Learning Action Representations for Reinforcement Learning

Yash Chandak[*] ychandak@cs.umass.edu
Georgios Theocharous[†] theochar@adobe.com
James Kostas[*] jekostas@cs.umass.edu
Scott Jordan[*] sjordan@cs.umass.edu
Philip S. Thomas[*] pthomas@cs.umass.edu

Abstract

Most model-free reinforcement learning methods leverage state representations (embeddings) for generalization, but either ignore structure in the space of actions or assume the structure is provided *a priori*. We show how a policy can be decomposed into a component that acts in a low-dimensional space of action representations and a component that transforms these representations into actual actions. These representations improve generalization over large, finite action sets by allowing the agent to infer the outcomes of actions similar to actions already taken. We provide an algorithm to both learn and use action representations and provide conditions for its convergence. The efficacy of the proposed method is demonstrated on large-scale real-world problems.

1 Introduction

Reinforcement learning (RL) methods have been applied successfully to many simple and game-based tasks. However, their applicability is still limited for problems involving decision making in many real-world settings. One reason is that many real-world problems with significant human impact involve selecting a single decision from a multitude of possible choices. For example, maximizing long-term portfolio value in finance using various trading strategies [Jiang et al., 2017], improving fault tolerance by regulating voltage level of all the units in a large power system [Glavic et al., 2017], and personalized tutoring systems for recommending sequences of videos from a large collection of tutorials [Sidney et al., 2005]. Therefore, it is important that we develop RL algorithms that are effective for real problems, where the number of possible choices is large.

In this paper we consider the problem of creating RL algorithms that are effective for problems with large action sets. Existing RL algorithms handle large *state* sets (e.g., images consisting of pixels) by learning a representation or embedding for states (e.g., using line detectors or convolutional layers in neural networks), which allow the agent to reason and learn using the state representation rather than the raw state. We extend this idea to the set of actions: we propose learning a representation for the actions, which allows the agent to reason and learn by making decisions in the space of action representations rather than the original large set of possible actions. This setup is depicted in Figure 1, where an *internal policy*, π_i, acts in a space of action representations, and a function, f, transforms these representations into actual actions. Together we refer to π_i and f as the *overall policy*, π_o.

Recent work has shown the benefits associated with using action-embeddings [Dulac-Arnold et al., 2015], particularly that they allow for generalization over actions. For real-world problems where

[*]University of Massachusetts Amherst
[†]Adobe Research, San Jose

Preprint. Work in progress.

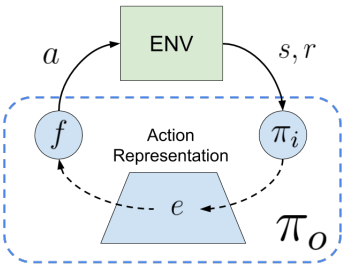

Figure 1: The structure of the proposed overall policy, π_o, consisting of f and π_i, that learns action representations to generalize over large action sets.

there are thousands of possible (discrete) actions, this generalization can significantly speed learning. However, this prior work assumes that fixed and predefined representations are provided. In this paper we present a method to autonomously learn the underlying structure of the action set by using the observed transitions. This method can both learn action representation from scratch and improve upon a provided action representation.

A key component of our proposed method is that it frames the problem of learning an action representation (learning f) as a *supervised* learning problem rather than an RL problem. This is desirable because supervised learning methods tend to learn more quickly and reliably than RL algorithms since they have access to instructive feedback rather than evaluative feedback [Sutton and Barto, 2018]. The proposed learning procedure exploits the structure in the action set by aligning actions based on the similarity of their impact on the state. Therefore, updates to a policy that acts in the space of learned action representation generalizes the feedback received after taking an action to other actions that have similar representations. Furthermore, we prove that our combination of supervised learning (for f) and reinforcement learning (for π_i) within one larger RL agent preserves the almost sure convergence guarantees provided by policy gradient algorithms [Borkar and Konda, 1997].

To evaluate our proposed method empirically, we study two real-world recommender system problems using data from Adobe HelpX and Adobe Photoshop. In both the applications, there are thousands of possible recommendations that could be given at each time step (e.g., which video to suggest the user watch next on the HelpX portal, or which tool to suggest to the user next in the Photoshop software). Our experimental results show our proposed system's ability to significantly improve performance relative to existing methods for these applications by quickly and reliably learning action representations that allow for meaningful generalization over the large discrete set of possible actions.

The rest of this paper is organized to provide in the following order: a background on RL, related work, and the following primary contributions:

- A new parameterization, called the *overall policy*, that leverages action representations. We show that for all optimal policies, π^*, there exist parameters for this new policy class that are equivalent to π^*.
- A proof of equivalence of the policy gradient update between the overall policy and the *internal policy*.
- A supervised learning algorithm for learning action representations (f in Figure 1). This procedure can be combined with any existing policy gradient method for learning the overall policy.
- An almost sure asymptotic convergence proof for the algorithm, which extends existing results for actor-critics [Borkar and Konda, 1997].
- Experimental results on real-world domains with thousands of actions using actual data collected from Adobe HelpX and Photoshop.

2 Background

We consider problems modeled as discrete-time *Markov decision processes* (MDPs) with discrete states and finite actions. An MDP is represented by a tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, d_0)$. $\mathcal{S}$ is the set of all possible states, called the state space, and $\mathcal{A}$ is a finite set of actions, called the action set. Though our notation assumes that the state set is finite, our primary results extend to MDPs with continuous states. In this work, we restrict our focus to MDPs with finite action sets, and $|\mathcal{A}|$ denotes the size of the action set. The random variables, $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$, and $R_t \in \mathbb{R}$ denote the state, action, and reward at time $t \in \{0, 1, \dots\}$. We assume that $R_t \in [-R_{\max}, R_{\max}]$ for some finite $R_{\max}$. The first state, S_0, comes from an initial distribution, d_0, and the reward function $\mathcal{R}$ is defined so that $\mathcal{R}(s, a) = \mathbf{E}[R_t | S_t = s, A_t = a]$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Hereafter, for brevity, we write P to denote both probabilities and probability densities, and when writing probabilities and expectations, write s, a, s' or e to denote both elements of various sets *and* the events $S_t = s, A_t = a, S_{t+1} = s'$, or $E_t = e$ (defined later). The desired meaning for s, a, s' or e should be clear from context. The reward discounting parameter is given by $\gamma \in [0, 1)$. $\mathcal{P}$ is the state transition function, such that $\forall s, a, s', t, \mathcal{P}(s, a, s') \coloneqq P(s'|s, a)$.

A policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a conditional distribution over actions for each state: $\pi(a, s) \coloneqq P(A_t = a | S_t = s)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$, and t. Although π is simply a function, we write $\pi(a|s)$ rather than $\pi(a, s)$ to emphasize that it is a conditional distribution. For a given $\mathcal{M}$, an agent's goal is to find a policy that maximizes the expected sum of discounted future rewards. For any policy π, the corresponding state-action value function is $q^\pi(s, a) = \mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi]$, where conditioning on π denotes that $A_{t+k} \sim \pi(\cdot|S_{t+k})$ for all A_{t+k} and S_{t+k} for $k \in [t + 1, \infty)$. The state value function is $v^\pi(s) = \mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, \pi]$. It follows from the Bellman equation that $v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q^\pi(s, a)$. An optimal policy is any $\pi^* \in \text{argmax}_{\pi \in \Pi} \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R_t | \pi]$, where Π denotes the set of all possible policies, and v^* is shorthand for v^{π^*}.

3 Related Work

Here we summarize the most related work and discuss how they relate to the proposed work.

Factorizing Action Space: To reduce the size of large action spaces, Pazis and Parr [2011] considered representing each action in binary format and learning a value function associated with each bit. A similar binary based approach was also used as an ensemble method to learning optimal policies for MDPs with large action sets [Sallans and Hinton, 2004]. For planning problems, Cui and Khardon [2016, 2018] showed how a gradient based search on a symbolic representation of the state-action value function can be used to address scalability issues. More recently, it was shown that better performances can be achieved on Atari 2600 games [Bellemare et al., 2013] when actions are factored into their primary categories [Sharma et al., 2017]. All these methods assumed that a handcrafted binary decomposition of raw actions was provided. To deal with discrete actions that might have an underlying continuous representation, Van Hasselt and Wiering [2009] used policy gradients with continuous actions and selected the nearest discrete action. This work was extended by Dulac-Arnold et al. [2015] for larger domains, where they performed action representation look up, similar to our approach. However, they assumed that the embeddings for the actions are given, *a priori*. We present a method that can learn action representations with no prior knowledge or further optimize available action representations. If no prior knowledge is available, our method learns these representations from scratch autonomously.

Auxiliary Tasks: Previous works showed empirically that supervised learning with the objective to predict a component of a transition tuple (s, a, r, s') from the others, can be useful as an auxiliary method to learn state representations [Jaderberg et al., 2016] or to obtain intrinsic rewards [Shelhamer et al., 2016, Pathak et al., 2017]. We show how the overall policy itself can be decomposed using an action representation module learned using a similar loss function.

Motor Primitives: Research in neuroscience suggests that animals decompose their plans into mid-level abstractions, rather than the exact low-level motor controls needed for each movement [Jing et al., 2004]. Such abstractions of behavior that form the building blocks for motor control are often called *motor primitives* [Lemay and Grill, 2004, Mussa-Ivaldi and Bizzi, 2000]. In the field of robotics, dynamical system based models have been used to construct *dynamic movement primitives* (DMPs) for continuous control [Ijspeert et al., 2003, Schaal, 2006]. Imitation learning can also be

used to learn DMPs, which can be fine-tuned online using RL [Kober and Peters, 2009b,a]. However, these are significantly different from our work as they are specifically parameterized for robotics tasks and produce an encoding for kinematic trajectory plans, not the actions.

Later, Thomas and Barto [2012] showed how a goal-conditioned policy can be learned using multiple motor primitives that control only useful sub-spaces of the underlying control problem. To learn binary motor primitives, Thomas and Barto [2011] showed how a policy can be modeled as a composition of multiple "coagents", each of which learns using only the local policy gradient information [Thomas, 2011]. Our work follows a similar direction, but we focus on automatically learning optimal continuous-valued action representations for discrete actions. For action representations, we present a method that uses supervised learning and restricts the usage of high variance policy gradients to train the internal policy only.

Other Domains: In supervised learning, representations of the output categories have been used to extract additional correlation information among the labels. Popular examples include learning label embeddings for image classification [Akata et al., 2016] and learning word embeddings for natural language problems [Mikolov et al., 2013]. In contrast, for an RL setup, the policy is a function whose outputs correspond to the available actions. We show how learning action representations can be beneficial as well.

4 Generalization over Actions

The benefits of capturing the structure in the underlying state space of MDPs is a well understood and a widely used concept in RL. State representations allow the policy to generalize across states. Similarly, there often exists additional structure in the space of actions that can be leveraged. We hypothesize that exploiting this structure can enable quick generalization across actions, thereby making learning with large action sets feasible. To bridge the gap, we introduce an action representation space, $\mathcal{E} \subseteq \mathbb{R}^d$, and consider a factorized policy, π_o, parameterized by an embedding-to-action mapping function, $f \colon \mathcal{E} \to \mathcal{A}$, and an internal policy, $\pi_i \colon \mathcal{S} \times \mathcal{E} \to [0, 1]$, such that the distribution of A_t given S_t is characterized by:

$$E_t \sim \pi_i(\cdot|S_t), \qquad\qquad A_t = f(E_t).$$

Here, π_i is used to sample $E_t \in \mathcal{E}$, and the function f deterministically maps this representation to an action in the set $\mathcal{A}$. Both these components together form an *overall policy*, π_o. Figure 2 illustrates the probability of each action under such a parameterization. With a slight abuse of notation, we use $f^{-1}(a)$ to denote the set of representations that are mapped to the action a by the function f, i.e., $f^{-1}(a) \coloneqq \{e \in \mathcal{E} : f(e) = a\}$.

In the following sections we discuss the existence of an optimal policy π_o^* and the learning procedure for π_o. To elucidate the steps involved, we split it into four parts. First, we show that there exists f and π_i such that π_o is an optimal policy. Then we present the supervised learning process for the function f when π_i is fixed. Next we give the policy gradient learning process for π_i when f is fixed. Finally, we combine these methods to learn f and π_i simultaneously.

4.1 Existence of π_i and f to Represent An Optimal Policy

In this section, we aim to establish a condition under which π_o can represent an optimal policy. Consequently, we then define the optimal set of π_o and π_i using the proposed parameterization. To establish the main results we begin with the necessary assumptions.

The characteristics of the actions can be naturally associated with how they influence state transitions. In order to learn a representation for actions that captures this structure, we consider a standard Markov property, often used for learning probabilistic graphical models [Ghahramani, 2001], and make the following assumption that the transition information can be sufficiently encoded to infer the action that was executed.

Assumption A1. *Given an embedding E_t, A_t is conditionally independent of S_t and S_{t+1}:*

$$P(A_t|S_t, S_{t+1}) = \int_{\mathcal{E}} P(A_t|E_t = e)P(E_t = e|S_t, S_{t+1})\,\mathrm{d}e.$$

Assumption A2. *Given the embedding E_t the action, A_t is deterministic and is represented by a function $f : \mathcal{E} \to \mathcal{A}$, i.e., $\exists a$ such that $P(A_t = a|E_t = e) = 1$.*

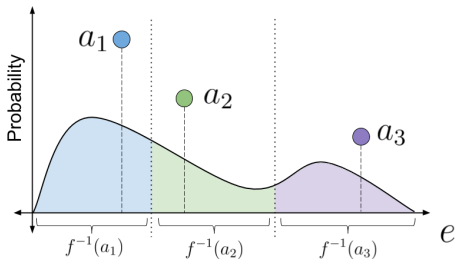

Figure 2: Illustration of the probability induced for three actions by the probability density of $\pi_i(e, s)$ on a 1-D embedding space. The x-axis represents the embedding, e, and the y-axis represents the probability. The colored regions represent the mapping $a = f(e)$, where each color is associated with a specific action.

We now establish a necessary condition under which our proposed policy can represent an optimal policy. This condition will also be useful later when deriving learning rules.

Lemma 1. *Under Assumptions* (A1)–(A2)*, which defines a function f, for all π, there exists a π_i such that*

$$v^{\pi}(s) = \sum_{a \in \mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s) q^{\pi}(s, a) \, \mathrm{d}e.$$

The proof is deferred to the Appendix A. Following Lemma 1, we use π_i and f to define the overall policy as

$$\pi_o(a|s) \coloneqq \int_{f^{-1}(a)} \pi_i(e|s) \, \mathrm{d}e. \tag{1}$$

Theorem 1. *Under Assumptions* (A1)–(A2)*, which defines a function f, there exists an overall policy, π_o, such that $v^{\pi_o} = v^{\star}$.*

Proof. This follows directly from Lemma 1. Because the state and action sets are finite, the rewards are bounded, and $\gamma \in [0, 1)$, there exists at least one optimal policy. For any optimal policy $\pi^{\star}$, the corresponding state-value and state-action-value functions are the unique $v^{\star}$ and $q^{\star}$, respectively. By Lemma 1 there exist f and π_i such that

$$v^{\star}(s) = \sum_{a \in \mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s) q^{\star}(s, a) \, \mathrm{d}e. \tag{2}$$

Therefore, there exists π_i and f, such that the resulting π_o has the state-value function $v^{\pi_o} = v^{\star}$, and hence it represents an optimal policy. □

Note that Theorem 1 establishes existence of an optimal overall policy based on equivalence of the state-value function, but does *not* ensure that all optimal policies can be represented by an overall policy. Using (2), we define $\Pi_o^{\star} \coloneqq \{\pi_o : v^{\pi_o} = v^{\star}\}$. Correspondingly, we define the set of *optimal internal policies* as $\Pi_i^{\star} \coloneqq \{\pi_i : \exists \pi_o^{\star} \in \Pi_o^{\star}, \exists f, \pi_o^{\star}(a|s) = \int_{f^{-1}(a)} \pi_i(e|s) \, \mathrm{d}e\}$.

4.2 Supervised Learning of f For a Fixed π_i

Theorem 1 shows that there exist π_i and a function f, which helps in predicting the action responsible for the transition from S_t to S_{t+1}, such that the corresponding overall policy is optimal. However, such a function, f, may not be known *a priori*. In this section, we present a method to estimate f using data collected from interactions with the environment.

By Assumptions (A1)–(A2), $P(A_t|S_t, S_{t+1})$ can be written in terms of f and $P(E_t|S_t, S_{t+1})$. We propose searching for an estimator, $\hat{f}$, of f and an estimator, $\hat{g}(E_t|S_t, S_{t+1})$, of $P(E_t|S_t, S_{t+1})$ such

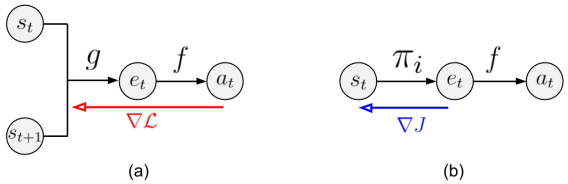

Figure 3: (a) Given a state transition tuple, functions g and f are used to estimate the action taken. The red arrow denotes the gradients of the supervised loss (5) for learning the parameters of these functions. (b) During execution, an internal policy, π_i, can be used to first select an action representation, e. The function f, obtained from previous learning procedure, then transforms this representation to an action. The blue arrow represents the internal policy gradients (7) obtained using Lemma 2 to update π_i.

that a reconstruction of $P(A_t|S_t, S_{t+1})$ is accurate. Let this estimate of $P(A_t|S_t, S_{t+1})$ based on $\hat{f}$ and $\hat{g}$ be

$$\hat{P}(A_t|S_t, S_{t+1}) = \int_{\mathcal{E}} \hat{f}(A_t|E_t = e)\hat{g}(E_t = e|S_t, S_{t+1})\,\mathrm{d}e. \tag{3}$$

One way to measure the difference between $P(A_t|S_t, S_{t+1})$ and $\hat{P}(A_t|S_t, S_{t+1})$ is using the expected (over states coming from the on-policy distribution) Kullback-Leibler (KL) divergence

$$\begin{aligned} &= -\,\mathbf{E}\left[\sum_{a\in\mathcal{A}} P(a|S_t, S_{t+1}) \ln\left(\frac{\hat{P}(a|S_t, S_{t+1})}{P(a|S_t, S_{t+1})}\right)\right] \\ &= -\,\mathbf{E}\left[\ln\left(\frac{\hat{P}(A_t|S_t, S_{t+1})}{P(A_t|S_t, S_{t+1})}\right)\right]. \end{aligned} \tag{4}$$

Since the observed transition tuples, (S_t, A_t, S_{t+1}), contain the action responsible for the given S_t to S_{t+1} transition, an on-policy sample estimate of the KL-divergence can be computed readily using (4). We adopt the following loss function based on the KL divergence between $P(A_t|S_t, S_{t+1})$ and $\hat{P}(A_t|S_t, S_{t+1})$:

$$\mathcal{L}(\hat{f}, \hat{g}) = -\mathbf{E}\left[\ln\left(\hat{P}(A_t|S_t, S_{t+1})\right)\right], \tag{5}$$

where the denominator in (4) is not included in (5) because it does not depend on $\hat{f}$ or $\hat{g}$. If $\hat{f}$ and $\hat{g}$ are parameterized, their parameters can be learned by minimizing the loss function, $\mathcal{L}$, using a supervised learning procedure.

A computational graph for this model is shown in Figure 3. We refer the reader to the Appendix D for the parameterizations of $\hat{f}$ and $\hat{g}$ used in our experiments. Note that, while $\hat{f}$ will be used for f in an overall policy, $\hat{g}$ is only used to find $\hat{f}$, and will not serve an additional purpose.

As this supervised learning process only requires estimating $P(A_t|S_t, S_{t+1})$, it does not require (or depend on) the rewards. This partially mitigates the problems due to sparse and stochastic rewards, since an alternative informative supervised signal is always available. This is advantageous for making the action representation component of the overall policy learn quickly and with low variance updates.

4.3 Learning π_i For a Fixed f

A common method for learning a policy parameterized with weights θ is to optimize the discounted start-state objective function, $J(\theta) := \sum_{s\in\mathcal{S}} d_0(s)v^{\pi}(s)$. For a policy with weights θ, the expected performance of the policy can be improved by ascending the *policy gradient*, $\frac{\partial J(\theta)}{\partial \theta}$.

Let the state-value function associated with the internal policy, π_i, be $v^{\pi_i}(s) = \mathbf{E}[\sum_{t=0}^{\infty}\gamma^t R_t|s, \pi_i, f]$, and the state-action value function $q^{\pi_i}(s, e) = \mathbf{E}[\sum_{t=0}^{\infty}\gamma^t R_t|s, e, \pi_i, f]$. We then define the performance function for π_i as:

$$J_i(\theta) := \sum_{s\in\mathcal{S}} d_0(s)v^{\pi_i}(s). \tag{6}$$

Viewing the embeddings as the action for the agent with policy π_i, the policy gradient theorem [Sutton et al., 2000], states that the unbiased [Thomas, 2014] gradient of (6) is,

$$\frac{\partial J_i(\theta)}{\partial \theta} = \sum_{t=0}^{\infty} \mathbf{E}\left[\gamma^t \int_{\mathcal{E}} q^{\pi_i}(S_t, e) \frac{\partial}{\partial \theta} \pi_i(e|S_t)\, \mathrm{d}e\right], \tag{7}$$

where, the expectation is over states from d^{π}, as defined by Sutton et al. [2000] (which is not a true distribution, since it is not normalized). The parameters of the internal policy can be learned by iteratively updating its parameters in the direction of $\partial J_i(\theta)/\partial\theta$. Since there are no special constraints on the policy π_i, any policy gradient algorithm designed for continuous control, like DPG [Silver et al., 2014], PPO [Schulman et al., 2017], NAC [Bhatnagar et al., 2009] etc., can be used out-of-the-box.

However, note that the performance function associated with the overall policy, π_o (consisting of function f and the internal policy parameterized with weights θ), is:

$$J_o(\theta, f) = \sum_{s \in \mathcal{S}} d_0(s) v^{\pi_o}(s).$$

The ultimate requirement is the improvement of this overall performance function, $J_o(\theta, f)$, and not just $J_i(\theta)$. So, how useful is it to update the internal policy, π_i, by following the gradient of its own performance function? The following lemma answers this question.

Lemma 2. *For all deterministic functions, f, which map each point, $e \in \mathbb{R}^d$, in the representation space to an action, $a \in \mathcal{A}$, the expected updates to θ based on $\frac{\partial J_i(\theta)}{\partial \theta}$ are equivalent to updates based on $\frac{\partial J_o(\theta, f)}{\partial \theta}$. That is,*

$$\frac{\partial J_o(\theta, f)}{\partial \theta} = \frac{\partial J_i(\theta)}{\partial \theta}.$$

The proof is deferred to the Appendix B. The chosen parameterization for the policy has this special property, which allows π_i to be learned using its internal policy gradient. Since this gradient update does not require computing the value of any $\pi_o(a|s)$ explicitly, the potentially intractable computation of f^{-1} in (1) required for π_o can be avoided. Instead, $\partial J_i(\theta)/\partial\theta$ can be used directly to update the parameters of the internal policy while still optimizing the overall policy's performance, $J_o(\theta, f)$.

4.4 Learning π_i and f Simultaneously

Since the supervised learning procedure for f does not require rewards, a few initial trajectories can contain enough information to begin learning a useful action representation. As more data becomes available it can be used for fine-tuning and improving the action representations.

4.4.1 Algorithm

We call our algorithm **p**olicy **g**radients with **r**epresentations for **a**ctions (PG-RA). PG-RA first initializes the parameters in the action representation component by sampling a few trajectories using a random policy and using the supervised loss defined in (5). If additional information is known about the actions, as assumed in prior work [Dulac-Arnold et al., 2015], it can also be considered when initializing the action representations. Optionally, once these action representations are initialized, they can be kept fixed.

In the Algorithm 1, Lines 2-9 illustrate the online update procedure for all of the parameters involved. Each time step in the episode is represented by t. For each step, an action representation is sampled and is then mapped to an action by $\hat{f}$. Having executed this action in the environment, the observed reward is then used to update the internal policy, π_i, using *any* policy gradient algorithm. Depending on the policy gradient algorithm, if a critic is used then semi-gradients of the TD-error are used to update the parameters of the critic. In other cases, like in REINFORCE [Williams, 1992] where there is no critic, this step can be ignored. The observed transition is then used in Line 9 to update the parameters of $\hat{f}$ and $\hat{g}$ so as to minimize the supervised learning loss (5). In our experiments, Line 9 uses a stochastic gradient update.

Algorithm 1: Policy Gradient with Representations for Action (PG-RA)

Initialize action representations
for $episode = 0, 1, 2...$ **do**
  for $t = 0, 1, 2...$ **do**
    Sample action embedding, E_t, from $\pi_i(\cdot|S_t)$
    $A_t = \hat{f}(E_t)$
    Execute A_t and observe S_{t+1}, R_t
    Update π_i using *any* policy gradient algorithm
    Update critic (if any) to minimize TD error
    Update $\hat{f}$ and $\hat{g}$ to minimize $\mathcal{L}$ defined in (5)

4.4.2 PG-RA Convergence

If the action representations are held fixed while learning the internal policy, then as a consequence of Property 2, convergence of our algorithm directly follows from previous two-timescale results [Borkar and Konda, 1997, Bhatnagar et al., 2009]. Here we show that learning both π_i and f simultaneously using our PG-RA algorithm can also be shown to converge by using a three-timescale analysis.

Similar to prior work [Bhatnagar et al., 2009, Degris et al., 2012, Konda and Tsitsiklis, 2000], for analysis of the updates to the parameters, $\theta \in \mathbb{R}^{d_\theta}$, of the internal policy, π_i, we use a projection operator $\Gamma : \mathbb{R}^{d_\theta} \to \mathbb{R}^{d_\theta}$ that projects any $x \in \mathbb{R}^{d_\theta}$ to a compact set $\mathcal{C} \subset \mathbb{R}^{d_\theta}$. We then define an associated vector field operator, $\hat{\Gamma}$, that projects any gradients leading outside the compact region, $\mathcal{C}$, back to $\mathcal{C}$. We refer the reader to the Appendix C.3 for precise definitions of these operators and the additional standard assumptions (A3)–(A5). Practically, however, we do not project the iterates to a constraint region as they are seen to remain bounded (without projection).

Theorem 2. *Under Assumptions* (A1)–(A5)*, the internal policy parameters* θ_t*, converge to* $\hat{\mathcal{Z}} = \left\{ x \in \mathcal{C} | \hat{\Gamma} \left(\frac{\partial J_i(x)}{\partial \theta} \right) = 0 \right\}$ *as* $t \to \infty$*, with probability one.*

Proof. (Outline) We consider three learning rate sequences, such that the update recursion for the internal policy is on the slowest timescale, the critic's update recursion is on the fastest, and the action representation module's has an intermediate rate. With this construction, we leverage the three-timescale analysis technique [Borkar, 2009] and prove convergence. The complete proof is in the Appendix C. □

5 Empirical Analysis

A core motivation of this work is to provide an algorithm that can be used as a drop-in extension for improving the action generalization capabilities of existing policy gradient methods for problems with large action spaces. We consider two standard policy gradient methods: actor-critic (AC) and deterministic-policy-gradient (DPG) [Silver et al., 2014] in our experiments. Just like previous algorithms, we also ignore the γ^t terms and perform the biased policy gradient update to be practically more sample efficient [Thomas, 2014]. We believe that the reported results can be further improved by using the proposed method with other policy gradient methods; we leave this for future work. For detailed discussion on parameterization of the function approximators and hyper-parameter search, see Appendix D.

5.1 Domains

Maze: As a proof-of-concept, we constructed a continuous-state maze environment where the state comprised of the coordinates of the agent's current location. The agent has n equally spaced actuators (each actuator moves the agent in the direction the actuator is pointing towards) around it, and it can choose whether each actuator should be on or off. Therefore, the size of the action set is exponential in the number of actuators, that is $|\mathcal{A}| = 2^n$. The net outcome of an action is the vectorial summation of the displacements associated with the selected actuators. The agent is rewarded with a

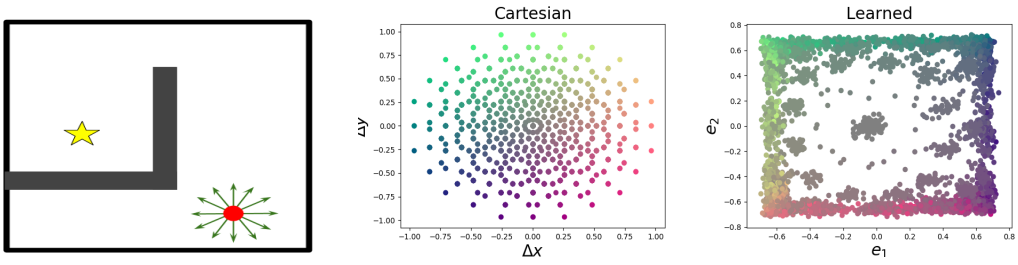

Figure 4: (a) The maze environment. The star denotes the goal state, the red dot corresponds to the agent and the arrows around it are the 12 actuators. Each action corresponds to a unique combination of these actuators. Therefore, in total 2^{12} actions are possible. (b) 2-D representations for the displacements in the Cartesian co-ordinates caused by each action, and (c) learned action embeddings. In both (b) and (c), each action is colored based on the displacement (Δx, Δy) it produces. That is, with the color [R= Δx, G=Δy, B=0.5], where Δx and Δy are normalized to $[0, 1]$ before coloring. Cartesian actions are plotted on co-ordinates (Δx, Δy), and learned ones are on the coordinates in the embedding space. Smoother color transition of the learned representation is better as it corresponds to preservation of the *relative* underlying structure. The 'squashing' of the learned embeddings is an artifact of a non-linearity applied to bound its range.

small penalty for each time step, and a reward of 100 is given upon reaching the goal position. To make the problem more challenging, random noise was added to the action 10% of the time and the maximum episode length was 150 steps.

This environment is a useful test bed as it requires solving a long horizon task in an MDP with a large action set and a single goal reward. Further, we know the Cartesian representation for each of the actions, and can thereby use it to visualize the learned representation, as shown in Figure 4.

Real-word recommender systems: We consider two real-world applications of recommender systems that require decision making over *multiple time steps*.

First, Adobe HelpX, a web-based video-tutorial platform, which has a recommendation engine that suggests a series of tutorial videos on various Adobe software products. The aim is to meaningfully engage the users in learning how to use these software products and convert novice users into experts in their respective areas of interest. The tutorial suggestion at each time step is made from a large pool of available tutorial videos on several products.

The second application is Adobe Photoshop, a professional multi-media editing software. Modern multimedia editing software often contain many tools that can be used to manipulate the media, and this wealth of options can be overwhelming for users. In this Adobe Photoshop domain, an agent suggests which of the available tools the user may want to use next. The objective is to increase user productivity and assist in achieving their end goal.

For both of these applications, an existing log of user's click stream data was used to create an n-gram based MDP model for user behavior [Shani et al., 2005]. In the Adobe HelpX tutorial recommendation task, user activity for a three month period was observed. Sequences of user interaction were aggregated to obtain over 29 million clicks. Similarly, for a month long duration, sequential usage patterns of the tools in the Adobe Photoshop software were collected to obtain a total of over 1.75 billion user clicks. Tutorials and tools that had less than 100 clicks in total were discarded. The remaining 1498 tutorials and 1843 tools for the HelpX platform and Adobe Photoshop, respectively, were used to create the action set for the MDP model.

The state for the MDP consists of the feature descriptors associated with each item (tutorial or tool) in the current n-gram. Rewards were chosen based on a surrogate measure for difficulty level of tutorials on HelpX portal and popularity of final outcomes of user interactions in Photoshop, respectively. Since such data is sparse, only 5% of the items had rewards associated with them, and the maximum reward for any item was 100.

Often the problem of recommendation is formulated as a contextual bandit or collaborative filtering problem, but as shown by Theocharous et al. [2015] these approaches fail to capture the long term

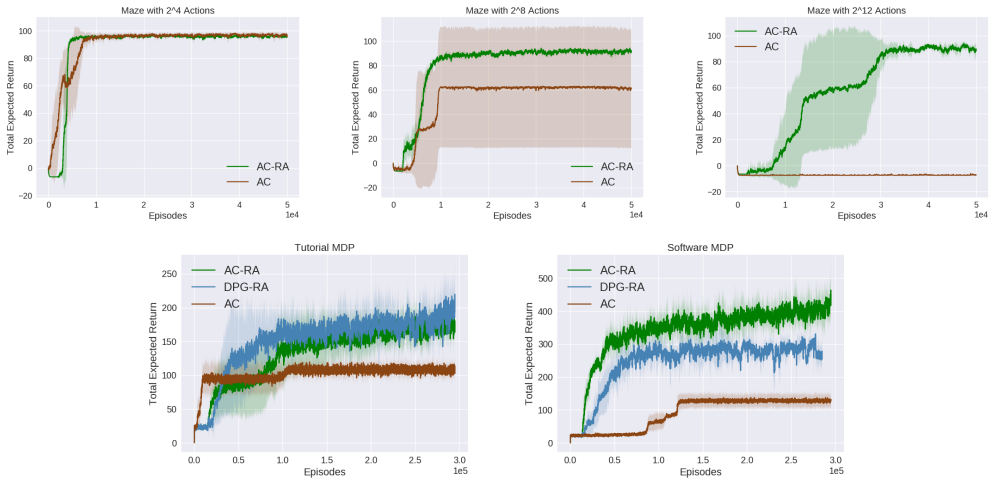

Figure 5: (Top) Results on the Maze domain with $2^4, 2^8$, and 2^{12} actions respectively. (Bottom) Results on a) Adobe HelpX MDP b) Adobe Photoshop MDP. AC-RA and DPG-RA are the variants of PG-RA algorithm that uses actor-critic (AC) and DPG, respectively.

value of the prediction. Solving this problem for a longer time horizon with a large number of actions (tutorials/tools) makes this real-life problem a useful and a challenging domain for RL algorithms.

5.2 Results

Visualizing the learned action representations

To understand the internal working of our proposed algorithm, we present visualizations of the learned action representations on the maze domain. A pictorial illustration of the environment is provided in Figure 4. Here, the underlying structure in the set of actions is related to the displacements in the Cartesian coordinates. This provides an intuitive base case against which we can compare our results.

In Figure 4, we provide a comparison between the action representations learned using our algorithm and the underlying Cartesian representation of the actions. It can be seen that the proposed method extracts useful structure in the action space. Actions which correspond to settings where the actuators on the opposite side of the agent are selected result in relatively small displacements to the agent. These are the ones in the center of plot. In contrast, maximum displacement in any direction is caused by only selecting actuators facing in that particular direction. Actions corresponding to those are at the edge of the representation space. The smooth color transition indicates that not only the information about magnitude of displacement but the direction of displacement is also represented. Therefore, the learned representations efficiently preserve the relative transition information among all the actions. To make exploration step tractable in the internal policy, π_i, we bound the representation space along each dimension to the range $[-1, 1]$ using *Tanh* non-linearity. This results in 'squashing' of these representations around the edge of this range.

Performance Improvement

The plots in Figure 5 for the Maze domain show how the performance of standard actor-critic (AC) method deteriorates as the number of actions increases, even though the goal remains the same. However, with the addition of an action representation module it is able to capture the underlying structure in the action space and consistently perform well across all settings. Similarly, for both Adobe HelpX and Adobe Photoshop MDPs, standard AC methods fail to reason over longer time horizons under such an overwhelming number of actions, choosing mostly one-step actions that have high returns. In comparison, instances of our proposed algorithm are not only able to achieve significantly higher return, up to $2\times$ and $4\times$ in the respective tasks, but they do so much quicker. These results reinforce our claim that learning action representations allow implicit generalization of feedback to other actions embedded in proximity to executed action.

Further, under the PG-RA algorithm, only a fraction of total parameters, the ones in the internal policy, are learned using the high variance policy gradient updates. The other set of parameters associated with action representations are learned by a supervised learning procedure. This reduces the variance of updates significantly, thereby making the PG-RA algorithms learn a better policy faster. This is evident from the plots in the Figure 5. These advantages allow the internal policy, π_i, to quickly approximate an optimal policy without succumbing to the curse of large actions sets.

6 Summary and Future Work

In this paper, we built upon the core idea of leveraging the structure in the space of actions and showed its importance for enhancing generalization over large action sets in real-world large-scale applications. Our approach has three key advantages. (a) Simplicity: by simply using the observed transitions, an additional supervised update rule can be used to learn action representations. (b) Theory: we showed that the proposed overall policy class can represent an optimal policy and derived the associated learning procedures for its parameters. (c) Extensibility: as the PG-RA algorithm indicates, our approach can be easily extended using other policy gradient methods to leverage additional advantages, while preserving the convergence guarantees.

An interesting future direction would be to extend the results for capturing the structure of a high dimensional continuous action space ($\in \mathbb{R}^m$) into a lower dimensional representation space ($\in \mathbb{R}^n$) as well. Unlike finite set of actions that can be embedded in a continuous space, the key challenge is that learning lower dimensional space for continuous action inevitably results in the inability to represent some sections of the original action space (as $n < m$).

References

Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2016.

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, 2013.

S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

V. S. Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.

V. S. Borkar and V. R. Konda. The actor-critic algorithm as multi-time-scale stochastic approximation. *Sadhana*, 22(4):525–543, 1997.

H. Cui and R. Khardon. Online symbolic gradient-based optimization for factored action mdps. In *IJCAI*, pages 3075–3081, 2016.

H. Cui and R. Khardon. Lifted stochastic planning, belief propagation and marginal MAP. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Z. Ghahramani. An introduction to hidden markov models and bayesian networks. *International journal of pattern recognition and artificial intelligence*, 15(01):9–42, 2001.

M. Glavic, R. Fonteneau, and D. Ernst. Reinforcement learning for electric power system decision and control: Past considerations and perspectives. *IFAC-PapersOnLine*, 50(1):6918–6927, 2017.

B. D. Haeffele and R. Vidal. Global optimality in neural network training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7331–7339, 2017.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554, 2003.

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.

Z. Jiang, D. Xu, and J. Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

J. Jing, E. C. Cropper, I. Hurwitz, and K. R. Weiss. The construction of movement with behavior-specific and behavior-independent modules. *Journal of Neuroscience*, 24(28):6315–6325, 2004.

K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.

J. Kober and J. Peters. Learning motor primitives for robotics. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2112–2118. IEEE, 2009a.

J. Kober and J. R. Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009b.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

G. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation in reinforcement learning using the fourier basis. In *AAAI*, volume 6, page 7, 2011.

M. A. Lemay and W. M. Grill. Modularity of motor output evoked by intraspinal microstimulation in cats. *Journal of neurophysiology*, 91(1):502–514, 2004.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

F. A. Mussa-Ivaldi and E. Bizzi. Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 355(1404):1755–1769, 2000.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

J. Pazis and R. Parr. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1185–1192, 2011.

B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5(Aug):1063–1088, 2004.

S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

G. Shani, D. Heckerman, and R. I. Brafman. An MDP-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

S. Sharma, A. Suresh, R. Ramesh, and B. Ravindran. Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. *arXiv preprint arXiv:1705.07269*, 2017.

E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

K. D. Sidney, S. D. Craig, B. Gholson, S. Franklin, R. Picard, and A. C. Graesser. Integrating affect sensors in an intelligent tutoring system. In *Affective Interactions: The Computer in the Affective Loop Workshop at*, pages 7–13, 2005.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

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

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.

G. Theocharous, P. S. Thomas, and M. Ghavamzadeh. Ad recommendation systems for life-time value optimization. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1305–1310. ACM, 2015.

P. Thomas. Bias in natural actor-critic algorithms. In *International Conference on Machine Learning*, pages 441–448, 2014.

P. S. Thomas. Policy gradient coagent networks. In *Advances in Neural Information Processing Systems*, pages 1944–1952, 2011.

P. S. Thomas and A. G. Barto. Conjugate Markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 137–144, 2011.

P. S. Thomas and A. G. Barto. Motor primitive discovery. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximationtechnical. Technical report, Report LIDS-P-2322. Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1996.

H. Van Hasselt and M. A. Wiering. Using continuous action spaces to solve discrete problems. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 1149–1156. IEEE, 2009.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Z. Zhu, D. Soudry, Y. C. Eldar, and M. B. Wakin. The global optimization geometry of shallow linear neural networks. *arXiv preprint arXiv:1805.04938*, 2018.

Appendix

A Proof of Lemma 1

Lemma 1. *Under Assumptions* (A1)–(A2)*, which defines a function f, for all π, there exists a π_i such that*

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s) q^\pi(s,a)\,\mathrm{d}e.$$

Proof. The Bellman equation associated with a policy, π, for any MDP, $\mathcal{M}$, is:

$$\begin{aligned} v^\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) q^\pi(s,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} P(s'|s,a)[\mathcal{R}(s,a) + \gamma v^\pi(s')]. \end{aligned}$$

G is used to denote $[\mathcal{R}(s,a) + \gamma v^\pi(s')]$ hereafter. Re-arranging terms in the Bellman equation,

$$\begin{aligned} v^\pi(s) &= \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \pi(a|s) \frac{P(s',s,a)}{P(s,a)} G \\ &= \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \pi(a|s) \frac{P(s',s,a)}{\pi(a|s)P(s)} G \\ &= \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \frac{P(s',s,a)}{P(s)} G \\ &= \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \frac{P(a|s,s')P(s,s')}{P(s)} G. \end{aligned}$$

Using the law of total probability, we introduce a new variable e such that:[3]

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \int_e \frac{P(a,e|s,s')P(s,s')}{P(s)} G\,\mathrm{d}e.$$

After multiplying and dividing by $P(e|s)$, we have:

$$\begin{aligned} v^\pi(s) &= \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \int_e P(e|s) \frac{P(a,e|s,s')P(s,s')}{P(e|s)P(s)} G\,\mathrm{d}e \\ &= \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} \frac{P(a,e|s,s')P(s,s')}{P(s,e)} G\,\mathrm{d}e \\ &= \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} \frac{P(a,e,s,s')}{P(s,e)} G\,\mathrm{d}e \\ &= \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} P(s',a|s,e) G\,\mathrm{d}e \\ &= \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} P(s'|s,a,e) P(a|s,e) G\,\mathrm{d}e. \end{aligned}$$

Since the transition to the next state, S_{t+1}, is conditionally independent of E_t, given the previous state, S_t, and the action taken, A_t,

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} P(s'|s,a) P(a|s,e) G\,\mathrm{d}e.$$

[3] Note that a and e are from a joint distribution over a discrete and a continuous random variable. For simplicity, we avoid measure-theoretic notations to represent its joint probability.

Similarly, using the Markov property, action A_t is conditionally independent of S_t given E_t,

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \int_e P(e|s) \sum_{s'\in\mathcal{S}} P(s'|s,a)P(a|e)G \,\mathrm{d}e.$$

As $P(a|e)$ evaluates to 1 for representations, e, that map to a and 0 for others (Assumption A2),

$$\begin{aligned} v^\pi(s) &= \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} P(e|s) \sum_{s'\in\mathcal{S}} P(s'|s,a)G \,\mathrm{d}e \\ &= \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} P(e|s)\; q^\pi(s,a) \,\mathrm{d}e. \end{aligned} \tag{8}$$

In (8), note that the probability density, $P(e|s)$ is the internal policy, $\pi_i(e|s)$. Therefore,

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s)\; q^\pi(s,a) \,\mathrm{d}e.$$

□

B Proof of Lemma 2

Lemma 2. *For all deterministic functions, f, which map each point, $e \in \mathbb{R}^d$, in the representation space to an action, $a \in \mathcal{A}$, the expected updates to θ based on $\frac{\partial J_i(\theta)}{\partial \theta}$ are equivalent to updates based on $\frac{\partial J_o(\theta,f)}{\partial \theta}$. That is,*

$$\frac{\partial J_o(\theta,f)}{\partial \theta} = \frac{\partial J_i(\theta)}{\partial \theta}.$$

Proof. Recall from (1) that the probability of an action given by the overall policy, π_o, is

$$\pi_o(a|s) \coloneqq \int_{f^{-1}(a)} \pi_i(e|s) \,\mathrm{d}e.$$

Using Lemma 1, we express the performance function of the overall policy, π_o, as:

$$\begin{aligned} J_o(\theta,f) &= \sum_{s\in\mathcal{S}} d_0(s) v^{\pi_o}(s) \\ &= \sum_{s\in\mathcal{S}} d_0(s) \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s) q^{\pi_o}(s,a) \mathrm{d}e. \end{aligned}$$

The gradient of the performance function is therefore

$$\frac{\partial J_o(\theta,f)}{\partial \theta} = \frac{\partial}{\partial \theta} \left[\sum_{s\in\mathcal{S}} d_0(s) \sum_{a\in\mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|s) q^{\pi_o}(s,a) \mathrm{d}e \right].$$

Using the policy gradient theorem [Sutton et al., 2000] for the overall policy, π_o, the partial derivative of $J_o(\theta,f)$ w.r.t. θ is,

$$\begin{aligned} \frac{\partial J_o(\theta,f)}{\partial \theta} &= \sum_{t=0}^{\infty} \mathbf{E} \left[\sum_{a\in\mathcal{A}} \gamma^t q^{\pi_o}(S_t,a) \frac{\partial}{\partial \theta} \left(\int_{f^{-1}(a)} \pi_i(e|S_t) \mathrm{d}e \right) \right] \\ &= \sum_{t=0}^{\infty} \mathbf{E} \left[\sum_{a\in\mathcal{A}} \gamma^t \int_{f^{-1}(a)} \frac{\partial}{\partial \theta} \left(\pi_i(e|S_t) \right) q^{\pi_o}(S_t,a) \mathrm{d}e \right] \\ &= \sum_{t=0}^{\infty} \mathbf{E} \left[\sum_{a\in\mathcal{A}} \gamma^t \int_{f^{-1}(a)} \pi_i(e|S_t) \frac{\partial}{\partial \theta} \ln \left(\pi_i(e|S_t) \right) q^{\pi_o}(S_t,a) \mathrm{d}e \right]. \end{aligned}$$

Note that since e deterministically maps to a, $q^{\pi_o}(S_t, a) = q^{\pi_i}(S_t, e)$. Therefore,

$$\frac{\partial J_o(\theta, f)}{\partial \theta} = \sum_{t=0}^{\infty} \mathbf{E}\left[\gamma^t \sum_{a \in \mathcal{A}} \int_{f^{-1}(a)} \pi_i(e|S_t) \frac{\partial}{\partial \theta} \ln\left(\pi_i(e|S_t)\right) q^{\pi_i}(S_t, e) \mathrm{d}e\right].$$

Finally, since each e is mapped to a unique action by the function f, the nested summation over a and its inner integral over $f^{-1}(a)$ can be replaced by an integral over the entire domain of e. Hence,

$$\begin{aligned}
\frac{\partial J_o(\theta, f)}{\partial \theta} &= \sum_{t=0}^{\infty} \mathbf{E}\left[\gamma^t \int_e \pi_i(e|S_t) \frac{\partial}{\partial \theta} \ln\left(\pi_i(e|S_t)\right) q^{\pi_i}(S_t, e) \mathrm{d}e\right] \\
&= \sum_{t=0}^{\infty} \mathbf{E}\left[\gamma^t \int_e q^{\pi_i}(S_t, e) \frac{\partial}{\partial \theta} \pi_i(e|S_t) \, \mathrm{d}e\right] \\
&= \frac{\partial J_i(\theta)}{\partial \theta}.
\end{aligned}$$

□

C Convergence of PG-RA

To analyze the convergence of PG-RA, we first briefly review existing two-timescale convergence results for actor-critics. Afterwards, we present a general setup for stochastic recursions of three dependent parameter sequences. Asymptotic behavior of the system is then discussed using three different timescales, by adapting existing multi-timescale results by Borkar [2009]. This lays the foundation for our subsequent convergence proof. Finally, we prove convergence of the PG-RA method, which extends standard actor-critic algorithms using a new action prediction module, using a three-timescale approach. This technique for the proof is not a novel contribution of the work. We leverage and extend the existing convergence results of actor-critic algorithms [Borkar and Konda, 1997] for our algorithm.

C.1 Actor-Critic Convergence Using Two-Timescales

In the actor-critic algorithms, the updates to the policy depends upon a critic that can estimate the value function associated with the policy at that particular instance. One way to get a good value function is to fix the policy temporarily and update the critic in an inner-loop that uses the transitions drawn using only that fixed policy. While this is a sound approach, it requires a possibly large time between successive updates to the policy parameters and is severely sample-inefficient. Two-timescale stochastic approximation methods [Bhatnagar et al., 2009, Konda and Tsitsiklis, 2000] circumvent this difficulty. The faster update recursion for the critic ensures that asymptotically it is always a close approximation to the required value function before the next update to the policy is made.

C.2 Three-Timescale Setup

In our proposed algorithm, to update the action prediction module, one could have also considered an inner loop that uses transitions drawn using the fixed policy for supervised updates. Instead, to make such a procedure converge faster, we extend the existing two-timescale actor-critic results and take a three-timescale approach.

Consider the following system of stochastic ordinary differential equations (ODE):

$$X_{t+1} = X_t + \alpha_t^x (F_x(X_t, Y_t, Z_t) + \mathcal{N}_{t+1}^1), \tag{9}$$

$$Y_{t+1} = Y_t + \alpha_t^y (F_y(Y_t, Z_t) + \mathcal{N}_{t+1}^2), \tag{10}$$

$$Z_{t+1} = Z_t + \alpha_t^z (F_z(X_t, Y_t, Z_t) + \mathcal{N}_{t+1}^3), \tag{11}$$

where, F_x, F_y and F_g are Lipschitz continuous functions and $\{\mathcal{N}_t^1\}, \{\mathcal{N}_t^2\}, \{\mathcal{N}_t^3\}$ are the associated martingale difference sequences for noise w.r.t. the increasing σ-fields $\mathcal{F}_t = \sigma(X_n, Y_n, Z_n, \mathcal{N}_n^1, \mathcal{N}_n^2, \mathcal{N}_n^3, n \leq t), t \geq 0$, satisfying

$$\mathbf{E}[||N_{t+1}^i||^2 | \mathcal{F}_t] \leq D_1(1 + ||X_t||^2 + ||Y_t||^2 + ||Z_t||^2),$$

for $i = 1, 2, 3$, $t \geq 0$ and any constant $D < \infty$ such that the quadratic variation of noise is always bounded. To study the asymptotic behavior of the system, consider the following standard assumptions,

Assumption B1 (Boundedness). $\sup_t (||X_t|| + ||Y_t|| + ||Z_t||) < \infty$, *almost surely.*

Assumption B2 (Learning rate schedule). *The learning rates α_t^x, α_t^y and α_t^z satisfy:*

$$\sum_t \alpha_t^x = \infty, \sum_t \alpha_t^y = \infty, \sum_t \alpha_t^z = \infty,$$
$$\sum_t (\alpha_t^x)^2 < \infty, \sum_t (\alpha_t^y)^2 < \infty, \sum_t (\alpha_t^z)^2 < \infty,$$
$$As \;\; t \to \infty, \quad \frac{\alpha_t^z}{\alpha_t^y} \to 0, \; \frac{\alpha_t^y}{\alpha_t^x} \to 0. \tag{12}$$

Assumption B3 (Existence of stationary point for Y). *The following ODE has a globally asymptotically stable equilibrium $\mu_1(Z)$, where $\mu_1(\cdot)$ is a Lipschitz continuous function.*

$$\dot{Y} = F_y(Y(t), Z) \tag{13}$$

Assumption B4 (Existence of stationary point for X). *The following ODE has a globally asymptotically stable equilibrium $\mu_2(Y, Z)$, where $\mu_2(\cdot, \cdot)$ is a Lipschitz continuous function.*

$$\dot{X} = F_x(X(t), Y, Z), \tag{14}$$

Assumption B5 (Existence of stationary point for Z). *The following ODE has a globally asymptotically stable equilibrium $Z^\star$,*

$$\dot{Z} = F_z(\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t)). \tag{15}$$

Assumptions B1–B2 are required to bound the values of the parameter sequence and make the learning rate well-conditioned, respectively. Assumptions B3-B4 ensure that there exists a global stationary point for the respective recursions, individually, when other parameters are held constant. Finally, Assumption B5 ensures that there exists a global stationary point for the update recursion associated with Z, if between each successive update to Z, X and Y have converged to their respective stationary points.

Lemma 3. *Under Assumptions B1-B5, $(X_t, Y_t, Z_t) \to (\mu_2(\mu_1(Z^\star), Z^\star), \mu_1(Z^\star), Z^\star)$ as $t \to \infty$, with probability one.*

Proof. We adapt the multi-timescale analysis by Borkar [2009] to analyze the above system of equations using three-timescales. First we present an intuitive explanation and then we formalize the results.

Since these three updates are not independent at each time step, we consider three step-size schedules: $\{\alpha_t^x\}$, $\{\alpha_t^y\}$ and $\{\alpha_t^z\}$, which satisfy Assumption B2. As a consequence of (12), the recursion (10) is 'faster' than (11), and (9) is 'faster' than both (10) and (11). In other words, Z moves on the slowest timescale and the X moves on the fastest. Such a timescale is desirable since Z_t converges to its stationary point if at each time step the value of the corresponding converged X and Y estimates are used to make the next Z update (Assumption B5).

To elaborate on the previous points, first consider the ODEs:

$$\dot{Y} = F_y(Y(t), Z(t)), \tag{16}$$
$$\dot{Z} = 0. \tag{17}$$

Alternatively, one can consider the ODE

$$\dot{Y} = F_y(Y(t), Z),$$

in place of (16), because Z is fixed (17). Now, under Assumption B3 we know that the iterative update (10) performed on Y, with a fixed Z, will eventually converge to a corresponding stationary point.

Now, with this converged Y, consider the following ODEs:

$$\dot{X} = F_x(X(t), Y(t), Z(t)), \tag{18}$$

$$\dot{Y} = 0, \tag{19}$$

$$\dot{Z} = 0. \tag{20}$$

Alternatively, one can consider the ODE

$$\dot{X} = F_x(X(t), Y, Z),$$

in place of (18), as Y and Z are fixed (19)-(20). As a consequence of Assumption B4, X converges when both Y and Z are held fixed.

Intuitively, as a result of Assumption B2, in the limit, the learning-rate, α_t^z becomes very small relative to α_t^y. This makes Z 'quasi-static' compared to Y and has an effect similar to fixing Z_t and running the iteration (10) forever to converge at $\mu_1(Z_t)$. Similarly, both α_t^y and α_t^z become very small relative to α_t^x. Therefore, both Y and Z are 'quasi-static' compared to the critic, which has an effect similar to fixing Y_t and Z_t, and running the iteration (9) forever. In turn, this makes Z_t see X_t as a close approximation to $\mu_2(\mu_1(Z(t)), Z(t))$ always, and thus Z_t converges to $Z^\star$ due to Assumption B5.

Formally, define three real-valued sequences $\{i_t\}, \{j_t\}$ and $\{k_t\}$ as $i_t = \sum_{n=0}^{t-1} \alpha_t^y, j_t = \sum_{n=0}^{t-1} \alpha_t^x$ and $k_t = \sum_{n=0}^{t-1} \alpha_t^z$, respectively. These are required for tracking the continuous time ODEs, in the limit, using discretized time. Note that $(i_t - i_{t-1}), (j_t - j_{t-1}), (k_t - k_{t-1})$ almost surely converge to 0 as $t \to \infty$.

Define continuous time processes $\bar{Y}(i), \bar{Z}(i), i \geq 0$ as $\bar{Y}(i_t) = Y_t, \bar{Z}(i_t) = Z_t$, respectively with linear interpolations in between. For $s \geq 0$, let $Y^s(i), Z^s(i), i \geq s$ denote the trajectories of (16)–(17) with $Y^s(s) = \bar{Y}(s)$ and $Z^s(s) = \bar{Z}(s)$. Note that because of (17), $\forall_i \geq s$ $Z^s(i) = \bar{Z}(s)$. Now consider re-writing (10)–(11) as,

$$\begin{aligned}
Y_{t+1} &= Y_t + \alpha_t^y (F_y(Y_t, Z_t) + \mathcal{N}_{t+1}^2), \\
Z_{t+1} &= Z_t + \alpha_t^y \left(\frac{\alpha_t^z}{\alpha_t^y} (F_z(X_t, Y_t, Z_t) + \mathcal{N}_{t+1}^3) \right).
\end{aligned}$$

When the time discretization corresponds to $\{i_t\}$, this shows that (10)–(11) can be seen as 'noisy' Euler discretizations of the ODE (13) (or, equivalently of ODEs (16)–(17)), but as $\dot{Z} = 0$ this ODE has an approximation error of $\frac{\alpha_t^z}{\alpha_t^y}(F_z(X_t, Y_t, Z_t) + \mathcal{N}_{t+1}^3)$. However, asymptotically, this error vanishes as $\frac{\alpha_t^z}{\alpha_t^y} \to 0$. Now using results by Borkar [2009], it can be shown that, for any given $T \geq 0$, as $s \to \infty$,

$$\begin{aligned}
\sup_{i \in [s, s+T]} ||\bar{Y}(i) - Y^s(i)|| &\to 0, \\
\sup_{i \in [s, s+T]} ||\bar{Z}(i) - Z^s(i)|| &\to 0,
\end{aligned}$$

with probability one. Hence, in the limit, the discretization error also vanishes and $(Y(t), Z(t)) \to (\mu_1(Z(t)), Z(t))$. Similarly for (9)–(11), with $\{j_t\}$ as time discretization, and using the fact that both $\frac{\alpha_t^z}{\alpha_t^y} \to 0$, and $\frac{\alpha_t^y}{\alpha_t^x} \to 0$, a noisy Euler discretization can be obtained for ODE (14) (or equivalently for ODEs (18)–(20)). Hence, in the limit, $(X(t), Y(t), Z(t)) \to (\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t))$.

Now consider re-writing (11) as:

$$\begin{aligned}
Z_{t+1} = Z_t & \\
&+ \alpha_t^z (F_z(\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t))) \\
&- \alpha_t^z (F_z(\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t))) \\
&+ \alpha_t^z (F_z(X_t, Y_t, Z_t)) \\
&+ \alpha_t^z (\mathcal{N}_{t+1}^3).
\end{aligned} \tag{21}$$

This can be seen as a noisy Euler discretization of the ODE (15), along the time-line $\{k_t\}$, with the error corresponding to the third, fourth and fifth terms on the RHS of (21). We denote these

error terms as I, II and III, respectively. In the limit, using the result that $(X(t), Y(t), Z(t)) \rightarrow (\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t))$ as $t \rightarrow \infty$, the error $I + II$ vanishes. Similarly, martingale noise error, III, vanishes asymptotically as a consequence of bounded Z values and $\sum_t (\alpha_t^z)^2 < \infty$. Now using sequence of approximations using Gronwall's inequality, it can be shown that (21) converges to Z^* asymptotically [Borkar, 2009].

Therefore, under the Assumptions B1-B5, $(X_t, Y_t, Z_t) \rightarrow (\mu_2(\mu_1(Z^\star), Z^\star), \mu_1(Z^\star), Z^\star)$ as $t \rightarrow \infty$. □

C.3 PG-RA Convergence Using Three- Timescales:

Let the parameters of the critic and the internal policy be denoted as ω and θ respectively. Also, let ϕ denote all the parameters of $\hat{f}$ and $\hat{g}$. Similar to prior work [Bhatnagar et al., 2009, Degris et al., 2012, Konda and Tsitsiklis, 2000], for analysis of the updates to the parameters, we consider the following standard assumptions required to ensure existence of gradients and bound the parameter ranges.

Assumption A3. *For any state action-representation pair (s,e), internal policy, $\pi_i(e|s)$, is continuously differentiable in the parameter θ.*

Assumption A4. *The updates to the parameters, $\theta \in \mathbb{R}^{d_\theta}$, of the internal policy, π_i, includes a projection operator $\Gamma : \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_\theta}$ that projects any $x \in \mathbb{R}^{d_\theta}$ to a compact set $\mathcal{C} = \{x | c_i(x) \leq 0, i = 1, ..., n\} \subset \mathbb{R}^{d_\theta}$, where $c_i(\cdot), i = 1, ..., n$ are real-valued, continuously differentiable functions on $\mathbb{R}^{d_\theta}$ that represents the constraints specifying the compact region. For each x on the boundary of $\mathcal{C}$, the gradients of the active c_i are considered to be linearly independent.*

Assumption A5. *The iterates ω_t and ϕ_t satisfy $\sup_t (||\omega_t||) < \infty$ and $\sup_t (||\phi_t||) < \infty$.*

Let $v(\cdot)$ be the gradient vector field on $\mathcal{C}$. We define another vector field operator $\hat{\Gamma}$,

$$\hat{\Gamma}(v(\theta)) \coloneqq \lim_{h \rightarrow 0} \frac{\Gamma(\theta + hv(\theta)) - \theta}{h},$$

that projects any gradients leading outside the compact region, $\mathcal{C}$, back to $\mathcal{C}$.

Theorem 2. *Under Assumptions (A1)-(A5), the internal policy parameters: θ_t converge to $\hat{\mathcal{Z}} = \left\{ x \in \mathcal{C} | \hat{\Gamma} \left(\frac{\partial J_i(x)}{\partial \theta} \right) = 0 \right\}$ as $t \rightarrow \infty$, with probability one.*

Proof. PG-RA algorithm considers the following stochastic update recursions for the critic, action representation modules, and the internal policy, respectively:

$$\begin{aligned}
\omega_{t+1} =& \omega_t + \alpha_t^\omega \delta_t \frac{\partial v(s)}{\partial \omega} \\
\phi_{t+1} =& \phi_t + \alpha_t^\phi \frac{-\partial \log \hat{P}(a|s, s')}{\partial \phi} \\
\theta_{t+1} =& \theta_t + \alpha_t^\theta \hat{\Gamma} \left(\delta_t \frac{\partial \log \pi_i(e|s)}{\partial \theta} \right),
\end{aligned}$$

where, δ_t is the TD-error and is given by:

$$\delta_t = r + \gamma v(s') - v(s).$$

We now establish how these updates can be mapped to the three ODEs (9)–(11) satisfying Assumptions (B1)–(B5), so as to leverage the result from Lemma 3. To do so, we must consider how the recursions are dependent on each other. Since the reward observed is a consequence of the action executed using the internal policy and the action representation module, it makes δ dependent on both ϕ and θ. Due to the use of bootstrapping and a baseline, δ is also dependent on ω. As a result, the updates to both ω and θ are dependent on all three sets of parameters. In contrast, notice that the updates to the action representation module is independent of the rewards/critic and is thus dependent only on θ and ϕ. Therefore, the ODEs that govern the update recursions for PG-RA parameters are of the form (9)–(11), where (ω, ϕ, θ) correspond directly to (X, Y, Z). The functions F_x, F_y and F_z in (9)–(11) correspond to the semi-gradients of TD-error, gradients of self-supervised loss, and

policy gradients, respectively. Lemma 3 can now be leveraged if the associated assumptions are also satisfied by our PG-RA algorithm.

For requirement B1, as a result of the projection operator Γ, the internal policy parameters, θ, remain bounded. Further, by assumption, ω and ϕ always remain bounded as well. Therefore, we have that $\sup_t (||\omega_t|| + ||\phi_t|| + ||\theta_t||) < \infty$.

For requirement B2, the learning rates $\alpha_t^\omega, \alpha_t^\phi$ and α_t^θ are hyper-parameters and can be set such that as $t \to \infty$,

$$\frac{\alpha_t^\theta}{\alpha_t^\phi} \to 0, \frac{\alpha_t^\phi}{\alpha_t^\omega} \to 0,$$

to meet the three-timescale requirement in Assumption B2.

For requirement B3, recall that when the internal policy has fixed parameters, θ, the updates to the action representation component follows a supervised learning procedure. For linear parameterization of estimators $\hat{f}$ and $\hat{g}$, the action prediction module is equivalent to a bi-linear neural network. Multiple works have established that for such models, there are no spurious local minimas and the Hessian at every saddle point has at least one negative eigenvalue [Kawaguchi, 2016, Haeffele and Vidal, 2017, Zhu et al., 2018]. Further, the global minima can be achieved by stochastic gradient descent. This ensures convergence to the required critical point and satisfies Assumption B3.

For requirement B4, given a fixed policy (fixed action representations and fixed internal policy) the proof of convergence of a linear critic to the stationary point $\mu_2(\phi, \theta)$ using TD(λ) is a well established result [Tsitsiklis and Van Roy, 1996]. We use $\lambda = 0$ in our algorithm, the proof however carries through for $\lambda > 0$ as well. This satisfies Assumption B4.

For requirement B5, the condition can be relaxed to a local rather than global asymptotically stable fixed point, because we only need convergence. Under the optimal critic and action representations for every step, the internal policy follows its internal policy gradient. Using Lemma 2, we established that this is equivalent to following the policy gradient of the overall policy and thus the internal policy converges to its local fixed point as well.

This completes the necessary requirements, the remaining proof now follows from Lemma 3. □

D Implementation Details

D.1 Parameterization

In our experiments, we consider a parameterization that minimizes the computational complexity of the algorithm. Learning the parameters of the action representation module, as in (5), requires computing the value $\hat{P}(a|s, s')$ in (3). This involves a complete integral over e. Due to the absence of any closed form solution, we need to rely on a stochastic estimate. Depending on the dimensions of e, an extensive sample based evaluation of this expectation can be computationally expensive. To make this more tractable, we approximate (3) by mean-marginalizing it using the estimate of the mean from $\hat{g}$. That is, we approximate (3) as $\hat{f}(a|\hat{g}(s, s'))$. We then parameterize $\hat{f}(a|\hat{g}(s, s'))$ as,

$$\hat{f}(a|\hat{g}(s, s')) = \frac{\mathbf{e}^{z_a/\tau}}{\sum_{a'} \mathbf{e}^{z_{a'}/\tau}},$$

where,

$$z_a = W_a^\top \hat{g}(s, s'). \tag{22}$$

This estimator, $\hat{f}$, models the probability of any action, a, based on its similarity with a given representation e. In (22), $W \in \mathbb{R}^{d_e \times |\mathcal{A}|}$ is a matrix where each column represents a learnable action representation of dimension $\mathbb{R}^{d_e}$. $W_a^\top$ is the transpose of the vector corresponding to the representation of the action a, and z_a is its measure of similarity with the embedding from $\hat{g}(s, s')$. To get valid probability values, a Boltzmann distribution is used with τ as a temperature variable. In the limit when $\tau \to 0$ the conditional distribution over actions becomes the required deterministic estimate for $\hat{f}$. That is, the entire probability mass would be on the action, a, which has the most similar representation to e. To ensure empirical stability during training, we relax τ to 1. During execution, the action, a, which has the most similar representation to e, is chosen for execution. In

practice, the linear decomposition in (22) is not restrictive as $\hat{g}$ can still be any differentiable function approximator, like a neural network.

D.2 Hyper-parameters

For the maze domain, single layer neural networks were used to parameterize both the actor and critic, and the learning rates were searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$. State features were represented using the 3rd order coupled Fourier basis [Konidaris et al., 2011]. The discounting parameter γ was set to 0.99 and λ to 0.9. Since it was a toy domain, the dimensions of action representations were fixed to 2. 2000 randomly drawn trajectories were used to learn an initial representation for the actions. Action representations were only trained once in the beginning and kept fixed from there on.

For Adobe HelpX and Photoshop environments, 2 layer neural networks were used to parameterize both the actor and critic, and the learning rates were searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$. Similar to prior work, the module for encoding state features was shared to reduce the number of parameters, and the learning rate for it was additionally searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$. The dimension of the neural network's hidden layer was searched over $\{64, 128, 256\}$. The discounting parameter γ was set to 0.9. For actor-critic based results λ was set to 0.9 and for DPG the target actor and policy update rate was fixed to its default setting of 0.001. The dimension of action representations were searched over $\{16, 32, 64\}$. Initial $10{,}000$ randomly drawn trajectories were used to learn an initial representation for the actions. The action prediction component was continuously improved on the fly, as given in PG-RA algorithm.

For all the results of the PG-RA based algorithms, since π_i was defined over a continuous space, it was parameterized as the isotropic normal distribution. The value for variance was searched over $\{0.1, 0.25, 1, -1\}$, where -1 represents learned variance. Function $\hat{g}$ was parameterized to concatenate the state features of both s and s' and project to the embedding space using a single layer neural network with *Tanh* non-linearity. To keep the effective range of action representations bounded, they were also transformed by *Tanh* non-linearity before computing the similarity scores. Though the similarity metric is naturally based on the dot product, other distance metrics are also valid. We found squared Euclidean distance to work best in our experiments. The learning rates for functions $\hat{f}$ and $\hat{g}$ were jointly searched over $\{1e-2, 1e-3, 1e-4\}$. All the results were obtained for 3 different seeds to get the variance. The architecture and hyper-parameter search for the baselines were done in the same way.