Learning Action Representations for Reinforcement Learning

Yash Chandak [1] **Georgios Theocharous** [2] **James E. Kostas** [1] **Scott M. Jordan** [1] **Philip S. Thomas** [1]

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.

[1]University of Massachusetts, Amherst, USA. [2]Adobe Research, San Jose, USA.. Correspondence to: Yash Chandak <ychandak@cs.umass.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

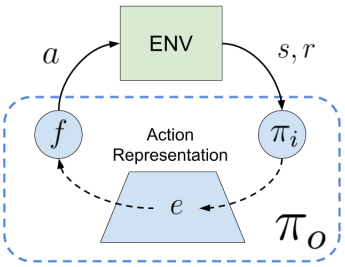

Figure 1. The structure of the proposed overall policy, π_o, consisting of f and π_i, that learns action representations to generalize over large action sets.

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 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 & 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 & Konda, 1997).

To evaluate our proposed method empirically, we study two real-world recommendation problems using data from widely used commercial applications. In both 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, or which tool to suggest to the user next in multi-media editing 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 & Konda, 1997).
- Experimental results on real-world domains with thousands of actions using actual data collected from recommender systems.

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 $\forall s, a, s', t$, $\mathcal{P}(s, a, s') := P(s'|s, a)$.

A policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a conditional distribution over actions for each state: $\pi(a, s) := P(A_t = a | S_t = s)$ 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 $\pi^* \in \operatorname{argmax}_{\pi \in \Pi} \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R_t | \pi]$, 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 & 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 & Hinton, 2004). For planning problems, Cui & 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 & 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*. Recent work also showed how action representations can be learned using data from expert demonstrations (Tennenholtz & Mannor, 2019). 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; François-Lavet et al., 2018) 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 & Grill, 2004; Mussa-Ivaldi & 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 & 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 & 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 & 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:

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

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 $f^{-1}(a)$ as a one-to-many function that denotes the set of representations that are mapped to the action a by the function f, i.e., $f^{-1}(a) := \{e \in \mathcal{E} : f(e) = a\}$.

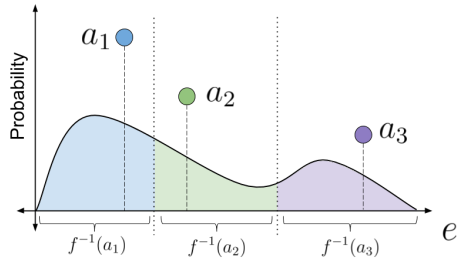

Figure 2. Illustration of the probability induced for three actions by the probability density of $\pi_i(e, s)$ on a 1-D embedding space. The x-axis represents the embedding, e, and the y-axis represents the probability. The colored regions represent the mapping $a = f(e)$, where each color is associated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

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 $P(A_t = a|E_t = e) = 1$.*

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

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

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 $\Pi_o^\star := \{\pi_o : v^{\pi_o} = v^\star\}$. Correspondingly, we define the set of *optimal internal policies* as $\Pi_i^\star := \{\pi_i : \exists \pi_o^\star \in \Pi_o^\star, \exists f, \pi_o^\star(a|s) = \int_{f^{-1}(a)} \pi_i(e|s)\,\mathrm{d}e\}$.

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 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

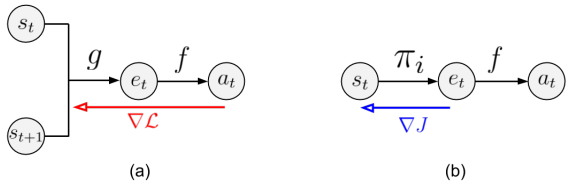

Figure 3. (a) Given a state transition tuple, functions g and f are used to estimate the action taken. The red arrow denotes the gradients of the supervised loss (5) for learning the parameters of these functions. (b) During execution, an internal policy, π_i, can be used to first select an action representation, 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.

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.

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

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

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.

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 & 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 & 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 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

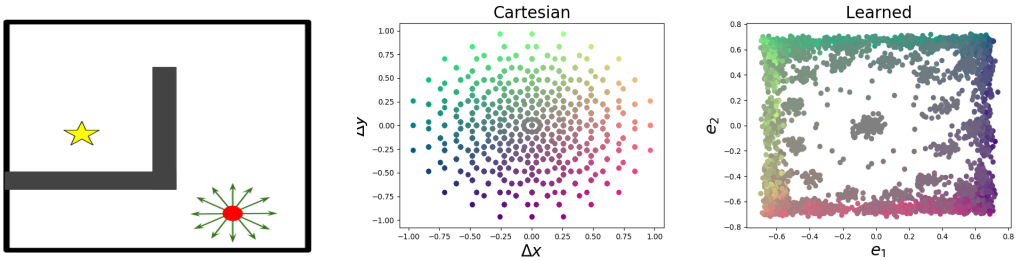

Figure 4. (a) The maze environment. The star denotes the goal state, the red dot corresponds to the agent and the arrows around it are the 12 actuators. Each action corresponds to a unique combination of these actuators. Therefore, in total 2^{12} actions are possible. (b) 2-D representations for the 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.

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, a web-based video-tutorial platform, which has a recommendation engine that suggests a series of tutorial videos on various software. The aim is to meaningfully engage the users in learning how to use these software 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 software.

The second application is 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 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 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 multi-media editing 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 web-based tutorial platform and the multi-media software, respectively, were used to create the action set for the MDP model. The MDP had continuous state-space, where each state consisted 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 and popularity of final outcomes of user interactions in the multi-media editing software, 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 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

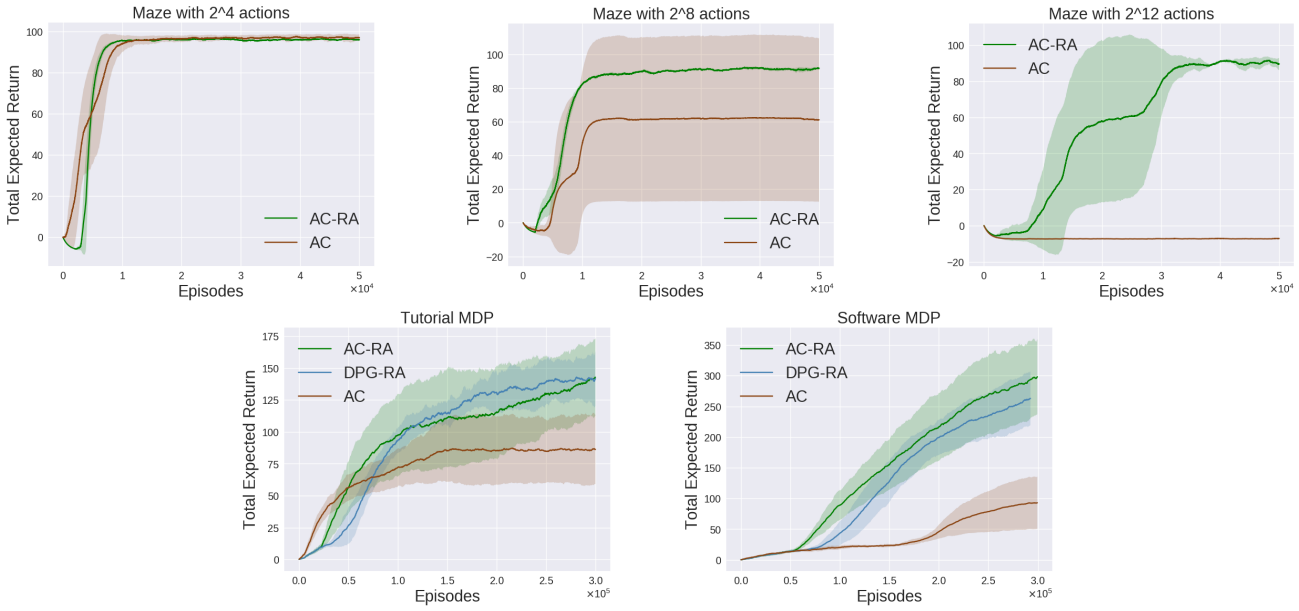

Figure 5. (Top) Results on the Maze domain with 2^4, 2^8, and 2^{12} actions respectively. (Bottom) Results on a) Tutorial MDP b) Software MDP. AC-RA and DPG-RA are the variants of PG-RA algorithm that uses actor-critic (AC) and DPG, respectively. The shaded regions correspond to one standard deviation and were obtained using 10 trials.

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 the tutorial and the software 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 $3\times$ 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. Conclusion

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.

Acknowledgement

The research was supported by and partially conducted at Adobe Research. We thank our colleagues from the Autonomous Learning Lab (ALL) for the valuable discussions that greatly assisted the research. We also thank Sridhar Mahadevan for his valuable comments and feedback on the earlier versions of the manuscript. We are also immensely grateful to the four anonymous reviewers who shared their insights to improve the paper and make the results stronger.

References

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

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

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

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

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

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

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

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

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

François-Lavet, V., Bengio, Y., Precup, D., and Pineau, J. Combined reinforcement learning via abstract representations. *arXiv preprint arXiv:1809.04506*, 2018.

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

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

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

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

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Shani, G., Heckerman, D., and Brafman, R. I. An MDP-based recommender system. *Journal of Machine Learning Research*, 2005.

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

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

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

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

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

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Tennenholtz, G. and Mannor, S. The natural language of actions. *International Conference on Machine Learning*, 2019.

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

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

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

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

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

Tsitsiklis, J. and Van Roy, B. 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.

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

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

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