ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/281607283

Reinforcement Learning with Parameterized Actions

Article · September 2015

Source: arXiv

CITATIONS
49

READS
6,266

2 authors, including:

George Konidaris
Brown University
133 PUBLICATIONS **3,518** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Portable Representations View project

All content following this page was uploaded by George Konidaris on 21 January 2016.

The user has requested enhancement of the downloaded file.

Reinforcement Learning with Parameterized Actions

Warwick Masson
School of Computer Science and Applied Mathematics
University of Witwatersrand
Johannesburg, South Africa

George Konidaris
Department of Computer Science
Duke University
Durham, North Carolina 27708

arXiv:1509.01644v1 [cs.AI] 5 Sep 2015

Abstract

We introduce a model-free algorithm for learning in Markov decision processes with parameterized actions—discrete actions with continuous parameters. At each step the agent must select both which action to use and which parameters to use with this action. This models domains where there are distinct actions which can be adjusted to a particular state. We introduce the Q-PAMDP algorithm for learning in these domains. We show that Q-PAMDP converges to a local optima, and compare different approaches in a robot soccer goal-scoring domain and a platformer domain.

1 Introduction

In reinforcement learning, we typically consider either a discrete or a continuous action space (Sutton and Barto 1998). With a discrete action space, the agent makes decisions about which distinct action to perform from a finite action set. With a continuous action space, the selected action is expressed as a real-valued vector. If we use a continuous action space, we lose the ability to consider differences in kind: all actions must be expressible as a single vector. However, if we only use discrete actions, we are restricted to discrete actions or suffer a blow-up in the number of actions to represent a wide range of actions.

A parameterized action is a discrete action parameterized by a real-valued vector. Modeling actions this way introduces structure into the action space by treating different kinds of continuous actions as distinct. At each step an agent must choose both which action to use and what parameters to execute it with. For example, consider a soccer playing robot which can kick, pass, or run. We can associate a continuous parameter vector to each of these actions: we can kick the ball to a given target position with a given force, pass to a specific position, and run with a given velocity. Each of these actions is parameterized in its own way, so expressing an action as a single vector containing all the parameters for each of these movements would be redundant, as only a small subset of the parameters are applicable.

We formally define parameterized action Markov decision processes (PAMDPs) to model the situations where we have distinct actions that require parameters to adjust the action to different situations, or where there are multiple incompatible continuous actions. In this paper, we are concerned with how to learn a policy in domain with pre-defined parameterized actions. A naive approach would treat this as a direct policy search problem, where we optimize the weights for a parameterized policy. We compare this approach against two methods: Q-PAMDP(1) and Q-PAMDP(∞). We show that with appropriate update rules both methods converge to a local optimum. The three methods are compared in a robot soccer goal domain and a platformer domain.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2 Background

A Markov decision process (MDP) is a tuple $\langle S, A, P, R, \gamma \rangle$, where S is a set of states, A is a set of actions, $P(s, a, s')$ is the probability of transitioning to state s' from state s after taking action a, $R(s, a, r)$ is the probability of receiving reward r for taking action a in state s, and γ is a discount factor (Sutton and Barto 1998). We wish to find a policy, π, which selects an action for each state so as to maximize the expected sum of discounted rewards (the return).

The value function $V^\pi(s)$ is defined as the expected discounted return and is given by

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right],$$

for the policy π starting at state s (Sutton and Barto 1998). Similarly, we can define the action-value function

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[r_0 + \gamma V^\pi(s) \right],$$

as the expected discounted return for taking action a in state s following policy π. While using the value function in control requires a model, we would prefer to do so without needing such a model. We can approach this problem by learning Q, as knowing the action-value function allows us to select the action which maximizes $Q^\pi(s, a)$. We can learn Q for an optimal policy using a method such as Q-learning (Watkins and Dayan 1992). In domains with a continuous state space, we can represent $Q(s, a)$ using parametric function approximation with a set of parameters ω. Algorithms such as gradient descent SARSA(λ) (Sutton and Barto 1998) can learn such an approximation of $Q(s, a)$.

For problems with a continuous action space ($A \subseteq \mathbb{R}^m$), selecting the optimal action with respect to $Q(s, a)$ is non-trivial, as it requires finding a global maximum for a function in a continuous space. We can avoid this problem using a policy search algorithm, where a class of policies parameterized by a set of parameters θ is given, which transforms the problem into one of direct optimization over θ for an objective function $J(\theta)$. Several approaches to the policy search problem exist, including policy gradient methods, entropy-based approaches, path integral approaches, and sample-based approaches (Deisenroth, Neumann, and Peters 2013).

Parameterized Tasks

A parameterized task is a problem that is determined by task parameters τ. These parameters are fixed throughout an episode, and the goal is to optimize a task dependent policy. Kober *et al.* (2012) developed algorithms to adjust motor primitives to different task parameters. They apply this to learn table-tennis and darts with different starting positions and targets. Da Silva *et al.* (2012) introduced the idea of a parameterized skill as a task dependent parameterized policy. They sample a set of tasks, learn their associated parameters, and determine a mapping from tasks to parameters (da Silva, Konidaris, and Barto 2012). Deisenroth *et al.* (2014) applied a model-based method to learn a task dependent parameterized policy. This is used to learn task dependent policies for ball-hitting task, and for solving a block manipulation problem.

3 Parameterized Actions Markov Decision Processes

We consider MDPs where the state space is continuous ($S \subseteq \mathbb{R}^n$) and the actions are parameterized: there is a finite set of actions $A_d = \{a_1, a_2, \dots, a_k\}$, and each action $a \in A_d$ has a set of continuous parameters $X_a \subseteq \mathbb{R}^{m_a}$. The action space is then given by

$$A = \bigcup_{a \in A_d} \{(a, x) \mid x \in X_a\}.$$

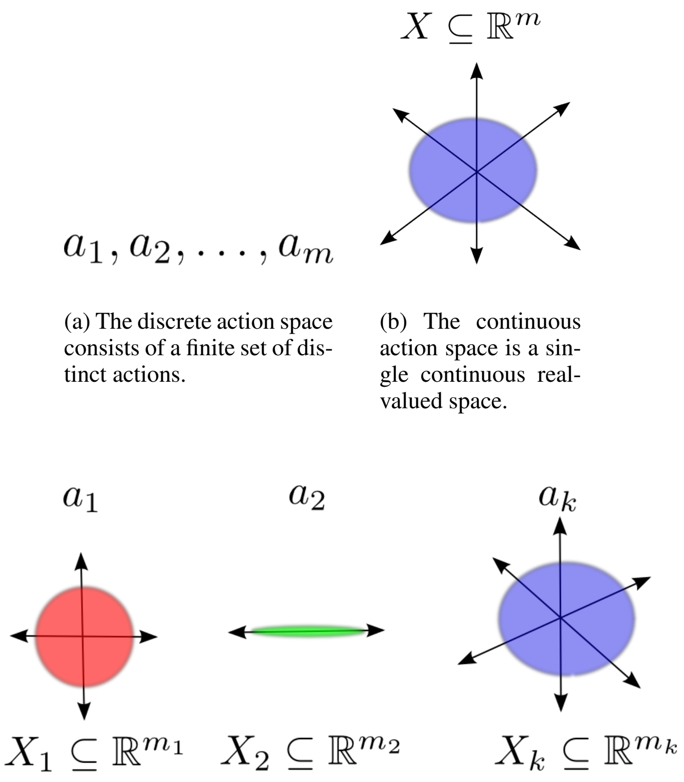
We refer to such MDPs as parameterized action MDPs (PAMDPs). Figure 1 depicts the different action spaces.

For example, a quad-rotor delivery system might have continuous actions for moving around and for dropping its payload on a target. We would not want to combine these actions into a single continuous action lest we experiment with dropping payloads while moving. One solution would be to use discretization where we consider a fixed parameterization for the different actions. This approach has the problem of both selecting an appropriate granularity for the policy, and in the blow-up in the number of actions considered.

We apply a two-tiered approach for domains with parameterized actions: first, selecting the parameterized action, then providing the parameters for that action. The discrete-action policy is denoted $\pi^d(a|s)$. To select the parameters for the action, we define the action-parameter policy for action a as $\pi^a(x|s)$. The policy is then given by

$$\pi(a, x|s) = \pi^d(a|s)\pi^a(x|s).$$

In other words, to select a complete action (a, x), we sample an action a from $\pi^d(a|s)$ and then sample a parameter x from $\pi^a(x|s)$.


$a_1, a_2, \dots, a_m$
$X \subseteq \mathbb{R}^m$
a_1
a_2
a_k
$X_1 \subseteq \mathbb{R}^{m_1}$
$X_2 \subseteq \mathbb{R}^{m_2}$
$X_k \subseteq \mathbb{R}^{m_k}$

(a) The discrete action space consists of a finite set of distinct actions.

(b) The continuous action space is a single continuous real-valued space.

(c) The parameterized action space has discrete actions, each of which has a continuous action space.

Figure 1: Three types of action spaces: discrete, continuous, and parameterized.

The first approach we consider is one of direct policy search. The action policy is defined by the parameters ω and is denoted by $\pi^d_\omega(a|s)$. The action-parameter policy for action a is determined by a set of parameters θ_a, and is denoted $\pi^a_\theta(x|s)$. We use a direct policy search method to optimize the objective function.

$$J(\theta, \omega) = \mathbb{E}_{s_0 \sim D}[V^{\pi_\Theta}(s_0)].$$

with respect to (θ, ω), where s_0 is an initial state sampled according to the initial state distribution D. J is the expected return for a given policy starting at an initial state.

Our second approach is to alternate updating the parameter-policy and learning an action-value function. For any PAMDP $M = \langle S, A, P, R, \gamma \rangle$ with a fixed parameter-policy π^a_θ, there exists a corresponding discrete action MDP, M_θ. We define $M_\theta = \langle S, A_d, P_\theta, R_\theta, \gamma \rangle$ where A_d is the discrete action set and

$$P_\theta(s, a, s') = \int_{x \in X_a} \pi^a_\theta(x|s) P(s, (a, x), s') dx$$

$$R_\theta(s, a, r) = \int_{x \in X_a} \pi^a_\theta(x|s) R(s, (a, x), r) dr.$$

We represent the action-value function for M_θ using function approximation with parameters ω. For M_θ, there exists

Algorithm 1 Q-PAMDP(k)

```
Input:
Initial parameters $\theta$
Parameter update method P-UPDATE
Q-learning algorithm Q-LEARN
Algorithm:
$\omega \leftarrow$ Q-LEARN$^{(\infty)}(M_\theta, -)$
repeat
  $\theta \leftarrow$ P-UPDATE$^{(k)}(J_\omega, \theta)$
  $\omega \leftarrow$ Q-LEARN$^{(\infty)}(M_\theta, \omega)$
until $\theta$ converges
```

an optimal set of representation weights ω_{θ^*} which maximizes $J(\theta, \omega)$ with respect to ω. Let

$$W(\theta) = \arg\max_{\omega} J(\theta, \omega) = \omega_{\theta^*}.$$

We can compute $W(\theta)$ using a Q-learning algorithm such as Greedy-GQ (Maei et al. 2010) or gradient-descent Sarsa(λ) (Sutton and Barto 1998). Finally, we let

$$J_\omega(\theta) = J(\theta, \omega),$$
$$H(\theta) = J(\theta, W(\theta)).$$

Intuitively, $H(\theta)$ is the performance of the best discrete policy for M_θ.

Algorithm 1 describes a method for alternating updating θ and ω. The algorithm takes two input methods: P-UPDATE and Q-LEARN. It takes a positive integer parameter k, which determines the number of P-UPDATE(f, θ) should be a policy search method that optimizes θ with respect to f. Q-LEARN can be any algorithm for Q-learning with function approximation, and is left as a design choice. We consider two main cases of the Q-PAMDP algorithm: Q-PAMDP(1) and Q-PAMDP(∞).

For Q-PAMDP(1), we perform a single update of θ, then relearn ω to $W(\theta)$. The idea is that to optimize J, we can define a function

$$H(\theta) = J(\theta, W(\theta)),$$

which represents the best possible performance for parameters θ. In the next section we show that if we can find a local optima θ with respect to H, then we have found a local optima with respect to J. If at each step we only update θ once, then update ω until convergence, then we can optimize θ with respect to H.

With Q-PAMDP(∞) each step performs a full optimization on θ and then a full optimization of ω. The θ step would optimize $J(\theta, \omega)$, not $H(\theta)$, as we do update ω while we update θ. This approach can be problematic due to premature optimization, and the requirement of full convergence (Thomas and Barto 2011). Full convergence is a problem because it requires a convergence test for every single update, as well as a test for the sequence as a whole.

4 Theoretical Results

Now we show that Q-PAMDP(1) converges to a local or global optimum with mild assumptions. We assume that if we iterate P-UPDATE, it will converge to some θ^* with respect to a given objective function. As the P-UPDATE step is a design choice, it can be selected with the appropriate convergence property. We consider the sequence

$$\omega_{t+1} = W(\theta_t)$$
$$\theta_{t+1} = \text{P-UPDATE}(J_{\omega_t}, \theta_t),$$

for some initial parameters θ. Q-PAMDP(1) is equivalent to this sequence if iteration of Q-LEARN converges to $W(\theta)$ for each given θ. Next we show that if P-UPDATE can locally optimizes θ with respect to some objective function, then Q-PAMDP(1) converges to a local optimum.

Theorem 4.1 (Convergence to a Local Optimum)**.** *If the sequence*

$$\vartheta_{t+1} = \textit{P-UPDATE}(H, \theta_0),$$

converges to a local optima with respect to H, then Q-PAMDP(1) converges to a local optima with respect to J.

Proof. By definition of the sequence above $\omega_t = W(\theta_t)$, so it follows that

$$J_{\omega_t} = J(\theta, W(\theta)) = H(\theta).$$

In other words, the objective function J is equivalent to the objective function H if the condition on ω is satisfied. Therefore, we can replace J with H in our update for θ, to obtain the update rule

$$\theta_{t+1} = \text{P-UPDATE}(H, \theta_t).$$

Therefore by conditions of this theorem the sequence θ_t converges to a local optima θ^* with respect to H. Let $\omega^* = W(\theta^*)$. As θ^* is a local optimum with respect to H, by definition there exists $\epsilon > 0$, $s.t.$

$$||\theta^* - \theta||_2 < \epsilon \implies H(\theta) \leq H(\theta^*).$$

Therefore for any ω,

$$\begin{aligned} \left|\left| \begin{pmatrix} \theta^* \\ \omega^* \end{pmatrix} - \begin{pmatrix} \theta \\ \omega \end{pmatrix} \right|\right|_2 < \epsilon &\implies ||\theta^* - \theta||_2 < \epsilon \\ &\implies H(\theta) \leq H(\theta^*) \\ &\implies J(\theta, \omega) \leq J(\theta^*, \omega^*). \end{aligned}$$

Therefore by definition (θ^*, ω^*) is a local optima with respect to J. □

In summary, if we can locally optimize θ, and $\omega = W(\theta)$ at each step, then this will find a local optima for a function $H(\theta)$ and consequently for $J(\theta, \omega)$. We can make a similar argument that if the sequence θ_t converges to a global optimum with respect to H, then Q-PAMDP(1) converges to a global optimum (θ^*, ω^*). The conditions for the previous theorem can be met by assuming that P-UPDATE is a local optimization method such as gradient ascent.

One problem is that at each step we must re-learn $W(\theta)$ for the updated value of θ. Now we show that if updates to θ are bounded, and $W(\theta)$ is continuous function, then the required updates to ω will also be bounded. Intuitively, we are supposing that a small update to θ results in a small change in the weight specifying which discrete action to choose. The assumption that $W(\theta)$ is continuous is a strong one, and we do not claim that this will be satisfied by all PAMDPs. It is not necessary for the operation of Q-PAMDP(1), but it is convenient to assume that we do not have to completely re-learn ω after each update to θ.

Theorem 4.2 (Bounded Updates to ω)**.** *If W is continuous with respect to θ, and*

$$\theta_{t+1} = \theta_t + \alpha_t \textit{P-UPDATE}(\theta_t, \omega_t),$$

with $0 < ||\textit{P-UPDATE}(\theta_t, \omega_t)||_2 < \delta$ for some $\delta > 0$, then $\forall \epsilon > 0$, $\exists \alpha_0 > 0$ such that

$$\alpha_t < \alpha_0 \implies ||\omega_{t+1} - \omega_t||_2 < \epsilon.$$

Proof. Let $\epsilon > 0$. Let $\alpha_0 = \delta / ||\text{P-UPDATE}(\theta_t, \omega_t)||_2$. So if $\alpha_t < \alpha_0$,

$$\begin{aligned} \delta &> \alpha_t ||\text{P-UPDATE}(\theta_t, \omega_t)||_2 \\ &= ||\alpha_t \text{P-UPDATE}(\theta_t, \omega_t)||_2 \\ &= ||\theta_{t+1} - \theta_t||_2 . \end{aligned}$$

As W is continuous, this means that

$$||W(\theta_{t+1}) - W(\theta_t)||_2 = ||\omega_{t+1} - \omega_t||_2 < \epsilon.$$

□

In other words, if our update to θ is bounded and W is continuous, we can always adjust the learning rate α so that the difference ω_t and ω_{t+1} is bounded. If this is the case then we don't have to completely re-learn ω at each step, but only adjust slightly. This simplifies the operation of the Q-PAMDP(1) algorithm significantly.

With Q-PAMDP(1) we want P-UPDATE to optimize $H(\theta)$. One logical choice would be to use a gradient update. The next theorem shows that gradient of H is equal to the gradient of J if $\omega = W(\theta)$. This is useful as we can apply existing gradient-based methods to compute the gradient of J with respect to θ. The proof follows from the fact that we are at a global optima of J with respect to ω, and so the gradient $\nabla_\omega J$ is zero. This theorem requires that W is differentiable (and therefore also continuous).

Theorem 4.3 (Gradient of $H(\theta)$)**.** *If $J(\theta, \omega)$ is differentiable with respect to θ and ω and $W(\theta)$ is differentiable with respect to θ, then the gradient of H is given by $\nabla_\theta H(\theta) = \nabla_\theta J(\theta, \omega^*)$, where $\omega^* = W(\theta)$.*

Proof. If $\theta \in \mathbb{R}^n$ and $\omega \in \mathbb{R}^m$, then we can compute the gradient of H by the chain rule:

$$\begin{aligned} \frac{\partial H(\theta)}{\partial \theta_i} &= \frac{\partial J(\theta, W(\theta))}{\partial \theta_i} \\ &= \sum_{j=1}^{n} \frac{\partial J(\theta, \omega^*)}{\partial \theta_j} \frac{\partial \theta_j}{\partial \theta_i} + \sum_{k=1}^{m} \frac{\partial J(\theta, \omega^*)}{\partial \omega_k^*} \frac{\partial \omega_k^*}{\partial \theta_i} \\ &= \frac{\partial J(\theta, \omega^*)}{\partial \theta_i} + \sum_{k=1}^{m} \frac{\partial J(\theta, \omega^*)}{\partial \omega_k^*} \frac{\partial \omega_k^*}{\partial \theta_i}, \end{aligned}$$

where $\omega^* = W(\theta)$. Note that as by definitions of W,

$$\omega^* = W(\theta) = \arg\max_{\omega} J(\theta, \omega),$$

we have that the gradient of J with respect to ω is zero $\partial J(\theta, \omega^*) / \partial \omega_k^* = 0$, as ω is a global maximum with respect to J for fixed θ. Therefore, we have that $\nabla_\theta H(\theta) = \nabla_\theta J(\theta, \omega^*)$. □

With this result, if we perform a gradient update on θ with respect to J, and after updating $\omega = W(\theta)$, the gradient update will be the same as it would be for H. Therefore Q-PAMDP(1) with P-UPDATE being a single gradient update is equivalent to performance gradient ascent on H.

To conclude, if $W(\theta)$ is continuous and P-UPDATE converges to a global or local optimum, then Q-PAMDP(1) will converge to a global or local optimum, respectively, and the Q-LEARN step will be bounded if the update rate of the P-UPDATE step is bounded. As such, if P-UPDATE is a policy gradient update step then Q-PAMDP by Theorem 4.3 will converge to a local optimum and by Theorem 4.4 the Q-LEARN step will require a bounded number of updates. This policy gradient step can use the gradient of J with respect to θ.

Theorem 4.4 (Local Convergence of Q-PAMDP(∞))**.** *If at each step of Q-PAMDP(∞) for some bounded set Θ:*

$$\theta_{t+1} = \arg\max_{\theta \in \Theta} J(\theta, \omega_t)$$

and

$$\omega_{t+1} = W(\theta_{t+1}),$$

then Q-PAMDP(∞) converges to a local optimum.

Proof. By definition of W,

$$\omega_{t+1} = \arg\max_{\omega} J(\theta_{t+1}, \omega).$$

Therefore this algorithm takes the form of direct alternating optimization. As such, it converges to a local optimum (Bezdek and Hathaway 2002). □

Q-PAMDP(∞) has weaker convergence properties than Q-PAMDP(1), as it requires a globally converging P-STEP method. However, it has the potential to bypass nearby local optima (Bezdek and Hathaway 2002).

5 Experiments

First, we consider a simplified robo-cup problem (Kitano et al. 1997) where a single striker attempts to score a goal. Each episode starts with the player at a random position along the left bound of the field. The player starts with the ball in possession, and the keeper is positioned between the ball and the goal. The game takes place in a 2D environment where the player and the keeper have a position, velocity and orientation and the ball has a position and velocity resulting in 14 continuous state variables. An episode ends when the keeper possesses the ball, the player scores a goal, or the ball leaves the field. The reward for an action is 0 for non-terminal state, 50 for a terminal goal state, and $-d$ for a terminal non-goal state, where d is the distance of the ball to the goal. The player has two parameterized actions: kick-to(x, y), which kicks to ball towards position (x, y); and shoot-goal(h), which shoots the ball towards a position h along the goal line. If the player is not in possession of the ball, it moves towards it. The keeper has a fixed policy: it moves towards the ball, and if the player shoots at the goal, the keeper moves to intercept the ball. To score a goal, the player must shoot around the keeper. This means that at some positions we must shoot left past the keeper, and at others shoot to the right past the keeper. However at no point do we shoot at the keeper, so an optimal policy is discontinuous. This policy would be difficulty to represent in a purely continuous action space, but is simpler in a parameterized action domain. We split the action into two parameterized actions: shoot-goal-left, shoot-goal-right. This allows us to use a simple action selection policy instead of complex continuous action policy.

We represent the action-value function for the discrete action a using linear function approximation: $Q_\omega(s,a) = \omega_a^T \phi_a(s)$, where ω_a is a vector of weights, and $\phi_a(s)$ gives the features for state s. For this domain, we use Fourier basis features (Konidaris, Osentoski, and Thomas 2011). As we have 14 state variables, we must be selective in which basis functions to use. We only use basis functions with two non-zero elements and exclude all velocity state variables. We use the soft-max discrete action policy (Sutton and Barto 1998) $\pi_\omega^d(a|s) = \exp(Q_\omega(s,a)/\tau)/\sum_b \exp(Q_\omega(s,b)/\tau)$, where τ is the action selection temperature. We represent the action-parameter policy π_θ^a as a normal distribution around a weighted sum of features $\pi_\theta^a(x|s) = \mathcal{N}(\theta_a^T \psi_a(s), \Sigma)$, where θ_a is a matrix of weights, and $\psi_a(s)$ gives the features for state s, and Σ is a fixed covariance matrix. We use specialized features for each action. For the shoot-goal actions we use using a simple linear basis $(1, g)$, where g is the projection of the keeper onto the goal line. For kick-to we use linear features $(1, bx, by, bx^2, by^2, (bx-kx)/\left\|b-k\right\|_2, (by-ky)/\left\|b-k\right\|_2)$, where (bx, by) is the position of the ball and (kx, ky) is the position of the keeper. These linear features allow for a policy that directs the ball around the keeper and towards the goal.

The policy is differentiable, allowing us to use a policy gradient method. For the direct policy search approach, we use the episodic natural actor critic (eNAC) algorithm (Peters and Schaal 2008), optimizing $J(\omega, \theta)$ with respect to (ω, θ). This approach shows the performance of currently available methods. For the Q-PAMDP(1) approach we use the gradient-descent Sarsa(λ) algorithm for Q-learning, and the eNAC algorithm for policy search (Peters and Schaal 2008). At each step we perform one eNAC update based on 50 episodes and then refit Q_ω using 50 gradient descent Sarsa(λ) episodes. We also consider the same algorithms using Q-PAMDP(∞).

Return is directly correlated with goal scoring probability, so their graphs are close to indentical. We plot goal scoring probability in figure 2. We can see that direct eNAC is outperformed by Q-PAMDP(1) and Q-PAMDP(∞). This is likely due to the difficulty of optimizing the action selection parameters directly, rather than with Q-learning.

For both methods, the goal probability is greatly increased: while the initial policy rarely scores a goal, both Q-PAMDP(1) and Q-PAMDP(∞) increase the probability of a goal to roughly 35%. Direct eNAC increases the probability of scoring a goal to 15%. This suggests that a direct optimization method is insufficient to handle a parameterized action domain. Finally, we depict the performance of SARSA where the parameters are fixed at θ_0. This achieves roughly 20% goal scoring probability. Both Q-PAMDP(1) and Q-PAMDP(∞) strongly out-perform fixed parameter SARSA, but eNAC does not. Figure 3 depicts a single episode using a converged Q-PAMDP(1) policy — the player draws the keeper out and strikes when the goal is open.

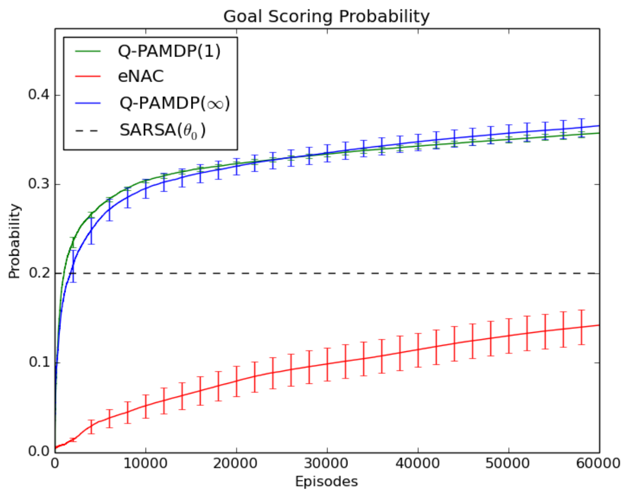


Figure 2: Average goal scoring probability, averaged over 20 runs for Q-PAMDP(1), Q-PAMDP(∞), and eNAC in the goal domain. The performance of fixed parameter SARSA using the initial parameters θ_0 is shown using a dotted black line. Intervals show standard error.

Next we consider a platformer domain. In this domain, the agent starts on a platform and must reach a goal platform while avoiding enemies. If the player reaches the goal platform, touches an enemy, or falls into a gap between platforms, the episode is ended. The reward for a step is the change in x value for that step, divided by the total length of all the platforms and gaps. The player has two primitive actions: run or jump, which continue for a fixed period or until the player lands again respectively. There are two different kinds of jumps: a high jump to get over enemies, and a long jump to get over gaps between platforms. The domain therefore has three parameterized actions: run(dx), hop(dx), and leap(dx).

The player only takes actions while on the ground, and enemies only move when the player is on their platform, the state space consists of only four variables $(x, \dot{x}, ex, \dot{ex})$. These provide the player position, player speed, enemy position, and enemy speed respectively. For learning Q_ω, as in the previous domain, we use linear function approximation with the Fourier basis. We apply a softmax discrete action policy based on Q_ω, and a Gaussian parameter policy based on parameter features $\psi_a(s)$.

6 Related Work

Hauskrecht *et al.* (2004) introduced an algorithm for solving factored MDPs with a hybrid discrete-continuous action

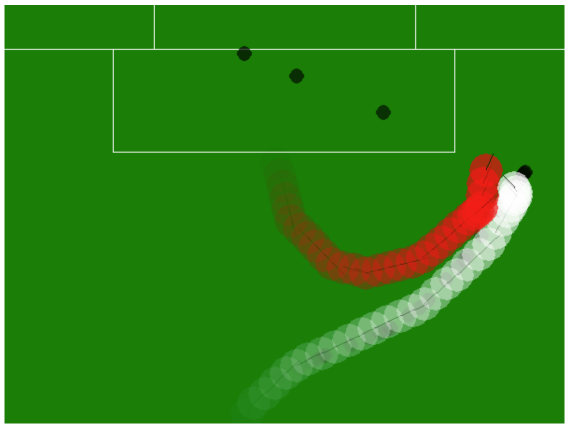


Figure 3: A robot soccer goal episode using a converged Q-PAMDP(1) policy. The player runs to one side, then shoots immediately upon overtaking the keeper.

space. However, their formalism has an action space with a mixed set of discrete and continuous components, whereas our domain has distinct actions with a different number of continuous components for each action. Furthermore, they assume the domain has a compact factored representation, and only consider planning.

Rachelson (2009) encountered parameterized actions in the form of an action to wait for a given period of time in his research on time dependent, continuous time MDPs (TMDPs). He developed XMDPs, which are TMDPs with a parameterized action space (Rachelson 2009). He developed a Bellman operator for this domain, and in a later paper mentions that the TiMDP$_{poly}$ algorithm can work with parameterized actions, although this specifically refers to the parameterized wait action (Rachelson, Fabiani, and Garcia 2009). This research also takes a planning perspective, and only considers a time dependent domain. Additionally, the size of the parameter space for the parameterized actions is the same for all actions.

Hoey *et al.* (2013) considered mixed discrete-continuous actions in their work on Bayesian affect control theory. They model affect control theory, a formalization of interpersonal interaction, as a POMDP with hybrid actions. To approach this problem they use a form of POMCP, a Monte Carlo sampling algorithm, using domain specific adjustments to compute the continuous action components (Silver and Veness 2010). They note that the discrete and continuous components of the action space reflect different control aspects: the discrete control provides the “what”, while the continuous control describes the “how” (Hoey, Schroder, and Alhothali 2013).

In their research on symbolic dynamic programming (SDP) algorithms, Zamani *et al.* (2012) considered domains with a set of discrete parameterized actions. Each of these actions has a different parameter space. Symbolic dynamic programming is a form of planning for relational or first-order MDPs, where the MDP has a set of logical relationships defining its dynamics and reward function. Their algorithms represent the value function as an extended algebraic decision diagram (XADD). As such, this work is limited to MDPs with predefined logical relations.

7 Conclusion

The PAMDP formalism models reinforcement learning domains with parameterized actions. PAMDPs allow for new kinds of domains, and for new approaches for old domains. Parameterized actions give us the adaptibility of continuous domains and the use of distinct actions. They also allow for simple representation of discontinuous policies without complex parameterizations. Three approaches for model-free learning in PAMDPs have been presented: direct optimization, and two variants of the Q-PAMDP algorithm. We have shown that Q-PAMDP(1), with an appropriate P-UPDATE method, converges to a local or global optima. Q-PAMDP(∞) with a global optimization step converges to a local optima.

We have examined the performance of the three approaches in the robot soccer goal domain. The robot soccer goal domain models the situation where a striker must out-maneuver a keeper to score a goal. Of these, Q-PAMDP(1) and Q-PAMDP(∞) outperformed eNAC. We can conclude that direct optimization is an ineffective approach for PAMDPs. Q-PAMDP(1) and Q-PAMDP(∞) performed similarly well in terms of goal scoring, learning policies that score goals roughly 35% of the time.

References

[2002] Bezdek, J., and Hathaway, R. 2002. Some notes on alternating optimization. In *Advances in Soft Computing*. Springer. 288–300.

[2012] da Silva, B.; Konidaris, G.; and Barto, A. 2012. Learning parameterized skills. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, 1679–1686.

[2014] Deisenroth, M.; Englert, P.; Peters, J.; and Fox, D. 2014. Multi-task policy search for robotics. In *Proceedings of the Fourth International Conference on Robotics and Automation*, 3876–3881. IEEE.

[2013] Deisenroth, M.; Neumann, G.; and Peters, J. 2013. *A Survey on Policy Search for Robotics*, volume 2. Now Publishers.

[2004] Guestrin, C.; Hauskrecht, M.; and Kveton, B. 2004. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, 235–242.

[2013] Hoey, J.; Schroder, T.; and Alhothali, A. 2013. Bayesian affect control theory. In *Proceedings of the Fifth International Conference on Affective Computing and Intelligent Interaction*, 166–172. IEEE.

[1997] Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; Osawa, E.; and Matsubara, H. 1997. Robocup: A challenge problem for AI. *AI Magazine* 18(1):73.

[2012] Kober, J.; Wilhelm, A.; Oztop, E.; and Peters, J. 2012. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots* 33(4):361–379.

[2011] Konidaris, G.; Osentoski, S.; and Thomas, P. 2011. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth International Conference on Artificial Intelligence*, 380–385.

[2010] Maei, H.; Szepesvári, C.; Bhatnagar, S.; and Sutton, R. 2010. Toward off-policy learning control with function approximation. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, 719–726.

[2008] Peters, J., and Schaal, S. 2008. Natural actor-critic. *Neurocomputing* 71(7):1180–1190.

[2009] Rachelson, E.; Fabiani, P.; and Garcia, F. 2009. TiMDP-poly: an improved method for solving time-dependent MDPs. In *Proceedings of the Twenty First International Conference on Tools with Artificial Intelligence*, 796–799. IEEE.

[2009] Rachelson, E. 2009. *Temporal Markov Decision Problems: Formalization and Resolution*. Ph.D. Dissertation, University of Toulouse, France.

[2010] Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, volume 23, 2164–2172.

[1998] Sutton, R., and Barto, A. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press.

[2011] Thomas, P., and Barto, A. 2011. Conjugate Markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning*, 137–144.

[1992] Watkins, C., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

[2012] Zamani, Z.; Sanner, S.; and Fang, C. 2012. Symbolic dynamic programming for continuous state and action MDPs. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

View publication stats