Hierarchical Advantage for Reinforcement Learning in Parameterized Action Space

1st Zhejie Hu
Graduate School of Interdisciplinary Information Studies
the University of Tokyo
Tokyo, Japan
ko-setsushou@g.ecc.u-tokyo.ac.jp

2nd Tomoyuki Kaneko
Interfaculty Initiative in Information Studies
the University of Tokyo
Tokyo, Japan
kaneko@acm.org

***Abstract*—We propose a hierarchical architecture for the advantage function to improve the performance of reinforcement learning in parameterized action space, which consists of a set of discrete actions and a set of continuous parameters corresponding to each discrete action. The hierarchical architecture extends the actor-critic architecture with two specialized advantage functions, one for discrete actions and the other for continuous parameters, to estimate a better baseline. We incorporate this architecture into proximal policy optimization, which is referred to as HA-PPO. We evaluated all of our methods on the Half Field Offense domain, and found that the hierarchical architecture of the advantage function, which is referred to as the hierarchical advantage, helps to stabilize the learning and leads to a better performance.**

***Index Terms*—parameterized action, hierarchical advantage, reinforcement learning**

I. Introduction

Most remarkable deep reinforcement learning algorithms are designed for either pure discrete action space or pure continuous action space. For example, deep Q-learning [15] has learned policies to play Atari games better than humans, and AlphaGo [18] has defeated world champions in the game of Go. Maximum entropy reinforcement learning frameworks, such as SAC [8] and Soft-Q learning [7], enable a robot to achieve a state-of-the-art performance on a set of different tasks.

Different from the action spaces in game domains and robotics domain, action space can be parameterized when modeling various real-world tasks, such as soccer and car racing. In such cases, the agent not only selects a discrete action but also needs to determine the corresponding continuous parameters. For example, when a player chooses to kick a ball, he also needs to determine the direction and the power, which are treated as continuous parameters. Thus, a parameterized action can be viewed as a combination of discrete action and corresponding continuous parameters, and it is harder than either pure continuous action space or discrete action space to deal with for two reasons. First, it is widely known that instead of selecting two elements from two independent sets of actions, the dependency between discrete actions and continuous parameters increases the difficulty of action selection. Second, as each of the continuous parameters has its own bound, enforcing these bounds while updating the policy is hard to handle, which is referred to as the bounded parameter space learning problem in [10].

978-1-6654-3886-5/21/$31.00

In this work, following the research in [21] and [6], we divide the whole parameterized action into two parts, discrete action and continuous parameters, as a hierarchical structure for parameterized action selection to ensure dependency. Then, we propose a hierarchical structure of advantage that is used to learn both discrete actions policy and continuous parameters policy faster and more stably. Specially, we incorporate the hierarchical advantage into PPO, which we called Hierarchical Advantage Proximal Policy Optimization (HA-PPO). To describe our method clearly, we first develop a method named Parameterized Action Proximal Policy Optimization to properly handle parametrized actions on top of PPO, and then present HA-PPO by augmenting PAPPO with a hierarchical structure of advantage. Our experimental results show that HA-PPO outperforms existing methods on two tasks in the HFO domain [11] [9].

II. Background

In this section, we first introduce the concepts of parameterized action space MDPs and policy gradient methods as our proposed methods are based on them. We then discuss existing studies that are also based on parameterized action space MDPs.

A. Parameterized Action Space MDPs

Most parameterized action problems can be modeled as a *Parameterized Action Space Markov Decision Process (PAMDP)* [14] with finite states and a set of parameterized actions. A PAMDP is defined as a tuple $(S, A, P, R, \gamma, \rho_0)$. Except for the action space A, other variables are the same as those in MDPs. S is the set of all possible states, and A is the set of all possible parameterized actions. Each parameterized action can be defined by one discrete action $a \in A_d$ and the corresponding continuous parameters $(x_1^a, x_2^a, \ldots, x_m^a) \in P_a \subseteq \mathcal{R}^m$, where A_d is defined as the set of discrete actions $\{a_1, a_2, \ldots, a_k\}$. Thus, a parameterized action can be represented by a tuple $(a, x_1^a, x_2^a, \ldots, x_m^a)$, and the whole parameterized action space $A = \cup_{a \in A_d, p_i^a \in P_a}(a, x_1^a, x_2^a, \ldots, x_m^a)$. R is a reward function defined as $r(s_t, a_t, s_{t+1})$, where $s_t \in S$,

$a_t \in A$, and $s_{t+1} \in S$ are the state, action, and the next state at time step t respectively. P is the transition dynamics defined as $P(s_{t+1}|s_t, a_t)$. The future reward discount factor is $\gamma \in [0, 1]$, and ρ_0 is the initial state distribution of s_0. An agent selects a parameterized action based on a policy $\pi_\theta(a^h|s)$, which is a probability distribution parameterized by θ over all possible parameterized actions. Typically, the goal of reinforcement learning is to find a set of parameters θ to maximize the discounted cumulative reward as the following objective function

$$J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0} \gamma^t r_t \right] \tag{1}$$

. Here, τ represents the entire action trajectory, where $s_0 \sim \rho_0$, $a_t \sim \pi$ and $s_t \sim P$.

Following the research [4] [21], a parameterized action policy can be treated as a hierarchical structure where the choice of continuous parameters depends on the selected discrete action. Thus, we decompose the parameterized action policy into a discrete action policy and a continuous parameter policy, as

$$\pi(a^h|s) = \pi(a^d, \boldsymbol{a^c}|s) = \pi^d(a^d|s)\pi^c(\boldsymbol{a^c}|a^d, s), \tag{2}$$

where the parameterized action is represented by $a^h = (a^d, \boldsymbol{a^c})$. $\boldsymbol{a^c}$ denotes the corresponding continuous parameters of a^d and is usually represented as a vector.

B. Policy Gradient Methods

Policy gradient methods [19] are reinforcement learning algorithms that learn optimal policies π that are parameterized by θ to maximize the expected returns $J(\theta)$. In this work, we utilize popular policy gradient methods (DDPG [13] and PPO [17]) to calculate policy gradients via the advantage function $A^{\pi_\theta}(s, a)$. Then, the policy gradient with respect to θ can be written as

$$\nabla J(\theta) = E_{a\sim\pi_\theta, s_0\sim\rho_0, \tau} \left[A^{\pi_\theta}(s, a) \nabla_\theta \log \pi_\theta(a|s) \right], \tag{3}$$

where

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s). \tag{4}$$

Here, the advantage function A is defined as the difference between the returns for a given state-action pair ($Q^{\pi_\theta}(s, a)$) and a baseline V^{π_θ}, which represents the return from the given state s until the end of the episode and leaves the policy gradient unchanged [19]. In other words, the advantage function identifies the estimated additional returns that could be obtained by taking a particular action for the given state. Furthermore, it is observed that better advantage functions can lead to significant improvements in performances [2]. Thanks to the baseline, the advantage function helps to reduce the variance of the policy gradients and thus enable faster and more stable learning compared with function $Q^{\pi_\theta}(s, a)$.

C. Existing Studies in PAMDPs

To the best of our knowledge, the earliest method that combines the policy gradient method and PAMDPs is PADDPG [10]. It extends the DDPG [13] into the parameterized action space and prepares two neural networks to output discrete actions and continuous parameters respectively. As DDPG is designed for continuous action space on purpose, the discrete action is relaxed into a continuous set, which may become more complex and difficult to learn. In contrast to PADDPG, P-DQN [23] takes advantage of the hierarchical structure in parameterized action space and works without relaxation. It first determines all the continuous parameters and then gives the Q-values of all the parameterized actions. However, since the whole parameter space is shared in P-DQN, each Q-value may produce gradients for unrelated continuous parameters. This issue of false gradients can be mitigated by MP-DQN [3]. To correct gradients, it performs multiple forward passes to produce multiple gradient vectors. Each of them correspond to one Q-value of a parameterized action concerning its own continuous parameters. Another study that determines the Q-values of parameterized actions via continuous parameters and states is Hybrid-SAC [4]. Although Hybrid-SAC does not utilize Monte-Carlo returns, its performance is better than that of MP-DQN without Monte-Carlo returns.

In contrast to methods that rely on parameters to learn the discrete action of parameterized action space, PATRPO [21] and H-PPO [6] take states only to generate both discrete action policy and continuous parameter policy at the end of the forward pass. In such methods, the agent can select the discrete action firstly, then determines the remaining corresponding continuous parameters. Both of PATRPO and H-PPO have achieved impressive performances in the tasks with parameterized action space.

III. Proposed Methods

This section introduces the method proposed in this study: Hierarchical Advantage Proximal Policy Optimization (HA-PPO). we develop this mthod on top of PPO, because PPO is a state-of-the-art policy gradient method effective in many domains, such as Atari, Mujoco [20], and Roboschool. In the proposed methods, two policy networks are learned; one for discrete action policy π^d and the other for continuous parameter policy π^c, to handle the parameterized actions.

To clearly describe HA-PPO, we first introduce PAPPO that is an extension of PPO for parametrized actions. The objective functions in terms of discrete actions and continuous parameters are integrated in PAPPO, while they are separated in H-PPO. HA-PPO is constructed on top of PAPPO with a hierarchical structure of advantage, which is our main contribution. In HA-PPO, we introduce a new estimator $Q^d(s_t, a_t^d)$ for the expected return of each discrete action to improve performance by more accurate advantage estimates.

A. Parameterized Action Proximal Policy Optimization

PAPPO adopts an integrated objective function that updates both the discrete action policy and the continuous parameter

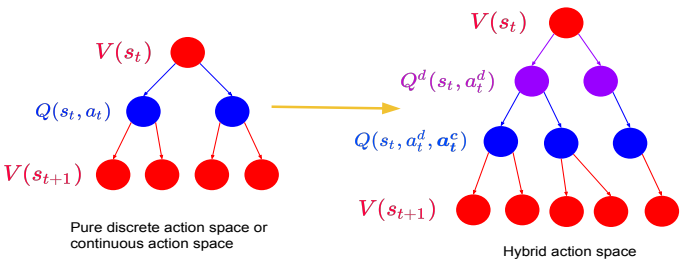

Fig. 1. A backup diagram in typical RL (left) and our hierarchical architecture for parametrized action space (left)

policy at the same time. This gives us the clipped surrogate objective function for the integrated objective, as follows:

$$\begin{aligned} L^{CLIP}(\theta_d, \theta_c) =& \hat{\mathbb{E}}_t[\min(r_t^d(\theta_d) r_t^c(\theta_c)\hat{A}_t, \\ & \text{clip}(r_t^d(\theta_d), 1-\epsilon, 1+\epsilon) \\ & \text{clip}(r_t^c(\theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \end{aligned} \tag{5}$$

where $r_t^d(\theta_d) = \frac{\pi_{\theta_d}(a_t^d|s_t)}{\pi_{\theta_d(old)}(a_t^d|s_t)}$ and $r_t^c(\theta_c) = \frac{\pi_{\theta_c}(\boldsymbol{a_t^c}|a_t^d, s_t)}{\pi_{\theta_c(old)}(\boldsymbol{a_t^c}|a_c^d, s_t)}$. This clipped surrogate objective, which is the core part of PAPPO, allows us to update both policies with the whole advantage estimator. Furthermore, following PPO, PAPPO uses a generalized advantage estimator (GAE) [16] for $\hat{A}_t$ to stabilize the learning process, where GAE is expected to be better than the n-step bootstrap return adopted in H-PPO. The learning of value function $V(s_t)$ in PAPPO is not modified from PPO or H-PPO.

B. Hierarchical Advantage Proximal Policy Optimization

We improve the advantage estimator of PAPPO to yield HA-PPO. The key idea is a hierarchical design of advantage functions that are compatible with the factorization of $\pi(a^h|s)$ into $\pi^d(a^d|s)\pi^c(\boldsymbol{a^c}|a^d, s)$ introduced in Eq. (2). To introduce hierarchical advantage in PPO, we first illustrate how the backup diagram changes from pure discrete action space or pure continuous action space to the parameterized action space in Fig. 1. Here, we define a partial action-value function $Q^d(s_t, a_t^d)$ as an action-value function for discrete actions only:

$$Q^{d^\pi}(s_t, a_t^d) = \mathbb{E}_{\boldsymbol{a_t^c} \sim \pi^c(\boldsymbol{a_t^c}|s_t, a_t^d)} \left[Q^\pi(s_t, a_t^d, \boldsymbol{a_t^c}) \right], \tag{6}$$

$$V^\pi(s_t) = \mathbb{E}_{a_t^d \sim \pi^d(a_t^d|s_t)} \left[Q^{d^\pi}(s_t, a_t^d) \right]. \tag{7}$$

Then, we can take $Q^{d^\pi}(s_t, a_t^d)$ as the baseline to evaluate how advantageous continuous parameters $\boldsymbol{a_t^c}$ are, and give the gradient function of the continuous parameter policy:

$$\begin{aligned} \nabla J(\theta_c) =& \mathbb{E}_{a^d \sim \pi^d, a^c \sim \pi^c, s \sim \rho^\pi, \tau}[[Q^\pi(s, a^d, \boldsymbol{a^c}) - Q^{d^\pi}(s, a^d)] \\ & \nabla_{\theta_c} \log \pi(\boldsymbol{a^c}|a^d, s)]. \end{aligned} \tag{8}$$

The remaining proof is provided in Appendix A. For discrete action, we use state value function $V(s)$ as the baseline. In general, two advantage estimators are utilized in HA-PPO, $\hat{A}_t^c$ for continuous parameters $\boldsymbol{a^c}$, and $\hat{A}_t$ for discrete action a^d, where both of them are based on GAE.

$$\hat{A}_t = \Sigma_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{9}$$

$$\hat{A}_t^c = -Q^d(s_t, a_t^d) + r_t + \gamma V(s_{t+1}) + \Sigma_{l=1}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{10}$$

We here define $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ (the mathematical derivation of $\hat{A}_t^c$ is provided in Appendix B). Then, we obtain two clipped surrogate objectives for the two respective policies and use a squared-error loss to learn the baseline $Q^d(s_t, a_t^d)$.

$$L^{CLIP}(\theta_d) = \hat{\mathbb{E}}_t \left[\min(r_t^d(\theta_d)\hat{A}_t, \text{clip}(r_t^d(\theta_d), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{11}$$

$$L^{CLIP}(\theta_c) = \hat{\mathbb{E}}_t \left[\min(r_t^c(\theta_c)\hat{A}_t^c, \text{clip}(r_t^c(\theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t^c) \right] \tag{12}$$

$$L^{Q^dF}(\theta_q) = (Q_{\theta_q}^d(s_t, a_t^d) - Q_t^{d \text{ target}})^2 \tag{13}$$

By combining Eq. (11), (12) and (13) with l2 regularizations $L^R(\theta_d, \theta_c, \theta_v, \theta_q)$, the final objective to be maximized is

$$\begin{aligned} L^{CLIP+Q^dF+R}(\theta_d, \theta_c, \theta_v, \theta_q) =& \hat{\mathbb{E}}_t[L^{CLIP}(\theta_d) + L^{CLIP}(\theta_c) \\ & - c_1 L^{VF}(\theta_v) - c_2 L^{Q^dF}(\theta_q) \\ & + c_3 L^R(\theta_d, \theta_c, \theta_v, \theta_q)], \end{aligned} \tag{14}$$

where c_1, c_2, and c_3 are coefficients, and $L^{VF}(\theta_v)$ is the same as that of PPO. We follow a similar algorithm to Algorithm 1 in PPO for training but change it slightly for updating multiple sets of parameters.

IV. Experiments

A. Environment

Half Field Offense (HFO) is a soccer game originated from Robocup 2D Soccer [12], where the offense team plays against defense opponents to shoot goals in a half football field. Since this domain is treated as the most complex one in MP-DQN, Hybrid-SAC, and PATRPO, we chose it to evaluate our methods. We used the source code of HFO available at: https://github.com/LARG/HFO. The game environment has several variations[1] but we used the most difficult one; low-level features for state representation and low-level actions. Low-level means that there is less heuristic knowledge embedded. There are two different hand-coded build-in AI teams in HFO for the convenience of training and evaluation. One is built on Agent2D [1], which is a simple but competitive AI. The other is built on Helios' champion agent, which won the first place in the 2012 Robocup-2D Soccer Competition. In other words, Helios' champion agent represents a top-level agent

[1]https://github.com/LARG/HFO/blob/master/doc/manual.pdf

with a policy of very high competitiveness. We used Agent2D as the defense player in training and Helios' champion agent in evaluation.

To conduct reinforcement learning in HFO, the authors of MP-DQN made an interface compatible with OpenAI Gym, and we basically followed their implementation[2]. At the beginning of each episode, the ball is positioned randomly inside an specified area where two parameters, normalized minimum initial position and normalized maximum initial position, configure the area. The agent always plays as the offense, and the episode ends in one of the following situations:

- the agent scores a goal,
- the ball moves outside the valid area,
- the ball is captured by the defense,
- the agent does not touch the ball within a certain amount of time, or
- the time limit is exceeded.

B. Experiment Settings

Following existing studies on MP-DQN, H-PPO, and PADDPG, we evaluated HA-PPO and PAPPO with the low-level state set and the low-level action set. In addition, we used the authors' implementation for MP-DQN and H-PPO for comparison purposes. Unfortunately, the source code and implementation details of Hybrid-SAC are not available, so we were not able to include Hybrid-SAC in our experiments. We set up two tasks in our experiments: 1vs0 task and 1vs1 task. The 1vs0 task is easier because there are no defense players, while the 1vs1 task is more difficult because there is a goalie controlled by Agent2D. We use the term *difficult* 1vs1 task to emphasize the difficulty compared to the similar task implemented by the authors of MP-DQN, and give the details of our tasks' settings in Appendix D. The parameterized actions in both tasks are Dash (power, direction), Turn (direction), and Kick (power, direction). Since the performances for the difficult 1vs1 task were poor in the preliminary experiments, we set up an easy 1vs1 task where we weaken the goalie by forcing a no-op like action with a probability of 30% into the Agent2D players. Then we test the agents in the difficult 1vs1 task without no-op action. More details about the implementation of the no-op action are explained in Appendix E. We adopted the reward function utilized in PADDPG as it has proven effective in several existing studies mentioned before.

We followed the implementation of PPO given by OpenAI baselines [5] in our algorithms (HA-PPO and PAPPO) and H-PPO. As a small exception, we turned off *per minibatch advantage normalization* in the easy 1vs1 task so as to visualize advantages better. Note that per minibatch advantage normalization does not make any significant difference in performance as discussed in [2]. The hyperparameters are listed in Appendix C. The networks of PAPPO and H-PPO implemented in this work are the same size, and the structure is illustrated in Fig. 2. For HA-PPO, the network is enlarged

[2]https://github.com/cycraig/gym-soccer

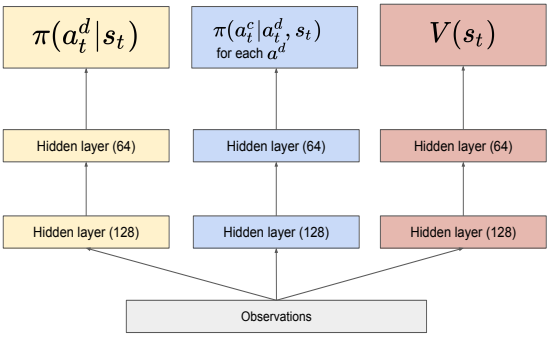

Fig. 2. PAPPO and H-PPO network architecture.

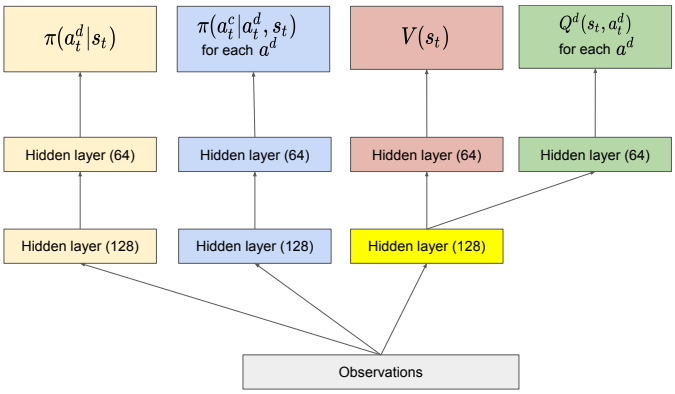

Fig. 3. HA-PPO network architecture.

a bit to present partial state value function $Q^d(s_t, a_t^d)$ (Fig. 3). Our network is smaller than that in the original H-PPO, though discrete policy $\pi(a^d|s)$ and continuous parameters $\pi(a^c|a^d, s)$ do not share their intermediate layers. We used the ReLU function for activation of all hidden layers, and used the Softmax function for the output layers of the discrete action policy. As in the most existing works, the continuous parameter policy is presented in the Gaussian distribution where the mean is yielded by tanh and the standard deviation is yielded via softplus in logarithmic form. While some existing studies need an invert gradient method [10] or similar to guarantee that an agent learn the valid range of continuous parameters, our methods worked well without such methods though it might depend on the small learning rate.

C. Experiment Results

We trained five different agents for each algorithm of HA-PPO, PAPPO, and H-PPO in the 1vs0 task and the easy 1vs1 task. In the 1vs0 task, every agent of the PPO-based algorithms was trained for five million steps. The learning curve of HA-PPO, PAPPO, and H-PPO in the 1vs0 task are calculated by Wilson score interval [22] and shown in Fig. 4. The agents of both PAPPO and HA-PPO rapidly mastered this task, though the HA-PPO agents needed a few more steps to train extra neurons for the partial action-value function $Q^d(s_t, a_t^d)$. All three algorithms achieved a maximum success

rate of 100% that was evaluated on the last 1,000 episodes during the training. After the training, we evaluated all agents by the success rate of scoring a goal and the average steps to the goal (shorter is better) over 1,000 independent episodes in the 1vs0 task. Note that each agent plays according to its own trained stochastic policy. The results are shown in Table I, where the ones of MP-DQN in the 1vs0 task are those reported in the original work. We also tested Helios' champion agent as an offense player for 1,000 episodes in the 1vs0 task, and the results were averaged over five trials. Specifically, the performances of HA-PPO and PAPPO almost reached that of Helios' champion agent in the evaluation. Among the three

TABLE I
EVALUATION OF 1VS0 TASK.

Algorithm	Success rate			Avg. steps to goal		
HA-PPO	0.9972	±	0.0013	75.6	±	0.6
PAPPO	0.9947	±	0.0031	76.5	±	1.5
H-PPO	0.9910	±	0.0118	77.9	±	4.0
Helios	0.9970	±	0.0020	74.9	±	0.5
MP-DQN [3]	0.9130	±	0.0700	99.0	±	12.0
H-PPO [6]	0.9540	±	0.0480		–	

PPO-based algorithms, we found that the HA-PPO agents performed best in both success rate of scoring a goal and average steps to the goal. We conclude that HA-PPO provides a better way to learn continuous parameters. The difference between our methods (PAPPO and HA-PPO) and H-PPO was not significant in this task, so we show the difference more larger in 1vs1 task which is much more difficult. Note that the performance of H-PPO implemented by us is slightly better than the original work. Our smaller neural network and fine-tuned hyperparameters may be the reasons.

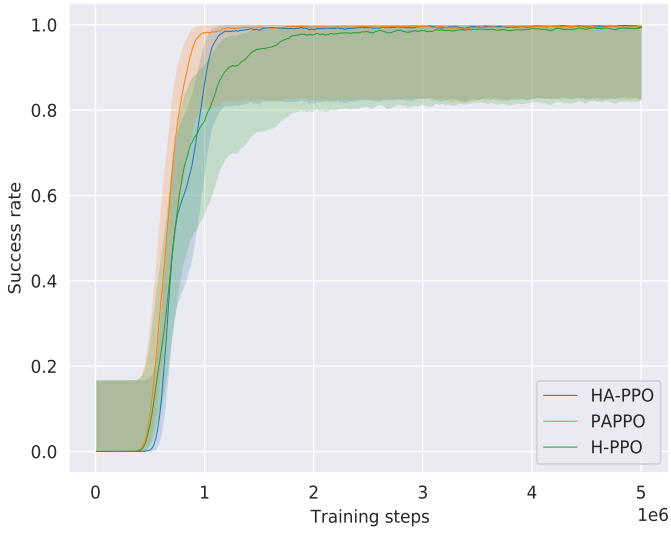

Fig. 4. Results of HA-PPO, PAPPO, and H-PPO in 1vs0 task. The colored area represents Wilson score interval for each algorithm.

In the easy 1vs1 task, every agent of the PPO-based algorithms and MP-DQN was trained by fifteen million steps. Note that according to the source code of MP-DQN[1], the length of the learning procedure set in MP-DQN is calculated by episode, so we modified the source code to record the success rate by training steps to be consistent with other results. The results of HA-PPO, PAPPO, H-PPO, and MP-DQN in easy 1vs1 task are calculated by Wilson score interval and shown in Fig. 5. HA-PPO achieved a maximum mean success rate of 82.7%, which was the best among all algorithms. However, PAPPO, H-PPO and MP-DQN only obtained maximum mean success rates of 77.3%, 70.0% and 50.3%, respectively.

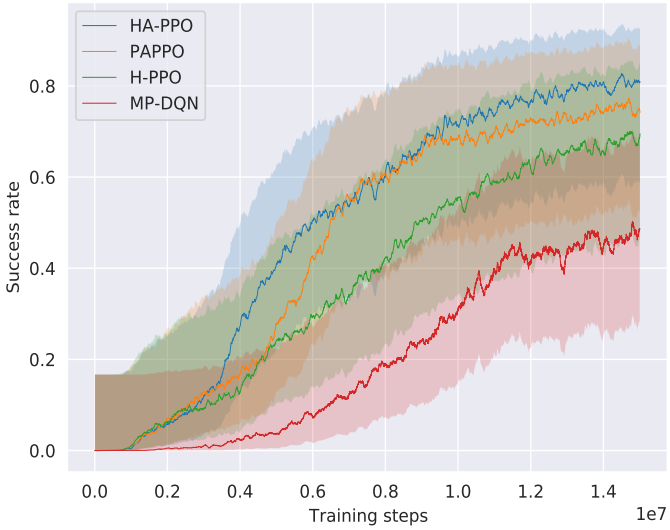

Fig. 5. Results of HA-PPO, PAPPO, H-PPO, and MP-DQN in easy 1vs1 task. The colored area represents Wilson score interval for each algorithm.

To evaluate the proposed baseline $Q^d(s_t, a_t^d)$ of HA-PPO, we compared the mean of the advantage of continuous parameters (Eq. (10)) with the mean of the whole advantage (Eq. (9)) for each discrete action. Table II shows the mean observed with 0.1 million steps after the training. These means are averaged over three different random seeds for each agent of each algorithm. We found that large differences among discrete actions in both the advantage of continuous paramters and the whole advantage. The results are consistent with our speculation that $Q^d(s_t, a_t^d)$ gives better advantage estimates of continuous parameters for each discrete action. In addition, Fig. 6 in Appendix G shows the standard deviation of the advantage (Eq. (9)) during the training for each method. The standard deviation is calculated with a sliding window of size 100. The values increase as the agents receive more returns and then decrease as the value functions become accurate. We can see that all methods have similar values that indicate the differences in Table II are inherent to the environment. With a close look, HA-PPO's standard deviation increases faster than others because the agents trained by HA-PPO receive more returns in the early stage of the training. To clarify whether the hierarchical advantage helps to stabilize training,

[1]https://github.com/cycraig/MP-DQN

TABLE II
THE MEAN OF THE ADVANTAGE IN EASY 1VS1 TASK

Algorithm	Discrete action	Mean
HA-PPO[a]	Dash	-0.0253
HA-PPO[a]	Kick	-0.1063
HA-PPO[a]	Turn	-0.0424
HA-PPO[b]	Dash	-0.0299
HA-PPO[b]	Kick	-0.1155
HA-PPO[b]	Turn	-0.0441
PAPPO[b]	Dash	-0.0190
PAPPO[b]	Kick	-0.0842
PAPPO[b]	Turn	-0.0462
H-PPO[b]	Dash	-0.0319
H-PPO[b]	Kick	-0.0919
H-PPO[b]	Turn	-0.0663

[a]The mean of the advantage of continuous parameters for each discrete action.
[b]The mean of the whole advantage for each discrete action.

TABLE III
THE MEAN OF THE WIDTS OF WILSON SCORE INTERVAL IN EASY 1VS1 TASK

Algorithm	Mean of the length of Wilson score intervals
HA-PPO	0.3399
PAPPO	0.3626
H-PPO	0.3856

we investigated the mean of the width of Wilson score interval over the last one million steps of the training for all PPO-based algorithms. The results are shown in Table III. We found that HA-PPO has the lowest one across three PPO-based algorithms, and thus HA-PPO learns most stably. We think that because the hierarchical advantage method helps pick up better individual advantages for the discrete action policy and continuous parameter policy, HA-PPO learns faster and more stably than PAPPO and H-PPO.

After the training, we evaluated each agent by three different random seeds in the difficult 1vs1 task including MP-DQN as its source code provides methods for saving and loading models. In addition, we tested the Helios' champion agent in the difficult 1vs1 task for 1,000 episodes and obtained the mean over three trials. The details of the evaluation are listed in Table IV. Since the performance fluctuation of some algorithms is large, we give details about the mean of success rate of each agent in Appendix F. Generally, the performance in the difficult 1vs1 task was consistent with the corresponding performance of the training in the easy 1vs1 task. We believe that the difficult 1vs1 task is tough, so some agents may obtain a success rate over 50% in the easy 1vs1 task, but perform poorly in the difficult 1vs1 task.

TABLE IV
EVALUATIONS OF DIFFICULT 1VS1 TASK

Algorithm	Success rate			Avg. steps to goal		
HA-PPO	0.7080	$\pm$	0.1353	85.2	$\pm$	4.3
PAPPO	0.5989	$\pm$	0.1751	92.8	$\pm$	12.3
H-PPO	0.4825	$\pm$	0.2196	95.3	$\pm$	8.1
MP-DQN	0.3418	$\pm$	0.1235	111.8	$\pm$	4.6
Helios	0.9590	$\pm$	0.0110	70.8	$\pm$	0.7

V. CONCLUSION

In this work, we investigated reinforcement learning in environments with parametrized action space (e.g., 'kick'+power), and proposed a hierarchical architecture for advantage functions to improve the policy gradient methods. In our experiments, we demonstrated that the hierarchical advantage makes learning faster and more stable compared to existing work with the ordinary advantage function. While the hierarchical advantage is applicable to any policy gradient method, we chose PPO as a representative and proposed HA-PPO. Agents trained by HA-PPO outperformed those trained by PAPPO and other existing methods in two offense tasks with respect to the success rate and the average steps to the goal. Moreover, in the task without a goalie, the performance of agents trained by HA-PPO almost reached that of Helios's champion agent. In the task with a goalie, the success rate of HA-PPO agents was much higher than other agents but still has room for improvement compared to Helios' champion agent. An interesting future direction would be to incorporate methods for mitigating the issue of bounded parameter space learning to improve performance further.

REFERENCES

[1] H. Akiyama and T. Nakashima. Helios base: An open source package for the robocup soccer 2d simulation. volume 8371, pages 528–535, 01 2014.

[2] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[3] C. J. Bester, S. D. James, and G. D. Konidaris. Multi-pass q-networks for deep reinforcement learning with parameterised action spaces. *arXiv preprint arXiv:1905.04388*, 2019.

[4] O. Delalleau, M. Peter, E. Alonso, and A. Logut. Discrete and continuous action representation for practical rl in video games. In *Association for the Advancement of Artificial Intelligence Workshop on Reinforcement Learning in Games*, 2020.

[5] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Z. Tan, Y. Wu, and P. Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[6] Z. Fan, R. Su, W. Zhang, and Y. Yu. Hybrid actor-critic reinforcement learning in parameterized action space. In *International Joint Conference on Artificial Intelligence*, 2019.

[7] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[9] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*. sn, 2016.

[10] M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. In *International Conference on Learning Representations(ICLR)*, 2016.

[11] S. Kalyanakrishnan, Y. Liu, and P. Stone. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 72–85. Springer Verlag, Berlin, 2007.

[12] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347, 1997.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] W. Masson, P. Ranchod, and G. Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

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

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

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

[20] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[21] E. Wei, D. Wicke, and S. Luke. Hierarchical approaches for reinforcement learning in parameterized action space. In *Association for the Advancement of Artificial Intelligence*, 2018.

[22] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[23] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

APPENDIX A

For simplicity, we take out the part of the baseline and prove that this part does not change the expected value of the continuous paramteter policy gradients.

$$
\begin{aligned}
&\mathbb{E}_{a^d\sim\pi^d,a^c\sim\pi^c,s\sim\rho^\pi,\tau}[Q^{d^\pi}(s,a^d)\nabla_{\theta_c}\log\pi(\boldsymbol{a^c}|a^d,s)]\\
&=\int_{a^d,\boldsymbol{a^c},s}\rho^\pi(s)Q^{d^\pi}(s,a^d)\nabla_{\theta_c}\log\pi(\boldsymbol{a^c}|a^d,s)d\boldsymbol{a^c}da^dds\\
&=\int_s\rho^\pi(s)\int_{a^d}\pi(a^d|s)Q^{d^\pi}(s,a^d)\int_{\boldsymbol{a^c}}\pi(\boldsymbol{a^c}|a^d,s)\\
&\quad\nabla_{\theta_c}\log\pi(\boldsymbol{a^c}|a^d,s)d\boldsymbol{a^c}da^dds\\
&=\int_s\rho^\pi(s)\int_{a^d}\pi(a^d|s)Q^{d^\pi}(s,a^d)\int_{\boldsymbol{a^c}}\nabla_{\theta_c}\pi(\boldsymbol{a^c}|a^d,s)d\boldsymbol{a^c}da^dds\\
&=\int_s\rho^\pi(s)\int_{a^d}\pi(a^d|s)Q^{d^\pi}(s,a^d)\nabla_{\theta_c}\int_{\boldsymbol{a^c}}\pi(\boldsymbol{a^c}|a^d,s)d\boldsymbol{a^c}da^dds\\
&=\int_s\rho^\pi(s)\int_{a^d}\pi(a^d|s)Q^{d^\pi}(s,a^d)\nabla_{\theta_c}1da^dds\\
&=0
\end{aligned}
$$

Thus, our partial state-value function $Q^d(s_t,a_t^d)$ can be compatible with any policy gradient methods without bias to reduce the variance in parameterized action space.

APPENDIX B

For simplicity, we define $\delta_t^V=r_t+\gamma V(s_{t+1})-V(s_t)$. We introduce the generalized advantage estimator of parameters $\boldsymbol{a_t^c}$, as follows.

$$\hat{A}_t^1=-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1}) \tag{15}$$

$$\hat{A}_t^2=-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\gamma\delta_{t+1}^V \tag{16}$$

$$
\begin{aligned}
\hat{A}_t^k=&-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\gamma\delta_{t+1}^V+\gamma^2\delta_{t+2}^V+\dots\\
&+\gamma^k\delta_{t+k}^V
\end{aligned}
\tag{17}
$$

If we take $k\to\infty$, we obtain:

$$\hat{A}_t^\infty=-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\Sigma_{l=1}^\infty\gamma^l\delta_{t+l}^V \tag{18}$$

, which is the difference between empirical returns and the $Q^d(s_t,a_t^d)$ function baseline. Then, the generalized advantage estimator $GAE(\gamma,\lambda)$ of the discrete action can be defined as the exponentially-weighted average of the following k-step estimators.

$$
\begin{aligned}
&\hat{A}_t^{c\ GAE(\gamma,\lambda)}=(1-\lambda)(\hat{A}_t^1+\lambda\hat{A}_t^2+\lambda^2\hat{A}_t^3+\dots)\\
&=(1-\lambda)[(-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1}))\\
&+\lambda[-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\gamma\delta_{t+1}^V]\\
&+\lambda^2[-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\gamma\delta_{t+1}^V+\gamma^2\delta_{t+2}^V]\\
&\dots\\
&=-Q^d(s_t,a_t^d)+r_t+\gamma V(s_{t+1})+\Sigma_{l=1}^\infty(\gamma\lambda)^l\delta_{t+l}^V
\end{aligned}
$$

APPENDIX C

TABLE V
HYPERPARAMETERS OF HA-PPO, PAPPO, AND H-PPO

Hyper-parameters	Value
Learning rate	2e-5
Number of workers	16
Mini-batch size	64
Horizon	512
V baseline loss scaling c_1	0.5
Q baseline loss scaling c_2	0.5
Discount γ	0.99
GAE parameterλ	0.9
Clipping range ϵ	0.2
Clipped global gradient norm	0.5
L2 regularization coefficient c_3	5e-5

APPENDIX D

TABLE VI
SETTINGS OF HFO TASKS

Parameter	Value
Untouched time limit	100
Total time limit	500
Normalized minimum initial position (1vs0)	0.0
Normalized maximum initial position (1vs0)	0.2
Normalized minimum initial position (1vs1)	0.2
Normalized maximum initial position (1vs1)	0.4
offense on ball (1vs1)	false

APPENDIX E

To introduce a no-op action for the goalie in the easy 1vs1 task, we modified two files in the original HFO implementation available on github: /src/main_player.cpp, and /src/role_goalie.cpp. To create a random seed for generating a random value α, we insert the following source code into the *main* function of /src/main_player.cpp.

```
srand(time(NULL));
```

Then, we calculate α each time by

```
float random_n = rand()%10/float(10);
```

when the program excutes the *execute* function of /src/-role_goalie.cpp. When α is less than or equal to 0.3, the goalie will perform a Dash with 0 power, which means no movements, and turn to the ball via the following source code.

```
double power=0.0;
agent->doDash(power);
agent->setNeckAction(new Neck_TurnToBall());
```

Otherwise, the goalie will follow the policy of Agent2D.

APPENDIX F

TABLE VII
DETAILS OF RESULTS IN 1VS1 TASK

Experiment	Success rate[a]	Success rate for training[b]
HA-PPO$_1$	0.732	0.822
HA-PPO$_2$	0.793	0.878
HA-PPO$_3$	0.485	0.725
HA-PPO$_4$	0.879	0.892
HA-PPO$_5$	0.656	0.802
PAPPO$_1$	0.384	0.631
PAPPO$_2$	0.699	0.841
PAPPO$_3$	0.764	0.818
PAPPO$_4$	0.389	0.613
PAPPO$_5$	0.743	0.817
H-PPO$_1$	0.378	0.693
H-PPO$_2$	0.743	0.811
H-PPO$_3$	0.743	0.839
H-PPO$_4$	0.331	0.513
H-PPO$_5$	0.217	0.589
MP-DQN$_1$	0.196	0.415
MP-DQN$_2$	0.398	0.458
MP-DQN$_3$	0.564	0.522
MP-DQN$_4$	0.256	0.521
MP-DQN$_5$	0.310	0.521

[a]Mean of success rate over three trials in difficult 1vs1 task.
[b]Success rate of last 1,000 episodes at the end of training in easy 1vs1 task.

APPENDIX G

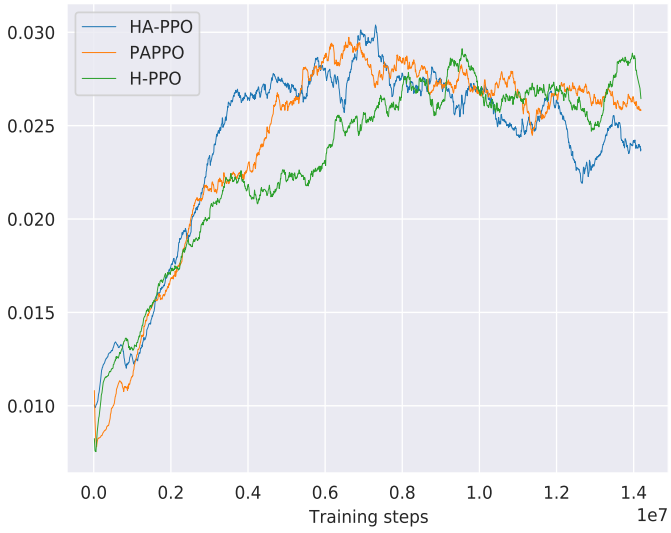

Fig. 6. Standard deviation of the whole advantage in easy 1vs1 task during the training.