Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space

Zhou Fan[*] , **Rui Su**[*] , **Weinan Zhang**[†] and **Yong Yu**
Shanghai Jiao Tong University
zhou.fan@sjtu.edu.cn, {surui, wnzhang, yyu}@apex.sjtu.edu.cn

Abstract

In this paper we propose a hybrid architecture of actor-critic algorithms for reinforcement learning in parameterized action space, which consists of multiple parallel sub-actor networks to decompose the structured action space into simpler action spaces along with a critic network to guide the training of all sub-actor networks. While this paper is mainly focused on parameterized action space, the proposed architecture, which we call hybrid actor-critic, can be extended for more general action spaces which has a hierarchical structure. We present an instance of the hybrid actor-critic architecture based on proximal policy optimization (PPO), which we refer to as hybrid proximal policy optimization (H-PPO). Our experiments test H-PPO on a collection of tasks with parameterized action space, where H-PPO demonstrates superior performance over previous methods of parameterized action reinforcement learning.

1 Introduction

Reinforcement learning (RL) has achieved impressive performance on a wide range of tasks including game playing, robotics and natural language processing. Most of recent exciting achievements is obtained by the combination of deep learning and reinforcement learning, known as deep reinforcement learning [Mnih *et al.*, 2013]. In game playing domains, deep Q-network (DQN) [Mnih *et al.*, 2013] is capable of learning control policies directly from high-dimensional sensory input in Atari games, and AlphaGo [Silver *et al.*, 2016] has defeated world champions in the game of Go and could achieve superhuman performance even without human knowledge for training [Silver *et al.*, 2017]. Robotics is also a significant aspect of applications of RL, where RL enables a robot to autonomously learn a sophisticated behavior through interactions with its environment [Kober *et al.*, 2013].

In the general setup of RL, an agent interacts with an environment in the following way: at each time step t, it observes (either fully or partially) a state s_t and takes an action a_t, then receives a reward signal r_t as well as the next state s_{t+1}. Here the action a_t is selected by the agent from its action space $\mathcal{A}$. The type of the action space is an important characteristic of the setup of an RL problem, and problems with different types of action space are usually solved with different algorithms. A typical RL setup may come with a discrete action space or a continuous one, and most RL algorithms are designed for either one of these two types. The agent simply selects its actions from a finite set of discrete actions if the action space is discrete, or from a single continuous space in the case of a continuous action space. However, action space could also have some hierarchical structure instead of being a flat set. The most common class of structured action space is known as parameterized action spaces, where a parameterized action is a discrete action parameterized by a continuous real-valued vector [Masson *et al.*, 2016]. With a parameterized action space, the agent not only selects an action from a discrete set, but selects the parameter to use with that action from the continuous parameter set of that action as well.

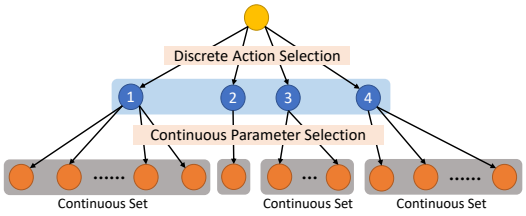


Figure 1: Illustration of a parameterized action space.

Figure 1 shows an example of parameterized action space. The hierarchically structured action space contains four types of discrete actions shown in blue, and every discrete action has a continuous parameter space marked with rounded rectangles in grey. In this example, the discrete action with index 2 is actually not parameterized. It can also be viewed as a special case that the parameter space of discrete action 2 only has one element. Parameterized action space perfectly models the scenarios where there are different categories of continuous actions. Many games as well as real world tasks have a parameterized action space. For example, in the Half Field Offense (HFO) [Hausknecht *et al.*, 2016] domain, which is a subtask based on the RoboCup 2D simulation platform, the agent may choose the discrete action Kick and specify its real-valued parameters (power and direction). Moreover, pa-

[*]Equal contribution.
[†]Corresponding author.

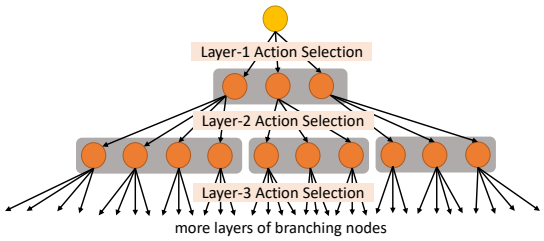


Figure 2: A hierarchically structured action space.

rameterized actions naturally exist in the context of robotics, where the action space can be constructed in a way that a set of meta-actions defines the higher-level selection of actions and every meta-action is controlled with fine-grained parameters [Kober *et al.*, 2013].

In addition to parameterized action spaces, action spaces may have more general hierarchical structures. For example, the parameters for the different actions are discretized in some game environments such as StarCraft II Learning Environment [Vinyals *et al.*, 2017]. Also, the action space may be manually constructed to have a hierarchical structure of more than two layers, which is a technique often used to reduce the size of an extremely large action space, with OpenAI Five[1] on Dota 2 as a remarkable example. While it is intractable to choose an action directly from a set that contains millions of discrete actions, we can tackle this problem by constructing a hierarchically structured action space with a hierarchical taxonomy. As is shown in Figure 2, the action space has a tree structure of multi-layer classifications of actions, in a way that each action selection node only has a small number of branches. Note that this tree structure could have more than two layers, and the external nodes of the tree structure could be continuous action-selection instead of discrete branching. In this view, the parameterized action space is a special case of hierarchical action space which has a discrete layer and then a continuous layer.

In this work, we propose a hybrid architecture of actor-critic algorithms for RL in parameterized action space. It is based on original architecture of actor-critic algorithms [Konda and Tsitsiklis, 2000], but contains multiple parallel sub-actor networks instead of one to solve multi-layer action selection respectively and has one global critic network to update the policy parameters of all sub-actor networks. Moreover, the hybrid actor-critic architecture we propose is flexible to the structure of the action space, such that it can also be generalized for other hierarchically structured action spaces. Specifically, we present an instance of the hybrid actor-critic architecture based on the proximal policy optimization (PPO) [Schulman *et al.*, 2017], which we call hybrid proximal policy optimization (H-PPO). We show that H-PPO outperforms previous methods on a collection of tasks with parameterized action space.

The rest of this paper is organized as follows: Section 2 introduces related work in the parameterized action space domain. The detailed architecture of hybrid actor-critic algorithms and H-PPO is presented in Section 3. Section 4 shows experiments and results. Finally, conclusion and future work are presented in Section 5.

[1]https://blog.openai.com/openai-five/

2 Related Work

Parameterized action spaces and other hierarchical action spaces are more difficult to deal with in RL compared to purely discrete or continuous action spaces for the following reasons. First, the action space has a hierarchical structure, which makes selecting an action more complicated than just choosing one element from a flat set of actions. Second, a parameterized action space involves both discrete action selection and continuous parameter selection, while most RL models are designed for only discrete action spaces or continuous action spaces.

2.1 RL Methods for Discrete Action Space and Continuous Action Space

The Q-learning algorithm [Watkins and Dayan, 1992] is a value-based method which updates the Q-function using the Bellman equation

$$Q(s,a) = \mathop{\mathbb{E}}_{r_t, s_{t+1}}[r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') \mid s_t = s, a_t = a]. \quad (1)$$

In the domain of discrete action space, deep Q-network (DQN) [Mnih *et al.*, 2013] takes the framework and uses a deep neural network to approximate the Q function. Some variations of DQN are also widely used in discrete action space, including asynchronous DQN [Mnih *et al.*, 2016], double DQN [Hasselt *et al.*, 2016] and dueling DQN [Wang *et al.*, 2016].

Policy gradient [Sutton *et al.*, 2000] is another class of RL algorithms which optimizes a stochastic policy π_θ parameterized by θ to maximize the expected policy value $J(\pi_\theta)$. The gradient of the stochastic policy is given by the policy gradient theorem [Sutton *et al.*, 2000] as

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{s,a}[\nabla_\theta \log \pi_\theta(a \mid s) Q^{\pi_\theta}(s,a)]. \quad (2)$$

As an alternative, the policy gradient could also be computed with the advantage function $A^{\pi_\theta}(s,a)$ as

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{s,a}[\nabla_\theta \log \pi_\theta(a \mid s) A^{\pi_\theta}(s,a)]. \quad (3)$$

Similarly in continuous action spaces, the deterministic policy gradient (DPG) algorithm [Silver *et al.*, 2014] and the DDPG algorithm [Lillicrap *et al.*, 2016] optimize a deterministic policy μ_θ parameterized by θ based on the deterministic policy gradient theorem [Silver *et al.*, 2014] as

$$\nabla_\theta J(\mu_\theta) = \mathop{\mathbb{E}}_{s}[\nabla_\theta \mu_\theta(s) \nabla_\theta Q^{\mu_\theta}(s,a) \mid_{a=\mu_\theta(s)}]. \quad (4)$$

Based on the policy gradient methods, trust region policy optimization (TRPO) [Schulman *et al.*, 2015] and proximal policy optimization (PPO) [Schulman *et al.*, 2017] improve the optimization techniques to achieve better performance.

2.2 RL Methods for Parameterized Action Space

To deal with the fact that a parameterized action space contains both discrete actions and continuous parameters, one

straightforward approach is to directly discretize the continuous part of the action space and turn it into a large discrete set (for example with the tile coding approach [Sherstov and Stone, 2005]). This trivial method loses the advantages of continuous action space for fine-grained control, and often ends up with an extremely large discrete action space.

Another direction is to convert the discrete action selection into a continuous space. Hausknecht and Stone [2016] used an actor network to output a value for each of the discrete actions, concatenated with all continuous parameters, and the discrete action is chosen to be the one with the maximum output value. The actor network is learned using the DDPG algorithm. By relaxing the structured action space into a continuous set, this method might significantly increase the complexity of the action space [Xiong *et al.*, 2018].

Masson *et al.* [2016] focused on how to learn an action-selection policy given fixed parameter-selection, and proposed the framework called Q-PAMDP, which alternately learns the discrete action selection with Q-learning and updates parameter-selection policies with policy search methods.

Wei *et al.* [2018] proposed a hierarchical approach for RL in parameterized action space where the parameter policy is conditioned on the discrete action policy and used TRPO and Stochastic Value Gradient [Heess *et al.*, 2015] to train such an architecture. Although they also found that this method could be unstable due to the joint-learning between the discrete action policy and parameter policy.

Xiong *et al.* [2018] proposed the parameterized deep Q-networks (P-DQN) algorithm, which can be viewed as a combination of DQN and DDPG. P-DQN has one network to select the continuous parameters for all discrete action. Another network takes the state and the chosen continuous parameters as input and outputs the Q-values for all discrete actions. The discrete action with the largest Q-value is chosen. However, the network that selects continuous parameters are updated to maximize the sum of the Q-values for all discrete actions, which might cause the algorithm being updated to improve the sum of the Q-values but decrease the largest Q-value.

3 Methodologies

This section introduces the proposed hybrid actor-critic architecture and presents the H-PPO algorithm as an instance of this architecture. Following the notations in [Masson *et al.*, 2016], we describe the parameterized action space in a mathematical way. We consider the following parameterized action space: the discrete actions are selected from a finite set $\mathcal{A}_d = \{a_1, a_2, \ldots, a_k\}$, and each $a \in \mathcal{A}_d$ has a set of real-valued continuous parameters $\mathcal{X}_a \subseteq \mathbb{R}^{m_a}$. In this way, a complete action is represented as a tuple (a, x), where $a \in \mathcal{A}_d$ is the chosen discrete action and $x \in \mathcal{X}_a$ is the chosen parameter to execute with action a. The whole action space $\mathcal{A}$ is then the union of each discrete action with all possible parameters for that action:

$$\mathcal{A} = \bigcup_{a \in \mathcal{A}_d} \{(a, x) \mid x \in \mathcal{X}_a\}. \tag{5}$$

A Markov decision process with a parameterized action space is referred to as parameterized-action Markov decision processes (PAMDPs) [Masson *et al.*, 2016].

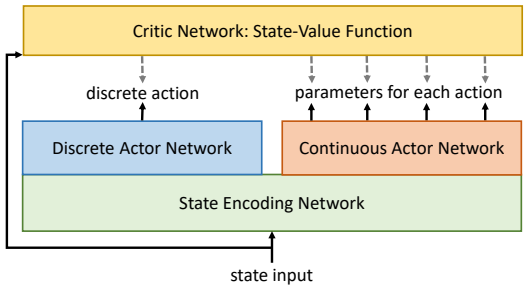


Figure 3: Hybrid actor-critic architecture for parameterized action space.

To design an algorithm for PAMDPs, we should first tackle the problem that parameterized action spaces are a class of discrete-continuous hybrid action spaces. Our method is based on the actor-critic architecture for the reason that many algorithms with actor-critic style, including policy gradient methods and PPO, are capable of learning stochastic policies in both discrete action spaces and continuous action spaces. Actor-critic algorithms usually have one actor network and one critic network, and the critic network is used to compute the gradient of the parameters of the actor network. By contrast, our proposed architecture for parameterized action space (shown in Figure 3) contains two parallel actor networks (or even more for general hierarchical action spaces, introduced later in subsection 3.2). The parallel actors perform action-selection and parameter-selection separately: one discrete actor network learns a stochastic policy π_{θ_d} to select the discrete action a and one continuous actor network learns a stochastic policy π_{θ_c} to choose the continuous parameters $x_{a_1}, x_{a_2}, \ldots, x_{a_k}$ for all discrete actions. The complete action to execute is the selected action a paired with the chosen parameter x_a corresponding to action a. The two actor networks shares the first few layers to encode the state information. We refer to the proposed architecture as hybrid actor-critic architecture since discrete actor and continuous actor both exist in this architecture.

There is a single critic network in the hybrid actor-critic architecture, which works as an estimator of the state-value function $V(s)$. One important reason for us to use the state-value function as the critic instead of the action-value function is that action-value function suffers from the over-parameterization problem in parameterized action space. Specifically, if the action-value function is used as the critic, the critic network in implementation would take the state s, the selected discrete action a and the chosen parameters for all discrete actions $x_{a_1}, x_{a_2}, \ldots, x_{a_k}$ as input. It is impossible to just feed the chosen parameter x_a for one specific discrete action a into the critic network since the parameter dimensions of different discrete actions could be different. In this way, the action-value function is represented in the form of $Q(s, a, x_{a_1}, x_{a_2}, \ldots, x_{a_k})$. However, the actual action to execute is not influenced by irrelevant parameters, so the true Q-function value is independent of $x_{a'}$ for all $a' \neq a$. Therefore, the action-value function would suffer from the problem

of over-parameterization that

$$Q(s, a, x_{a_1}, x_{a_2}, \ldots, x_{a_k}) = Q(s, a, x_a). \quad (6)$$

By contrast, the state-value function only takes the state s as input and does not have this problem. In our architecture, the state-value function $V(s)$ is used for computing a variance-reduced advantage function estimator $\hat{A}$. We follow the implementation used by Mnih *et al.* [2016], which runs the policy for T timesteps and computes the estimator $\hat{A}_t$ using the collected samples as

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T), \quad (7)$$

where $t \in [0, T]$ is the timestep index and T is much less than the length of an episode.

With a critic network providing estimation of the advantage function, the hybrid actor-critic architecture is flexible in the choice of the policy optimization method. The only requirement is that the optimization method should have an actor-critic style and updates stochastic policies with the advantage function provided by the critic. Although the complete action to execute (a, x_a) is decided by both of the actors, the discrete actor and the continuous actor are updated separately by their respective update rules at each timestep. The update rules of the two actor networks could follow policy gradient methods as Eq. (3) or other optimization methods for stochastic policies such as PPO. We can even use two different optimization methods for the discrete policy π_{θ_d} and the continuous policy π_{θ_c}.

Then we present the hybrid proximal policy optimization algorithm for parameterized action space, which is a specific instance of the hybrid actor-critic architecture based on PPO.

3.1 Hybrid Proximal Policy Optimization

The hybrid proximal policy optimization (H-PPO) takes the hybrid actor-critic architecture in Figure 3 and uses PPO as the policy optimization method for both its discrete policy π_{θ_d} and its continuous policy π_{θ_c}.

PPO is a state-of-the-art policy optimization method that learns a stochastic policy π_θ by minimizing a clipped surrogate objective [Schulman *et al.*, 2017] as

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \quad (8)$$

where $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and ϵ is a hyperparameter.

To generate the stochastic policy for discrete actions π_{θ_d}, the discrete actor network of H-PPO outputs k values $f_{a_1}, f_{a_2}, \ldots, f_{a_k}$ for the k discrete actions, and the discrete action a to take is randomly sampled from the $\text{softmax}(f)$ distribution. The continuous actor network of H-PPO generates the stochastic policy π_{θ_c} for continuous parameters by outputting the mean and variance of a Gaussian distribution for each of the parameters. On every iteration of training, H-PPO runs by its policies π_{θ_d} and π_{θ_c} in the environment for T timesteps and updates these two policies with the collected samples. The discrete policy π_{θ_d} and the continuous policy π_{θ_c} are updated separately by minimizing their respective clipped surrogate objective. The objective for the discrete policy π_{θ_d} is given by

$$L_d^{\text{CLIP}}(\theta_d) = \hat{\mathbb{E}}_t[\min(r_t^d(\theta_d)\hat{A}_t, \text{clip}(r_t^d(\theta_d), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \quad (9)$$

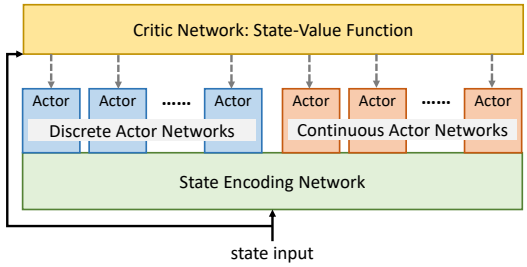


Figure 4: Hybrid actor-critic architecture for general hierarchically structured action space.

and similarly the objective for the continuous policy π_{θ_c} is

$$L_c^{\text{CLIP}}(\theta_c) = \hat{\mathbb{E}}_t[\min(r_t^c(\theta_c)\hat{A}_t, \text{clip}(r_t^c(\theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)]. \quad (10)$$

Here the probability ratio $r_t^d(\theta_d)$ only considers the discrete policy and $r_t^c(\theta_c)$ only considers the continuous policy. That is to say, even though the two policies work with each other to decide the complete action, their objectives are not explicitly conditioned on each other. In other words, π_{θ_d} and π_{θ_c} are viewed as two separate distributions instead of a joint distribution in policy optimization. For example, if the complete action executed at timestep t ($t \in [0, T]$) is denoted by $a_t = (a, x_a)$, $r_t^d(\theta_d)$ is defined as $\frac{\pi_{\theta_d}(a|s_t)}{\pi_{\theta_{d(\text{old})}}(a|s_t)}$ and $r_t^c(\theta_c)$ is defined as $\frac{\pi_{\theta_c}(x_a|s_t)}{\pi_{\theta_{c(\text{old})}}(x_a|s_t)}$.

3.2 Hybrid Actor-Critic Architecture for General Hierarchical Action Space

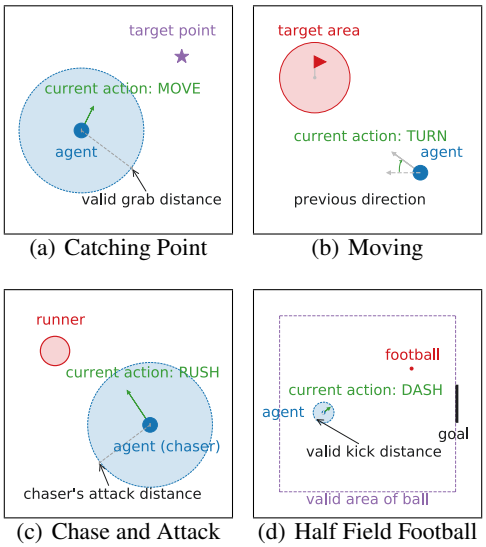
Apart from parameterized action space, the hybrid actor-critic architecture can be extended to solve RL problems with general hierarchical action space. Shown in Figure 2, the action-selection process in a general hierarchical action space could be represented with a tree structure. Each of the grey areas in Figure 2 stands for an action-selection sub-problem. All internal nodes of the tree structure should be discrete action-selection sub-problems, and each discrete action on an internal node corresponds to an action-selection sub-problem of the next layer. The leaf nodes of the tree could be either discrete action-selection or continuous action-selection.

The hybrid actor-critic architecture for general hierarchical action space contains (see Figure 4) multiple parallel actor networks and one critic network. There is one actor network for each of the action-selection sub-problems, either discrete or continuous. The critic network here is the same as the critic in the hybrid actor-critic architecture for parameterized action space. The actor networks share the first few layers to encode the state and each of them generates either a stochastic discrete policy or a stochastic continuous policy. During training, the actors are updated as separate policies using a chosen policy optimization method such as PPO.

4 Experiments

4.1 Environments

We create a collection of tasks with parameterized action space for the experiments, which is shown in Figure 5. The



(a) Catching Point (b) Moving

(c) Chase and Attack (d) Half Field Football

Figure 5: The four environments with parameterized action space used as the environments of the experiments. For clearness, the illustrations here show only a part of the whole field for the tasks except Football.

tasks are briefly described below, and a more detailed description of the environment settings can be found in the supplemental material[2]. Every task has a so-called "winning state", which is a final state of an episode indicating that the agent has succeeded in that episode.

Catching point. In this task, the agent should catch a target point within limited chances. The parameterized actions are MOVE($\text{direction}_\text{M}$) and CATCH, where MOVE means a movement of a constant distance in the specified direction and CATCH is an attempt to catch the target point. The agent could only try the CATCH action for up to 10 times. The episode ends if the agent catches up the target point (the winning state), all of its 10 chances ran out or the time limit exceeds.

Moving. In this scenario, the goal of the agent is to move towards a target area and stop in it. The agent can choose its action among ACCEL(power_A), TURN($\text{direction}_\text{T}$) and BRAKE. The movement of the agent is always in the direction of its current direction. An episode ends if the agent stops in the target area (the winning state), it moves out of the field or the time limit exceeds.

Chase and attack. In this task, the agent should chase a rule-based runner and attack it. The parameterized actions of the agent are RUSH($\text{direction}_\text{R}$), ATTACK($\text{direction}_\text{A}$). The runner has 3 lives at the beginning, and it loses one life every time the agent performs a successful attack. The episode ends when the runner loses all of its lives (the winning state for the agent) or the time limit exceeds.

Half field football. This environment is a similar self-implemented version of a sub-scenario in the Half Field Offense (HFO) [Hausknecht *et al.*, 2016]. The task of the agent is to score a goal in the half football field with no goalie, and the same task in the original HFO environment is used as test environment for RL algorithms in parameterized action space by Hausknecht and Stone [2016] and Xiong *et al.* [2018]. The parameterized actions are DASH(power_D, $\text{direction}_\text{D}$), TURN($\text{direction}_\text{T}$) and KICK($\text{power}_\text{K}$, $\text{direction}_\text{K}$). The episode ends when the agent scores a goal (the winning state), the ball is out of the valid area, or the time limit exceeds.

[2]https://www.dropbox.com/s/s0ut449i3e2fsk1/suppl.pdf

4.2 Experiment Settings and Results

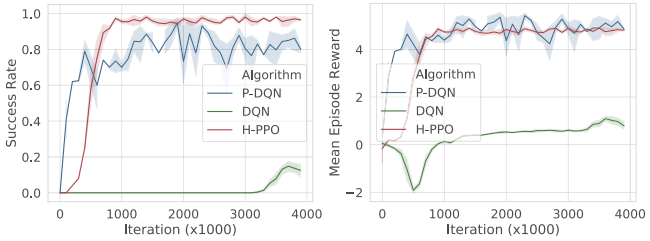
We evaluated the performance of the H-PPO on the four tasks above. In addition, we also implemented and tested the following three baseline algorithms: the extended DDPG for parameterized action space by Hausknecht and Stone [2016], the P-DQN algorithm [Xiong *et al.*, 2018] and DQN which first discretizes the parameterized action space.

The networks in the four algorithms are of the same size, and the hidden layer sizes for each network is $(256, 256, 128, 64)$. The replay buffer size for DDPG and DQN is 10000, and the batch size for sampling is 32. For DQN, we discretizes the action space of Chase and Attack into 30 actions, 16 discrete actions for Catching Point and 23 discrete actions for Moving. However, since the action space of Half Field Football task contains more parameters, the discretized action space has a relatively large size of 104 even if we only discretize each direction parameter into 8 values and each power parameter into 6 values.

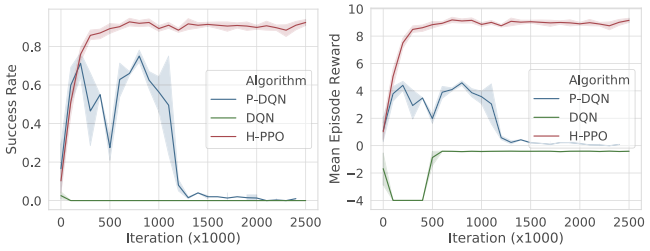
Figure 6 shows the experiments results, which contains both the success rate (the percentage of episodes which ends in the winning state) and mean episode reward during training of the methods in the four test environments. The experiment results of DDPG are not included here because the performance of DDPG in our experiments was far worse than in the original paper, demonstrated a large variance and it failed to learn reasonable policies, which is an issue also reported by Wei *et al.* [2018]. Table 1 shows the success rate, standard deviation of success rate and mean episode reward achieved by DQN, P-DQN and H-PPO after the same number of iterations of learning in the four environments. As we can see from the results, H-PPO showed stable learning and achieved high

Environment	Algorithm	SuccRate	SD of SuccRate	Mean Reward
Catching Point	DQN	6.13%	±8.21%	0.796
	P-DQN	82.52%	±11.60%	4.977
	H-PPO	**96.32%**	**±4.82%**	**4.790**
Moving	DQN	0.00%	±0.00%	-0.415
	P-DQN	1.56%	±2.78%	0.173
	H-PPO	**90.45%**	**±6.75%**	**8.955**
Chase and Attack	DQN	99.91%	±0.74%	5.664
	P-DQN	99.85%	±0.84%	5.589
	H-PPO	**99.98%**	**±0.30%**	**5.393**
Half Field Football	DQN	0.00%	±0.00%	0.000
	P-DQN	76.31%	±16.81%	8.762
	H-PPO	**95.39%**	**±4.81%**	**9.849**

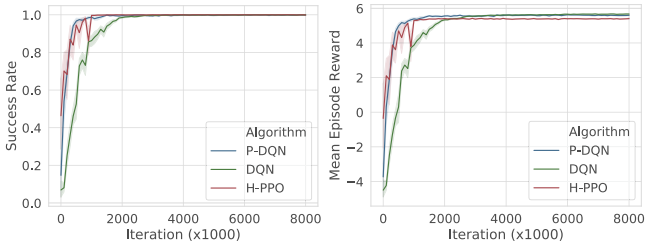
Table 1: Success rate, standard deviation of success rate and mean episode reward achieved by DQN, P-DQN and H-PPO in the experiment environments.



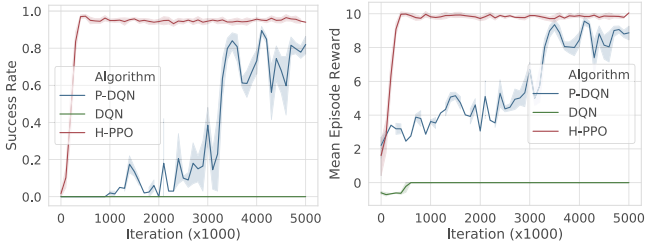
(a) Catching Point



(b) Moving



(c) Chase and Attack

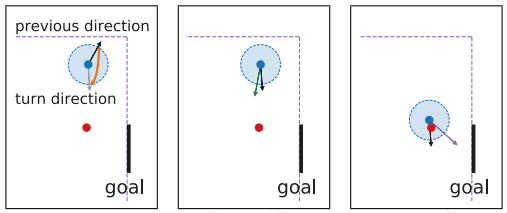


(d) Half Field Football

Figure 6: Results of the experiments on the four tasks. Left column: Success rate of each method during training. Right column: Mean episode reward of each method during training.

success rate on all the four tasks. Moreover, H-PPO outperformed other methods by a large margin in three of the four environments (except in Chase and Attack, where the three algorithms all achieved similar success rate and H-PPO had the lowest variance, see Table 1). It generally achieved higher success rate, faster convergence and lower variance than other methods in the experiments.

To illustrate the learned action-selection and parameter-selection policy of H-PPO from the micro perspective, Figure 7 shows the states of three frames (placed in the order of time) observed in an episode of Half Field Football and the parameterized actions selected by the H-PPO agent in these



(a) TURN (b) DASH (c) KICK

Figure 7: Illustration of the parameterized actions selected by the H-PPO agent in three frames of a Half Field Football episode.

frames. Black arrows in Figure 7 indicates the facing direction of the agent, and arrows in other colors show the selected actions and parameters. The agent was not close to the ball in frame (a) and its direction was not toward the ball, so it performed a TURN (the discrete action) with the specified angle (the continuous parameter, shown in orange) to get closer to the ball. Then it was roughly facing the ball in frame (b), and it chose DASH along with proper power and direction (shown in green). Finally, it took a KICK in the direction (shown in purple) toward the goal in frame (c), when it was close enough to the ball to perform the KICK. This example shows the capability of H-PPO to learn the discrete action-selection policy and the continuous parameter-selection policy in coordination with its hybrid actor-critic architecture. This coordination during training comes from the characteristics of H-PPO that not only the two parallel actors share the first few layers of their networks, they also share the same critic to perform policy optimization updates with similar objectives given by Eq. (9) and Eq. (10).

5 Conclusion and Future Work

This paper introduced a hybrid actor-critic architecture for reinforcement learning in parameterized action space where the discrete action policy and the continuous parameter policy are trained in parallel as separate actors with a global critic. As the hybrid actor-critic architecture is flexible in the choice of the policy optimization method, we also presented H-PPO, which is an implementation of the architecture based on PPO. Empirically, H-PPO achieves stable learning in all of the four tasks with parameterized action space and outperforms previous methods of parameterized action reinforcement learning.

Although this paper is mainly focused on reinforcement learning in parameterized action space, we also briefly presented an extended version of the hybrid actor-critic architecture for general hierarchical action spaces. More experiments are needed to test the performance of this architecture in general hierarchical action spaces, and we leave this investigation as future work.

Acknowledgments

This work is supported by Tencent AI Lab Joint Research Program. The corresponding author Weinan Zhang thanks the support of National Natural Science Foundation of China (61702327, 61772333, 61632017) and Shanghai Sailing Program (17YF1428200).

References

[Hasselt *et al.*, 2016] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16. AAAI Press, 2016.

[Hausknecht and Stone, 2016] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2016.

[Hausknecht *et al.*, 2016] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.

[Heess *et al.*, 2015] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in NIPS 28*. Curran Associates, Inc., 2015.

[Kober *et al.*, 2013] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[Konda and Tsitsiklis, 2000] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000.

[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. *CoRR*, abs/1509.02971, 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*, AAAI'16, pages 1934–1940. AAAI Press, 2016.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, New York, New York, USA, 2016. PMLR.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, Lille, France, 2015. PMLR.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Sherstov and Stone, 2005] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In Jean-Daniel Zucker and Lorenza Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.

[Silver *et al.*, 2016] David Silver, Aja Huang, Christopher J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[Sutton *et al.*, 2000] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[Vinyals *et al.*, 2017] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, et al. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

[Wang *et al.*, 2016] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, New York, New York, USA, 2016. PMLR.

[Watkins and Dayan, 1992] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[Wei *et al.*, 2018] Ermo Wei, Drew Wicke, and Sean Luke. Hierarchical approaches for reinforcement learning in parameterized action space. *CoRR*, abs/1810.09656, 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. *CoRR*, abs/1810.06394, 2018.