

# Multi-Pass Q-Networks for Deep Reinforcement Learning with Parameterised Action Spaces

Craig J. Bester<sup>1</sup>, Steven D. James<sup>1</sup>, George D. Konidaris<sup>2</sup>

<sup>1</sup>University of the Witwatersrand, Johannesburg

<sup>2</sup>Brown University, Providence RI

## Abstract

Parameterised actions in reinforcement learning are composed of discrete actions with continuous action-parameters. This provides a framework for solving complex domains that require combining high-level actions with flexible control. The recent P-DQN algorithm extends deep Q-networks to learn over such action spaces. However, it treats all action-parameters as a single joint input to the Q-network, invalidating its theoretical foundations. We analyse the issues with this approach and propose a novel method—multi-pass deep Q-networks, or MP-DQN—to address them. We empirically demonstrate that MP-DQN significantly outperforms P-DQN and other previous algorithms in terms of data efficiency and converged policy performance on the Platform, Robot Soccer Goal, and Half Field Offense domains.

## 1 Introduction

Reinforcement learning (RL) and deep RL in particular have demonstrated remarkable success in solving tasks that require either discrete actions, such as Atari [Mnih *et al.*, 2015], or continuous actions, such as robot control [Schulman *et al.*, 2015; Lillicrap *et al.*, 2016]. Reinforcement learning with parameterised actions [Masson *et al.*, 2016] that combine discrete actions with continuous action-parameters has recently emerged as an additional setting of interest, allowing agents to learn flexible behavior in tasks such as 2D robot soccer [Hausknecht and Stone, 2016a; Hussein *et al.*, 2018], simulated human-robot interaction [Khamassi *et al.*, 2017], and terrain-adaptive bipedal and quadrupedal locomotion [Peng *et al.*, 2016].

There are two main approaches to learning with parameterised actions: alternate between optimising the discrete actions and continuous action-parameters separately [Masson *et al.*, 2016; Khamassi *et al.*, 2017], or collapse the parameterised action space into a continuous one [Hausknecht and Stone, 2016a]. Both of these approaches fail to fully exploit the structure present in parameterised action problems. The former does not share information between the action and action-parameter policies, while the latter does not take into account which action-parameter is associated

with which action, or even which discrete action is executed by the agent. More recently, Xiong *et al.* [2018] introduced P-DQN, a method for learning behaviours directly in the parameterised action space. This leverages the distinct nature of the action space and is the current state-of-the-art algorithm on 2D robot soccer and King of Glory, a multiplayer online battle arena game. However, the formulation of the approach is flawed due to the dependence of the discrete action values on all action-parameters, not only those associated with each action. In this paper, we show how the above issue leads to suboptimal decision-making. We then introduce a novel multi-pass method to separate action-parameters, and demonstrate that the resulting algorithm—MP-DQN—outperforms existing methods on the Platform, Robot Soccer Goal, and Half Field Offense domains.

## 2 Background

Parameterised action spaces [Masson *et al.*, 2016] consist of a set of discrete actions,  $\mathcal{A}_d = [K] = \{k_1, k_2, \dots, k_K\}$ , where each  $k$  has a corresponding continuous action-parameter  $x_k \in \mathcal{X}_k \subseteq \mathbb{R}^{m_k}$  with dimensionality  $m_k$ . This can be written as

$$\mathcal{A} = \bigcup_{k \in [K]} \{a_k = (k, x_k) | x_k \in \mathcal{X}_k\}. \quad (1)$$

We consider environments modelled as a Parameterised Action Markov Decision Process (PAMDP) [Masson *et al.*, 2016]. For a PAMDP  $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ :  $\mathcal{S}$  is the set of all states,  $\mathcal{A}$  is the parameterised action space,  $P(s' | s, k, x_k)$  is the Markov state transition probability function,  $R(s, k, x_k, s')$  is the reward function, and  $\gamma \in [0, 1)$  is the future reward discount factor. An action policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  maps states to actions, typically with the aim of maximising Q-values  $Q(s, a)$ , which give the expected discounted return of executing action  $a$  in state  $s$  and following the current policy thereafter.

The Q-PAMDP algorithm [Masson *et al.*, 2016] alternates between learning a discrete action policy with fixed action-parameters using Sarsa( $\lambda$ ) [Sutton and Barto, 1998] with the Fourier basis [Konidaris *et al.*, 2011] and optimising the continuous action-parameters using episodic Natural Actor Critic (eNAC) [Peters and Schaal, 2008] while the discrete action policy is kept fixed. Hausknecht and Stone [2016a] apply artificial neural networks and the Deep Deterministic Policy

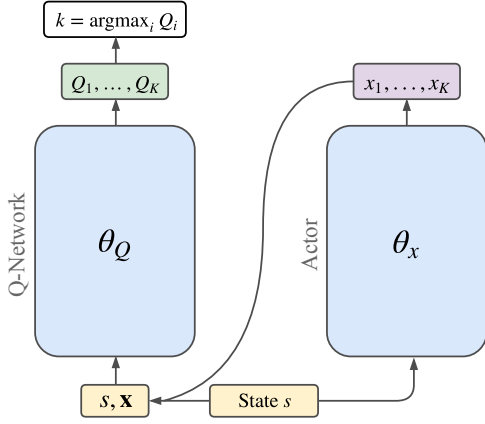


Figure 1: The P-DQN network architecture [Xiong *et al.*, 2018]. Note that the joint action-parameter vector  $\mathbf{x}$  is fed into the Q-network.

Gradients (DDPG) algorithm [Lillicrap *et al.*, 2016] to parameterised action spaces by treating both the discrete actions and their action-parameters as a joint continuous action vector. This can be seen as relaxing the parameterised action space (Equation 1) into a continuous one:

$$\mathcal{A} = \{(f_{1:K}, x_{1:K}) | f_k \in \mathbb{R}, x_k \in \mathcal{X}_k \forall k \in [K]\}, \quad (2)$$

where  $f_1, f_2, \dots, f_K$  are continuous values in  $[-1, 1]$ . An  $\epsilon$ -greedy or softmax policy is then used to select discrete actions. However, not only does this fail to exploit the disjoint nature of different parameterised actions, but optimising over the joint action and action-parameter space can result in premature convergence to suboptimal policies, as occurred in experiments by Masson *et al.* [2016]. We henceforth refer to the algorithm used by Hausknecht and Stone as PA-DDPG.

## 2.1 Parameterised Deep Q-Networks

Unlike previous approaches, Xiong *et al.* [2018] introduce a method that operates in the parameterised action space directly by combining DQN and DDPG. Their P-DQN algorithm achieves state-of-the-art performance using a Q-network to approximate Q-values used for discrete action selection, in addition to providing critic gradients for an actor network that determines the continuous action-parameter values for all actions. By framing the problem as a PAMDP directly, rather than alternating between discrete and continuous action MDPs as with Q-PAMDP, or using a joint continuous action MDP as with PA-DDPG, P-DQN necessitates a change to the Bellman equation to incorporate continuous action-parameters:

$$Q(s, k, x_k) = \mathbb{E}_{r, s'} \left[ r + \gamma \max_{k'} \sup_{x_{k'} \in \mathcal{X}_{k'}} Q(s', k', x_{k'}) \middle| s, k, x_k \right]. \quad (3)$$

To avoid the computationally intractable calculation of the supremum over  $\mathcal{X}_k$ , Xiong *et al.* [2018] state that when the Q function is fixed, one can view  $\operatorname{argsup}_{x_k \in \mathcal{X}_k} Q(s, k, x_k)$  as a function  $x_k^Q : S \rightarrow \mathcal{X}_k$  for any state  $s \in S$  and  $k \in [K]$ . This

allows the Bellman equation to be rewritten as:

$$Q(s, k, x_k) = \mathbb{E}_{r, s'} \left[ r + \gamma \max_{k'} Q(s', k', x_{k'}^Q(s')) \middle| s, k, x_k \right]. \quad (4)$$

P-DQN uses a deep neural network with parameters  $\theta_Q$  to represent  $Q(s, k, x_k; \theta_Q)$ , and a second deterministic actor network with parameters  $\theta_x$  to represent the action-parameter policy  $x_k(s; \theta_x) : S \rightarrow \mathcal{X}_k$ , an approximation of  $x_k^Q(s)$ . With this formulation it is easy to apply the standard DQN approach of minimising the mean-squared Bellman error to update the Q-network using minibatches sampled from replay memory  $D$  [Mnih *et al.*, 2015], replacing  $a$  with  $(k, x_k)$ :

$$L_Q(\theta_Q) = \mathbb{E}_{(s, k, x_k, r, s') \sim D} \left[ \frac{1}{2} (y - Q(s, k, x_k; \theta_Q))^2 \right], \quad (5)$$

where  $y = r + \gamma \max_{k' \in [K]} Q(s', k', x_{k'}(s'; \theta_x); \theta_Q)$  is the update target derived from Equation (4). Then, the loss for the actor network in P-DQN is given by the negative sum of Q-values:

$$L_x(\theta_x) = \mathbb{E}_{s \sim D} \left[ - \sum_{k=1}^K Q(s, k, x_k(s; \theta_x); \theta_Q) \right]. \quad (6)$$

Although this choice of loss function was not motivated by Xiong *et al.* [2018], it resembles the deterministic policy gradient loss used by PA-DDPG where a scalar critic value is used over all action-parameters [Hausknecht and Stone, 2016a]. During updates, the estimated Q-values are back-propagated through the critic to the actor, producing gradients indicating how the action-parameters should be updated to increase the Q-values.

## 3 Problems with Joint Action-Parameters

The P-DQN architecture inputs the joint action-parameter vector over all actions to the Q-network, as illustrated in Figure 1. This was pointed out by Xiong *et al.* [2018] but they did not discuss it further. While this may seem like an inconsequential implementation detail, it changes the formulation of the Bellman equation used for parameterised actions (Equation 4) since each Q-value is a function of the joint action-parameter vector  $\mathbf{x} = (x_1, \dots, x_K)$ , rather than only the action-parameter  $x_k$  corresponding to the associated action:

$$Q(s, k, \mathbf{x}) = \mathbb{E}_{r, s'} \left[ r + \gamma \max_{k'} Q(s', k', \mathbf{x}^Q(s')) \middle| s, k, \mathbf{x} \right]. \quad (7)$$

This in turn affects both the updates to the Q-values and the action-parameters. Firstly, we consider the effect on the action-parameter loss, specifically that each Q-value produces gradients for all action-parameters. Consider for demonstration purposes the action-parameter loss (Equation 6) over a single sample with state  $s$ :

$$L_x(\theta_x) = - \sum_{k=1}^K Q(s, k, \mathbf{x}(s; \theta_x); \theta_Q). \quad (8)$$

The policy gradient is then given by:

$$\nabla_{\theta_x} \mathbf{x}(s; \theta_x) = - \sum_{k=1}^K \nabla_{\mathbf{x}} Q(s, k, \mathbf{x}(s; \theta_x); \theta_Q) \nabla_{\theta_x} \mathbf{x}(s; \theta_x). \quad (9)$$

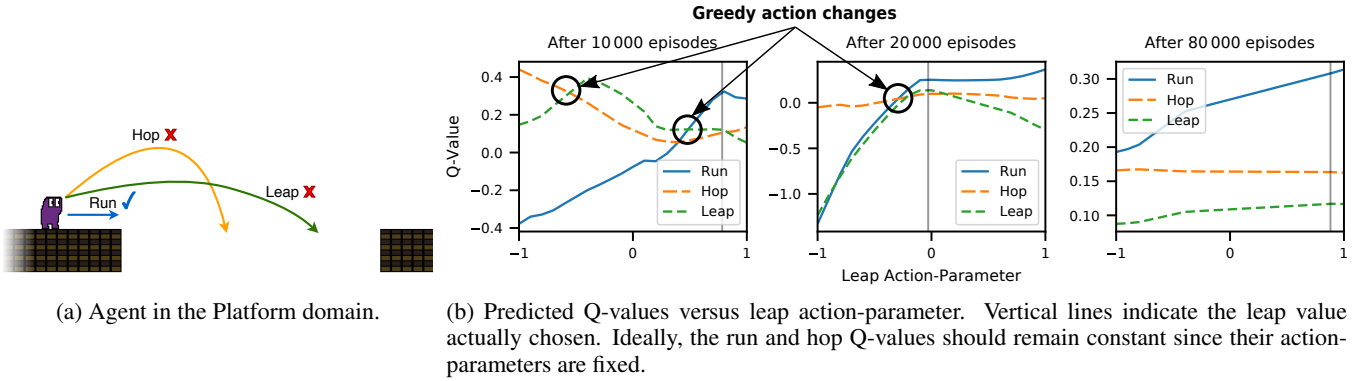


Figure 2: Example of dependence on unrelated action-parameters affecting discrete action selection on the Platform domain. Three parameterised actions are available: *run*, *hop*, and *leap*. In a particular state (a), the optimal action is to run forward to be able to traverse a gap, while choosing to leap would cause the agent to fall and die. The Q-value of the leap action should change with its action-parameter, but (b) shows that varying the leap action-parameter while the others are kept fixed changes the Q-values predicted by P-DQN for *all* actions. Near the start of training, this can alter the discrete policy such that a suboptimal action is chosen. After 80 000 episodes, P-DQN correctly learns to choose the optimal action regardless of the unrelated leap action-parameter, although the other Q-values still vary.

Expanding the gradients with respect to the action-parameters gives

$$\nabla_{\mathbf{x}} Q = \left( \frac{\partial Q_1}{\partial x_1} + \frac{\partial Q_2}{\partial x_1} + \dots + \frac{\partial Q_K}{\partial x_1}, \dots, \frac{\partial Q_1}{\partial x_K} + \dots + \frac{\partial Q_K}{\partial x_K} \right), \quad (10)$$

where  $Q_k = Q(s, k, \mathbf{x}(s; \theta_x); \theta_Q)$ . Theoretically, if each Q-value were a function of just  $x_k$  as the P-DQN formulation intended, then  $\partial Q_k / \partial x_j = 0 \forall k, j \in [K], j \neq k$  and  $\nabla_{\mathbf{x}} Q$  simplifies to:

$$\nabla_{\mathbf{x}} Q = \left( \frac{\partial Q_1}{\partial x_1}, \frac{\partial Q_2}{\partial x_2}, \dots, \frac{\partial Q_K}{\partial x_K} \right). \quad (11)$$

However this is not the case in P-DQN, so the gradients with respect to other action-parameters  $\partial Q_k / \partial x_j$  are not zero in general. This is a problem because each Q-value is updated only when its corresponding action is sampled, as per Equation 5, and thus has no information on what effect other action-parameters  $x_j, j \neq k$  have on transitions or how they should be updated to maximise the expected return. They therefore produce what we term *false gradients*. This effect may be mitigated by the summation over all Q-values in the action-parameter loss, since the gradients from each Q-value are summed and averaged over a minibatch.

The dependence of Q-values on all action-parameters also negatively affects the discrete action policy. Specifically, updating the continuous action-parameter policy of any action perturbs the Q-values of *all* actions, not just the one associated with that action-parameter. This can lead to the relative ordering of Q-values changing, which in turn can result in suboptimal greedy action selection. We demonstrate a situation where this occurs on the Platform domain in Figure 2.

## 4 Multi-Pass Q-Networks

The naïve solution to the problem of joint action-parameter inputs in P-DQN would be to split the Q-network into separate networks for each discrete action. Then, one can input only the state and relevant action-parameter  $x_k$  to the network corresponding to  $Q_k$ . However, this drastically increases the

computational and space complexity of the algorithm due to the duplication of network parameters for each action. Furthermore, the loss of the shared feature representation between Q-values may be detrimental.

We therefore consider an alternative approach that does not involve architectural changes to the network structure of P-DQN. While separating the action-parameters in a single forward pass of a single Q-network with fully connected layers is impossible, we can do so with multiple passes. We perform a forward pass once per action  $k$  with the state  $s$  and action-parameter vector  $\mathbf{x}e_k$  as input, where  $e_k$  is the standard basis vector for dimension  $k$ . Thus  $\mathbf{x}e_k = (0, \dots, 0, x_k, 0, \dots, 0)$  is the joint action-parameter vector where each  $x_j, j \neq k$  is set to zero. This causes all false gradients to be zero,  $\partial Q_k / \partial x_j = 0$ , and completely negates the impact of the network weights for unassociated action-parameters  $x_j$  from the input layer, making  $Q_k$  only depend on  $x_k$ . That is,

$$Q(s, k, \mathbf{x}e_k) \cong Q(s, k, x_k). \quad (12)$$

Both problems are therefore addressed without introducing any additional neural network parameters. We refer to this as the *multi-pass Q-network* method, or MP-DQN.

A total of  $K$  forward passes are required to predict all Q-values instead of one. However, we can make use of the parallel minibatch processing capabilities of artificial neural networks, provided by libraries such as PyTorch and Tensorflow, to perform this in a single parallel pass, or *multi-pass*. A multi-pass with  $K$  actions is processed in the same manner as a minibatch of size  $K$ :

$$\begin{pmatrix} Q(s, \cdot, \mathbf{x}e_1; \theta_Q) \\ \vdots \\ Q(s, \cdot, \mathbf{x}e_K; \theta_Q) \end{pmatrix} = \begin{pmatrix} Q_{11} & Q_{12} & \dots & Q_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{K1} & Q_{K2} & \dots & Q_{KK} \end{pmatrix}, \quad (13)$$

where  $Q_{ij}$  is the Q-value for action  $j$  generated on the  $i^{\text{th}}$  pass where  $x_i$  is non-zero. Only the diagonal elements  $Q_{ii}$  are valid and used in the final output  $Q_i \leftarrow Q_{ii}$ . This process is illustrated in Figure 3.

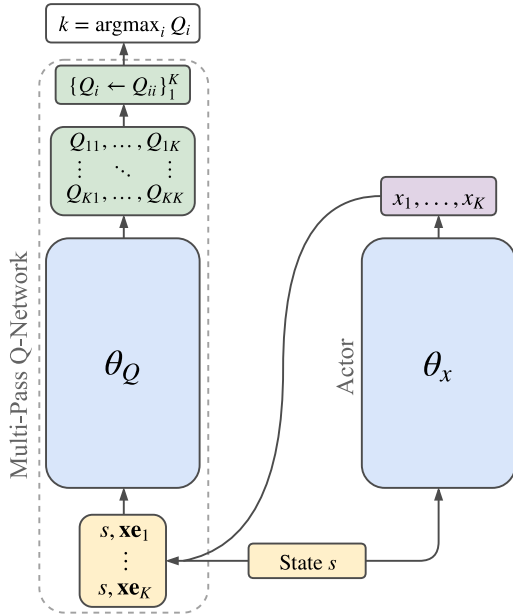


Figure 3: Illustration of the multi-pass Q-network architecture.

Compared to separate Q-networks, our multi-pass technique introduces a relatively minor amount of overhead during forward passes. Although minibatches for updates are similarly duplicated  $K$  times, backward passes to accumulate gradients are not duplicated since only the diagonal elements  $Q_{ii}$  are used in the loss function. The computational complexity of this overhead scales linearly with the number of actions and minibatch size during updates. Unlike separate Q-networks (and even when a larger Q-network with more hidden layers and neurons is used) if the number of actions does not change, then the overhead of multi-passes would be the same as with a smaller Q-network, provided the minibatch is of a reasonable size and can be processed in parallel.

## 5 Experiments

We compare the original P-DQN algorithm with a single Q-network against our proposed multi-pass Q-network (MP-DQN), as well as against separate Q-networks (SP-DQN). We also compare against Q-PAMDP and PA-DDPG, the former state-of-the-art approaches on their respective domains. We are unable to use King of Glory as a benchmark domain as it is closed-source and proprietary.

Similar to Mnih *et al.* [2015] and Hausknecht and Stone [2016a], we add target networks to P-DQN to compute the update targets  $y$  for stability. Soft updates (Polyak averaging) are used for the target networks. Adam [Kingma and Ba, 2014] with  $\beta_1 = 0.9, \beta_2 = 0.999$  is used to optimise the neural network parameters for P-DQN and PA-DDPG. Layer weights are initialised following the strategy of He *et al.* [2015] with rectified linear unit (ReLU) activation functions. We employ the inverting gradients approach to bound action-parameters for both algorithms, as Hausknecht and Stone [2016a] claim PA-DDPG is unable to learn without it on Half Field Offense. Action-parameters are scaled to

$[-1, 1]$ , as we found this increased performance for all algorithms.

We perform a hyperparameter grid search for Platform and Robot Soccer Goal over: the network learning rates  $\alpha_Q, \alpha_x \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$  s.t.  $\alpha_x \leq \alpha_Q$ ; Polyak averaging factors  $\tau_Q, \tau_x \in \{0.1, 0.01, 0.001\}$  s.t.  $\tau_x \leq \tau_Q$ ; minibatch size  $B \in \{32, 64, 128\}$ ; and number of hidden layers and neurons in  $\{(256, 128), (128, 64), (256), (128)\}$ . The hidden layers are kept symmetric between the actor and critic networks as in previous works. Each combination is tested over 5 random runs for P-DQN and PA-DDPG separately on each domain. The same hyperparameters are used for P-DQN, SP-DQN and MP-DQN.

To keep the comparison with PA-DDPG fair, we do not use dueling networks [Wang *et al.*, 2016] nor asynchronous parallel workers as Xiong *et al.* [2018] used for P-DQN. For each algorithm and domain, we train 30 agents with unique random seeds and evaluate them without exploration for another 1000 episodes. Our experiments are implemented in Python using PyTorch [Paszke *et al.*, 2017] and OpenAI Gym [Brockman *et al.*, 2016], and run on the following hardware: Intel Core i7-7700, 16GB DRAM, NVidia GTX 1060 GPU. Complete source code is available online.<sup>1</sup>

### 5.1 Platform

The Platform domain [Masson *et al.*, 2016] has three actions—run, hop, and leap—each with a continuous action-parameter to control horizontal displacement. The agent has to hop over enemies and leap across gaps between platforms to reach the goal state. The agent dies if it touches an enemy or falls into a gap. A 9-dimensional state space gives the position and velocity of the agent and local enemy along with features of the current platform such as length.

We train agents on this domain for 80 000 episodes, using the same hyperparameters for Q-PAMDP as Masson *et al.* [2016], except we reduce the learning rate for eNAC ( $\alpha_{\text{eNAC}}$ ) to 0.1, and exploration noise variance ( $\sigma$ ) to 0.0001, to account for the scaled action-parameters. For P-DQN, shallow networks with one hidden layer (128) were found to perform best with  $\alpha_Q = 10^{-3}, \alpha_x = 10^{-4}, \tau_Q = 0.1, \tau_x = 0.001$ , and  $B = 128$ . PA-DDPG uses two hidden layers (256, 128) with  $\alpha_Q = 10^{-3}, \alpha_\mu = 10^{-4}, \tau_Q = 0.01, \tau_\mu = 0.01$ , and  $B = 32$ . A replay memory size of 10 000 samples is used for both algorithms, update gradients are clipped at 10, and  $\gamma = 0.9$ .

We introduce a *passthrough layer* to the actor networks of P-DQN and PA-DDPG to initialise their action-parameter policies to the same linear combination of state variables that Masson *et al.* [2016] use to initialise the Q-PAMDP policy. The weights of the passthrough layer are kept fixed to avoid instability; this does not reduce the range of action-parameters available as the output of the actor network compensates before inverting gradients are applied. We use an  $\epsilon$ -greedy discrete action policy with additive Ornstein-Uhlenbeck noise for action-parameter exploration, similar to Lillicrap *et al.* [2016], which we found gives slightly better performance than Gaussian noise.

<sup>1</sup><https://github.com/cycraig/MP-DQN>

## 5.2 Robot Soccer Goal

The Robot Soccer Goal domain [Masson *et al.*, 2016] is a simplification of RoboCup 2D [Kitano *et al.*, 1997] in which an agent has to score a goal past a keeper that tries to intercept the ball. The three parameterised actions—kick-to, shoot-goal-left, and shoot-goal-right—are all related to kicking the ball, which the agent automatically approaches between actions until close enough to kick again. The state space consists of 14 continuous features describing the position, velocity, and orientation of the agent and keeper, and the ball’s position and distance to the keeper and goal.

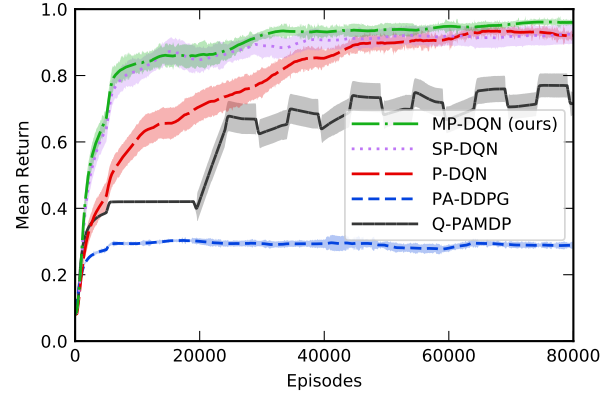
Training consisted of 100 000 episodes, using the same hyperparameters for Q-PAMDP as Masson *et al.* [2016] except we set  $\alpha_{eNAC} = 0.06$  and  $\sigma = 0.0001$ . P-DQN uses a single hidden layer (256), with  $\alpha_Q = 10^{-3}$ ,  $\alpha_x = 10^{-5}$ ,  $\tau_Q = 0.1$ ,  $\tau_x = 0.001$ , and  $B = 128$ . Two hidden layers (128, 64) are used for PA-DDPG, with  $\alpha_Q = 10^{-4}$ ,  $\alpha_\mu = 10^{-5}$ ,  $\tau_Q = 0.01$ ,  $\tau_\mu = 0.01$ , and  $B = 64$ . Both algorithms use a replay memory size of 20 000,  $\gamma = 0.95$ , gradients clipping at 1, and the same action-parameter policy initialisation as Q-PAMDP with additive Ornstein-Uhlenbeck noise.

## 5.3 Half Field Offense

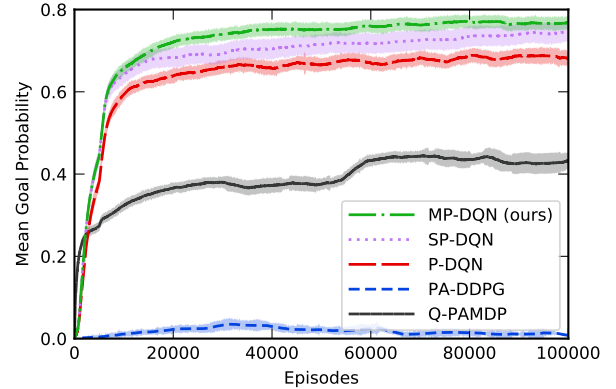
The third and final domain, Half Field Offense (HFO) [Hausknecht and Stone, 2016a], is also the most complex. It has 58 state features and three parameterised actions available: dash, turn, and kick. Unlike Robot Soccer Goal, the agent must first learn to approach the ball and then kick it into the goals, although there is no keeper in this task.

We use 30 000 episodes for training on HFO. This is more than the 20 000 episodes (or roughly 3 million transitions) used by Hausknecht and Stone [2016a] and Xiong *et al.* [2018] so that ample opportunity is given for the algorithms to converge in order to fairly evaluate the final policy performance. We use the same network structure as previous works with hidden layers of (256, 128, 64) neurons for P-DQN and (1024, 512, 256, 128) neurons for PA-DDPG. The leaky ReLU activation function with negative slope 0.01 is used on HFO because of these deeper networks. Xiong *et al.* [2018] use 24 asynchronous parallel workers for  $n$ -step returns on HFO. For fair comparison and due to the lack of sufficient hardware, we instead use mixed  $n$ -step return targets [Hausknecht and Stone, 2016b] with a mixing ratio of  $\beta = 0.25$  for both P-DQN and PA-DDPG, as this technique does not require multiple workers. The  $\beta$  value was selected after a search over  $\beta \in \{0, 0.25, 0.5, 0.75, 1\}$ . We otherwise use the same hyperparameters as Hausknecht and Stone [2016b] apart from the network learning rates:  $\alpha_Q = 10^{-3}$ ,  $\alpha_x = 10^{-5}$  for P-DQN and  $\alpha_Q = 10^{-3}$ ,  $\alpha_\mu = 10^{-3}$  for PA-DDPG. In the absence of an initial action-parameter policy, we use the same  $\epsilon$ -greedy with uniform random action-parameter exploration strategy as the original authors. In general we kept as many factors consistent between the two algorithms as possible for a fair comparison.

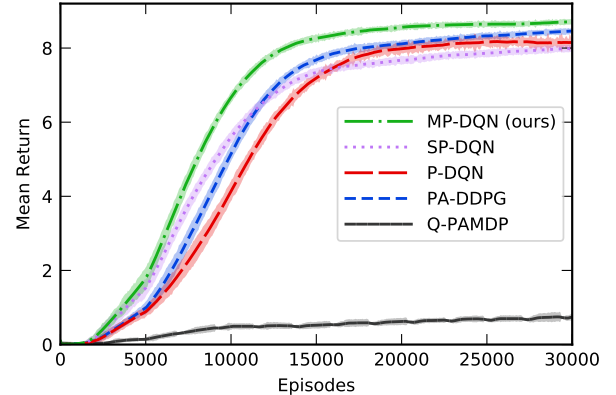
We select 10 of the most relevant state features for Q-PAMDP to avoid intractable Fourier basis calculations. These features include: player orientation, stamina, proximity to ball, ball angle, ball-kickable, goal centre position, and goal centre proximity. Even with this reduced selection, we



(a) Platform



(b) Robot Soccer Goal



(c) Half Field Offense

Figure 4: Learning curves on Platform (a), Robot Soccer Goal (b), and Half Field Offense (c). The running average scores—episodic return for Platform and HFO, and goal scoring probability for Robot Soccer Goal—are smoothed over 5000 episodes and include random exploration, higher is better. Shaded areas represent standard error of the running averages over the different agents. The oscillating behaviour of Q-PAMDP on Platform is a result of re-exploration between the alternating optimisation steps. MP-DQN clearly performs best overall.

	Platform	Robot Soccer Goal	Half Field Offense	
	Return	P(Goal)	P(Goal)	Avg. Steps to Goal
Q-PAMDP	0.789 ± 0.188	0.452 ± 0.093	0 ± 0	n/a
PA-DDPG	0.284 ± 0.061	0.006 ± 0.020	0.875 ± 0.182	<b>95 ± 7</b>
P-DQN	0.964 ± 0.068	0.701 ± 0.078	0.883 ± 0.085	111 ± 11
SP-DQN	0.941 ± 0.164	0.752 ± 0.131	0.718 ± 0.131	99 ± 7
MP-DQN	<b>0.987 ± 0.039</b>	<b>0.789 ± 0.070</b>	<b>0.913 ± 0.070</b>	99 ± 12
PA-DDPG <sup>2</sup>	-	-	0.923 ± 0.073	112 ± 5
Async. P-DQN <sup>3</sup>	-	-	0.989 ± 0.006	81 ± 3

Table 1: Mean evaluation scores over 30 random runs for each algorithm, averaged over 1000 episodes after training with no random exploration. We include previously published results from Hausknecht and Stone [2016a] and Xiong *et al.* [2018] on HFO, although they are not directly comparable with ours as we use a longer training period and have a much larger sample size of agents—30 versus 7 and 9 respectively—and asynchronous P-DQN uses 24 parallel workers to implement  $n$ -step returns rather than the mixing strategy we use.

found at most a Fourier basis of order 2 could be used. We use an adaptive step-size [Dabney and Barto, 2012] for Sarsa( $\lambda$ ) with an eNAC learning rate of 0.2. The Q-PAMDP agent initially learns with Sarsa( $\lambda$ ) for a period of 1000 episodes before alternating between  $\kappa = 50$  eNAC updates of 25 rollouts each, and 1000 episodes of discrete action re-exploration.

## 6 Results

The resulting learning curves of MP-DQN, SP-DQN, P-DQN, PA-DDPG, and Q-PAMDP on the three parameterised action benchmark domains are shown in Figure 4, with mean evaluation scores detailed in Table 1.

Our results show that MP-DQN learns significantly faster than baseline P-DQN with joint action-parameter inputs and achieves the highest mean evaluation scores across all three domains. SP-DQN similarly shows better performance than P-DQN on Platform and Robot Soccer Goal but to a slightly lesser extent than MP-DQN. Notably, SP-DQN exhibits fast initial learning on HFO but plateaus at a lower performance level than P-DQN. This is likely due to the aforementioned lack of a shared feature representation between the separate Q-networks and the duplicate network parameters which require more updates to optimise.

In general, we observe that P-DQN and its variants outperform Q-PAMDP on Platform and Robot Soccer Goal, while PA-DDPG consistently converges prematurely to suboptimal policies. Wei *et al.* [2018] observe similar behaviour for PA-DDPG on Platform. This highlights the problem with updating the action and action-parameter policies simultaneously and was also observed when using eNAC for direct policy search on Platform [Masson *et al.*, 2016]. On HFO, Q-PAMDP fails to learn to score any goals—likely due to its reduced feature space and use of linear function approximation rather than neural networks. Unexpectedly, baseline P-DQN appears to learn slower than PA-DDPG on HFO. This suggests that the dueling networks and asynchronous parallel workers used by Xiong *et al.* [2018] were major factors improving P-DQN in their comparisons.

<sup>2</sup>Average over 7 runs [Hausknecht and Stone, 2016a].

<sup>3</sup>Average over 9 runs with 24 workers [Xiong *et al.*, 2018].

## 7 Related Work

Many recent deep RL approaches follow the strategy of collapsing the parameterised action space into a continuous one. Hussein *et al.* [2018] present a deep imitation learning approach for scoring goals on HFO using long-short-term-memory networks with a joint action and action-parameter policy. Agarwal [2018] introduces skills for multi-goal parameterised action space environments to achieve multiple related goals; they demonstrate success on robotic manipulation tasks by combining PA-DDPG with hindsight experience replay and their skill library.

One can alternatively view parameterised actions as a 2-level hierarchy: Klimek *et al.* [2017] use this approach to learn a reach-and-grip task using a single network to represent a distribution over macro (discrete) actions and their lower-level action-parameters. The work most relevant to this paper is by Wei *et al.* [2018], who introduce a parameterised action version of TRPO (PATRPO). They also take a hierarchical approach but instead condition the action-parameter policy on the discrete action chosen to avoid predicting all action-parameters at once. While their preliminary results show the method achieves good performance on Platform, we omit comparison with PATRPO as it fails to learn to score goals on HFO.

## 8 Conclusion

We identified a significant problem with the P-DQN algorithm for parameterised action spaces: the dependence of its Q-values on all action-parameters causes false gradients and can lead to suboptimal action selection. We introduced a new algorithm, MP-DQN, with separate action-parameter inputs which demonstrated superior performance over P-DQN and former state-of-the-art techniques Q-PAMDP and PA-DDPG. We also found that PA-DDPG was unstable and converged to suboptimal policies on some domains. Our results suggest that future approaches should leverage the disjoint nature of parameterised action spaces and avoid simultaneous optimisation of the policies for discrete actions and continuous action-parameters.

## Acknowledgments

This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number: 113737).

## References

- [Agarwal, 2018] Arpit Agarwal. Deep reinforcement learning with skill library: Exploring with temporal abstractions and coarse approximate dynamics models. Master’s thesis, Carnegie Mellon University, Pittsburgh, PA, July 2018.
- [Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [Dabney and Barto, 2012] William Dabney and Andrew G Barto. Adaptive step-size for online temporal difference learning. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- [Hausknecht and Stone, 2016a] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations*, 2016.
- [Hausknecht and Stone, 2016b] Matthew Hausknecht and Peter Stone. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI Workshop*, July 2016.
- [He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [Hussein *et al.*, 2018] Ahmed Hussein, Eyad Elyan, and Chrisina Jayne. Deep imitation learning with memory for Robocup soccer simulation. In *Proceedings of the International Conference on Engineering Applications of Neural Networks*, pages 31–43. Springer, 2018.
- [Khamassi *et al.*, 2017] Mehdi Khamassi, George Velentzas, Theodore Tsitsimis, and Costas Tzafestas. Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task. In *Proceedings of the First IEEE International Conference on Robotic Computing*, pages 28–35. IEEE, 2017.
- [Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [Kitano *et al.*, 1997] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for AI. *AI Magazine*, 18:73–85, 1997.
- [Klimek *et al.*, 2017] Maciej Klimek, Henryk Michalewski, and Piotr Miłoś. Hierarchical reinforcement learning with parameters. In *Conference on Robot Learning*, pages 301–313, 2017.
- [Konidaris *et al.*, 2011] George D. Konidaris, Sarah Osentoski, and Philip S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, August 2011.
- [Lillicrap *et al.*, 2016] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.
- [Masson *et al.*, 2016] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1934–1940, 2016.
- [Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, 2017.
- [Peng *et al.*, 2016] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics*, 35(4):81:1–81:12, 2016.
- [Peters and Schaal, 2008] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.
- [Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference of Machine Learning*, volume 37, pages 1889–1897, 2015.
- [Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.
- [Wang *et al.*, 2016] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, pages 1995–2003, 2016.
- [Wei *et al.*, 2018] Ermo Wei, Drew Wicke, and Sean Luke. Hierarchical approaches for reinforcement learning in parameterized action space. In *2018 AAAI Spring Symposium Series*, 2018.
- [Xiong *et al.*, 2018] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep Q-networks learning: Reinforcement learning with

discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.