{-5} & \text{agent hit by ball} \\ 0 & \text{otherwise} \end{cases}$$

We propose a two level Feudal Reinforcement Learning agent shown in Figure 2. Feudal Reinforcement Learning (Feudal RL) is a type of Hierarchical Reinforcement Learning where a high-level controller sets a subtask that is executed by a lower-level controller [Dayan and Hinton, 1992; Pateria et al., 2021]. The state space $S = S_h \times S_o$ consists of the positions and velocities of the agent's joints as well as the position and velocity of the projectile ball. Both high-level and low-level controllers use SAC (soft-actor critic) [Haarnoja et al., 2018] as a base learning algorithm for the task objective.

The high-level controller receives the observation $z = [s_o, \phi(s_h)]$ and outputs a subgoal g to the low-level controller. A subgoal is a 2D point with coordinates relative to the agent. The low-level controller then receives this intent along with the fully observable part of the state s_o and outputs an action consisting of joint velocities for the 8-DOF agent.

668

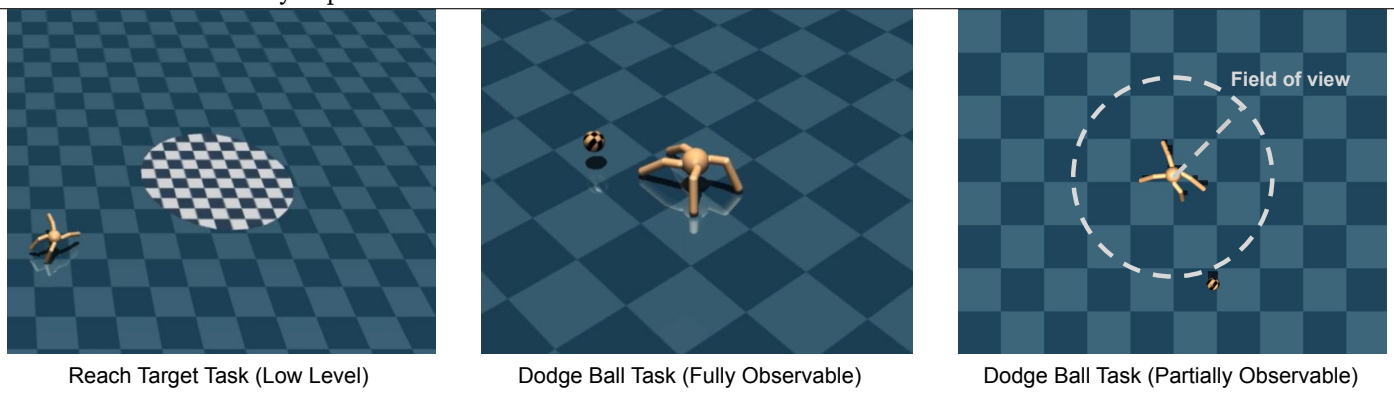


Figure 1: Left: The environment used to train the low-level controller. A target location is defined randomly near the agent (every 100 steps). The agent will gain reward proportional to the distance it traveled toward the goal. Center: In the dodge ball task, a single ball is spawned randomly in the environment (every 100 steps) and shot toward the agent along a linear trajectory. The agent receives a -5 reward every time it gets hit by the ball. Right: in the partially observable version of the dodge ball task, the agent does not receive information about the ball until it is within the field of view (a circle with 4 meter radius).

Similar to how animals possess prior locomotion skills such as running or flying, we first train the low-level controller in a different environment to gain basic locomotion skills for the agent. This environment, shown on the left side of Figure 1, requires the agent to move along a randomly specified direction vector within 300 steps, receiving reward proportional to the distance travelled in this direction. The agent receives an observation consisting of the internal configuration (joint positions and velocities) along with a subgoal g with the normalized direction vector. After training, the low-level controller is able to move the agent forward along an arbitrary direction specified by g.

Next, the low-level controller is "attached" to the high-level controller in the sense that the high-level controller outputs latent intents (similar to a subgoal direction g) which are passed into the low-level controller to compute the primitive action a. Note, because the low-level controller is trained agnostic to the high-level task, this module can be re-used – all that needs to be re-trained is the high-level controller.

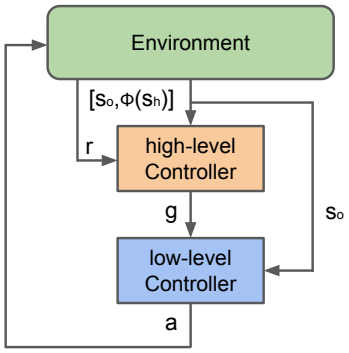


Figure 2: The high-level controller receives a state s from the environment and outputs a subgoal g to the low-level controller consisting of the direction vector it wants the agent to move along. The low-level controller takes in this subgoal and the internal state s_o and outputs a primitive action for execution in the environment.

3 Preliminary Experiments

3.1 Experiment Setup

We evaluate the performance of our algorithm in both fully and partially observable settings (Figure 1). In the fully observable setting, the agent will receive the complete information of the state s, including $[s_o, s_h]$. The full state of the ball s_h is a 6D vector that contains the 3D position (relative to the agent) and velocity of the ball (relative to the world frame). In the partially observable setting, the observation function ϕ_h is defined such that the agent can only "see" the ball once it is within a 4-meter radius (in all other circumstances 0 is populated for the elements corresponding to the obstacle in the low-dimensional state s_h). We use a standard SAC end-to-end trained agent (no low-level controller) as a baseline for comparison. Each agent is trained for 3 seeds (each lasting 2000 epochs on both environments). In each epoch, the agent collects 2000 samples of interaction from the environment and performs 200 gradient update steps.

3.2 Results and Discussion

Our results in Figure 3 show that under sparse reward, feudal HRL agents in both the fully observable and partially observable settings are able to learn reactive behavior well. We observe that the high-level controller learns to assign the direction it wants to go in as a subgoal to the low-level controller, and the low-level controller moves the robot toward the desired direction based on the subgoal, which allows the agent to avoid the ball. It's worth noting that the behavior of the HRL agent is slightly different between the partially observable setting and fully observable setting. Under the fully observable setting, the agent tends to react earlier to the ball when it is approaching compared to the agent under partial observability (Figure 4). This matches our

intuition since under partial observability, the agent can not "see" the ball until it is within 4 meters. As a result, the HRL agent under partial observability converges to a slightly lower average return. End-to-end agents struggle to learn anything meaningful under the sparse reward setting and end up with jittering policies for locomotion after extensive training.

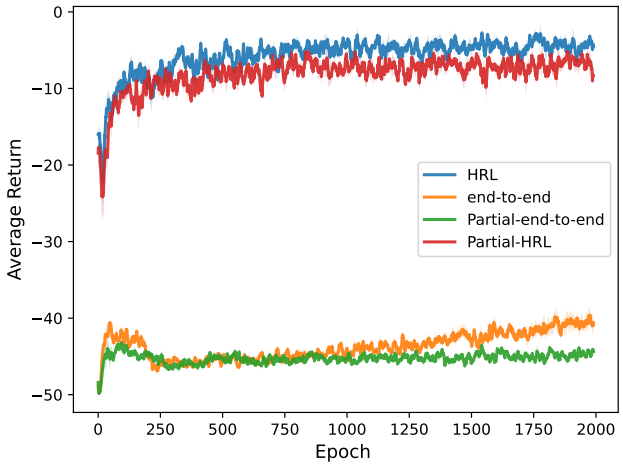


Figure 3: All agents are trained on 3 seeds. The results show that the HRL agents converged within the first 250 epochs to a cumulative reward close to 0, indicating that the agents are able to dodge most balls. Meanwhile, the agents trained end-to-end in both the partially observable and fully observable setting were not able to learn a reactive policy.

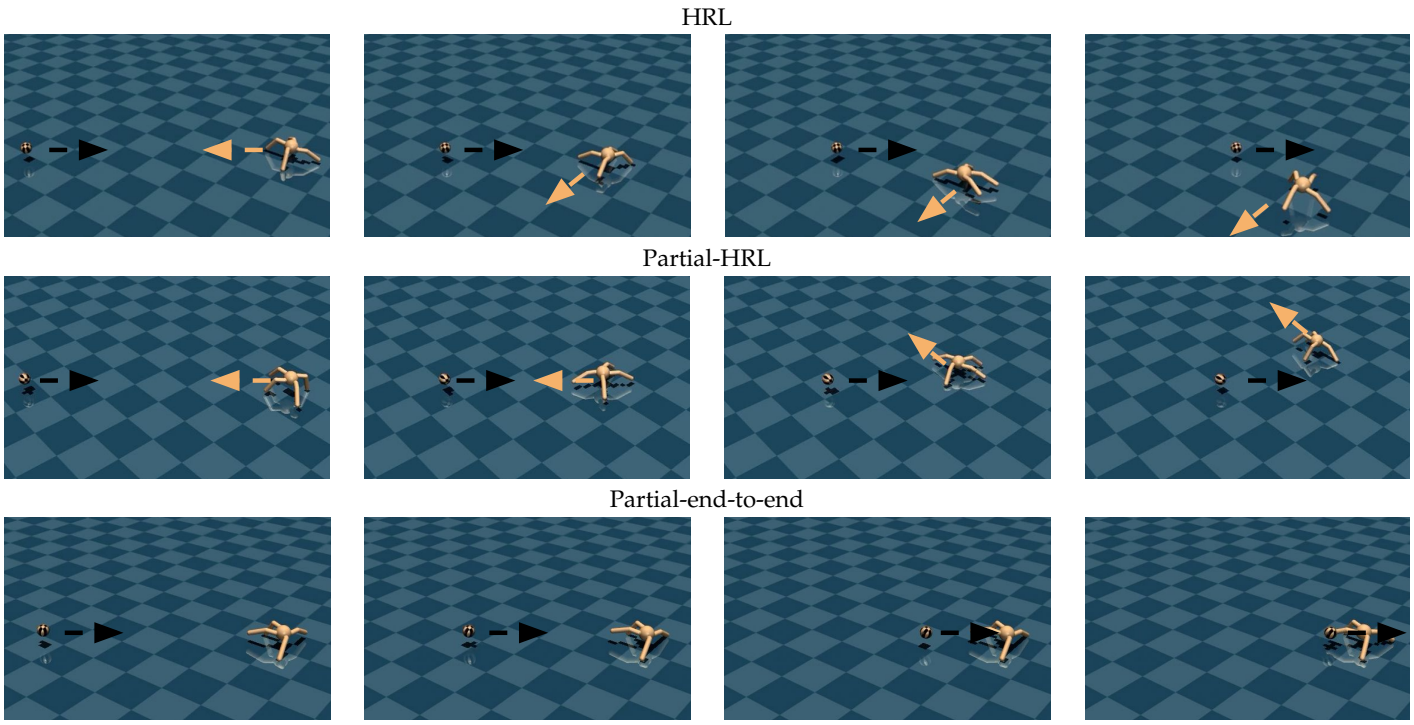


Figure 4: It takes less time for the HRL agent in the fully observable environment (1st row) to react when the ball is approaching compared to the HRL agent in the partially observable environment (2nd row), while the end-to-end agent fails to learn an effective policy in both environments. (3rd row)

3.3 Conclusion and Future Work

To study the task of reactive control, we developed a ball-dodging environment in MuJoCo that involves a subset of the challenges real world robots face such as learning a locomotion policy and acting under partial observability. We also presented a two level feudal hierarchical reinforcement learning framework for the environment and empirically demonstrated significant improvements over an end-to-end trained RL agent. We plan to extend our HRL framework to tasks that involve a variety of kinematic systems with more realistic assumptions of the agent's perception capabilities in the partial observable setting (e.g. optical flow). We also hope to increase the complexity of approaching objects (increasing the number and changing the dynamics) to create a more challenging domain. Eventually, we hope to realize our framework on a real quadrupedal robot.

References

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

Shixiang Shane Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Andrew Levy, George Dimitri Konidaris, Robert W. Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations (ICLR)*, 2019.

Siyuan Li, Rui Wang, Minxue Tang, and Chongjie Zhang. Hierarchical reinforcement learning with advantage-based auxiliary rewards. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1407–1417, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, 54(5):109:1–109:35, 2021.

P Ivan Pavlov. Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex. *Annals of neurosciences*, 2010.

Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Actor-Critic based Improper Reinforcement Learning

Mohammadi Zaki
Electrical Communication Engineering
Indian Institute of Science
Bangalore, India 560094
mohammadi@iisc.ac.in

Avinash Mohan
Department of Electrical and Computer Engineering
Boston University
Massachusetts, MA 02215
avimohan@bu.edu

Aditya Gopalan
Electrical Communication Engineering
Indian Institute of Science
Bangalore, India 560094
aditya@iisc.ac.in

Shie Mannor
Faculty of Electrical Engineering
Technion, Israel Institute of Technology
Haifa, Israel 3200003
shie@ee.technion.ac.il

Abstract

We consider an improper reinforcement learning setting where a learner is given M base controllers for an unknown Markov decision process, and wishes to combine them optimally to produce a potentially new controller that can outperform each of the base ones. This can be useful in tuning across controllers, learnt possibly in mismatched or simulated environments, to obtain a good controller for a given target environment with relatively few trials. Applications of this paradigm include the *Sim2Real* problem – simulators are typically (crude) approximations to real world scenarios, so optimal strategies devised for them may need further tweaking to perform well in reality.

Towards this, we propose an algorithm that can switch between a simple Actor-Critic (AC) based scheme and a Natural Actor-Critic (NAC) scheme depending on the available information. Both algorithms operate over a class of improper mixtures of the given controllers. For the AC-based approach we provide convergence rate guarantees to a stationary point. We provide sample complexity guarantee to achieve $\varepsilon-$(local) optimality of $\mathcal{O}\left(\frac{M}{\varepsilon^2}\log(1/\varepsilon)\right)$. For NAC, we provide sample complexity guarantee for achieving $\varepsilon-$ *global* optimality of $\mathcal{O}\left(\frac{M}{\varepsilon^3}\log(1/\varepsilon)\right)$. We validate our theory on two experimental setups (1) stabilizing the standard inverted pendulum, and (2) scheduling in constrained queueing networks. Numerical results show that that our improper policy optimization algorithm can achieve stability even when the base policies at its disposal are *unstable*, and when the optimal policy is one of the base controllers, our algorithm finds it. Further, even when the packet arrival rates to the queueing network are time-*varying* (i.e., the underlying MDP is *non-stationary*), our algorithm is able to *track* the optimal mixture of base controllers. Link to full paper:www.dropbox.com/s/pt1n7cw5zm1foaz/Improper_RL_AC_State_dependent.pdf?dl=0

Keywords: policy gradient, model-free reinforcement learning, improper learning, actor-critic methods, sample complexity, interpretability in learning.

1 Introduction

A natural approach to designing effective controllers for large, complex systems is to first approximate the system using a tried-and-true Markov decision process (MDP) model, such as the Linear Quadratic Regulator (LQR) [4] or tabular MDPs [1], and then compute (near-) optimal policies for the assumed model. Though this yields favorable results in principle, it is quite possible that errors in describing or understanding the system – leading to misspecified models – may lead to 'overfitting', resulting in subpar controllers in practice.

Moreover, in many cases, the stability of the designed controller may be crucial and more desirable than optimizing a fine-grained cost function. From the controller design standpoint, it is often easier, cheaper and more interpretable to specify or hardcode control policies based on domain-specific principles, e.g., anti-lock braking system (ABS) controllers [8]. For these reasons, we investigate in this paper a promising, *general-purpose* reinforcement learning (RL) approach towards designing controllers given pre-designed ensembles of *basic* or *atomic* controllers, which (a) allows for flexibly combining the given controllers to obtain richer policies than the atomic policies, and, at the same time, (b) can preserve the basic structure of the given class of controllers and confer a high degree of interpretability on the resulting hybrid policy.

Overview of the approach. We consider a situation where we are given 'black-box access' to M controllers (maps from state to action distributions) $\{K_1, \ldots, K_M\}$ for an unknown MDP. By this we mean that we can choose to invoke any of the given controllers at any point during the operation of the system. With the understanding that the given family of controllers is 'reasonable', we frame the problem of learning the best combination of the controllers by trial and error. We first set up an improper policy class of all randomized mixtures of the M given controllers – each such mixture is parameterized by a probability distribution over the M base controllers. Applying an improper policy in this class amounts to selecting independently at each time a base controller according to this distribution and implementing the recommended action as a function of the present state of the system. The learner's goal is to find the best performing mixture policy by iteratively testing from the pool of given controllers and observing the resulting state-action-reward trajectory.

Note that the underlying parameterization in our setting is over a set of given *controllers* which could be potentially abstract and defined for complex MDPs with continuous state/action spaces, instead of the (standard) policy gradient (PG) view where the parameterization directly defines the policy in terms of the state-action map. Our problem, therefore, hews more closely to a *meta RL* framework, in that we operate over a set of controllers that have themselves been designed using some optimization framework to which we are agnostic. This has the advantage of conferring a great deal of generality, since the class of controllers can now be chosen to promote any desirable secondary characteristic such as interpretability, ease of implementation or cost effectiveness. We corroborate our theory using extensive simulation studies in two different settings (a) a scheduling task in a constrained queueing system and (b) the well-known *Cartpole* system (see the full version of the paper:`www.dropbox.com/s/pt1n7cw5zm1foaz/Improper_RL_AC_State_dependent.pdf?dl=0`).

2 Motivating Example – Scheduling in Constrained Queueing Networks

We consider a system that comprises two queues fed by independent, stochastic arrival processes $A_i(t), i \in \{1,2\}, t \in \mathbb{N}$. The length of queue i, measured at the beginning of time slot t, is denoted by $Q_i(t) \in \mathbb{Z}_+$. A common server serves both queues and can drain at most one packet from the system in a time slot. The server, therefore, needs to decide which of the two queues it intends to serve in a given slot (we assume that once the server chooses to serve a packet, service succeeds with probability 1). The server's decision is denoted by the vector $\mathbf{D}(t) \in \mathcal{A} := \{[0,0],[1,0],[0,1]\}$, where a "1" denotes service and a "0" denotes lack thereof. Let $\mathbb{E}A_i(t) = \lambda_i$, and note that the arrival rate $\boldsymbol{\lambda} = [\lambda_1, \lambda_2]$ is unknown to the learner. We aim to find a (potentially randomized) policy π to minimize the discounted system backlog given by $J_\pi(\mathbf{Q}(0)) := \mathbb{E}^\pi_{\mathbf{Q}(0)} \sum_{t=0}^{\infty} \gamma^t \left(Q_1(t) + Q_2(t)\right)$. Any policy with $J_\pi(\cdot) < \infty$, is said to be *stabilizing* (or, equivalently, a *stable* policy). It is well known that there exist stabilizing policies iff $\lambda_1 + \lambda_2 < 1$ [9]. A policy π_{μ_1,μ_2} that chooses Queue i w.p. μ_i in every slot, can provably stabilize a system iff $\mu_i > \lambda_i, \forall\, i \in \{1,2\}$. Now, assume our control set consists of two stationary policies K_1, K_2 with $K_1 \equiv \pi_{\varepsilon,1-\varepsilon}$, $K_1 \equiv \pi_{1-\varepsilon,\varepsilon}$ and sufficiently small $\varepsilon > 0$. That is, we have $M = 2$ controllers K_1, K_2. Clearly, neither of these can, by itself, stabilize a network with $\boldsymbol{\lambda} = [0.49, 0.49]$. However, an *improper* mixture of the two that selects K_1 and K_2 each with probability $1/2$ can. In fact, as Fig. 1 shows, our improper learning algorithm can stabilize *all* arrival rates in $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \Delta ABC$, without prior knowledge of $[\lambda_1, \lambda_2]$. In other words, our algorithm enlarges the stability region by the triangle ΔABC, over and above $\mathcal{C}_1 \cup \mathcal{C}_2$. We will return to these examples in Sec. 5, and show, using experiments, (1) how our improper learner converges to the stabilizing mixture of the available policies and (2) if the optimal policy is among the available controllers, how our algorithm can find and converge to it.

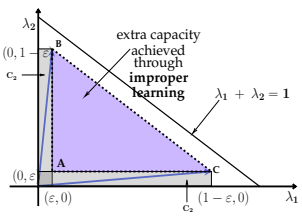


Figure 1: K_1 and K_2 by themselves can only stabilize $\mathcal{C}_1 \cup \mathcal{C}_2$ (gray rectangles). With improper learning, we enlarge the set of stabilizable arrival rates by the triangle ΔABC shown in purple, above.

3 Problem Statement and Notation

A (finite) Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathtt{P}, r, \rho, \gamma)$ is specified by a finite state space $\mathcal{S}$, a finite action space $\mathcal{A}$, a transition probability matrix $\mathtt{P}$, where $\mathtt{P}(\tilde{s}|s,a)$ is the probability of transitioning into state $\tilde{s}$ upon taking action $a \in \mathcal{A}$ in state s, a single stage reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a starting state distribution ρ over $\mathcal{S}$ and a discount factor $\gamma \in (0,1)$. A (stationary) *policy* or *controller* $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ specifies a decision-making strategy in which the learner chooses actions (a_t) adaptively based on the current state (s_t), i.e., $a_t \sim \pi(s_t)$. π and ρ, together with $\mathtt{P}$, induce a probability measure $\mathbb{P}_\rho^\pi$ on the space of all sample paths of the underlying Markov process and we denote by $\mathbb{E}_\rho^\pi$ the associated expectation operator. The value function of policy π (also called the value of policy π), denoted by V^π is the total discounted reward obtained by following π, i.e., $V^\pi(\rho) := \mathbb{E}_\rho^\pi \sum_{t=0}^\infty \gamma^t r(s_t, a_t)$.

Improper Learning. We assume that the learner is provided with a finite number of (stationary) controllers $\mathcal{C} := \{K_1, \cdots, K_M\}$ and, as described below, set up a parameterized improper policy class $\mathcal{I}_{soft}(\mathcal{C})$ that depends on $\mathcal{C}$. The aim therefore, is to identify the best policy for the given MDP within this class, i.e.,

$$\pi^* = \underset{\pi \in \mathcal{I}_{soft}(\mathcal{C})}{\operatorname{argmax}} \; V^\pi(\rho). \tag{1}$$

We now describe the construction of the class $\mathcal{I}_{soft}(\mathcal{C})$.

The Softmax Policy Class. We assign weights $\theta_m \in \mathbb{R}$, to each controller $K_m \in \mathcal{C}$ and define $\theta := [\theta_1, \cdots, \theta_M]$. The improper class $\mathcal{I}_{soft}$ is parameterized by θ as follows. In each round, the policy $\pi_\theta \in \mathcal{I}_{soft}(\mathcal{C})$ chooses a controller drawn from $\mathtt{softmax}(\theta)$, i.e., the probability of choosing Controller K_m is given by, $\pi_\theta(m) := e^{\theta_m} / \left(\sum_{m'=1}^M e^{\theta_{m'}}\right)$. Note, therefore, that in every round, our algorithm interacts with the MDP only *through* the controller sampled in that round. In the rest of the paper, we will deal exclusively with a fixed and given $\mathcal{C}$ and the resultant $\mathcal{I}_{soft}$. therefore, we overload the notation $\pi_{\theta_t}(a|s)$ for any $a \in \mathcal{A}$ and $s \in \mathcal{S}$ to denote the probability with which the algorithm chooses action a in state s at time t. For ease of notation, whenever the context is clear, we will also drop the subscript θ i.e., $\pi_{\theta_t} \equiv \pi_t$. Hence, we have at any time $t \geqslant 0 : \pi_{\theta_t}(a|s) = \sum_{m=1}^M \pi_{\theta_t}(m) K_m(s,a)$. Finally, let for any event E, $\mathbb{I}\{E\}$ denote the indicator that event E is true.

Comparison to the standard PG setting. The problem we define is different from the usual policy gradient setting where the parameterization completely defines the policy in terms of the state-action mapping. One can use the methodology followed in [6], by assigning a parameter $\theta_{s,m}$ for every $s \in \mathcal{S}, m \in [M]$. With basic calculations, it can be shown that this is equivalent to the tabular setting with S states and M actions, with the new 'reward' defined by $r(s,m) := \sum_{a \in \mathcal{A}} K_m(s,a) r(s,a)$ where $r(s,a)$ is the usual expected reward obtained at state s and playing action $a \in \mathcal{A}$. By following the approach in [6] on this modified setting, it can be shown that a vanilla policy gradient approach can converge for each $s \in \mathcal{S}$, $\pi_\theta(m^*(s) \mid s) \to 1$, for every $s \in \mathcal{S}$, which is the optimum policy. However, we address the problem to select *a single* controller (from within $\mathcal{I}_{soft}$, the convex hull of the given M controllers) , which would guarantee maximum return if one plays that single mixture for all time.

4 Actor-Critic based Improper Learning

We begin by noting that V^{π_θ}, as described in Section 3, is *nonconcave* in θ for both direct and softmax parameterizations, which renders analysis with standard tools of convex optimization inapplicable.

Lemma 4.1. *(Non-concavity of Value function) There is an MDP and a set of controllers, for which the maximization problem of the value function (i.e.* (1)*) is non-concave for both the SoftMax and direct parameterizations, i.e., $\theta \mapsto V^{\pi_\theta}$ is non-concave.*

In this section, we provide an algorithm based on an actor-critic framework for solving our problem. Actor-Critic methods are well-known to have low variance than their Monte-carlo counterparts [5].

We begin by proposing modifications to the standard Q-function and advantage function definitions. Recall that we wish to solve for the following optimization problem: $\max_{\pi \in \mathcal{I}_{\mathtt{soft}}} \mathbb{E}_{s\sim\rho}[V^\pi(s)]$, where π is some distribution over the M base controllers. Let $\tilde{Q}^\pi(s,m) := \sum_{a\in\mathcal{A}} K_m(s,a) Q^\pi(s,a)$. Let $\tilde{A}^\pi(s,m) := \sum_{a\in\mathcal{A}} K_m(s,a) A^\pi(s,a) = \sum_{a\in\mathcal{A}} K_m(s,a) Q^\pi(s,a) - V^\pi(s)$, where Q^π and A^π are the usual action-value functions and advantage functions respectively. We also define the new reward function $\tilde{r}(s,m) := \sum_{a\in\mathcal{A}} K_m(s,a) r(s,a)$ and a new transition kernel $\tilde{P}(s'|s,m) := \sum_{a\in\mathcal{A}} K_m(s,a) P(s'|s,a)$. Then, following the distribution π over the controllers induces a Markov Chain on the state space $\mathcal{S}$. Define $\nu_\pi(s,m)$ as the state-controller visitation measure induced by the *policy* π: $\nu_\pi(s,m) := (1-\gamma)\sum_{t\geqslant 0} \gamma^t \mathbb{P}^\pi(s_t = s, m_t = m) = d_\rho^\pi(s)\pi(m)$. With these definitions, we have the following variant of the policy-gradient theorem. For the gradient ascent update of the parameters θ we need to estimate $\tilde{A}^{\pi_\theta}(s,m)$ where (s,m) are drawn according to $\nu_{\pi_\theta}(\cdot,\cdot)$. We recall how to sample from ν_π. Following [5, 2] and casting into our setting, observe that ν_π is a stationary distribution of a Markov chain over the pair $(s,m) \mapsto (s',m')$ with transition kernel defined by $\bar{P}(s'|s,m) := \gamma \tilde{P}(s'|s,m) + (1-\gamma)\rho(s')$ and $m' \sim \pi(.)$.

Algorithm Description. We present the algorithm in detail in Algorithm 1 along with a subroutine Alg 2 which updates the critic's parameters. ACIL is a single-trajectory based algorithm, in the sense that it does not have to be reset along

the run. We begin with the critic's updates. The **critic** uses linear function approximation $V_w(s) := \varphi(s)^\top w$, and uses TD learning to update its parameters $w \in \mathbb{R}^d$. We assume that $\varphi(\cdot) : \mathcal{S} \to \mathbb{R}^d$ is a known feature mapping. Let Φ be the corresponding $|S| \times d$ matrix. We assume that the columns of Φ are linearly independent. Next, based on the critic's parameters, the **actor** approximates the $\tilde{A}(s,m)$ function using the TD error: $\mathcal{E}_w(s,m,s') = \tilde{r}(s,m) + (\gamma\varphi(s') - \varphi(s))^\top w$.

In order to provide guarantees of the convergence rates of Algorithm ACIL, we make the following assumptions, which are standard in RL literature [5, 3, 10].

Assumption 4.2 (Uniform Ergodicity)**.** For any $\theta \in \mathbb{R}^M$, consider the Markov Chain induced by the policy π_θ, and following the transition kernel $\tilde{P}(.|s,m)$. Let ξ_{π_θ} be the stationary distribution of this Markov Chain. We assume that there exists constants $\kappa > 0$ and $\xi \in (0,1)$ such that

$$\sup_{s\in\mathcal{S}} \left\|\mathbb{P}\left(s_t \in \cdot | s_0 = s, \pi_\theta\right) - \xi_{\pi_\theta}(\cdot)\right\|_{TV} \leqslant \kappa\xi^t.$$

Further, let $L_\pi := \mathbb{E}_{\nu_\pi}[\varphi(s)(\gamma\varphi(s') - \varphi(s))^\top]$ and $v_\pi := \mathbb{E}_{\nu_\pi}[r(s,m,s')\varphi(s)]$. The optimal solution to the critic's TD learning is now $w^* := -L_\pi^{-1} v_\pi$.

Assumption 4.3. There exists a positive constant Γ_L such that for all $w \in \mathbb{R}^d$, we have $\langle w - w^*, A(w - w^*)\rangle \leqslant -\Gamma_L \left\|w - w^*\right\|_2^2$.

Based on the above two assumptions, let $L_V := (2\sqrt{2}C_{\kappa\xi} + 1)/(1-\gamma)$, where $C_{\kappa\xi} = \left(1 + \lceil \log_\xi \frac{1}{\kappa} \rceil + \frac{1}{1-\xi}\right)$.

Theorem 4.4. *Consider the Actor-Critic improper learning algorithm ACIL (Alg 1). Assume* $\sup_{s\in\mathcal{S}} \|\varphi(s)\|_2 \leqslant 1$. *Under Assumptions 4.2 and 4.3 with step-sizes chosen as* $\alpha = \left(\frac{1}{4L_V\sqrt{M}}\right)$, $\beta = \min\left\{\mathcal{O}\left(\Gamma_L\right), \mathcal{O}\left(1/\Gamma_L\right)\right\}$, *batch-sizes* $H = \mathcal{O}\left(\frac{1}{\varepsilon}\right)$, $B = \mathcal{O}\left(1/\varepsilon\right)$, $T_c = \mathcal{O}\left(\frac{\sqrt{M}}{\Gamma_L}\log(1/\varepsilon)\right)$, $T = \mathcal{O}\left(\frac{\sqrt{M}}{(1-\gamma)^2\varepsilon}\right)$, *we have* $\mathbb{E}[\left\|\nabla_\theta V(\theta_{\hat{T}})\right\|_2^2] \leqslant \varepsilon + \mathcal{O}(\Delta_{critic})$. *Hence, the total sample complexity is* $\mathcal{O}\left(M(1-\gamma)^{-2}\varepsilon^{-2}\log(1/\varepsilon)\right)$.

Algorithm 1 Actor-Critic based Improper RL (ACIL)

Input: φ, actor stepsize α, critic stepsize β, regularization parameter λ, 'AC' or 'NAC'
Initialize: $\theta_0 = (1,1,\ldots,1)_{M\times 1}$, $s_0 \sim \rho$
flag $= \mathbb{I}\{\text{NAC}\}$ {Selects AC or NAC}
for $t \leftarrow 0$ **to** $T-1$ **do**
  $s_{init} = s_{t-1,B}$ (when $t = 0$, $s_{init} = s_0$)
  $w_t, s_{t,0} \leftarrow \texttt{Critic-TD}(s_{init}, \pi_{\theta_t}, \varphi, \beta, T_c, H)$
  $F_t(\theta_t) \leftarrow 0$.
  for $i \leftarrow 0$ **to** $B-1$ **do**
    $m_{t,i} \sim \pi_{\theta_t}$, $a_{t,i} \sim K_{m_{t,i}}(s_{t,i}, \cdot)$
    $s_{t,i+1} \sim \tilde{P}(.|s_{t,i}, m_{t,i})$
    $\mathcal{E}_{w_t}(s_{t,i}, m_{t,i}, s_{t,i+1}) = \tilde{r}(s_{t,i}, m_{t,i}) + (\gamma\varphi(s_{t,i+1}) - \varphi(s_{t,i}))^\top w_t$
    $F_t(\theta_t) \leftarrow F_t(\theta_t) + \frac{1}{B}\psi_{\theta_t}(m_{t,i})\psi_{\theta_t}(m_{t,i})^\top$
  end for
  if {flag} **then**
    $G_t := [F_t(\theta_t) + \lambda I]$
    $\theta_{t+1} = \theta_t + G_t^{-1}\frac{\alpha}{B}\left(\sum_{i=0}^{B-1}\mathcal{E}_{w_t}(s_{t,i}, m_{t,i}, s_{t,i+1})\psi_{\theta_t}(m_{t,i})\right)$
  else
    $\theta_{t+1} = \theta_t + \frac{\alpha}{B}\sum_{i=0}^{B-1}\mathcal{E}_{w_t}(s_{t,i}, m_{t,i}, s_{t,i+1})\psi_{\theta_t}(m_{t,i})$
  end if
  $\pi_{\theta_{t+1}} = \texttt{softmax}(\theta_{t+1})$
end for
Output: $\theta_{\hat{T}}$ with $\hat{T}$ chosen uniformly at random from $\{1,\ldots,T\}$

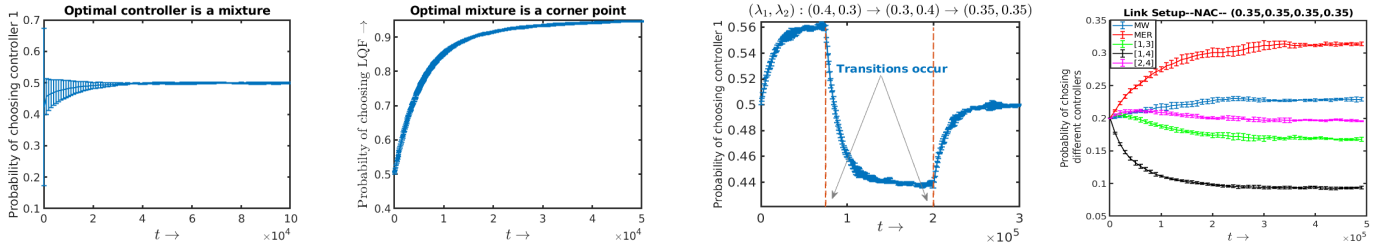
Here, $\Delta_{critic} := \max_{\theta\in\mathbb{R}^M} \mathbb{E}_{\nu_{\pi_\theta}}\left[\left|V^{\pi_\theta}(s) - V^{w^*_{\pi_\theta}}\right|^2\right]$, which equals zero, if the value function lies in the linear space spanned by the features.

Algorithm 2 Critic-TD Subroutine

Input: $s_{init}, \pi, \varphi, \beta, T_c, H$
Initialize: w_0
for $k \leftarrow 0$ **to** $T_c - 1$ **do**
  $s_{k,0} = s_{k-1,H}$ (when $k = 0$, $s_{k,0} = s_{init}$)
  for $j \leftarrow 0$ **to** $H-1$ **do**
    $m_{k,j} \sim \pi(.)$, $a_{k,j} \sim K_{m_{k,j}}(s_{k,j}, \cdot)$, $s_{k,j+1} \sim \tilde{P}(.|s_{k,j}, m_{k,k})$
    $\mathcal{E}_{w_k}(s_{k,j}, m_{k,j}, s_{k,j+1}) = \tilde{r}(s_{k,j}, m_{k,j}) + (\gamma\varphi(s_{k,j+1}) - \varphi(s_{k,j}))^\top w_k$
  end for
  $w_{k+1} = w_k + \frac{\beta}{H}\sum_{i=0}^{H-1}\mathcal{E}_{w_k}(s_{k,i}, m_{k,i}, s_{k,i+1})\varphi(s_{k,i})$
end for
Output: $w_{T_c}, s_{T_c-1,H}$

Next we provide the global optimality guarantee for the Natural-Actor-Critic version of ACIL.

Theorem 4.5. *Assume* $\sup_{s\in\mathcal{S}} \|\varphi(s)\|_2 \leqslant 1$. *Under Assumptions 4.2 and 4.3 with step-sizes chosen as* $\alpha = \left(\frac{\lambda^2}{2\sqrt{M}L_V(1+\lambda)}\right)$, $\beta = \min\left\{\mathcal{O}\left(\Gamma_L\right), \mathcal{O}\left(1/\Gamma_L\right)\right\}$, *batch-sizes* $H = \mathcal{O}\left(\frac{1}{\Gamma_L\varepsilon^2}\right)$, $B = \mathcal{O}\left(\frac{1}{(1-\gamma)^2\varepsilon^2}\right)$, $T_c = \mathcal{O}\left(\frac{\sqrt{M}}{\Gamma_L}\log(1/\varepsilon)\right)$, $T = \mathcal{O}\left(\frac{\sqrt{M}}{(1-\gamma)^2\varepsilon}\right)$ *and* $\lambda = \mathcal{O}(\Delta_{critic})$ *we have* $V(\pi^*) - \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[V(\pi_{\theta_t})\right] \leqslant \varepsilon + \mathcal{O}\left(\sqrt{\frac{\Delta_{actor}}{(1-\gamma)^3}}\right) + \mathcal{O}(\Delta_{critic}) + \mathcal{O}(\Delta_{critic})$. *Hence, the total sample complexity is* $\mathcal{O}\left(\frac{M}{(1-\gamma)^4\varepsilon^3}\log\frac{1}{\varepsilon}\right)$. *Here* $\Delta_{actor} := \max_{\theta\in\mathbb{R}^M}\min_{w\in\mathbb{R}^d}\mathbb{E}_{\nu_{\pi_\theta}}[[\psi_\theta^\top w - A_{\pi_\theta}(s,m)]^2]$ *and* Δ_{critic} *is same as before.*



(a) Arrival rate:$(\lambda_1, \lambda_2) = (0.4, 0.4)$ (b) Arrival rate:$(\lambda_1, \lambda_2) = (0.35, 0.35)$ (c) (Estimated) 2 queue system with time-varying arrival rates (d) Path Graph Network setting with 4 links (queues)

Figure 2: NACIL algorithm applied to various queuing networks show convergence to the best mixture policy.

5 Numerical Results

We perform some queueing theoretic simulations on the natural actor critic version of ACIL, which we will call NACIL.

Stationary arrival rates. Recall the example that we discussed in Sec. 2. We consider the case with Bernoulli arrivals with rates $\boldsymbol{\lambda} = [\lambda_1, \lambda_2]$ and are given two base/atomic controllers $\{K_1, K_2\}$, where controller K_i serves Queue i with probability 1, $i = 1, 2$. As can be seen in Fig. 2(a) when $\boldsymbol{\lambda} = [0.4, 0.4]$ (equal arrival rates), NACIL converges to an improper mixture policy that serves each queue with probability $[0.5, 0.5]$. Next in Fig 2(b) shows a situation where one of the base controllers, i.e., the "Longest-Queue-First" (LQF) is the optimal controller. NACIL converges correctly to the corner point.

Non-stationary arrival. Next, Fig. 2(c) shows a setting where the scheduler is now given two base/atomic controllers $\mathcal{C} := \{K_1, K_2\}$, i.e. $M = 2$. Controller K_i serves Queue i with probability 1, $i = 1, 2$. As can be seen in Fig. 2(d), the arrival rates $\boldsymbol{\lambda}$ to the two queues vary over time (adversarially) during the learning. In particular, $\boldsymbol{\lambda}$ varies from $(0.4, 0.3) \to (0.3, 0.4) \to (0.35, 0.35)$. The transition of (λ_1, λ_2) occur precisely at $t = 75000$ and at $t = 2 \times 10^5$ which is *unknown* to the learner. We show the probability of choosing K_1. NACIL successfully tracks the changing arrival rates.

Path Graph Network. The scheduling constraints in the first network we study dictate that Queues i and $i + 1$ cannot be served simultaneously for $i \in [N - 1]$ in any round $t \geqslant 0$. Such queueing systems are called *path graph* networks [7]. We work with $N = 4$. Therefore, sets of queues which can be served simultaneously are $\mathcal{A} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 3\}, \{2, 4\}, \{1, 4\}\}$. The constituents of $\mathcal{A}$ are called *independent sets* in the literature. In each round t, the scheduler selects an independent set to serve the queues therein. Let $Q_j(t)$ be the backlog of Queue j at time t. We use the following base controllers: (i) K_1 : Max Weight (MW) controller [9] chooses a set $s_t := \text{argmax}_{\underline{\mathbf{S}} \in \mathcal{A}} \sum_{j \in \underline{\mathbf{S}}} Q_j(t)$, i.e, the set with the largest backlog, (ii) K_2 : Maximum Egress Rate (MER) controller chooses a set $s_t := \text{argmax}_{\underline{\mathbf{S}} \in \mathcal{A}} \sum_{j \in \underline{\mathbf{S}}} \mathbb{I}\{Q_j(t) > 0\}$, i.e, the set which has the maximum number of non-empty queues.We also choose K_3, K_4 and K_5 which serve the sets $\{1, 3\}, \{2, 4\}, \{1, 4\}$ respectively with probability 1. We fix the arrival rates as $(0.35, 0.35, 0.35, 0.35)$. The plot in Fig. 2(d) show NACIL converges to a improper mixture of the 5 base controllers.

References

[1] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 89–96. Curran Associates, Inc., 2009.

[2] Anas Barakat, Pascal Bianchi, and Julien Lehmann. Analysis of a target-based actor-critic algorithm with linear function approximation. *CoRR*, abs/2106.07472, 2021.

[3] Jalaj Bhandari, Daniel Russo, and Raghav Singal. A finite time analysis of temporal difference learning with linear function approximation. *Oper. Res.*, 69:950–973, 2018.

[4] Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the Sample Complexity of the Linear Quadratic Regulator. *arXiv e-prints*, October 2017.

[5] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[6] Jincheng Mei, Chenjun Xiao, Csaba Szepesvari, and Dale Schuurmans. On the global convergence rates of softmax policy gradient methods. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6820–6829. PMLR, 2020.

[7] Avinash Mohan, Aditya Gopalan, and Anurag Kumar. Throughput optimal decentralized scheduling with single-bit state feedback for a class of queueing systems. *ArXiv*, abs/2002.08141, 2020.

[8] Mircea-Bogdan Radac and Radu-Emil Precup. Data-driven model-free slip control of anti-lock braking systems using reinforcement q-learning. *Neurocomput.*, 275(C):317–329, January 2018.

[9] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, 1992.

[10] Tengyu Xu, Zhe Wang, and Yingbin Liang. Improving sample complexity bounds for (natural) actor-critic algorithms. In *NeurIPS*, volume 33, pages 4358–4369, 2020.

Supporting End Users in Defining Reinforcement-Learning Problems for Human-Robot Interactions

Valerie Zhao
Department of Computer Science
University of Chicago
Chicago, IL 60637
vzhao@uchicago.edu

Michael L. Littman
Department of Computer Science
Brown University
Providence, RI 02912
michael_littman@brown.edu

Shan Lu
Department of Computer Science
University of Chicago
Chicago, IL 60637
shanlu@uchicago.edu

Sarah Sebo
Department of Computer Science
University of Chicago
Chicago, IL 60637
sarahsebo@uchicago.edu

Blase Ur
Department of Computer Science
University of Chicago
Chicago, IL 60637
blase@uchicago.edu

Abstract

Reinforcement learning (RL) can help agents learn complex tasks that would be hard to specify using standard imperative programming. However, end users may have trouble personalizing their technology using RL due to a lack of technical expertise. Prior work has explored means of supporting end users after a problem for the RL agent to solve has been defined. Little work, however, has explored how to support end users *when* defining this problem. We propose a tool to provide structured support for end users defining problems for RL agents. Through this tool, users can (i) directly and indirectly specify the problem as a Markov decision process (MDP); (ii) receive automatic suggestions on possible MDP changes that would enhance training time and accuracy; and (iii) revise the MDP after training the agent to solve it. We believe this work will help reduce barriers to using RL and contribute to the existing literature on designing human-in-the-loop systems.

Keywords: End-User Programming, Reinforcement Learning, Markov Decision Process, Human-Robot Interaction, Human-in-the-Loop

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants CCF-1837120 and CCF-1836948. We thank Yunfei Shen and Lilith Yu for their contributions to this work.

1 Introduction

Reinforcement learning (RL) is applicable to a variety of domains [Sutton and Barto, 1998, Barto et al., 2017, Kober et al., 2013], but end users without sufficient technical expertise may have trouble applying it to their own needs. They may face challenges such as determining how to train the RL agent efficiently and how to define and administer rewards. Existing work has attempted to support end users by incorporating user feedback through policy shaping, guided exploration, and augmented value functions [Arzate Cruz and Igarashi, 2020].

While most work supports end users in training the RL agent on a defined task, little work has attempted to address end-user challenges in *defining* that task. It may appear straightforward to define an RL task as a Markov decision process (MDP) with a set of states, a set of actions, and a set of rewards. However, users might find it difficult to anticipate the appropriate state space and action space. Specifying too many states or actions can cause the agent to learn over an excessively long period of time. Specifying too few, on the other hand, may cause the agent to learn a suboptimal policy.

We propose a tool for assisting end users in defining and refining RL problems. Our tool consists of a graphical interface that guides the user in defining RL problems both before and after training an agent. Using built-in primitives, users will first directly or indirectly specify the states, actions, and rewards of the MDP. The tool will generate recommendations for refining this MDP, which the user can accept or reject. The RL agent will then train on the resulting MDP. (For our tool, we assume a Q-learning agent.) The interface shows the policy at each timestep to help the user evaluate whether their MDP appropriately captures the intended task. When training terminates, the user can accept the final policy, revise it, or revise the MDP definition and try again.

Our work focuses on RL applications in Human-Robot Interaction (HRI), which concern social interactions between people and robots. In HRI tasks, robots must behave in a socially appropriate manner by adapting to personal preferences, cultural norms, and other context-specific factors [Andrist et al., 2015, Wang et al., 2010, Lee et al., 2012, Leyzberg et al., 2014]. One example in HRI is the issue of robot abuse. Multiple public incidents have occurred in which people purposefully blocked the robots' path or sensors, hurled obscenities against them, or resorted to violent behaviors like kicking and hitting [Nomura et al., 2015, Brščić et al., 2015, Salvini et al., 2010]. Robots designed to accomplish tasks unsupervised must somehow mitigate these incidents.

How can a user help the robot navigate complex social interactions, such as avoiding or pacifying incidents of robot abuse? Programming languages like Python for the Pepper robot[1] require substantial technical knowledge. End-user programming paradigms and systems, such as Choreographe [Pot et al., 2009], demands less technical knowledge but still requires the user to anticipate the ideal robot behavior under a specific circumstance for all possible circumstances. A similar problem arises for Learning from Demonstration (LfD).

In contrast, an RL approach can help the robot adapt to the many scenarios it will encounter without requiring the user to detail each possibility, making it especially well suited to those with minimal programming and domain expertise. Unfortunately, to leverage RL for this problem, the user still needs advanced knowledge of the environment. In our example, users may neglect to realize that states related to bystanders may affect the likelihood of robots being bullied, choosing instead to only consider states that relate to the bullies and the robot. Our tool seeks to address this gap, with a focus on HRI tasks but may also be extended to other applications.

2 Vision for the Interaction

We envision a multi-stage interaction between the user and the RL system. Via a graphical interface, the user will first specify the states, actions, and rewards of the MDP (Figure 1) by selecting from a set of system-defined choices (Figure 2). However, some users may find it difficult to think about RL tasks in the terms of MDP. The interface will offer these users the option to, instead, define the task by predicting parts of the optimal policy. They will describe these predictions in the form of IF-THEN (trigger-action) statements, and the system will extract a potential MDP based on these statements. For example, if the user writes "IF a person starts hitting the robot THEN the robot should run away," the system will infer that "whether there is a person hitting the robot" is a relevant state and "robot running away from people" is a relevant action for the MDP overall.

The system will review the MDP and recommend possible changes to improve the training efficiency and accuracy. These recommendations may include removing redundant state features and actions, or adding those that seem relevant to the task. The user accepts or rejects these recommendations based on their understanding of their intended task, producing a tentative MDP on which to train the robot.

Training the robot will produce information for debugging the MDP. During training, the interface will show the policy of the robot at each timestep. The user can monitor this to understand the trends in the robot's learning, such as how quickly they acquire the right behaviors. When the robot has completed learning, the interface visualizes the final policy.

[1] https://www.softbankrobotics.com/emea/en/pepper

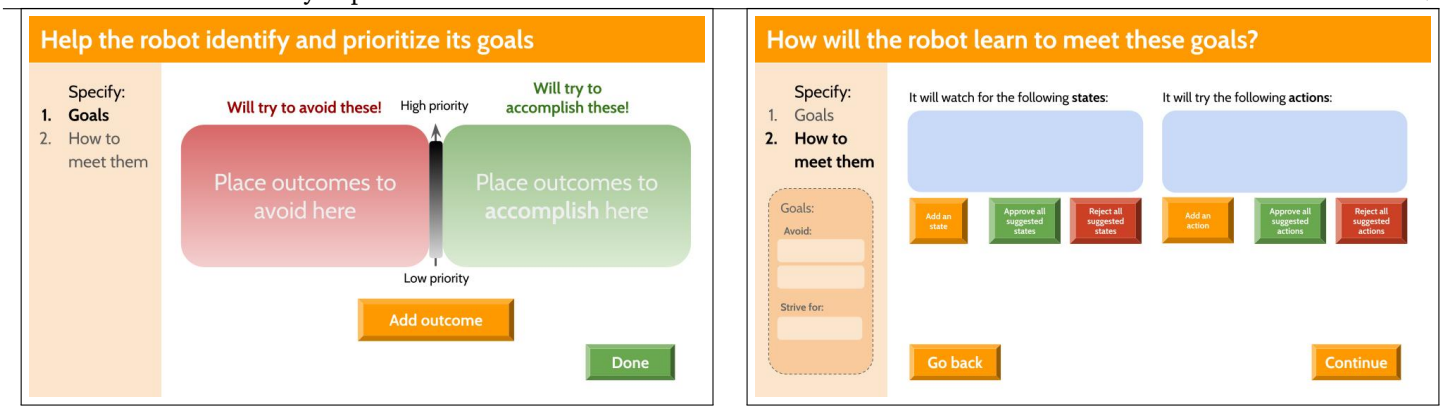


Figure 1: Design sketches showing selected stages of the user interaction with our graphical interface. Left: The user can specify outcomes and assign reward values as part of the MDP. Right: The user can specify states and actions as well as review recommendations from the system.

State Features

- **Person's Location**: their position and distance relative to the robot;
- **Person's Mood**: how happy, engaged, and/or frustrated they are;
- **Person's Actions**: whether they are looking at the robot or speaking, what they are saying, or other user-specified actions;
- **Miscellaneous**: what the robot is doing, is the environment noisy, is there trash around the robot.

(Robot's) Actions

- **Gaze**: look at a specific person, object, or somewhere elsewhere;
- **Speech**: say a specified sample of text, stop speaking;
- **Gesture**: exhibit, touch, present, point to, sweep over, or group (an) object(s);
- **Navigate**: start traveling a specified distance toward a specified direction at a specified speed, stop traveling;
- **Miscellaneous**: rest, shut down.

Outcomes to Reward

- **Person's Mood**: whether they are happy, engaged, and/or frustrated;
- **Person's Speech**: whether they refer to the robot positively, whether they respond to it and whether this response matches a specified sample of text;
- **Person's Actions**: whether they acknowledge or ignore the robot; whether their actions match a specified action (e.g. if the robot is giving directions).

Figure 2: A work-in-progress list of interface primitives we plan to support, deduced from reviewing existing tasks studied in HRI literature (e.g. [Sauppé and Mutlu, 2014]). "Person" refers to a single individual interacting with the robot, such as a conversation partner.

The user may even choose to see the robot execute this policy, either in simulation or in a real-world deployment. The user can modify this final policy, or return to the beginning to revise the MDP and repeat the process.

To illustrate a use case of our proposed tool, we walk through the following example. Suppose a robot is deployed at a store and tasked with maximizing average customer satisfaction, as measured by an in-store exit survey. A user who is comfortable with the ideas of MDP might use our interface to specify state features related to the mood of the customers. They might specify actions such as greeting customers and approaching customers to offer assistance. They might consider each customer to be an episode, and the positive or negative survey rating of the customer to be the reward for that episode. A different user might choose to to define the tasks by writing IF-THEN statements such as "IF a customer begins to get frustrated THEN approach the customer to offer assistance," allowing the system to extract a similar MDP as above.

Based on the MDP, the tool might recommend additions to the states, actions, and rewards to account for potential robot abuse. For example, it might recommend the age of the customer as a state feature since children might be more unruly and more likely to hinder the robot, and the ability to run away as a possible action. The user decides to accept these changes, and then train the robot. When training has concluded, suppose the user sees that the final policy has the robot walk away from people intentionally blocking its path. The user wants to know if it will be more effective in raising customer moods and reducing antagonistic encounters if the robot instead asks the customer to politely move, which was not an action included in the original MDP. They can, therefore, return to the starting MDP, modify it to add this new action, and re-train the robot.

3 Development and Evaluation

Our development road map consists of low-fidelity prototyping, an implementation stage, and then a final user study evaluation. We will first hold remote, moderated sessions with participants of various levels of technical expertise to prototype our designs with them. After iterating on the designs based on participants' feedback, we will implement them. We evaluate the implemented system with a user study.

In the user study, participants will use our tool to train a robot for a set of HRI tasks. We may select tasks applicable to either the NAO[2] or the Stretch[3] robots. There may be a baseline condition in which participants use a simplified version of tool, without receiving automatic support in the form of MDP recommendations or policy visualizations. We will ask participants to rate the difficulty of the tasks, the convenience of the tool, and the confidence in their answers. We will administer the System Usability Scale and ask participants to describe their thought process during the experience.

4 Related Work

In this section, we review selected existing work on applying RL to HRI and on RL tools for end users.

4.1 HRI Applications for RL

Hemminghaus and Kopp [2017] explored how Q-learning can help a robot learn when and how to provide assistance to the user without overwhelming them or distracting them from the task at hand. Park et al. [2019] designed a robot for expanding childrens' language skills by using Q-learning to determine the complexity of the stories it tells to children. Ramírez et al. [2016] used inverse RL to train robots to approach humans in a socially appropriate manner. In these works, the authors defined the RL problem for the robot to solve, rather than asking novices to determine the problem, which our work will enable.

4.2 RL Support for End Users

A large body of work has investigated the potential for interactive RL, in which the user is able to adjust the reward function and state features during the agent's learning process [Amershi et al., 2014, Arzate Cruz and Igarashi, 2020]. Thomaz and Breazeal [2006] found that study participants used rewards to provide feedback on the robot's past actions and provide guidance on future actions. Incorporating these reward variants helped RL non-experts efficiently train a virtual robot to successfully bake a cake. In addition, Thomaz et al. [2006] found that study participants preferred giving positive rewards, and often adapted their reward behavior as they became more knowledgeable about the robot's training process. MacGlashan et al. [2017] showed that people construe the reward they deliver as being a critique relative to the agent's recent behavior. Judah et al. [2010] explored means of combining agent exploration with end-user critique after each session to label certain actions as "good" or "bad." These studies largely focus on end-user involvement during the training process. Our work instead enables users to explicitly define the problem being learned, thus allowing them to dictate the purpose of the training and to do so before training starts.

5 Conclusion

A large body of work has examined supporting end users and experts in improving their RL system after it is deployed, yet little has attempted to help at the very beginning of this lifecycle. We propose supporting end users in defining RL problems through structured templates, automatic recommendations, and iterative debugging. We will prototype these designs, implement them, and then evaluate the system by conducting a user study with HRI tasks.

References

Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4), Dec. 2014. doi: 10.1609/aimag.v35i4.2513.

Sean Andrist, Micheline Ziadee, Halim Boukaram, Bilge Mutlu, and Majd Sakr. Effects of culture on the credibility of robot speech: A comparison between english and arabic. HRI '15, 2015. doi: 10.1145/2696454.2696464.

Christian Arzate Cruz and Takeo Igarashi. *A Survey on Interactive Reinforcement Learning: Design Principles and Open Challenges*. 2020.

[2] https://www.softbankrobotics.com/emea/en/nao

[3] https://hello-robot.com/product

Andrew G Barto, Philip S Thomas, and Richard S Sutton. Some recent applications of reinforcement learning. In *Proceedings of the Eighteenth Yale Workshop on Adaptive and Learning Systems*, 2017.

Dražen Brščić, Hiroyuki Kidokoro, Yoshitaka Suehiro, and Takayuki Kanda. Escaping from children's abuse of social robots. HRI '15, 2015. doi: 10.1145/2696454.2696468.

Jacqueline Hemminghaus and Stefan Kopp. Towards adaptive social behavior generation for assistive robots using reinforcement learning. HRI '17, 2017. doi: 10.1145/2909824.3020217.

Kshitij Judah, S. Roy, Alan Fern, and Thomas G. Dietterich. Reinforcement learning via practice and critique advice. In *AAAI*, 2010.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 2013. doi: 10.1177/0278364913495721.

Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. HRI '12, 2012. doi: 10.1145/2157689.2157804.

Daniel Leyzberg, Samuel Spaulding, and Brian Scassellati. Personalizing robot tutors to individuals' learning differences. HRI '14, 2014. doi: 10.1145/2559636.2559671.

James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive learning from policy-dependent human feedback. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning*, 2017.

Tatsuya Nomura, Takayuki Uratani, Takayuki Kanda, Kazutaka Matsumoto, Hiroyuki Kidokoro, Yoshitaka Suehiro, and Sachie Yamada. Why do children abuse robots? HRI'15 Extended Abstracts, 2015. doi: 10.1145/2701973.2701977.

Hae Won Park, Ishaan Grover, Samuel Spaulding, Louis Gomez, and Cynthia Breazeal. A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education. AAAI'19/IAAI'19/EAAI'19, 2019. doi: 10.1609/aaai.v33i01.3301687.

E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009. doi: 10.1109/ROMAN.2009.5326209.

Omar A. Islas Ramírez, Harmish Khambhaita, Raja Chatila, Mohamed Chetouani, and Rachid Alami. Robots learning how and where to approach people. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2016. doi: 10.1109/ROMAN.2016.7745154.

P. Salvini, G. Ciaravella, W. Yu, G. Ferri, A. Manzi, B. Mazzolai, C. Laschi, S.R. Oh, and P. Dario. How safe are service robots in urban environments? bullying a robot. In *19th International Symposium in Robot and Human Interactive Communication*, 2010. doi: 10.1109/ROMAN.2010.5654677.

Allison Sauppé and Bilge Mutlu. Robot deictics: How gesture and context shape referential communication. HRI '14, 2014. doi: 10.1145/2559636.2559657.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 1998.

Andrea L. Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. AAAI'06, 2006.

Andrea L. Thomaz, Guy Hoffman, and Cynthia Breazeal. Reinforcement learning with human teachers: Understanding how people want to teach robots. In *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006. doi: 10.1109/ROMAN.2006.314459.

Lin Wang, Pei-Luen Patrick Rau, Vanessa Evers, Benjamin Krisper Robinson, and Pamela Hinds. When in rome: The role of culture amp; context in adherence to robot recommendations. HRI '10, 2010.

Characterizing the Action-Generalization Gap in Deep Q-Learning

Zhiyuan Zhou
Department of Computer Science
Brown University
Providence, RI 02912
zhouzy@brown.edu

Cameron Allen
Department of Computer Science
Brown University
Providence, RI 02912
csal@brown.edu

Kavosh Asadi
Amazon Web Services
Santa Clara, CA 95054
kavasadi@amazon.com

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02912
gdk@cs.brown.edu

Abstract

We study the action generalization ability of deep Q-learning in discrete action spaces. Generalization is crucial for efficient reinforcement learning (RL) because it allows agents to use knowledge learned from past experiences on new tasks. But while function approximation provides deep RL agents with a natural way to generalize over state inputs, the same generalization mechanism does not apply to discrete action outputs. And yet, surprisingly, our experiments indicate that Deep Q-Networks (DQN), which use exactly this type of function approximator, are still able to achieve modest action generalization. Our main contribution is twofold: first, we propose a method of evaluating action generalization using expert knowledge of action similarity, and empirically confirm that action generalization leads to faster learning; second, we characterize the action-generalization gap (the difference in learning performance between DQN and the expert) in different domains. We find that DQN can indeed generalize over actions in several simple domains, but that its ability to do so decreases as the action space grows larger.

Keywords: Action Generalization, Deep Reinforcement Learning, Action Abstraction, Deep Q Learning

Acknowledgements

This research was supported by a Brown University Undergraduate Teaching and Research Award, as well as by the ONR under the PERISCOPE MURI Contract N00014-17-1-2699, and by the NSF under grant 1955361.

Introduction

In reinforcement learning, generalization (Ponsen et al., 2009) is crucial for achieving efficient learning and leads to better performance over unseen data. Generalization allows agents to extrapolate from environment data they have observed and to adapt to unseen situations. It is well known that deep learning supports generalization (Kawaguchi et al., 2017), and deep reinforcement learning agents, which use deep learning to maximize reward, can therefore be expected to generalize as well. However, this kind of generalization is assumed to come from parameter sharing in the function approximator, and thus only provides a natural way of generalizing over inputs.

In discrete-action domains, deep reinforcement learning (DRL) agents typically only use states as inputs, because incorporating actions as inputs becomes computationally expensive as the size of the action space grows. Therefore, such agents, typified by DQN (Mnih et al., 2015), cannot rely on the same parameter sharing mechanism for generalizing over actions. And yet, many of the domains on which DQN has been shown to perform well have action spaces where generalization is not only possible, but essential. In particular, many of the Atari 2600 games (Machado et al., 2018) include multiple actions with identical effects or actions (e.g. RIGHTFIRE) that combine the effects of two or more other actions (e.g. RIGHT and FIRE). DQN's ability to cope with such action spaces is therefore surprising, given that it is unclear exactly how it is generalizing over these kinds of "similar" actions.

In this work, we seek to better understand DQN's (arguably counter-intuitive) ability to generalize over actions. We first empirically confirm the widely-held belief that action generalization leads to faster learning. We introduce an oracle for characterizing "perfect" action generalization with DQN using expert knowledge: when it is known which actions lead to the same transition effects, all action values for that subset are updated in one Q-update. Our experiments indicate that such generalization indeed improves learning speed. Next, we study action generalization in unmodified DQN by measuring its learning performance against the oracle; the difference is what we term the *action-generalization gap*. We experiment on classic Gym control environments (Brockman et al., 2016) and Atari 2600 games, and find that DQN's ability to generalize over actions depends on the size of the action space. In small action spaces, DQN performs nearly as well as the expert; however, when the action space gets large, DQN performs poorly because of its inability to generalize.

Expert Action Generalization

One way of achieving action generalization is through action abstraction. In the context of reinforcement learning, abstraction is a method to map the representation of the original problem to a new, simpler representation where irrelevant properties are filtered and only properties relevant to decision-making are kept (Abel, 2020; Ponsen et al., 2009). Action abstraction, in particular, is the technique of grouping similar actions together into one abstract action, ignoring their small differences that are not relevant to decision-making. It facilitates generalization by abstracting similar experiences, and allowing information about one experience to apply to related (unseen) experiences.

We introduce an oracle for characterizing perfect action generalization that uses expert knowledge to abstract over actions that are similar to each other. More precisely, in a Markov Decision Process with state space S, action space A, reward function R, and discount factor γ, the expert knowledge is a symmetric $|A| \times |A|$ similarity matrix K, where each index $K(i,j)$ is a value between 0 and 1 indicating the similarity score of actions a_i and a_j. A score of 1 means two actions are fully similar, and 0 means fully different. Given this expert knowledge, we can adjust the Q-update process during Q-learning: for an experience tuple (s, a, r, s'), we update

$$Q(s,\tilde{a}) = Q(s,\tilde{a}) + \alpha * K(a,\tilde{a}) * [(r + \gamma V(s')) - Q(s,\tilde{a})] \quad \forall\, \tilde{a} \in A.$$

In words, during each update step, we not only update $Q(s,a)$ from the experienced action a, but also $Q(s,\tilde{a})$, proportionally to how similar a and $\tilde{a}$ are. So, a fully similar action $\tilde{a}$ will get a full Q update, and a fully dissimilar action $\tilde{a}$ will not be updated at all, and those in between will be updated proportionally. This process allows generalization to $\tilde{a}$ despite only experiencing a in the environment.

This oracle provides a way of measuring the degree of action generalization through learning performance. To see how much action generalization DQN can do, we can simply compare it with the oracle, which represents a best-case performance ceiling for action generalization methods. Since it is hard to quantify action generalization directly, we compare learning performance instead, using it as a proxy for generalization. We define this performance difference between the oracle and a DRL agent to be the *action-generalization gap*. The rest of this paper aims to characterize the action-generalization gap for DQN.

Action Space Augmentation for Evaluation

In order to evaluate action generalization, we need environments with action spaces where actions may be similar to each other, so that it makes sense for actions to generalize. To that end, we propose to augment the action space of a base environment to include additional similar actions. We propose three such "action augmentation" methods:

1. *Duplicate actions ($N\times$)*: augment the original action set $N-1$ times, so the new action space contains N copies of every action in the original action space.
2. *Semi-duplicate actions*: augment the original action set with 4 sets of reduced-magnitude actions, $|A|$ in each set (where $|A|$ is the size of the original action space), for a total of $5 \cdot |A|$ actions. Each set of these semi-duplicate actions has similar transition effects to the original action set, differing only in the magnitude. The magnitude similarity is controlled by a similarity score $h \in [0, 1]$, where $h = 1$ corresponds to the same-magnitude action and $h = 0$ corresponds to a zero-magnitude action (No-op). For example, in the Pendulum environment, we can apply a 0.8 torque to the left, or 0.5 torque to the right, etc.
3. *Random actions*: augment the original action set with $4 \cdot |A|$ stochastic actions for a total of $5 \cdot |A|$ actions. Each stochastic action is a uniform random distribution over the actions in the original action space.

Here we are concerned with discrete action spaces, so we take discrete-action versions of CartPole, Pendulum, and LunarLander as the base environments for our experiments, and apply the three action augmentation methods to expand the action space.

Because the action-augmentations above are artificially created, we can supply the oracle with knowledge of which similar actions should be abstracted together. For the "duplicate actions" augmentation, all duplicate copies of each actions are fully similar ($K(i, j) = 1$) to each other, and not to anything else. For the "semi-duplicate actions" augmentation, $K(i, j) = h$ if a_i and a_j are semi-duplicates of each other, else $K(i, j) = 0$. For the "random actions" augmentation, all the random actions are fully similar to each other, and nothing else is similar. For all augmentations, $K(i, i) = 1$. The oracle uses this information to guide its Q updates.

Evaluating Performance

We now experimentally characterize the action-generalization gap between DQN and the oracle. In addition to the augmented action space, we provide the original action space as a baseline for comparison, which we call "baseline". For the "duplicate actions" augmentation in this section, we set $N = 5$. For the semi-duplicate augmentation, we use similarity scores $h \in \{0.2, 0.5, 0.8\}$.

In the "duplicate actions" environments, having duplicate actions slows down learning, with the exception of Pendulum, where 5x duplicate actions performs about the same as the baseline (Figure 1). By contrast, the oracle is unaffected by the larger action space and performs just as well as the baseline. This confirms the hypothesis that action generalization does help speed up learning.

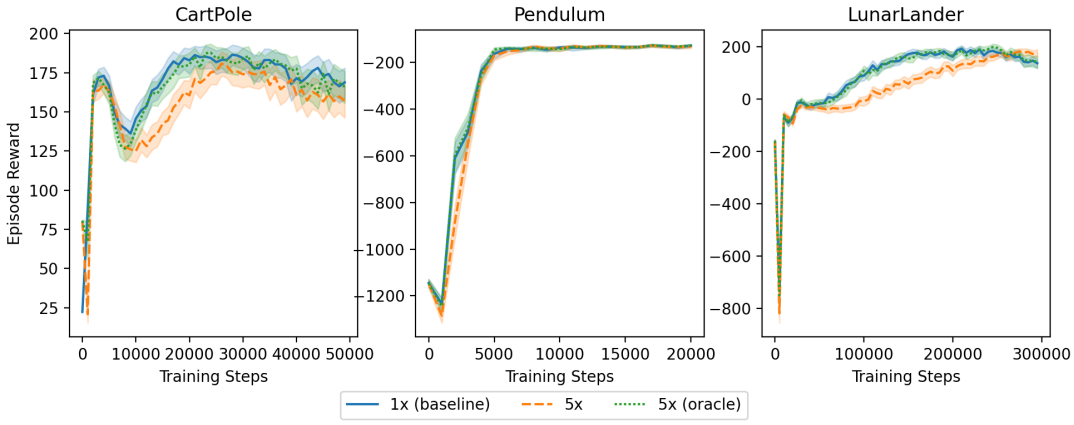


Figure 1: Action-generalization gap for DQN with **5x duplicate actions** on CartPole, Pendulum, and LunarLander

The experiment also provides evidence that DQN is doing some implicit action generalization. Notice the difference in performance between the oracle and unmodified DQN in Figure 1: DQN takes at most twice as long as the oracle to learn with 5x duplicate actions. This is a bit surprising given that the oracle performs the same as the unaugmented environment (baseline), whose action space is one-fifth as large. If DQN isn't doing any action generalization, then learning time should scale linearly with the size of the action space. The fact that it scales sub-linearly suggests that DQN is able to generalize over actions to some degree.

These 5x duplicate action results are somewhat surprising, because even though there are more actions to choose from, the probability of choosing the optimal action under a uniform random policy remains the same. So why does the learning problem become harder? We hypothesize two potential explanations. First, it could be a result of the overestimation

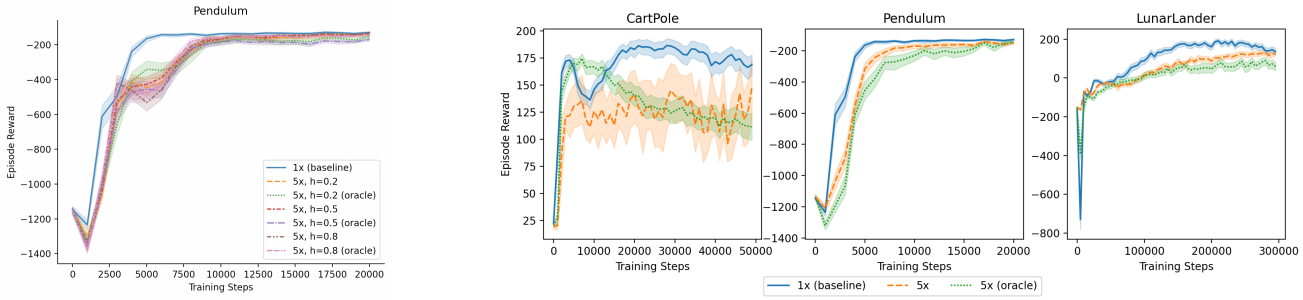


Figure 2: (Left) Action-generalization gap for DQN with **5x semi-duplicate actions**, for similarity score $h \in \{0.2, 0.5, 0.8\}$, on Pendulum. (Right) Action-generalization gap for DQN with **5x random actions** on CartPole, Pendulum, and LunarLander.

bias in Q-Learning (Thrun and Schwartz, 1993), which increases with the size of the action space. Second, it could be that a larger action space means greater opportunity for noise in neural network parameters to produce noisy gradient updates. We leave a full investigation of these hypotheses for future work.

For the semi-duplicate action augmentations, all the experiments take about twice as long to converge, even the ones using an oracle (Figure 2, left). This indicates that semi-duplicate actions form a harder learning problem than exactly-duplicated actions, as expected. But still, we can note here that there is basically no action-generalization gap. This is more evidence to suggest that DQN is able to generalize over actions in small action spaces.

The "random actions" (Figure 2, right) results tell a more complicated story: when random actions are introduced, the learning performance is worse than "baseline", regardless of whether the oracle is used. This shows that having random actions makes learning substantially harder, which is expected because this stochasticity is hard to account for during learning. What's surprising is that in Pendulum and LunarLander the action-generalization gap is actually negative: the oracle performs worse than unmodified DQN. We suspect this may be due to an incorrect assumption: the oracle treats all random actions as similar to each other. However, the random actions are not always similar; in fact, they frequently select from original actions that have completely opposite effects. Performing Q-updates with the assumption that all random actions are fully similar may at times lead the function approximator to incorrectly update shared internal parameters governing the Q-values of the *non-stochastic* actions. If we instead assume the random actions are fully dissimilar, we simply recover the baseline performance. So, in the case of random actions, "baseline" is a better performance ceiling because none of its "abstract" actions contains actions with opposite effects. Using this ceiling, the action-generalization gap is quite noticeable, indicating that DQN is not good at generalizing over stochastic actions.

Evaluation on Large Action Spaces

The experiments from the last section suggest that DQN has some robustness to action augmentation in domains with small action spaces. Here we investigate whether that robustness extends to environments with larger action spaces, and find that, for both Pendulum and Atari 2600 games, it does not.

For Pendulum, we again augment with duplicate actions, and increase N to discover the point at which learning performance starts to degrade (see Figure 3, left). When $N = 5$, we do not observe any action-generalization gap, but when $N \in \{15, 50\}$, the gap becomes quite noticeable. Moreover, when $N = 50$, learning is not just slow, but converges to worse final performance. This deterioration in DQN's performance is not due to the inherent hardness of the domain, because the oracle continues to match the performance of learning on the unaugmented environment. This suggests that the large action-generalization gap must come from DQN's inability to generalize over actions is large action spaces.

For Atari 2600, we chose six commonly-used games to evaluate the action-generalization gap: Beam Rider, Breakout, Pong, Ms Pacman, Qbert, and Space Invaders. Our experiments use three types of action augmentation:

1. *Full action set*: There are 18 legal actions that are shared across all Atari games. However, since some of the actions are not meaningful in certain games, the default action space of each game is usually pruned to only leave the meaningful action space (which we refer to as the "baseline" actions). In the full action set setting, we use all of the 18 legal actions in Atari games as the action space, which increases the size of the action space for all the games we investigate.
2. *Duplicate actions*: augment the baseline actions with $N - 1$ sets of duplicate actions. Here we use $N = 5$.
3. *Noop actions*: augment the baseline actions with N noop actions. Here we use $N = 2$.

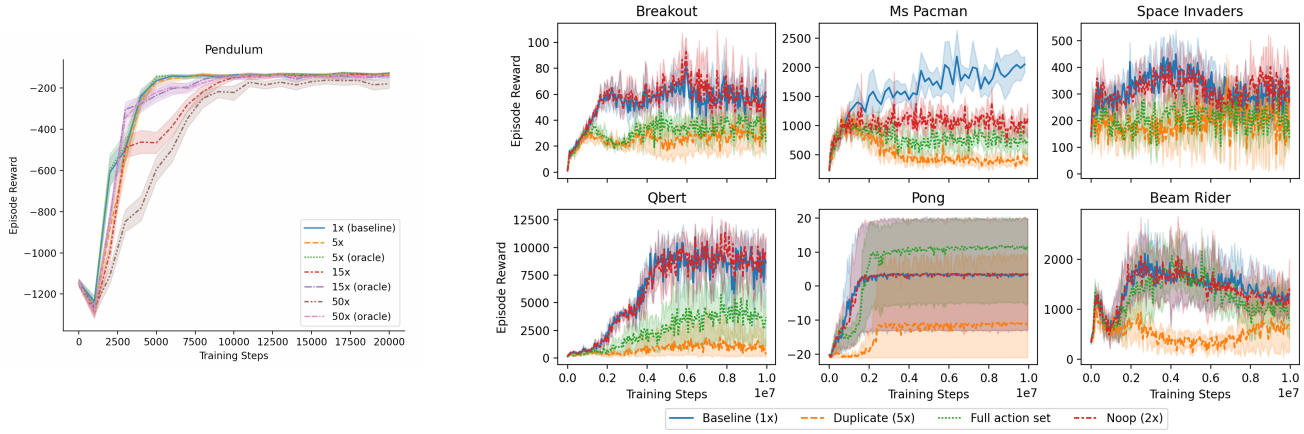


Figure 3: (Left) Action-generalization gap for DQN with **5x, 15x, and 50x duplicate actions** on Pendulum. (Right) Action-generalization gap for DQN on six Atari games.

In Atari games, we don't have expert knowledge of which actions are similar, because such information is game-specific and often highly state-dependent. Fortunately, we can still use the unmodified baseline action space as a performance ceiling, similar to the oracle in the preceding experiments. Even though this is not a perfect oracle because Atari actions are context-dependent and can't simply be treated as one abstract action group, we can still use this approach to characterize the action-generalization gap.

Figure 3 (right) shows a large action-generalization gap for the duplicate and full action set augmentations, across the majority of games. We hypothesize that this degradation occurs because 1) Atari games are more complicated domains and 2) Atari games have a larger action space to begin with, both of which make it harder for DQN to generalize over actions. Surprisingly, the noop augmentation only leads to an action-generalization gap on Ms Pacman. We suspect that which specific actions lead to worse performance may be game-dependent, and that in the other games, noop actions may more frequently be the optimal choice.

Conclusion

Overall, we have obtained preliminary evidence suggesting that DQN has some ability to do action generalization in small to medium action spaces, but for the most part that ability does not extend to large action spaces. However, given enough time, DQN may still be able to recover the optimal policy in those large spaces. A direction for future work is to pinpoint the exact reason for DQN not being able to generalize over actions in large action spaces and to provide a remedy that improves performance.

References

Abel, D. (2020). *A theory of abstraction in reinforcement learning*. PhD thesis, Brown University.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Ponsen, M., Taylor, M. E., and Tuyls, K. (2009). Abstraction and generalization in reinforcement learning: A summary and framework. In *AAMAS Workshop on Adaptive and Learning Agents*, pages 1–32.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, volume 6.

Three systems interact in one-shot reinforcement learning

Amy R. Zou
Department of Psychology
University of California, Berkeley
Berkeley, CA 94720
amyzou@berkeley.edu

Anne G. E. Collins
Department of Psychology, Helen Wills Neuroscience Institute
University of California, Berkeley
Berkeley, CA 94720
annecollins@berkeley.edu

Abstract

Human adaptive decision-making recruits multiple cognitive processes for learning stimulus-action (SA) associations: reinforcement learning (RL) represents gradual estimation of values of choices relevant for future reward-driven decisions, episodic memory (EM) stores precise event information for long-term retrieval, and working memory (WM) serves as flexible but temporary, capacity-limited storage. However, we have limited understanding of how these systems work together. Here, we designed a new one-shot RL task to disentangle their respective roles: 144 human adult participants used one-shot rewards to learn SA associations across independent training blocks. The first half of each block provided one chance to obtain feedback for pressing one of two keys for each stimulus, followed by a chance in the second half of the block to use this feedback to choose the correct key in a short-term association task, primarily targeting WM. In a subsequent testing phase designed to assess long-term retention through RL or EM, all stimuli were shown again in randomized order and subjects were asked to press the correct key for each. Training block performance revealed WM-dependent strategy effects on choice accuracy, as well as a role for both RL and EM when WM is overwhelmed. Testing phase performance depended on feedback interacting with presentation order, revealing signatures of both RL and EM in learning from one-shot rewards. Computational modeling suggests that a mixture model combining RL and EM components best fitted group-level testing phase behavior. Our results show that our new protocol can identify signatures of each of the three memory systems' contributions to reward-based learning. With this approach, we create new possibilities to better understand how each integrates a single bit of information, what their exact contributions to choice are, and how they interact.

Keywords: Human reinforcement learning; episodic memory; working memory; computational cognitive modeling

Acknowledgements

This work was supported by NSF Grant 2020844.

1 Introduction

Learning is often studied under the framework of Reinforcement Learning (RL) (Sutton & Barto, 2018; Eckstein, Wilbrecht, & Collins, 2021), which posits that agents learn from past outcomes of their actions to inform future decisions. However, recent work has shown important additional contributions to RL from other cognitive processes like working memory (WM) (Collins, 2018; Yoo & Collins, 2021) and episodic memory (EM) (Bornstein, Khaw, Shohamy, & Daw, 2017). WM is a temporary, capacity-limited, effortful, yet extremely flexible form of short-term memory storage. EM stores high-precision memories for long-term retrieval and is associated *primacy* and *recency* effects that improve retrieval at the beginning or end of an episode, respectively (Ebbinghaus, 1913). Previous studies have demonstrated load and temporal effects, characteristic of WM capacity limitation and EM respectively, on learning associations (Bornstein et al., 2017), but the precise information stored and used in WM or EM during learning remains unclear. Furthermore, these systems interact with one another, (Collins, 2018; Poldrack & Packard, 2003; Wimmer, Braun, Daw, & Shohamy, 2014), but the nature of those interactions remains unclear.

We designed a new experiment to simultaneously identify contributions of all three systems in a single learning context to better qualify them individually and in interaction (Fig. 1). Participants learned stimulus-action (SA) associations from a single instance of feedback, and we manipulated trial-features (e.g. reward, load, and temporal order) known to impact RL, WM, and EM to assess their effects on recall. We predicted that RL, WM, and EM would play roles in short-term memory performance, while long-term memory performance will be predominantly RL- and EM-driven, and can be captured by computational modeling.

2 Methods

Participants: The research was approved by UC Berkeley's Institutional Review Board. Following exclusion criteria, our final sample consisted of 66 MTurk participants (22 female, 31.8±5.42 years) and 78 online undergraduate students (RPP; 59 female, 20.8±2.68 years).

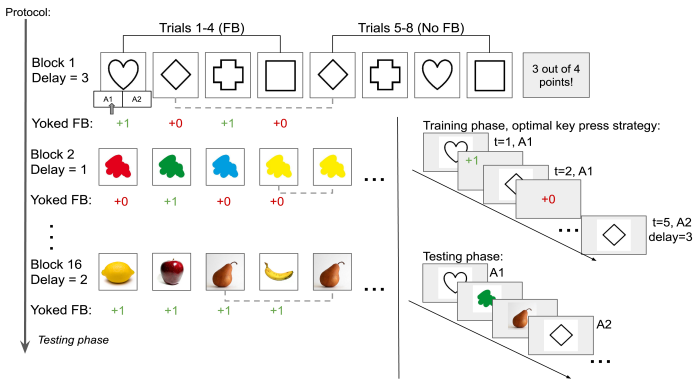


Figure 1: Participants performed a one-shot RL task comprising a *training phase* and a *testing phase*. The training phase contained 16 categorically independent blocks; in each block, participants used one-shot rewards to learn 4 new SA associations across 8 trials. On trials 1–4, participants saw 4 distinct images sequentially, pressed either "J" or "K" for each, and immediately received a truthful, positive (+1) or negative (0) feedback (FB) for their action. Using this FB, participants had to press the correct key when they saw these images again on trials 5–8, albeit in different order and receiving no immediate FB. A trial was considered correct if a rewarded action was repeated or an unrewarded action was avoided, and incorrect otherwise. In the testing phase, participants saw all 64 stimuli in a randomized order, and had to press the correct key again (immediate FB was withheld).

Task design: The *training phase* incorporated and counterbalanced critical features of each system—valenced feedback to target RL's asymmetrical learning mechanism, stimulus presentation order for EM's temporal sensitivity, and a load around the WM limit—to measure their individual contributions to and interactions in trial-by-trial learning. Furthermore, participants could adopt different choice policies on trials 1–4 to learn the correct key. For example, if only one key was pressed (henceforth called as the *same-key strategy*), an unrewarded outcome would always signal pressing the other key; pressing different keys, however, would require WM to track the initial key press and FB, then compute a switch if the initial action was wrong. The *same-key strategy* thus reduces WM load, and this potential variation in WM use would further affect the system's interactions with RL and EM. The *testing phase* assessed how FB-sensitive RL and temporally-sensitive EM contributed to long-term retention; we assumed randomization and delay in stimulus presentation would wash out WM effects.

Computational models: As a preliminary step, we modeled testing phase behavior using 4 computational models capturing different assumptions about separate or mixed contributions of RL and EM. Assuming no WM effect, we fitted datasets of only participants who predominantly used the *same-key strategy* (i.e., solely using the strategy after block 9 or earlier) to reduce effects from varying WM engagement. 688

RL. We implemented a classic delta-rule learning algorithm, where the expected value Q at time t of an action a for a stimulus s is updated by the reward prediction error—the difference between outcome r and prior expectations of $Q(a, s)$—scaled by learning rate α ($0 < \alpha < 1$):

$$Q_t(a, s) = Q_{t-1}(a, s) + \alpha(r_{t-1} - Q_{t-1}(a, s)) \tag{1}$$

Q-values were initialized at 0.5 and converted to action probabilities via softmax:

$$p(a|s) = \frac{\exp(_t(a, s))}{\exp(\beta \sum_i (Q_t(a_i, s)))} \tag{2}$$

We assumed that Q-values are updated following explicit feedback on trials 1–4 of training blocks. We also considered a variant model **RL2a**, which separates learning rates for gains and losses (α when $r_t = 1$ and α_{neg} when $r_t = 0$) (Katahira, 2018). In both RL models β was fixed at 10 because α and β are not jointly recoverable as their product uniquely influences choice policy. We also tested other RL models, such as one that iteratively updates $Q(s, a)$ of previous trials to account for RL potentially exhibiting temporal order effects, but they were excluded since they failed to capture key behaviors.

EM is a descriptive model that quantitatively implements a memory process of probabilistic storage and retrieval of a trial's information. We assumed that the probability of retrieving a memory of a trial, $p(ret)$, decreases exponentially with trial position within block (capturing within-block *primacy effect*), parameterized by time constant τ:

$$p(ret) = \exp(-\tau(s - 1)) \tag{3}$$

We note that this EM implementation does not account for experiment-wide memory effects (specifically *recency effects*). When we implemented a variant EM model with such a global recency mechanism, however, it failed to capture key behaviors in simulation, and was thus excluded.

We also assumed imperfect encoding and retrieval accuracies, parameterized by $0 < m_{FB} < 1$ (where FB=1/0). We used Bayes rule to combine the probability of correct encoding m_{FB} with $p(ret)$ to compute the final probability of accessing the knowledge of the correct action, $p(cor)$. We assumed random key choice (0.5) if the memory was not stored or retrieved, to translate into the EM policy:

$$p(cor) = m_{FB}p(ret) + 0.5(1 - p(ret)). \tag{4}$$

The model's policy p_{EM} is defined by $p(cor)$ for the correct action (as defined by FB on trials 1–4), and $1-p(cor)$ otherwise. Note that values of $m_{FB} < 0.5$ indicate worse-than-chance memory encoding, capturing the possibility that participants stored the wrong memory. This could happen, especially in the FB=0 case, if EM stored SA instead of SAO (stimulus-action-outcome) associations, reflecting an outcome-independent memory of a performed action.

RLEM assumes that RL and EM processes jointly contribute to testing phase behavior. The model's policy is a mixture of the RL2a model's policy p_{RL} and the EM model's policy p_{EM}, with mixture weight $(0 < \omega < 1)$:

$$p(a|s) = \omega p_{RL}(a|s) + (1 - \omega)p_{EM}(a|s) \tag{5}$$

Model fitting: We used standard maximum likelihood estimation to fit individual participants' data with each model (Wilson & Collins, 2019). Because we had very few trials per participant, we also fit our models at the group level, combining all participant data into one dataset. We used AIC scores for model comparison, and validated parameter identification and the model comparison procedure with simulation studies (Wilson & Collins, 2019), and calculated exceedance probabilities to further evaluate model selection (Rigoux, Stephan, Friston, & Daunizeau, 2014). We simulated 10 datasets per set of parameters and qualitatively compared their experimental result patterns to those of participant data (Palminteri, Wyart, & Koechlin, 2017).

3 Results

3.1 Behavioral Results

Participants' learning improved over blocks (Fig. 2a, block regressor $\beta = 0.0363, p < 0.001$), likely reflecting a general adoption of the the *same-key strategy* (Fig. 2b). A mixed effects logistic regression on training phase accuracy (trials 5–8; Fig. 2d) revealed significant main effects of strategy ($\beta = 1.71, p < 0.001$), presentation order (order; $\beta = 0.0806, p = 0.0323$), and feedback (FB; $\beta = 0.757, p < 0.001$), and two interactions of strategy with FB ($\beta = -0.788, p < 0.001$) and with order ($\beta = -0.147, p = 0.008$) (Fig. 2c; Fig. 2e, red). Main effects of FB and order were only observed in *non-same-key* blocks (Fig. 3a), whereas *same-key* blocks showed performance consistent with near-perfect WM use.

Extending the same analysis to the testing phase revealed significant effects of FB ($\beta = 1.44, p < 0.001$), order ($\beta = 0.100, p < 0.001$), and their interaction ($\beta = -0.307, p < 0.001$) (Fig. 2d; Fig. 2e, blue). Controlling for training phase

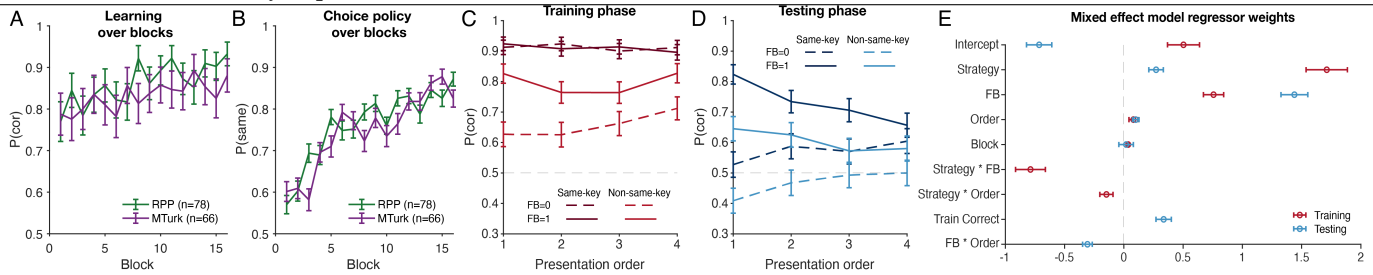


Figure 2: Behavioral results. A) Participants from both groups successfully learned the task, and B) adopted the *same-key strategy* over training blocks. Both groups achieved similar performance in both the training and the testing phase and were combined into one sample for subsequent analyses. C) In the training phase, choice strategy interacted with initial feedback (FB) received and presentation order. D) In the testing phase, there were main effects of choice strategy, FB, and order, and a significant interaction between FB and order. E) Regressor weights from linear mixed effects models predicting accuracy in training (red) and testing (blue) phases. Points represent estimates of main effects and interactions, bars represent standard error. All effects are significant (at $p < 0.05$).

accuracy of a given stimulus (Train Correct; $\beta = 0.336, p < 0.001$), training phase choice strategy still had a small yet significant effect on testing phase performance (strategy; $\beta = 0.281, p < 0.001$), hinting at deep interactions between WM-dependent strategies and the long-term memory systems RL and EM. Critically, testing phase performance revealed signatures of both RL and EM. Despite the fact that correct and incorrect FB provided the same information, accuracy was higher for rewarded SA pairs, reflecting a feedback valence sensitivity often associated with RL involvement. The performance difference between rewarded and unrewarded stimuli on trial 1 and trial 4 indicates a FB-dependent *primacy effect*, suggesting reward-independent, long-term EM storage of performed SA associations.

3.2 Modeling results

We fit all models on data from *same-key strategy* participants ($N = 75$; see methods). Subject-level AIC comparisons show that the EM model (3 parameters) performed best while the mixture model RLEM (6 parameters) performed the worst (Fig. 3a). We suspected that RLEM's poor performance was due to overfitting of insufficient trial data at the subject level. Indeed, EM model simulations could not capture the key FB-order interaction in testing phase (Palminteri et al., 2017). Group-level model comparison confirmed that RLEM was previously disadvantaged by a shortage of trials (Fig. 3b). In future work, we will use hierarchical modeling to overcome this issue while accounting for individual differences (Baribault & Collins, 2021).

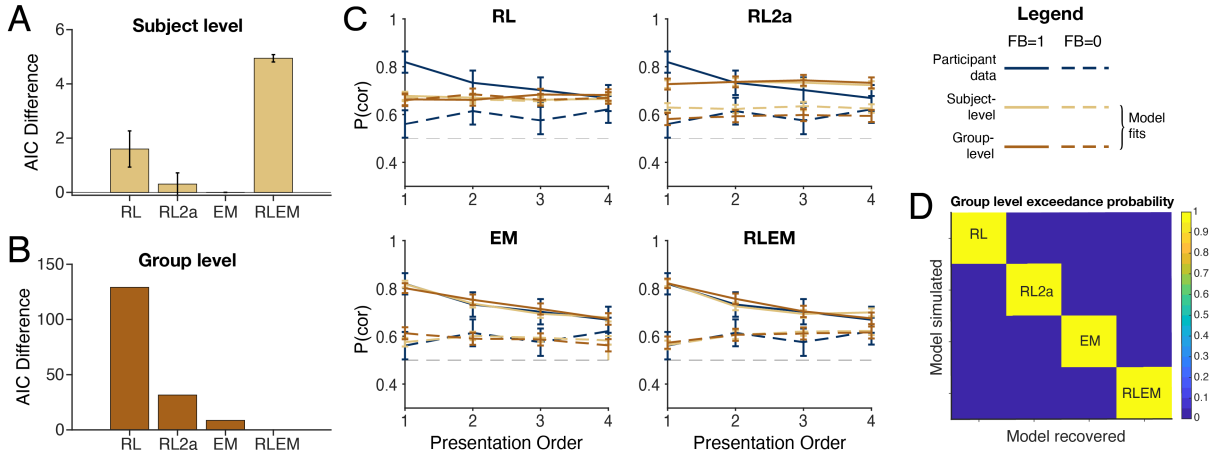


Figure 3: A) and B) show AIC score differences between the winning model and the other models fit at the subject-level and group-level, respectively. C) Model validation; RLEM simulation replicates qualitative trends observed in experimental data (blue lines) at the group and subject-level. D) Confusion matrix of exceedance probabilities.

RLEM's simulations best captured the qualitative trends observed in the participants' data, replicating the main effects of FB, order, and their interaction (Fig. 3c). Parameter recovery was also successful for the RLEM model, and verified via a model recovery study. Group fit model parameters confirmed an asymmetrical learning rate bias ($\alpha = 0.55$ vs.

$\alpha_{neg} = 0.33$). They revealed that the EM process was more likely to encode SA associations than SAO associations when FB=0 ($m_1 = 0.75$ vs. $m_0 = 0.41$), and that the EM process contributed more than RL after a single feedback ($\omega = 0.36$); individual parameter fits showed similar findings.

4 Discussion

Our model-independent analyses suggest that our task is able to identify distinct contributions of RL, WM, and EM in a one-shot reward learning paradigm. WM effects emerged strongly in the training phase via an efficient choice policy that minimized FB- and order-sensitive effects reflecting RL and EM involvement. While WM's role should have been limited (in both capacity and temporal range) during the testing phase, WM-efficient choice policy appeared to still mildly improve testing phase performance.

In the testing phase, FB effects led to better performance for rewarded stimuli than unrewarded stimuli, supporting the *positivity bias* often seen in previous studies measuring RL-based learning (Katahira, 2018; Master et al., 2020). Within-block primacy effects amplified this effect at trial 1, suggesting outcome-blind EM storage of SA associations. Our modeling results support our behavioral findings: the mixture model RLEM, which captured both RL's FB sensitivity and EM's FB and temporal sensitivity, best fit the data and replicated these qualitative features.

The main limitation to our study was insufficient data at the subject-level, resulting in minimal ability to consider individual differences. Future steps include implementing modeling under a Bayesian hierarchical framework to capture both subject- and group-level information, and designing follow-up experiments with more training phase trials. We will also develop the modeling to include the training phase and incorporate WM contributions in the testing phase.

In conclusion, we showed evidence of three systems contributing distinct characteristics to learning from one-shot rewards, as well as evidence for interactions between them. Disentangling how multiple systems contribute to adaptive decision-making is essential to better understanding where individual differences come from, in healthy individuals, across development, and in clinical populations. Our study offers an important step in this direction.

References

Baribault, B., & Collins, A. (2021, December). *Troubleshooting Bayesian cognitive models: A tutorial with matstanlib* (preprint). PsyArXiv. doi: 10.31234/osf.io/rtgew

Bornstein, A. M., Khaw, M. W., Shohamy, D., & Daw, N. D. (2017, December). Reminders of past choices bias decisions for reward in humans. *Nature Communications*, *8*(1), 15958. doi: 10.1038/ncomms15958

Collins, A. G. E. (2018, October). The Tortoise and the Hare: Interactions between Reinforcement Learning and Working Memory. *Journal of Cognitive Neuroscience*, *30*(10), 1422–1432. doi: 10.1162/jocn$_{a0}$1238

Ebbinghaus, H. (1913). *Memory: A contribution to experimental psychology.* (H. A. Ruger & C. E. Bussenius, Trans.). New York: Teachers College Press. doi: 10.1037/10011-000

Eckstein, M. K., Wilbrecht, L., & Collins, A. G. (2021, October). What do reinforcement learning models measure? Interpreting model parameters in cognition and neuroscience. *Current Opinion in Behavioral Sciences*, *41*, 128–137. doi: 10.1016/j.cobeha.2021.06.004

Katahira, K. (2018, December). The statistical structures of reinforcement learning with asymmetric value updates. *Journal of Mathematical Psychology*, *87*, 31–45. doi: 10.1016/j.jmp.2018.09.002

Master, S. L., Eckstein, M. K., Gotlieb, N., Dahl, R., Wilbrecht, L., & Collins, A. G. (2020, February). Disentangling the systems contributing to changes in learning during adolescence. *Developmental Cognitive Neuroscience*, *41*, 100732. doi: 10.1016/j.dcn.2019.100732

Palminteri, S., Wyart, V., & Koechlin, E. (2017, June). The Importance of Falsification in Computational Cognitive Modeling. *Trends in Cognitive Sciences*, *21*(6), 425–433. doi: 10.1016/j.tics.2017.03.011

Poldrack, R. A., & Packard, M. G. (2003, January). Competition among multiple memory systems: converging evidence from animal and human brain studies. *Neuropsychologia*, *41*(3), 245–251. doi: 10.1016/S0028-3932(02)00157-4

Rigoux, L., Stephan, K., Friston, K., & Daunizeau, J. (2014, January). Bayesian model selection for group studies — Revisited. *NeuroImage*, *84*, 971–985. doi: 10.1016/j.neuroimage.2013.08.065

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wilson, R. C., & Collins, A. G. (2019, November). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547. doi: 10.7554/eLife.49547

Wimmer, G. E., Braun, E. K., Daw, N. D., & Shohamy, D. (2014, November). Episodic Memory Encoding Interferes with Reward Learning and Decreases Striatal Prediction Errors. *Journal of Neuroscience*, *34*(45), 14901–14912. doi: 10.1523/JNEUROSCI.0204-14.2014

Yoo, A., & Collins, A. (2021, December). How Working Memory and Reinforcement Learning Are Intertwined: A Cognitive, Neural, and Computational Perspective. *Journal of Cognitive Neuroscience*, 1–17.

Lesions to Value-Responsive Brain Regions Lead to Impairments in Voluntary Persistence

Camilla van Geen
Department of Psychology
University of Pennsylvania
cvg@sas.upenn.edu

Rebecca Kazinka
Department of Psychology
University of Minnesota
kazin003@umn.edu

Avinash Vaidya
Department of Cognitive, Linguistic and Psychological Sciences
Brown University
avinash_vaidya@brown.edu

Joseph W Kable
Department of Psychology
University of Pennsylvania
kable@psych.upenn.edu

Joseph T McGuire
Department of Psychological and Brain Sciences
Boston University
jtmcg@bu.edu

Abstract

Deciding how long to keep waiting for uncertain future rewards is a complex problem. Previous research has shown that opting to stop waiting is the result of a rational value-maximizing process that pits the subjective value of quitting against the subjective value of waiting. As such, brain regions known to compute context-dependent value track the dynamics of this trade-off. Here, we provide causal evidence of the necessity of these brain regions for successful performance in a willingness-to-wait task. Patients with lesions to value-responsive areas of the brain (ventromedial and dorsomedial parts of the prefrontal cortex as well as the anterior insula) show deficits in their ability to adaptively calibrate persistence based on the temporal statistics of reward delivery. Conversely, patients with lesions to brain regions implicated in self-control but not value computation perform similarly to healthy controls. These findings support the idea that failures of persistence are driven by sophisticated cost-benefit analyses rather than unfortunate lapses in self-control.

Keywords: Decision making
Willingness-to-wait
Brain lesions

Acknowledgements

We would like to thank Lesley Fellows for facilitating access to participants in Montreal, Christine Déry and Eileen Cardillo for coordinating participants in Montreal and in Philadelphia, and all of the participants themselves, without whom this work would not be possible. This work was supported by National Institute on Drug Abuse (NIDA) R01-DA029149 to JWK, National Institutes of Health (NIH) F32-DA030870, National Science Foundation (NSF) BCS-1755757, and National Institutes of Health (NIH) uR21-MH124095 to JTM.

1 Introduction

Not all rewards are worth waiting for. Although the ability to delay gratification has often been touted as indicative of discipline and self-control [1,2,3], this line of research often fails to consider that in the pursuit of value maximization, waiting is not always more rewarding. Rather, contextual features of the environment can modulate the value of waiting such that the adaptive regulation of persistence, not persistence no matter what, should be the ultimate goal.

Previous research has shown that humans learn the temporal characteristics of reward delivery in order to adaptively calibrate their waiting behavior [4]. If the distribution of wait durations is heavy-tailed, such that reward is likely to arrive either almost immediately or not for a long time, people engage in a rational cost/benefit analysis and (rightfully) conclude that the financially optimal wait duration is short. Conversely, in a temporal environment that favors persistence, humans are more likely to wait longer, again in line with the predictions of a normative model. Performance on these sorts of willingness-to-wait (WTW) tasks thus appears to reflect the results of a dynamic comparison between the value of quitting and the value of waiting. To define failures of persistence as the outcomes of a rational computation that maximizes reward contradicts their characterization as unfortunate lapses in self-control.

Neural evidence further corroborates this idea: activity in areas of the brain known to be sensitive to subjective value, such as the vmPFC [5], evolves differently depending on the temporal environment participants are in [6]. If the distribution of wait durations favors persistence no matter what, the subjective value signal ramps up progressively as time goes by and the wait shortens. Conversely, if the distribution of wait times is such that the value of waiting decreases after a certain point, activity in the vmPFC is flat and then subsequently decays. Furthermore, a variety of brain regions – including the insula and dorsal regions of the prefrontal cortex – seem to track dynamic estimates of the value of quitting vs. the value of waiting, and show a ramping up of activity before the decision to quit is made [6]. Beyond providing insight into the neural mechanisms that underlie persistence in willingness-to-wait tasks, these findings are important because they reflect computations that feed into estimates of subjective value.

Although the fMRI analyses described thus far provide compelling evidence that value-responsive brain regions are involved in adaptive persistence, they cannot address the question of whether these regions of the brain are *necessary* for such behavior. Here, we provide the first instance of causal evidence in favor of subjective value maximization underlying performance in a WTW task: patients with lesions to parts of the brain known to track either the subjective value of waiting (vmPFC) or the relative value of quitting (dmPFC/AI) were significantly impaired in calibrating their waiting behavior to the temporal environment. Crucially, lesions to areas of the prefrontal cortex known to underlie self-control but not valuation [7,8] did not affect performance.

2 Methods

2.1 Participants

A total of 18 healthy controls and 31 patients with brain lesions were recruited from the University of Pennsylvania and McGill University to participate in this study. We further separated the patients into subgroups according to the localization of their lesions. To establish these groupings, we used a combination of the Automated Anatomical Labeling atlas (AAL) [9] and the results of the fMRI analyses reported in McGuire & Kable (2015) [6], which employed a similar willingness-to-wait task. We first assessed the overlap between each patient's lesion and the regions of the anterior insula (AI) that exhibited differential activation for "quit" vs. "reward" trials in McGuire & Kable (2015). We then calculated the conjunction of the brain areas identified in the paper and those labeled as belonging to the AI in the AAL atlas. The resulting brain map was used to define the anterior insula group, and any patient whose lesion comprised > 25% of the region of interest was classified as belonging to it (n = 6). For the remaining 25 participants, we first assessed whether at least 50% of each patient's lesion affected the frontal lobe, as defined by a combination of regions in the AAL atlas. This resulted in the exclusion of 3 participants whose lesions did not meet the threshold. We then assessed whether the localization of the remaining 21 participants' lesions significantly overlapped with the vmPFC as defined by the atlas. On the basis of this analysis, 10 participants were classified as belonging to the vmPFC group. Finally, we identified a dmPFC group, whose lesion significantly overlapped with regions of the dmPFC sensitive to "quit" vs. "reward" trials [6] and with the dmPFC as defined by the atlas. This procedure resulted in 6 patients in AI group, 10 frontal controls (FC) whose lesions only met the frontal lobe

cut off, 10 vmPFC patients, and 2 dmPFC patients. Because the relevant regions of the AI and dmPFC were both shown to be differentially active for "quit" vs. "reward" trials, we grouped the AI and dmPFC patients together to result in a combined sample of 8 dmPFC/AI patients (see Figure 1a).

2.2. Task

For this experiment, we used a willingness-to-wait task similar to the one described in McGuire & Kable (2015) [6] (Figure 1b). At the start of a trial, participants were shown a coin worth 0¢ on the center of the screen. They were told that after a certain amount of time, the length of which was not made explicit, the coin would mature and become worth 10¢. At that point, which was signaled by a change in the coin's color, they could sell the coin by pressing the space bar, leading to a 10¢ reward. However, participants were also told that at any point in the trial, they could elect to press the space bar and sell the coin, even if it had not yet matured. In that case, they would not receive 10¢, but they would move on to the next coin. Whenever participants decided to sell the coin, the word 'SOLD' would appear in red on the center of the screen for 1 second. A progress bar on the bottom of the screen indicated how long they had left in the block.

The blocks were defined according to the distribution of wait durations and each lasted a total of 12 minutes. In the high-persistence block (HP), wait times were sampled according to a uniform distribution of up to 20s. In this condition, the coin was equally likely to mature at any time within the 20-second period, and the optimal strategy was to always wait until it matured. Conversely, wait times in the limited-persistence (LP) environment were sampled according to a heavy-tailed generalized Pareto distribution truncated at 40 seconds. As is illustrated in Figure 1B, the optimal strategy in the limited-persistence environment was to wait only 2.3 seconds for the coin to mature.

The experiment was conducted over two sessions, with the order of the blocks held constant across participants so as to highlight individual differences. Session 1 began with a high-persistence block followed by a limited-persistence block. The next day, participants returned for session 2, which began with a limited-persistence block followed by a high-persistence block. The first blocks of each session (HP from session 1 and LP from session 2) were, because no task preceded them, taken as instances "uncontaminated" learning and are the focus of the upcoming analyses. Participants were able to tell the difference between the two blocks because the color of the coin varied, but they were not explicitly told about the characteristics of each distribution of wait times.

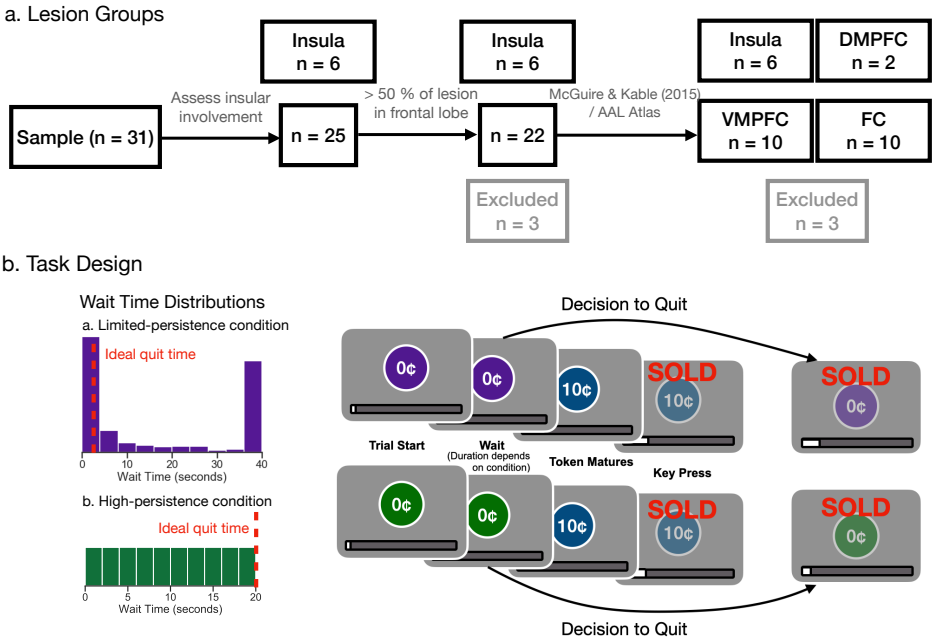


Figure 1. a. Graphical illustration of the pipeline used to separate lesion patients into subgroups. Groups were defined on the basis of the localization of the patient's lesion with respect to the AAL atlas and fMRI results from McGuire & Kable (2015). b. Structure of the willingness-to-wait task. High- and limited-persistence conditions were defined on the basis of the distribution of wait times. At any point during the waiting period, participants could decide to quit and move on to the next trial.

3 Results

Replicating previous findings [4,6,10], we found that healthy controls were able to differentiate between the two temporal environments and adapt their waiting behavior accordingly. As predicted by the difference in ideal quit times (Figure 1b), participants waited longer for the coin to mature in the high than the limited-persistence environment (mean wait time: 7.8 ± 0.14 in the HP environment vs. 4.1 ± 0.12 in the LP environment). To further quantify this effect, we ran a mixed effects Cox proportional hazard model predicting the likelihood that a participant "survived" until time t in each temporal environment, assuming the coin had not yet matured. The results of this survival analysis corroborated the finding that healthy controls waited longer in the HP than the LP condition, as is in line with normative predictions ($\beta = 0.79$; $p =$

0.023*). Finally, we found that the Area Under the Curve (AUC), an estimate of how many of the first 20s a participant is likely to keep waiting for on any given trial, was significantly higher in the HP than the LP environment in the healthy control group (t = 4.61; p = 0.0002*, Figure 2b).

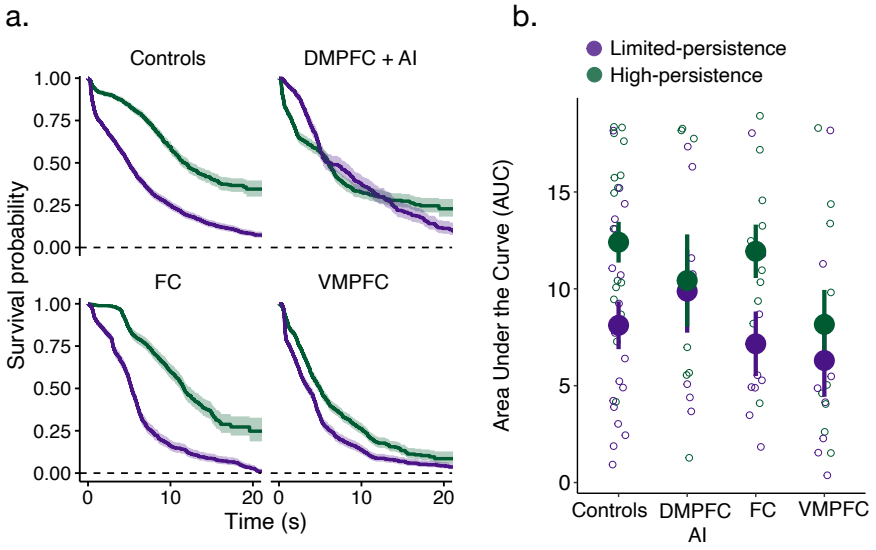


Figure 2. Within-trial willingness-to-wait for each group of lesion patients and healthy controls. a. Mean survival curves averaged across participants within each group. Purple lines correspond to the LP environment while green lines correspond to the HP environment. b. AUC for each group of participants in the high- and limited- persistence environments. Filled circles indicate the mean across the group and each open circle represents a participant.

The Frontal Control (FC) group exhibited waiting behavior that was quantitatively similar to that of the healthy controls (Figure 2a and b). Much like the control group, FC patients waited longer in the HP than the LP condition (mean wait time: 8.01 ± 0.17 in the HP environment vs. 4.01 ± 0.13 in the LP environment). In a mixed effects proportional hazard model that included a term for the interaction between environment and lesion group as well as separate main effects, there was no significant difference between the frontal controls and the healthy participants ($\beta_{FC} = 0.018$; p = 0.80; β_{FC*env}=0.30; p = 0.58). Further mirroring the behavior of the healthy controls, participants in the FC group waited for more of the first 20 seconds in the HP than the LP condition (difference in AUC: t = 5.3; p < 0.0001*).

Unlike the healthy controls and the patients whose lesions did not include our regions of interest, the remaining patients showed deficits in their ability to adaptively calibrate their waiting behavior. The results of the proportional hazard model suggest that both the vmPFC and the dmPFC/AI group behaved in a way that was significantly different from the healthy controls (β_{vmPFC}=1.09, p < 0.0001; $\beta_{dmPFC/AI}$=0.91, p < 0.0001, Figure 2a). Further analyses suggest that although both groups were suboptimal in their decisions to persist, they exhibited different biases with respect to their sensitivity to environmental differences vs. overall reward. In the case of the vmPFC group, participants retained some sensitivity to the difference between the HP and LP conditions ($\beta_{vmPFC*env}$ (compared to controls) = -0.46; p = 0.38; difference in AUC: t = 2.23; p = 0.056*, Figure 2b). However, their overall tendency to wait was greatly reduced (mean wait time: 4.9 ± 0.13 in the HP environment and 3.19 ± 0.12 in the LP environment). Conversely, participants in the dmPFC/AI group waited an average of 5.48 ± 0.15 seconds across the two conditions (compared to an overall mean wait time of 5.54 ± 0.10 for the healthy controls), but they did not calibrate their waiting behavior based on the characteristics of the temporal environment. This reduced sensitivity to the demands of the high- and limited-persistence conditions was evident in the effect on wait duration of the interaction between environment and dmPFC/AI lesion group compared to controls ($\beta_{dmPFC/AI*env}$ = -1.15; p = 0.049*). Finally, there was no significant difference across environments in the AUC for the dmPFC/AI group (t = 0.74, p = 0.49, Figure 2b).

Thus far, we have focused on trial-wise analyses of waiting behavior that collapse across the entirety of the learning block. While it is clear that lesion location modulates willingness-to-wait overall, the process by which it might affect how temporal environments are learned remains unclear. Figure 3 shows a local WTW estimate, averaged across 1s timepoints and across participants, as it evolves throughout learning. This estimate is dynamically updated based on participants' trial-wise quitting behavior, or if they wait for a reward that took longer to appear than their current WTW estimate. By the end of the 12 minutes, both the healthy participants and the frontal controls had learned to wait longer in the high- than the limited-persistence environment (t_C = 11.01; p < 0.001; t_{FC} = 17.1; p < 0.001). The evolution of WTW for the dmPFC/AI and vmPFC groups followed a more unusual pattern, however – for the dmPFC/AI patients, learning appeared to be somewhat erratic, with many reversals and no sensitivity to temporal environment by the end

(t = -0.13; p = 0.89). There was little evidence of learning across the block for the vmPFC group: as is also implied by Figure 2b, the WTW estimates remained significantly lower than the controls' throughout.

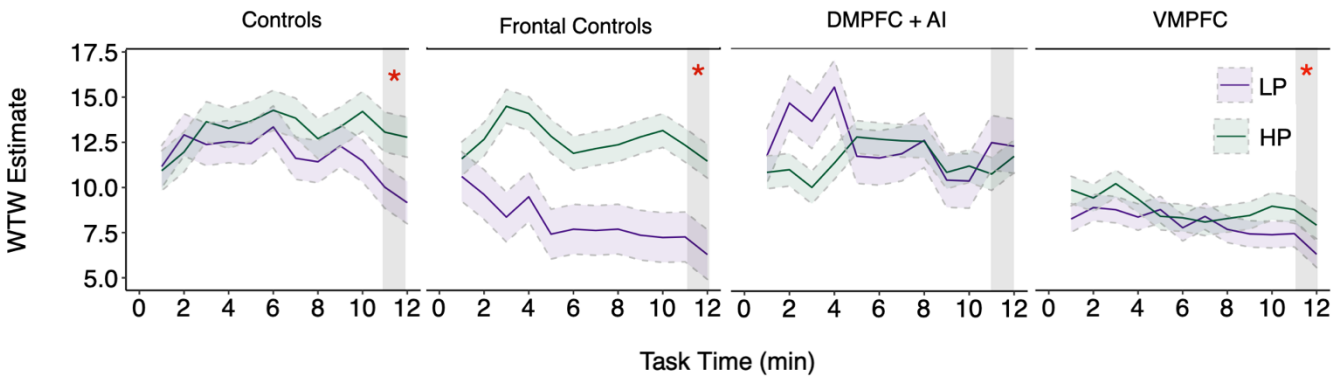


Figure 3. Mean estimate of running WTW sampled at 1s intervals, and collapsed into 60-second bins for legibility. Ribbon corresponds to the standard error of the mean and asterisk denotes significantly different WTW in each environment by the last minute of the learning block (t-test on the last 60 seconds, $p < 0.05$).

4. Discussion

The goal of this study was to establish a causal role for the computation of subjective value in performance on a willingness-to-wait task. Patients with lesions to areas of the brain known to compute context-dependent estimates of value were impaired in their ability to adaptively curtail persistence. Conversely, patients with lesions to other regions of the frontal lobe performed similarly to healthy controls. These findings suggest that failures of persistence may be governed by principles of rational value-maximization rather than lapses in patience or self-control.

Although we have shown that lesions to certain areas of the brain result in clear deficits on a willingness-to-wait task, the underlying mechanisms by which these deficits emerge remains unclear. To this end, ongoing work that fits reinforcement learning models to WTW data [11] may help systematically uncover where in the learning process lesion patients and healthy controls diverge.

References

[1] Metcalfe J, Mischel W. A hot/cool-system analysis of delay of gratification: dynamics of willpower. *Psychol Rev*. 1999;106(1):3.
[2] Mischel W, Ebbesen EB. Attention in delay of gratification. *J Pers Soc Psychol*. 1970;16(2):329.
[3] Baumeister RF, Vohs KD, Tice DM. The strength model of self-control. *Curr Dir Psychol Sci*. 2007; 16(6):351-355.
[4] McGuire JT, Kable JW. Decision makers calibrate behavioral persistence on the basis of time-interval experience. *Cognition*. 2012;124(2):216-226. doi:10.1016/J.COGNITION.2012.03.008
[5] Bartra O, McGuire JT, Kable JW. The valuation system: a coordinate-based meta-analysis of BOLD fMRI experiments examining neural correlates of subjective value. *Neuroimage*. 2013;76:412-427.
[6] Mcguire JT, Kable JW. Medial prefrontal cortical activity reflects dynamic re-evaluation during voluntary persistence. 2015;18(5). doi:10.1038/nn.3994
[7] Casey BJ, Somerville LH, Gotlib IH, et al. Behavioral and neural correlates of delay of gratification 40 years later. *Proc Natl Acad Sci*. 2011;108(36):14998-15003.
[8] Figner B, Knoch D, Johnson EJ, et al. Lateral prefrontal cortex and self-control in intertemporal choice. *Nat Neurosci*. 2010;13(5):538-539.
[9] Rolls ET, Huang C-C, Lin C-P, Feng J, Joliot M. Automated anatomical labelling atlas 3. *Neuroimage*. 2020;206:116189
[10] Lang EA, van Geen C, Tedeschi E, Marvin CB, Shohamy D. *J Exp Psychol Gen*. 2021.
[11] Chen, Y., Li, T., Lynch, J., & McGuire, J.T. (in prep). *A reinforcement learning model of adaptive behavioral persistence.*

PROGRAM COMMITTEE

We would like to thank our area chairs, Quentin Huys and Marc Bellemare for their tremendous efforts in assembling an outstanding program. We would further like to thank the following people who graciously agreed to form our program committee. Their hard work in reviewing the abstracts is essential to the success of this conference.

Aaron Bornstein aaron.bornstein@uci.edu UCI
Adam White amw8@ualberta.ca University of Alberta
Agnes Norbury agnes.norbury.10@ucl.ac.uk UCL
Aleksandra Faust sandrafaust@google.com Google Brain
Alessandro Lazaric alessandro.lazaric@inria.fr FAIR
Alex Kacelnik alex.kacelnik@zoo.ox.ac.uk Oxford University
Alexandra Kearney hi@alexkearney.com Alberta
Amitai Shenhav amitai_shenhav@brown.edu Brown University
Amy Zhang amy.x.zhang@mail.mcgill.ca McGill University
Anastasia Christakou a.christakou@reading.ac.uk Reading University
Andre Barreto andrebarreto@google.com DeepMind
Anna Konova abk288@nyu.edu Rutgers University - New Brunswick
Balaraman Ravindran ravi@cse.iitm.ac.in Indian Institute of Technology, Madras
Benjamin Van Roy bvr@stanford.edu Stanford University
Bo Liu boliu@auburn.edu Auburn University
Bruno C. da Silva bsilva@cs.umass.edu University of Massachusetts
Carlos Diuk carlosdiuk@gmail.com Princeton University
Caroline Charpentier ccharpen@caltech.edu California Institute of Technology
Caswell Barry caswell.barry@ucl.ac.uk University College London
Cedric Colas cedric.colas@inria.fr INRIA
Christian Ruff christian.ruff@econ.uzh.ch University of Zurich
Colin Camerer camerer@hss.caltech.edu Caltech
Daniela Schiller daniela.schiller@mssm.edu Icahn School of Medicine at Mount Sinai
David Abel dmabel@deepmind.com DeepMind
Dominik Bach dominik.bach@uzh.ch University of Zurich
Elliot Ludvig E.Ludvig@Warwick.ac.uk University of Warwick
Emma Brunskill ebrun@cs.stanford.edu Stanford University
Erie Boorman edboorman@ucdavis.edu UC Davis
Evan Russek evrussek@gmail.com Princeton University
Falk Lieder falk.lieder@gmail.com UC Berkeley
Francois Rivest francois.rivest@rmcc-cmrc.ca Royal Military College of Canada
Frederike Petzschner frederike_petzschner@brown.edu Brown University
G Elliott Wimmer e.wimmer@ucl.ac.uk University College London
Genela Morris gmorris@sci.haifa.ac.il University of Haifa
Georg Ostrovski ostrovski@deepmind.com DeepMind
George Konidaris gdk@cs.brown.edu Brown University
Glen Berseth glen.berseth@mila.quebec Mila
Guillermo Horga horgag@nyspi.columbia.edu Columbia University
Ian Krajbich krajbich.1@osu.edu The Ohio State University
Igor Mordatch imordatch@google.com Google
Isabel Berwian iberwian@princeton.edu Princeton University
Jane Wang wangjane@google.com DeepMind
Jesse Hoey jhoey@cs.uwaterloo.ca University of Waterloo
Jian Li li.jian@pku.edu.cn Peking University
John Martin jdmartin86@gmail.com University of Alberta
John Murray john.murray@yale.edu Yale University

Joshua Berke Joshua.Berke@ucsf.edu University of California San Francisco
Josiah Hanna jphanna@cs.wisc.edu University of Wisconsin – Madison
Karl Friston k.friston@ucl.ac.uk University College London
Katya Kudashkina ekudashkina@gmail.com University of Guelph
Kenji Doya doya@oist.jp Okinawa Institute of Science and Technology
Kenway Louie klouie@cns.nyu.edu NYU
Kimberly Stachenfeld stachenfeld@google.com DeepMind
Kory Mathewson korymath@gmail.com Alberta
Kozuno Tadashi tadashi.kozuno@gmail.com University of Alberta
Laura Graesser lauragraesser@google.com Google
Marc G. Bellemare bellemare@google.com Google Brain
Marcelo Mattar marcelomattar@gmail.com Princeton University
Mark Crowley mcrowley@uwaterloo.ca University of Waterloo
Mark Rowland markrowland@google.com DeepMind
Marlos C. Machado marlosm@google.com DeepMind
Martha White whitem@ualberta.ca University of Alberta
Matteo Pirotta matteo.pirotta@gmail.com Facebook
Matthew Botvinick botvinick@google.com Google
Michael Browning michael.browning@psych.ox.ac.uk University of Oxford
Michael Grubb michael.grubb@trincoll.edu Trinity College
Michael Kaisers Michael.Kaisers@cwi.nl Centrum Wiskunde & Informatica
Nan Jiang nanjiang@illinois.edu University of Illinois at Urbana-Champaign
Nicolas Le Roux nicolas@le-roux.name Microsoft
Nicolas Schuck schuck@mpib-berlin.mpg.de Max Planck Institute Human Development
Ofir Nachum ofirnachum@google.com Google
Olivier Sigaud olivier.sigaud@upmc.fr UPMC
Özgür Simsek o.simsek@bath.ac.uk University of Bath
Pablo Samuel Castro psc@google.com Google
Patrick Pilarski patrick.pilarski@ualberta.ca University of Alberta
Payam Piray ppiray@princeton.edu Princeton University
Peggy Series pseries@inf.ed.ac.uk University of Edinburgh
Peter Latham pel@gatsby.ucl.ac.uk"Gatsby Unit, UCL
Phil Corlett philip.corlett@yale.edu Yale University
Philipp Schwartenbeck philipp.schwartenbeck@tuebingen.mpg.de University of Tuebingen
Philippe Tobler phil.tobler@econ.uzh.ch University of Zurich
Pierre-Yves Oudeyer pierre-yves.oudeyer@inria.fr INRIA
Quentin Huys q.huys@ucl.ac.uk University College London
Richard Valenzano rick.valenzano@ryerson.ca Ryerson University
Robert Wilson bob@email.arizona.edu University of Arizona
Ross Otto ross.otto@mcgill.ca McGill University
Roy Fox roy.d.fox@gmail.com UC Irvine
Ryan Smith rsmith@laureateinstitute.org Laureate Institute for Brain Research
Samuel Gershman gershman@fas.harvard.edu Harvard University
Sarath Chandar sarath.chandar@mila.quebec Mila
Sephora Madjiheurem sephora.madjiheurem.17@ucl.ac.uk University College London
Stefano Panzeri stefano.panzeri@iit.it Istituto Italiano di Tecnologia
Steve Fleming stephen.fleming@ucl.ac.uk UCL
Tim Behrens behrens@fmrib.ox.ac.uk Oxford University
Timothy Mann mann.timothy@acm.org Deepmind
Tobias Hauser t.hauser@ucl.ac.uk UCL
Tom Schaul schaul@google.com DeepMind
Ulrik Beierholm ulrik.beierholm@durham.ac.uk Durham University

Vanessa Brown brownvm2@upmc.edu University of Pittsburgh
Vincent Francois-Lavet vincent.francois@gmail.com RLDM
Warren Powell powell@princeton.edu Princeton University
Will Dabney wdabney@google.com DeepMind
William de Cothi w.decothi@ucl.ac.uk UCL
Xiaosi Gu xiaosi.gu@mssm.edu MSSM
Zhihao Zhang zhihao.zhang@haas.berkeley.edu Berkeley