Reinforcement Learning with a Dynamic Action Set

Anonymous

Abstract

Reinforcement learning has been successfully applied to many sequential decision making problems, where the set of possible actions (possible decisions) is fixed. However, in many real-world settings, the set of possible actions can change over time. We present a model-free method to continually adapt to a dynamic set of possible actions. We show how a policy can be decomposed into an *internal policy* that acts in a space of action representations and a *reward-independent* component that transforms these representations into actual actions. These representations not only make the internal policy parameterization invariant to the cardinality of the action set, but also improve generalization by allowing the agent to infer the outcomes of actions similar to actions already taken. We provide an algorithm to autonomously adapt to this dynamic action set by exploiting structure in the space of actions using supervised learning while learning the internal policy using policy gradient. The efficacy of the proposed method is demonstrated on large-scale real-world continual learning problems.

1 Introduction

Real-world problems are often non-stationary. That is, parts of the problem specification change over time. We desire autonomous systems that continually adapt by capturing the regularities in such changes, without the need to learn from scratch after every change. In this work, we address one form of *continual learning* (CL) [Ring, 1994] for sequential decision making problems, wherein the set of possible actions (decisions) varies over time. Such a situation is omnipresent in many real-world problems. For example, in robotics it is natural to keep adding more control components over the lifetime of a robot to enhance its interaction with the environment. In recommender systems, new products are added to the stock and some are removed periodically. In email marketing, promotional emails are based on the current offers and outdated ones are removed from the system. In drug prescription, new medications are continually discovered and earlier ones are retired.

Reinforcement learning (RL) has emerged as a successful method for solving sequential decision making problems. However, its application have been limited to settings where the set of actions is fixed. One of the reasons is that the policy parameters are directly dependent on the number of actions available. This makes it hard to adapt the parameters of the policy when the number of actions changes over time. To address this problem, we introduce an abstract space of action representations, and decompose the policy into two parts. The first, an internal policy, acts in the space of action representations, and the second maps these representations to actual actions. A diagram of our approach is provided in Figure 1.

The proposed formulation has two major advantages: a) It makes the internal policy invariant to the cardinality of the set of possible actions, and b) it improves generalization by allowing the agent to infer the outcomes of actions similar to actions already taken. These advantages make our proposed method ideal for continual learning problems where the action set changes dynamically over time, and where quick adaptation to these changes, via generalization of prior knowledge about the impact of actions, is beneficial.

Our primary contributions include:

- A policy parameterization for continual reinforcement learning problems with a dynamic set of possible actions.
- Proof that for all optimal policies, π^*, there exist parameters under this new policy class that are equivalent to π^*.
- An algorithm to learn both components autonomously. We also show that one component can be learned independent of the rewards using supervised learning.
- Proof of convergence of the algorithm.
- Experimental analysis of the proposed method on large-scale real-world continual learning problems.

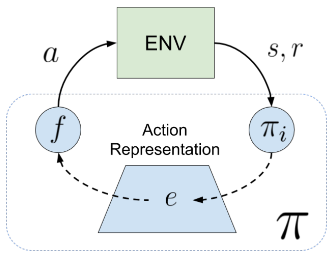

Figure 1: In the proposed method an internal policy, π_i, acts in the space of action representations and a function, f, transforms these representations into actual actions.

2 Action Cardinality Invariant Policies

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 help avoid modeling the policy in a tabular form. For domains with discrete state spaces, they not only make it easier to generalize across unseen states, but also make the policy parameterization independent of the exact number of states in the MDP. In other words, they also make the policy 'invariant' to the cardinality of the state space. Similarly, there 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 continual learning feasible in a dynamically changing set of actions. To bridge the gap, we introduce a space of action representation, $E_t \subseteq \mathbb{R}^{d_e}$, and consider a new class of policy, π_o, parameterized as:

$$\begin{aligned} E_t &\sim \pi_i(S_t, \cdot) \\ A_t &= f(E_t). \end{aligned}$$

Here, π_i represents an *internal policy* that is used to select E_t, and the function f deterministically maps this representation to an action in the set $\mathcal{A}$. This new class of policy parameterization, π_o, which we call an *overall policy* can be used to equivalently represent any optimal policy.

Theorem 1. *Under Assumption 1, which defines a function f, for all optimal policies, $\pi^\star$, there exists an internal policy, $\pi_i^\star$, such that the overall policy, π_o, consisting of $\pi_i^\star$ and f is also optimal.*

With action representations, the internal policy, π_i, is now invariant to the cardinality of the action set. Though the function f is still dependent on cardinality, we show (refer to the Appendix) that only the learning procedure for π_i is dependent on the rewards, and f can just be learned using a supervised learning procedure. Figure 2 illustrates the probability of each action under such a parameterization.

3 Continual learning with a dynamic action set

In contrast to the standard approach that treats each action as a discrete one-hot quantity, there are several advantages of our method that make it suitable for a continual learning setting.

- The internal policy, π_i, does not need to be re-initialized when the action set size is changed as it is invariant to the cardinality of the action set.
- Representation for the actions in the new action set can be adapted quickly as the updates to function f depend on reward-independent supervised learning procedure.
- Our learning procedure makes the actions aligned in the representation space based on their transition correlations. Therefore, updates in this space also provide the agent with additional information about other actions embedded in proximity to the action executed, even if they are not sampled for execution. Thereby, enhancing its generalization ability.

Leveraging these advantages, we propose a new algorithm (Algo 1) to learn an optimal policy over dynamic set of actions. We call this **C**ontinual **L**earning with **R**epresentations for **A**ctions (CL-RA). It has two primary roles: a) Quickly adapt the representations associated with the new actions, whenever there is a change in the current set of actions available. b) Learn the overall policy for this new

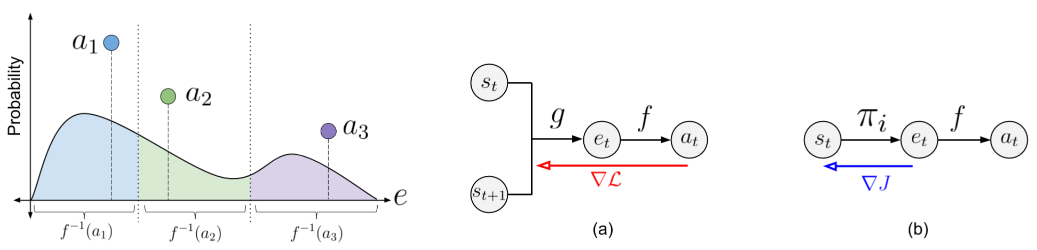

Figure 2: [Left] 1-D illustration for the discrete action's probability (3) induced due to underlying continuous space of representations e. Here, three actions are considered and representations are mapped to the nearest action by a function f. We use $f^{-1}(a)$ to denote the set of representations that are mapped to the action a by the function f. [Right (a)] Given a state transition tuple, functions g and f are used to estimate the action taken. The red arrow denotes the gradients of self-supervised loss (5) for learning these functions and satisfying Assumption 1. [Right (b)] During execution, as a consequence of Theorem 1, an internal policy, π_i, is 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 local policy gradients (7) obtained using Property 1 to learn π_i.

action set. For adaptively initializing the action representations, few random transition samples are drawn and used for self-supervision. At a high level, this process can be seen as learning the inverse dynamics. A function, g, first encodes the transition information between s and s', then a function, f, predicts the action, a, based on the similarity of the encoded information to representations of the available actions. Since the action responsible for the transition is already available, we update the parameters f and g by minimizing the action inference loss (5). This step is abstracted under the function `Self-supervise` (Algo 2). To get the optimal internal policy under the current set of actions, any policy gradient algorithm can be used. This process is abstracted under the function `PG-RA` (Algo 3), which stands for **P**olicy **G**radients with **R**epresentations for **A**ctions.

Let every point after which the set of actions are changed be indexed using $\mathit{Change} \in \{0, 1, 2, \ldots\}$ and let the parameters of critic and the internal policy be denoted as $\hat{V}$ and θ respectively. Also, let all the parameters of f and g be together represented as ϕ. Then the proposed algorithm converges to a locally optimal set of parameters, denoted as $\hat{V}^\star, \phi^\star$ and $\theta^\star$, for solving an MDP. (Detailed justification for the learning process and the proof of convergence is deferred to the Appendix.)

Theorem 2. *Under Assumptions 1-6, PG-RA parameters:* $(\hat{V}_t, \phi_t, \theta_t) \rightarrow (\hat{V}^\star, \phi^\star, \theta^\star)$ *as* $t \rightarrow \infty$*, with probability one.*

Corollary 1. *Under Assumptions 1-6,* $\forall i \in \mathit{Change}$*, CL-RA parameters:* $(\hat{V}_t, \phi_t, \theta_t) \rightarrow (\hat{V}_i^\star, \phi_i^\star, \theta_i^\star)$ *as* $t \rightarrow \infty$*, with probability one.*

Algorithm 1: CL-RA

```
1 Initialize $\pi_i, f, g$
2 for $Change = 0, 1, 2...$ do
3     Add/Remove action representations in/from $f$
4     for $episode = 0, 1, 2...$ do
5         for $t = 0, 1, 2...$ do
6             Execute random $a_t$ and observe $s_{t+1}, r_t$
7             Self-supervise($s_t, a_t, s_{t+1}, f, g$)
8     PG-RA($\pi_i, f, g$)
```

Algorithm 2: Self-supervise

```
1 Input: $s, a, s', f, g$
2 $\bar{P}_a = softmax(f(g(s, s')))$        ▷ (4)
3 Minimize $-\log \bar{P}_a$                  ▷ (5)
```

Algorithm 3: PG-RA

```
1 Input: $\pi_i, f, g$
2 for $episode = 0, 1, 2...$ do
3     for $t = 0, 1, 2...$ do
4         Sample action embedding from $\pi_i(e|s_t)$
5         Map embedding to an action $a_t$ using $f(e)$
6         Execute $a_t$ and observe $s_{t+1}, r_t$
7         Update $\pi_i$, critic using any policy gradient algo
8         Self-supervise($s_t, a_t, s_{t+1}, f, g$)
```

4 Empirical Analysis

To demonstrate the effectiveness of our proposed method on continual learning problems, we consider a maze environment and two real-world applications domains, all with large set of dynamic actions. Total number of actions for each of the domains are randomly split into three equal sets. Initially, the agent only has the actions available in the first set. During each change, 10% of the existing actions are removed and all the actions in the next set are made additionally available to the agent.

Maze: As a didactic example, we constructed a continuous-state maze environment where the state is 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 the agent can choose any possible combination of these actuators. For CL experiments 8 actuators were considered, that is, a total of $|\mathcal{A}| = 2^8$ possible number of actions. The net outcome of an action is the displacement produced by a vectorial summation of the displacements associated with the selected actuators. To make the problem more challenging, random noise was added to the action 10% of the time and maximum episode length was 150 time steps. The agent received a small penalty for each time step and a reward of 100 on reaching the goal position. The goal position was kept the same throughout. The Maze environment is a useful test bed as it requires solving a long horizon task using a high-dimensional dynamically changing action set, in presence of a single goal reward.

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

- A web-based video-tutorial platform, that has a recommendation engine to suggest a series of tutorial videos. The aim is to meaningfully engage the users in a learning activity. In total, 1498 tutorials were considered for recommendation.
- A professional multi-media editing software, where sequences of tools inside the software need to be recommended. The aim is to increase user's productivity and assist them in quickly achieving their end goal. In total, 1843 tools were considered for recommendation.

For both of these applications, the 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]. State for the MDP comprised real-valued feature descriptors associated with each item (tutorial/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 on that 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 contextual bandit or collaborative filtering problem, but as shown in [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 large, dynamically changing action (tutorials/tools) set makes this real-life problem an interesting hurdle for RL algorithms.

5 Results

Though any policy gradient algorithm can be used in our algorithm, we restricted our focus to two standard methods: Actor-Critic (AC) and DPG [Silver et al., 2014]. Instances of CL-RA that use AC and DPG are called AC-RA and DPG-RA, respectively. The plots in Fig 3 illustrates the advantage of the proposed continual learning algorithm over the standard approach. Since the internal policy is invariant to the action cardinality, it can be readily used after every change without having to be re-initialized.

In the maze domain, the knowledge about the optimal policy under the previous available action set is efficiently carried forward. AC-RA almost maintains optimal performance throughout, invariant to the change in the action set. Once the new actions are aligned using self-supervision, AC-RA algorithm immediately learns how to use the best alternative actions to achieve the optimal policy.

In the real-world application domains, standard AC methods fails to reason over longer time horizons under such overwhelming number of actions. Its learning gets stuck, choosing mostly one-step actions that have high returns. In comparison, CL-RA methods perform much better, even within each change window. This can be attributed to the fact that action representations allow the agent to generalize

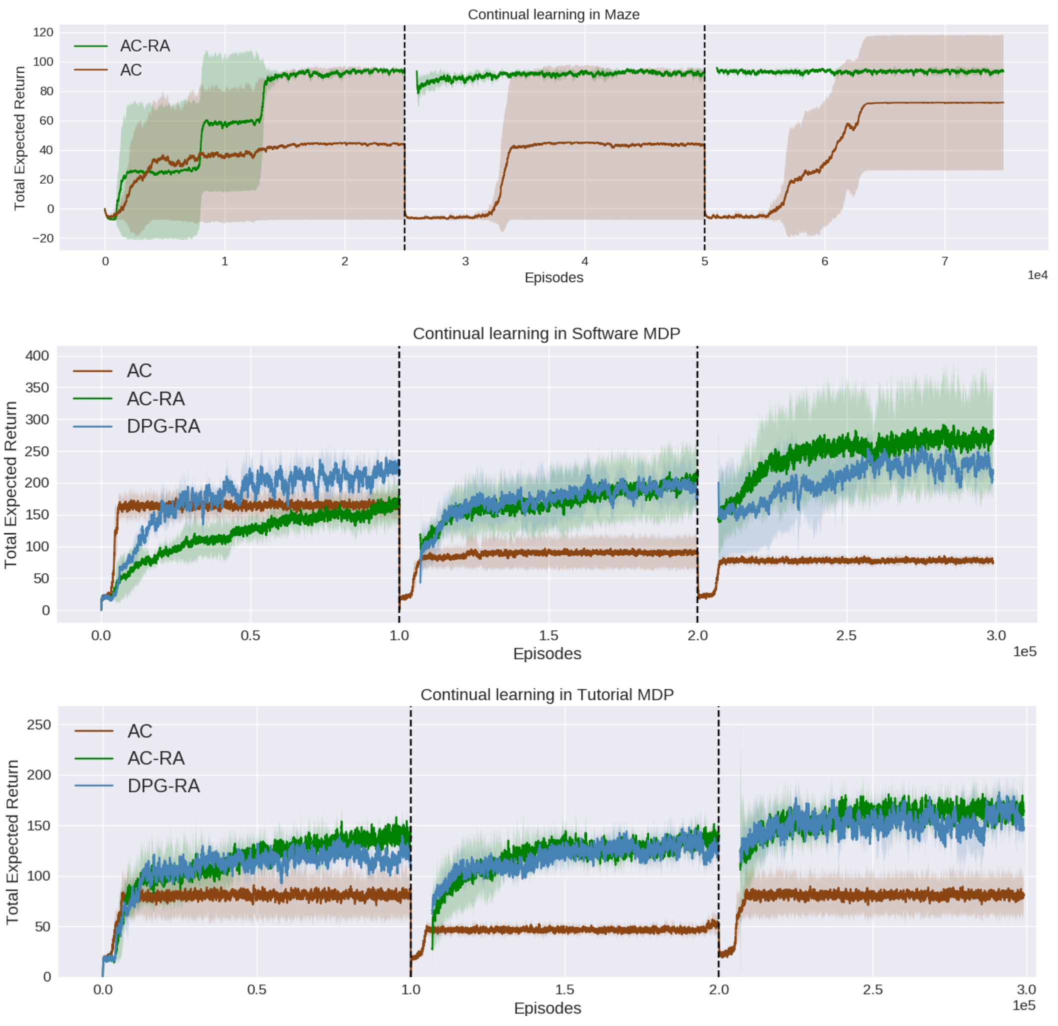

Figure 3: Continual learning experiments with dynamic set of actions in (Top) the maze domain, (Middle) software MDP, and (Bottom) Tutorial MDP. Vertical dotted bars indicate when the set of actions was changed. AC corresponds to the standard ActorCritic method, AC-RA and DPG-RA are instances of CL-RA with ActorCritic and DPG used for policy gradients. The breaks in the plots correspond to randomly drawn episodes for self-supervised adaptation.

the feedback across actions embedded in proximity to the executed action. By leveraging this added generalization capability, CL-RA algorithms can efficiently search even in this space of large number of actions. Since the rewards are associated with the actions in these domains, dynamically changing the possible set of actions impacts the total expected returns achievable. This makes the performance of CL-RA algorithms drop after each change. However, leveraging the prior knowledge about the structure in the action space, it quickly learns a policy under the new action set that is able to perform nearly $2\times$ to $3\times$ better than AC. Additional experiments and detailed implementation discussions are available in the Appendix.

6 Conclusion

We introduced a new class of policy which is invariant to the cardinality of action set and can generalize across the space of actions. The proposed learning method leverages the structre of this policy to quickly adapt to the available set of actions in a reward-independent supervised learning process. Superior performances on both synthetic and large-scale real-world environments make the CL-RA algorithm a promising method for continual learning tasks with a dynamic set of actions.

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.

H. B. Ammar, E. Eaton, P. Ruvolo, and M. E. Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proc. of AAAI*, 2015.

S. Bhatnagar. Adaptive multivariate three-timescale stochastic approximation algorithms for simulation based optimization. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 15(1):74–107, 2005.

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

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

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.

C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer, 2002.

A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

A. Gupta, B. Eysenbach, C. Finn, and S. Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

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.

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.

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.

A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

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.

W. Masson, P. Ranchod, and G. Konidaris. Reinforcement learning with parameterized actions. In *AAAI*, pages 1934–1940, 2016.

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.

M. B. Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

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.

A. A. Sherstov and P. Stone. Improving action selection in mdp's via knowledge transfer. In *AAAI*, volume 5, pages 1024–1029, 2005.

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

A. J. Smith. Applications of the self-organising map to reinforcement learning. *Neural networks*, 15 (8-9):1107–1124, 2002.

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.

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.

Appendix

Contents

A Background

We consider problems which can be modeled as a discrete-time Markov decision process (MDP) with continuous (or discrete) states and discrete actions. The MDP is represented by a tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, d_0)$. $\mathcal{S} \subseteq \mathbb{R}^d$ is the set of all possible states, called the state space, and $\mathcal{A} \subseteq \{0,1\}^{|\mathcal{A}|}$ is the set of discrete actions, called the action set. Capturing the primary structure of a higher dimensional continuous action space ($\in \mathbb{R}^m$) into a lower dimensional representation space ($\in \mathbb{R}^n$) can be beneficial as well, but this results in inability to represent some sections of the original action space (as $n < m$). In this work, we consider only discrete actions as any finite set of discrete items can be embedded in a continuous space.

The first state is drawn from an initial distribution, d_0, and each time step is associated with a tuple, (s_t, a_t, r_t, s_{t+1}), where $s_t, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}$ and the reward r_t, is scalar and bounded. When the specific time instance is not important, we represent this tuple as (s, a, r, s'). The reward discounting parameter is given by $\gamma \in [0, 1)$. Throughout the paper we use P for probability density function. The state transition probability $\mathcal{P}(s_t, a_t, s_{t+1}) = P(s_{t+1}|s_t, a_t)$.

For a given $\mathcal{M}$, an agent's goal is to find a policy, $\pi(a|s) \coloneqq \mathcal{S} \times \mathcal{A} \to [0,1]$, which results in maximum expected sum of discounted future rewards. For a policy π, the state-action value function $Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s, a, \pi]$. An optimal policy, $\pi^* \in \text{argmax}_{\pi \in \Pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \pi]$, where Π denotes the set of all possible policies. Here, the expectation are over an entire trajectory, $(s_0, a_0, ...)$, for which, $s_0 \sim d_0, a_t \sim \pi(s_t, \cdot)$ and $s_{t+1} \sim \mathcal{P}(s_t, a_t, \cdot)$. The value function, $V^\pi(s)$, associated with each state, s, is given by the Bellman equation: $V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$.

B Learning Procedure

To make the policy invariant to the cardinality of the action set, and to generalize feedback across actions, we introduced the concept of action representations. This decomposed the policy into two components: **a)** an internal policy, π_i, that selects an action representations and **b)** a function, f, that transforms the selected representation to an actual action. In this section, we discuss in details the learning procedure for this new class of policy.

To elucidate the steps involved, we split it into following sub-sections. First, we show that there exists function f and π_i that can be used to learn any optimal policy. Second, learning process for the function f when π_i is fixed. Followed by learning process for π_i when f is fixed, and finally, combining them to learn simultaneously.

B.1 Existence of π_i and f to represent any optimal policy.

The characteristics of the actions are naturally associated with their transition behavior in the MDP. 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. Given an embedding, $E_t = e$, $A_t = a$ is conditionally independent of $S_t = s$ and $S_{t+1} = s'$. That is, the transition information can be sufficiently encoded to infer the action that was executed. With this, we make the following assumption (and later show how this can be satisfied):

Assumption 1. *There exists a delta probability distribution, $P(a|e)$, represented by a function, f, such that*

$$P(a|s,s') = \int_e P(a|e) P(e|s,s') de. \tag{1}$$

Here, $P(a|s,s')$ and $P(e|s,s')$ are the probability of the action and the latent variable, respectively, given the consecutive states, s and s'. We now establish the necessary property under which our proposed policy can represent any optimal policy. Satisfying this necessary condition will be later useful in deriving the learning rules.

Lemma 1. *Under Assumption 1, which defines a function f, $\forall \pi, \exists \pi_i$ such that*

$$V^\pi(s) = \sum_a \int_{f^{-1}(a)} \pi_i(e|s) Q^\pi(s,a) de.$$

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

$$\begin{aligned}V^{\pi}(s) &= \sum_{a} \pi(a|s) Q^{\pi}(s,a) \\ &= \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[r + \gamma V(s')].\end{aligned}$$

Without loss of generality, we consider discrete states in the proof for notational simplicity. Further, for brevity, $[r + \gamma V(s')]$ is denoted as R from here on. We re-arrange some terms in Bellman equation to obtain:

$$\begin{aligned}V^{\pi}(s) &= \sum_{a}\sum_{s'} \pi(a|s) \frac{P(s',s,a)}{P(s,a)} R \\ &= \sum_{a}\sum_{s'} \pi(a|s) \frac{P(s',s,a)}{\pi(a|s)P(s)} R \\ &= \sum_{a}\sum_{s'} \frac{P(s',s,a)}{P(s)} R \\ &= \sum_{a}\sum_{s'} \frac{P(a|s,s')P(s,s')}{P(s)} R.\end{aligned}$$

Using the law of total probability, we introduce the new variable e such that:

$$V^{\pi}(s) = \sum_{a}\sum_{s'} \int_{e} \frac{P(a,e|s,s')P(s,s')}{P(s)} R \mathrm{d}e.$$

After multiplying and dividing with $P(e|s)$, it becomes:

$$\begin{aligned}V^{\pi}(s) &= \sum_{a}\sum_{s'} \int_{e} P(e|s) \frac{P(a,e|s,s')P(s,s')}{P(e|s)P(s)} R \mathrm{d}e \\ &= \sum_{a} \int_{e} P(e|s) \sum_{s'} \frac{P(a,e|s,s')P(s,s')}{P(s,e)} R \mathrm{d}e.\end{aligned}$$

We can further simplify it as:

$$\begin{aligned}V^{\pi}(s) &= \sum_{a} \int_{e} P(e|s) \sum_{s'} P(s',a|s,e) R \mathrm{d}e \\ &= \sum_{a} \int_{e} P(e|s) \sum_{s'} P(s'|s,a,e) P(a|s,e) R \mathrm{d}e.\end{aligned}$$

Since transition to s' in the environment is only dependent upon the previous state and action,

$$V^{\pi}(s) = \sum_{a} \int_{e} P(e|s) \sum_{s'} P(s'|s,a) P(a|s,e) R \mathrm{d}e.$$

Using Markov property, action a is conditionally independent of the states given e,

$$V^{\pi}(s) = \sum_{a} \int_{e} P(e|s) \sum_{s'} P(s'|s,a) P(a|e) R \mathrm{d}e.$$

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

$$\begin{aligned}V^{\pi}(s) &= \sum_{a} \int_{f^{-1}(a)} P(e|s) \sum_{s'} P(s'|s,a) R \mathrm{d}e \\ &= \sum_{a} \int_{f^{-1}(a)} P(e|s)\ Q^{\pi}(s,a) \mathrm{d}e. \qquad (2)\end{aligned}$$

□

Theorem 1. *Under Assumption 1, which defines a function f, for all optimal policies, $\pi^\star$, there exists an internal policy, $\pi_i^\star$ such that the overall policy, π_o, consisting of $\pi_i^\star$ and f is also optimal.*

Proof. Recall that any policy under which the optimal value function, V^*, can be obtained is an optimal policy. Following Lemma 1, it is now straightforward to see that under the function f obtained from Assumption 1, there exists an instantiation of our overall policy:

$$\pi_o(a|s) = \int_{f^{-1}(a)} \pi_i(e|s)\mathrm{d}e, \tag{3}$$

that can be used to optimally solve the MDP as well. $P(e|s)$ from Eqn (2) is renamed to $\pi_i(e|s)$ to represent the internal policy. □

B.2 Learning f when π_i is fixed

The function f in the overall policy, π_o, directly corresponds to $P(a|e)$. However, neither the probability $P(e|s,s')$ nor $P(a|e)$ is known *a priori*. Therefore, to get this function f, we need to learn a $P(a|e)$ which satisfies Assumption 1.

The required $P(a|e)$ can be achieved by estimating $P(a|s,s')$ itself as a surrogate objective. In this regard, we parameterize $P(a|e)$ and $P(e|s,s')$ using the learnable functions $f(e)$ and $g(s,s')$ respectively. A computational graph for this parameterized model is shown in Figure 2. Once these functions are learned, the mapping function, f, for the overall policy, π_o, becomes available as well. However, note that $P(e|s,s')$ is never required by π_o. Its only objective is to assist in learning $P(a|e)$ and can be discarded once that objective is accomplished.

Computing the exact value of $P(a|s,s')$ involves a complete integral over e. Due to the absence of any closed form solution, we need to rely on a stochastic estimate of $P(a|s,s')$. 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 (1) by mean-marginalizing it to $\bar{P}$ using the value of the mean from $g(s,s')$. That is, $\bar{P}(a|s,s') = P(a|g(s,s'))$. The function $f(e)$ is modeled such that it chooses an action based on how similar its representation is to e. Let, $W \in \mathbb{R}^{d_e \times |\mathcal{A}|}$ be such that each column of this matrix is a learnable action representation of dimension $\mathbb{R}^{d_e}$. The probability, $P(a|e)$, of any action a is then defined to be dependent upon a similarity score, z, such that:

$$z_a = f(g(s,s')) = W_a^T g(s,s').$$

Here, W_a^T is the column corresponding to the action a. Such linear decomposition is not restrictive as g can still be any differentiable function approximator (that is, neural networks, etc.). We normalize these scores, z, using the Boltzmann distribution to get a valid probability distribution.

$$P(a|g(s,s')) = \frac{\mathbf{e}^{z_a/\tau}}{\sum_{a'} \mathbf{e}^{z_{a'}/\tau}}. \tag{4}$$

Here, τ is the temperature variable and in the limit when $\tau \to 0$ the conditional distribution over actions becomes the required deterministic function 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.

To satisfy Assumption 1, we define the objective to ensure that $\bar{P}(a|s,s')$ obtained using functions f and g, is a close approximate to the true distribution, $P(a|s,s')$. One way to measure the distance between $P(a|s,s')$ and $\bar{P}(a|s,s')$ is by using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||\bar{P}) = -\sum_{\mathcal{A}} P(a|s,s') \log \frac{\bar{P}(a|s,s')}{P(a|s,s')}$$

Since the observed trajectories itself contains the transitions from the distribution $P(a|s,s')$, we can readily compute the loss using stochastic estimate of $D_{KL}(P||\bar{P})$:

$$\mathcal{L} = -\mathbb{E}_{\mathcal{A}}[\log \bar{P}(a|s,s')|s,s'] \tag{5}$$

where the expectation is over all the actions and the second term of KL is independent of the parameters being optimized and is thus ignored. The loss in (5) can be easily minimized using any gradient based update rule to learn the functions f and g. When the loss is reduced to zero, the Assumption 1 required by the Theorem 1 is perfectly satisfied. However, using function approximators, this will not be always possible and we only aim for local optima.

Since this learning process for the function f only requires estimating $P(a|s,s')$, it not only falls under the standard set-up of supervised training, it is also *independent of the reward.* Due to dense and more informative signal, this component of the overall policy can be learned much quicker and with less variance.

B.3 Learning π_i when f is fixed

One common method for learning a policy, π, parameterized with weights θ is to optimize for its performance function:

$$J(\theta) = \sum_s d_0(s) V^{\pi}(s),$$

where, $d_0(s)$ is the initial state distribution and $V^{\pi}(s)$ is the value function for the policy π. For a policy parameterized with weights θ, the expected performance of the policy can be improved by ascending the *policy gradient*, $\nabla J(\theta) := \frac{\partial J(\theta)}{\partial \theta}$, of its performance function. To learn the internal policy, π_i, we define its associated performance function as:

$$J_i(\theta) = \sum_s d_0(s) V^{\pi_i}(s). \tag{6}$$

Using the policy gradient theorem [Sutton et al., 2000], we know that the gradient of (6) is given by:

$$\nabla J_i(\theta) = \mathbb{E}_s \left[\int_e Q^{\pi}(s,e) \frac{\partial}{\partial \theta} \pi_i(e|s) \, \mathrm{d}e \right], \tag{7}$$

The internal policy can be learned by iteratively updating its parameters in the direction of $\nabla J_i(\theta)$. Since there are no special constraints on the policy π_i, any variant of the policy gradient algorithm: DPG [Silver et al., 2014], PPO [Schulman et al., 2017], NAC [Bhatnagar et al., 2009] etc, designed for continuous control 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:

$$J_o(\theta, f) = \sum_s d_0(s) V^{\pi_o}(s).$$

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

Property 1. *For a deterministic function, f, that maps each point, $e \in \mathbb{R}^{d_e}$, in the representation space to an action, $a \in \{0,1\}^{|\mathcal{A}|}$, the expected updates to θ based on $\nabla J_i(\theta)$ are equivalent to updates based on $\nabla J_o(\theta, f)$. That is,*

$$\nabla J_o(\theta, f) = \nabla J_i(\theta).$$

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

$$J_o(\theta, f) = \sum_s d_0(s) \sum_a \int_{f^{-1}(a)} P(e|s) Q^{\pi}(s,a) \mathrm{d}e.$$

The gradient of the performance function is:

$$\nabla J_o(\theta, f) = \mathbb{E}_s \Big[\sum_a \frac{\partial}{\partial \theta} \Big(\int_{f^{-1}(a)} \pi_i(e|s) Q^{\pi}(s,a) \mathrm{d}e \Big) \Big].$$

Where the expectation is over the initial state distribution. By unrolling the Q-function and using the policy gradient theorem [Sutton et al., 2000], we know that the biased gradient [Thomas, 2014] of $J_o(\theta, f)$ is:

$$\nabla J_o(\theta, f) = \mathbb{E}_s\Big[\sum_a \int_{f^{-1}(a)} \frac{\partial}{\partial\theta}\big(\pi_i(e|s)\big)Q^\pi(s,a)\mathrm{d}e\Big],$$

where, the expectation is over d^π, as defined in [Sutton et al., 2000] (which is not a true distribution, since it is not normalized). Continuing, we have that:

$$\nabla J_o(\theta, f) = \mathbb{E}_s\Big[\sum_a \int_{f^{-1}(a)} \pi_i(e|s)\frac{\partial}{\partial\theta}\ln\big(\pi_i(e|s)\big)Q^\pi(s,a)\mathrm{d}e\Big].$$

Note that for the set of e that maps to a, $Q^\pi(s,a) = Q^\pi(s,e)$. Therefore,

$$\nabla J_o(\theta, f) = \mathbb{E}_s\Big[\sum_a \int_{f^{-1}(a)} \pi_i(e|s)\frac{\partial}{\partial\theta}\ln\big(\pi_i(e|s)\big)Q^\pi(s,e)\mathrm{d}e\Big].$$

Finally, since each e gets 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,

$$\begin{aligned}\nabla J_o(\theta, f) &= \mathbb{E}_s\left[\int_e \pi_i(e|s)\frac{\partial}{\partial\theta}\ln\big(\pi_i(e|s)\big)Q^\pi(s,e)\mathrm{d}e\right],\\ &= \nabla J_i(\theta).\end{aligned}$$

□

The chosen parameterization for the policy has this special property which allows π_i to be learned using its local policy gradient. Hence, independent of the function f, we can directly use $\nabla J_i(\theta)$ to update the parameters of the internal policy, π_i, while still optimizing the overall performance $J_o(\theta, f)$.

B.4 Learning π_i and f simultaneously

Since the learning procedure for f is independent of rewards, few initial random trajectories can be leveraged to capture the primary structure in the action space. However, there are two primary reasons to keep updating the representations online:

- In cases where multiple actions can result in same state transitions, samples from on-policy distribution are required such that actions chosen by the current policy can be used to learn f.
- With more interactions more data becomes available, which can be used for fine-tuning and improving the action representations.

If the action representations are held fixed while learning the internal policy, then as a consequence of Property 1, convergence of our algorithm directly follows from previous two-time scale results [Borkar and Konda, 1997; Bhatnagar et al., 2009]. Here, we show that learning both π_i and f simultaneously can also be shown to converge by using the three-time scale analysis.

To analyze the convergence of CL-RA algorithm, we first present a general setup for stochastic recursion coupled among three parameter sequences and discuss its asymptotic behavior using three different time-scales. This would lay the foundation for our convergence proof. We then briefly discuss the existing asymptotic convergence results using two time-scales for the Actor-Critics. Finally, PG-RA (and consequently CL-RA) method that extends Actor-Critic using a new action prediction module are proved to converge by using three time-scales.

Consider the following system of stochastic ODEs:

$$X_{t+1} = X_t - \alpha_t^x(f(X_t, Y_t, Z_t) + \mathcal{N}_t^1), \tag{8}$$

$$Y_{t+1} = Y_t - \alpha_t^y(g(Y_t, Z_t) + \mathcal{N}_t^2), \tag{9}$$

$$Z_{t+1} = Z_t - \alpha_t^z(h(X_t, Y_t, Z_t) + \mathcal{N}_t^3), \tag{10}$$

where, $\{\mathcal{N}_t^1\}$, $\{\mathcal{N}_t^2\}$ and $\{\mathcal{N}_t^3\}$ are the associated Martingale difference sequences for noise. To study its asymptotic behavior, we consider the following standard assumptions:

Assumption 2 (Boundedness)**.** $\sup_t ||X_t||, \sup_t ||Y_t||, \sup_t ||Z_t|| < \infty$

Assumption 3 (Learning rate schedule)**.** *The learning rates α_t^x, α_t^y and α_t^z satisfy:*

$$\sum_t \alpha_t^x = \sum_t \alpha_t^y = \sum_t \alpha_t^z = \infty,$$

$$\sum_t (\alpha_t^x)^2, \sum_t (\alpha_t^y)^2, \sum_t (\alpha_t^z)^2 < \infty,$$

$$\textit{As } t \to \infty, \quad \frac{\alpha_t^z}{\alpha_t^y} \to 0, \frac{\alpha_t^y}{\alpha_t^x} \to 0. \tag{11}$$

Assumption 4 (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.*

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

Assumption 5 (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.*

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

Assumption 6 (Existence of stationary point for Z)**.** *The following ODE has a globally asymptotically stable equilibrium $Z^\star$,*

$$\dot{Z} = h(\mu_2(\mu_1(Z(t)), Z(t)), \mu_1(Z(t)), Z(t))$$

Assumptions 2–3 are required to bound the values of the parameter sequence and make the learning rate well-conditioned, respectively. Assumption 4-5 ensures that there exists a stationary point for the respective recursions, individually, when other parameters are held constant. Finally, Assumption 6 ensures that there exists a 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.

Property 2. *Under Assumptions 2-6, $(X_t, Y_t, Z_t) \to (\mu_2(\mu_1(Z^\star), Z^\star), \mu_1(Z^\star), Z^\star)$ as $t \to \infty$, with probability one.*

Proof. The result follows directly by extending two time-scale analysis [Bhatnagar et al., 2009; Borkar and Konda, 1997] using three time-scales [Bhatnagar, 2005]. We only present a sketch of the proof and discuss the intuition behind the key ideas. For detailed discussions on two and three time-scale convergence, we refer the readers to [Konda and Tsitsiklis, 2000; Bhatnagar et al., 2009; Bhatnagar, 2005].

Since these three updates are not independent on each time step we consider three step-size schedules: $\{\alpha_t^x\}$, $\{\alpha_t^y\}$ and $\{\alpha_t^z\}$ that satisfy Assumption 3. As a consequence of (11), the recursion (9) is 'faster' than (10), and (8) is 'faster' than both (9) and (10). In other words, Z moves on the slowest time-scale and the X moves on the fastest. Such a timescale is desirable since Z_t converges to its stationary point when, if at each time step the value of the corresponding converged X and Y estimates are used to make the next Z update (Assumption 6).

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

$$\dot{Y} = g(Y(t), Z(t)), \tag{12}$$

$$\dot{Z} = 0. \tag{13}$$

Alternatively, one can consider the ODE

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

in place of (12), because Z is fixed (13). Now, under Assumption 4 we know that the iterative update (9) performed on Y, with a fixed Z, will eventually converge to a corresponding stationary point.

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

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

$$\dot{Y} = 0, \tag{15}$$

$$\dot{Z} = 0. \tag{16}$$

Alternatively, one can consider the ODE

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

in place of (14), as Y and Z are fixed (15)-(16). As a consequence of Assumption 5, X converges to an optima when both Y and Z are held fixed.

Asymptotically, as a result of Assumption 3, 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 (9) 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 and has an effect similar to fixing Y_t, Z_t and running the iteration (8) forever. In turn, this makes Z_t sees X_t as a close approximation to $\mu_2(\mu_1(Z(t)), Z(t))$ always, and thus converges to $Z^\star$ due to Assumption 6.

Therefore, under the Assumptions 2-6, it can be seen that $(X_t, Y_t, Z_t) \to (\mu_2(\mu_1(Z^\star), Z^\star), \mu_1(Z^\star), Z^\star)$ as $t \to \infty$. When non-linear function approximators are used, this system converges to a local optimum. □

B.4.1 Actor-Critic convergence using two time-scales:

In the Actor-Critic method, 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 large time between successive updates to the policy parameters and is severely sample-inefficient. Two-time scale stochastic approximation methods [Bhatnagar et al., 2009; Konda and Tsitsiklis, 2000] circumvent this difficulty. The faster update recursion for critic ensures that asymptotically it is always a close approximation to the reuiqred value function before the next update to the policy is made.

B.4.2 PG-RA and CL-RA convergence using three time-scales:

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

Let the parameters of critic and the internal policy be denoted as $\hat{V}$ and θ respectively. Also, let all the parameters of f and g be represented as ϕ. Then a locally optimal set of parameters for solving an MDP is denoted as $\hat{V}^\star, \phi^\star$ and $\theta^\star$.

Theorem 2. *Under Assumptions 1-6, PG-RA parameters:* $(\hat{V}_t, \phi_t, \theta_t) \to (\hat{V}^\star, \phi^\star, \theta^\star)$ *as* $t \to \infty$, *with probability one.*

Proof. The result is a direct consequence of Property 2, when PG-RA parameters: $(\hat{V}, \phi, \theta)$ corresponds to (X, Y, Z) respectively. Also, $\phi^\star = \mu_1(\theta^\star)$ and $\hat{V}_t^\star = \mu_2(\phi^\star, \theta^\star)$. Note that the updates to the action prediction component are independent of rewards/critic. Therefore, the ODEs that govern the updates recursions for PG-RA parameters are of the form (8)–(10) that are used in Property 2. Here, f, g and h correspond to the semi-gradients of TD-error, gradients of self-supervised loss, and policy gradients, respectively.

All the required conditions that are needed for Property 2 are also satisfied by our PG-RA algorithm. The learning rates in Assumption 3 are hyper-parameters and can be easily set as per the three time-scale requirement. With the gradient update rule derived in Section B.2, for an internal policy with fixed parameters, θ, there exists the stationary point $\mu_1(\theta)$ to which the parameters of action prediction module converge under the self-supervised update. This satisfies Assumption 4. Similar to standard Actor-critic case, we know that for a fixed policy (fixed action representations and fixed internal policy), there exists a stationary point $\mu_2(\phi, \theta)$, to which the critic converges while minimizing the TD error. This satisfies Assumption 5. Finally, as applicable from Property 1, under the optimal critic and action representations for every step, the internal policy converges to the optima of the global objective by following its local policy gradient. This satisfies the Assumption 6. □

Corollary 1. *Under Assumptions 1-6,* $\forall i \in$ *Change, CL-RA parameters:* $(\hat{V}_t, \phi_t, \theta_t) \to (\hat{V}_i^\star, \phi_i^\star, \theta_i^\star)$ *as* $t \to \infty$, *with probability one.*

Proof. Each time PG-RA is invoked by CL-RA, the parameters of π, f and g can just be seen as adaptively initialized to new values conditioned on all the previous changes. The rest follows from Theorem 2 $\square$

C Relation to existing works

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

Continual Learning: One of the pioneering works to make an agent learn continuously over time was done by Ring [1994]. The authors showed how to constantly adapt a neural network architecture to learn in environments where the size of the state space changed over time. In order to make learning of new tasks easier in domains with large action spaces, Sherstov and Stone [2005] leveraged knowledge from previous tasks to selectively keep only a smaller subset of the original large action set for downstream tasks. To minimize the chance of discarding useful actions, *random task perturbation* method was introduced to let the agent have a more thorough exposure to the domain. More recently, to transfer knowledge from an agent to another agent having a different morphological structure (with possibly different action space), Gupta et al. [2017] learned invariant feature spaces of agents to make skill transfer possible.

Other recent works [Gupta et al., 2018; Ammar et al., 2015] also considered similar problem of quickly adapting RL algorithms by transferring previous acquired knowledge from few initial training tasks to test tasks. In such settings, all tasks were drawn from a common 'task distribution' with same action space. A detailed survey on such transfer learning tasks in reinforcement learning can be found in the work by Lazaric [2012]. Compared to transfer learning setup, our focus is on individual domains where the set of available actions continually changes over time.

Factorizing action space: To reduce 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 used as product of mixtures by Sallans and Hinton [2004] and also by Guestrin et al. [2002] for multi-RL setup to reduce the overall complexity of large action space. Masson et al. [2016] considered a pre-determined discrete set of continuous actions for MDPs with parameterized actions. More recently, Sharma et al. [2017] showed that when actions in ATARI domains are factored into their primary categories better performances can be achieved. All these methods assumed that handcrafted binary decomposition of raw actions was provided. To deal with discrete actions that might have 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, but they assumed that the embeddings for the actions are given.

Assumptions about *a priori* knowledge of action representations is often not applicable in practice. If such additional information is known about the actions, they can also be considered to initialize the action representations in our algorithm. Our method can further optimize any available representations to maximize the desired objective, as more data becomes available. Therefore, such works that do not involve learning action representations can be seen as a special instance of our algorithm. However, we don't assume availability of any side-information regarding action representations in this work and learn these representations from scratch autonomously.

For high dimensional continuous actions, Smith [2002] used self-organizing map (SOM) to capture the structure in action space. Since reducing a high dimensional continuous space to a lower dimensional space comes at a cost of inability to represent a large regions of original space, they directed the SOM updates towards the section of action space that resulted in higher return. Our method can also be extended for continuous action spaces under similar heuristics. However, in continual learning setup some actions might become useful later and since discarding them earlier would be disadvantageous, we restrict our focus in this work to discrete action spaces only.

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 systems have been extensively used to model dynamic motor primitives

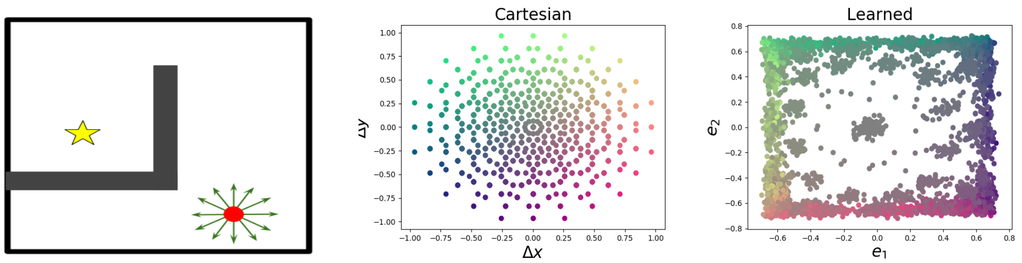

Figure 4: (a) The continuous state, discrete action maze environment. The star dentoes the goal state, the red dot corresponds to the agent and the arrows around it are the 12 actuators. Each action corresponds to a unique combination of these actuators. Therefore, in total 2^{12} number of actions are 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 co-ordinates 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 the $\tanh$ non-linearity applied to bound its range.

(DMPs) [Ijspeert et al., 2003; Schaal, 2006] for continuous control. It was show in [Kober and Peters, 2009b] how imitation learning can be used to learn DMPs and [Kober and Peters, 2009a] extended it to do fine-tuning these DMPs online using RL. However, these are significantly different from our work as they are specifically parameterized for robotics tasks and produce an entire policy. Later, Thomas and Barto [2012] showed how a goal-conditioned policy can be learned over multiple such motor primitives that controls only useful sub-spaces of the underlying continuous control problem. To learn binary motor primitives, Thomas [2011]; Thomas and Barto [2011] showed how a policy can be modeled as a composition of multiple "coagents", where each of them just learns using the local policy gradient information. Our work is motivated along similar directions, but we focus on automatically learning optimal continuous-valued action representations for discrete actions. For this we use supervised updates instead of high variance policy gradients.

Auxiliary tasks: Reconstructing various aspects of the transitions was shown to be a useful heuristic by Jaderberg et al. [2016] for getting additional gradient information to assist learning. Action prediction loss was also considered by Shelhamer et al. [2016]; Pathak et al. [2017] but was primarily motivated towards learning inverse dynamics for intrinsic rewards. We show how the overall policy itself can be decomposed using an action representation module learned using a similar loss function.

Other domains: In supervised learning, representations of the output categories have been used to extract additional correlation information among the labels from external data. Popular examples include learning label embeddings [Akata et al., 2016] for image classification and learning word embeddings [Mikolov et al., 2013] for natural language problems. In RL setup, the output of the policy is an action and we show how learning its representations can be beneficial as well.

D Additional Empirical Analysis

To dissect important aspects of our proposed algorithm, we perform additional analysis on the maze and the two real-world domains. In the following sections, we analyze the ability of our method to learn action representations that capture the underlying space of actions and the ability to use them for efficiently generalizing across the action space.

D.1 Visualizing the learned action representations

In this section, we present visualizations of the learned action representations to understand the internal working of our proposed algorithm. The maze domain acts as a didactic environment for this purpose. A pictorial illustration of the environment is provided in Fig: 4. Here, the underlying structure in the space of actions is related to the displacements in the cartesian coordinates when the

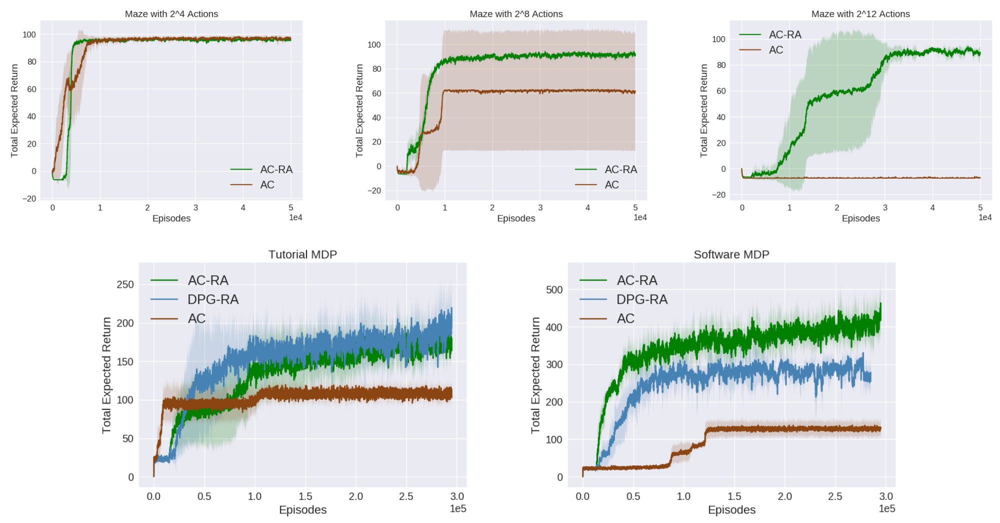

Figure 5: Experiments with all possible actions available to the agent throughout. (Top) Results on the Maze domain with $2^4, 2^8$, and 2^{12} actions respectively. (Bottom) Results on a) Tutorial MDP b) Software MDP. Here, AC-RA and DPG-RA are the variants of CL-RA algorithm (with $Change = \{0\}$) that uses ActorCritic and DPG, respectively.

respective action is executed by the agent. This provides a intuitive base case against which we can compare our results. In Figure 4 we compare the action representations learned using our algorithm to the underlying cartesian representation of the actions. It can be seen that the proposed method extracts the 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. Action corresponding to those are at the edge of the representation space. To make exploration step tractable in the internal policy, π_i, we bound the representation space along its each dimension to the range $[-1, 1]$ using Tanh non-linearity. This results in 'squashing' of these representations around the edge of this range. Notice, the smooth color transition indicates that not only the information about magnitude of displacement but the direction of displacement is also well captured. The learned representations therefore efficiently preserve the relative transition correlation information among all the actions to a great extent.

D.2 Generalizing across actions in high dimensional action space

State representations help in taking 'similar' actions for states with 'similar' representation. Once an action is found useful to be executed in a state, state representations help in quickly generalizing it across states. Though useful, this does not directly help in finding the useful action that needs to be taken in the first place. The impact of this is pronounced in domains that have large action sets or a set of actions where new actions get added over time. Using the standard way of one-hot encoding for each action, it would require significantly larger number of samples to learn about every single action's consequence in different states as it has no means to quickly generalize the observed feedback across other actions. Action representations help to bridge this gap.

In Figure 3 we present results of experiments on the same domains as considered earlier, but keep all the possible actions available to the agent throughout. The plots show how the performance of standard AC method deteriorates drastically as the number of actions increases, even though the goal remains the same. However, with the action representation module it is able to consistently perform well across all the settings. Similarly, for both the tutorial and the software MDPs, standard AC methods fails to reason over longer time horizons under such overwhelming number of actions. Its learning gets stuck, 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, 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 CL-RA algorithm only a small fraction of total parameters, the ones in the internal policy, are learned using the high variance policy gradient updates. The other big fraction of parameters associated with action representations are learned by a supervised learning procedure. This helps in reducing the variance significantly, thereby making the CL-RA algorithms learn a better policy much faster. These advantages allow the internal policy, π_i, quickly search and learn the optimal policy without succumbing to the curse of high dimensional action space. We believe that the results can be further improved by using the proposed method with other policy gradient methods; we leave these for future work.

E Implementation Details

For the maze domain, single layer neural networks were used to parameterize both the actor and critic, and the learning rates were searched over $\{1e\text{-}2, 1e\text{-}3, 1e\text{-}4, 1e\text{-}5\}$. State features were represented using 3^{rd} order coupled Fourier basis [Konidaris et al., 2011]. The discounting parameter γ was set to 0.99 and λ traces to 0.9. Since it was a toy domain, the dimensions of action representations were fixed to 2. For continual learning experiments, 1000 random trajectories were drawn to adapt the action representations after each change. For additional experiments, where all the actions were available throughout, 2000 randomly drawn trajectories were used to learn an initial representation for the actions.

For Tutorial and Software MDPs, 2 layer neural networks were used to parameterize both the actor and critic, and the learning rates were searched over $\{1e\text{-}2, 1e\text{-}3, 1e\text{-}4, 1e\text{-}5\}$. Similar to prior works, module for encoding state features was shared to reduce the number of parameters and the learning rate for it was additionally searched over $\{1e\text{-}2, 1e\text{-}3, 1e\text{-}4, 1e\text{-}5\}$. 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 traces were 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 $\{16, 32, 64\}$. For continual learning experiments, 5000 random trajectories were drawn to adapt the action representations after each change. For additional experiments, where all the actions were available throughout, initial $10,000$ randomly drawn trajectories were used to learn an initial representation for the actions.

For all the results of the CL-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 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 f and g were jointly searched over $\{1e\text{-}2, 1e\text{-}3, 1e\text{-}4\}$. The architecture and hyper-parameter search for the baselines were done in the same way. All the results were obtained for 3 different seeds to get the variance.