Distributed Reinforcement Learning with Self-Play in Parameterized Action Space

Jun Ma[1], Shunyi Yao[2], Guangda Chen[2], Jiakai Song[2], and Jianmin Ji[2*]
[1] School of Data Science, University of Science and Technology of China (USTC), Hefei 230026, China
[2] School of Computer Science and Technology, USTC, Hefei 230026, China
Email: {markjun, ustcysy, cgdsss, sjk1997}@mail.ustc.edu.cn, jianmin@ustc.edu.cn

***Abstract*— Self-play has been shown to be effective to provide a proper training curriculum for a reinforcement learning agent in competitive multi-agent environments without direct supervision. However, its performance is still unstable for problems with sparse rewards, e.g., the scoring task with goalkeeper for robots in RoboCup soccer. It is challenging to solve these tasks in reinforcement learning, especially for those that require combining high-level actions with flexible control. To address these challenges, we introduce a distributed self-play training framework for an extended proximal policy optimization (PPO) algorithm that learns to act in parameterized action space and plays against a group of opponents, i.e., a league. Experiments on the domain of simulated RoboCup soccer show that, the approach is effective and learns more robust policies against various opponents compared to existing reinforcement learning methods. A demonstration video is available online at https://youtu.be/BuL_li1vND4.**

***Index Terms*— reinforcement learning, parameterized action space, distributed, self-play, multi-agents**

I. INTRODUCTION

It is challenging for deep reinforcement learning (DRL) algorithms to solve competitive multi-agent tasks with sparse rewards and parameterized actions, e.g., the scoring task with goalkeeper for robots in RoboCup soccer [1], a.k.a. Half Field Offense (HFO) [2], as shown in Fig. 8. In particular, the offense agents attempt to score against the defense agents. Their parameterized actions are composed of discrete high-level actions, like 'dribbling' and 'running', with continuous action-parameters, like the corresponding coordinates and directions, for flexible control. In this paper, we introduce a distributed self-play training framework for an extended proximal policy optimization (PPO) algorithm to address these problems.

Self-play allows DRL to learn by playing against itself without requiring any direct supervision, which has been applied in many competitive multi-agent tasks with proposing results, such as playing the game of Go [3], Dota 2 [4], and StarCraft II [5], [6]. However, its performance is still unstable for problems with sparse rewards.

There are multiple self-play training paradigms. Fictitious play (FP) [7] generates fictitious players to repeatedly make the best response based on the opponent's average strategy, where the player's average strategy will converge to a Nash equilibrium. Fictitious self-play (FSP) [8] introduces a sample-based machine learning method, which learns an approximation of the best response through reinforcement learning and updates the average strategy through sample-based supervised learning. Neural network virtual self game (NFSP) [9], combining FSP and the neural network function approximation, uses DQN [10] to learn an approximation of Nash equilibrium in an incomplete game without any prior knowledge. Later, asymmetric self-play (ASP) [11] enables one of the two sides of a zero-sum game to set up problems for the other and improves the solution of one's agent. Recently, league self-play (LSP) [12] points out that it is challenging to discover novel strategies by using simple self-game exploration methods, while the alliance is composed of multiple agents iteratively trains the agents to defeat the others. LSP adjusts the mixed probability proportionally according to the winning rate of each opponent to the agent. In this paper, we follow the idea of LSP and train the policy by playing against a group of opponents, i.e., a league.

On the other hand, DRL in parameterized action space has attracted widespread attention [13]. A simple method to handle parameterized actions is to only consider the discretization of the continuous action-parameters, which sacrifices its performance. Another way is to extend the discrete-continuous action space into a continuous set [14], which would increase the difficulty of solving problems. Recently, parameterized deep Q network (P-DQN) [15] and multi-channel deep Q network (MP-DQN) [16] are proposed to directly learn policies based on parameterized actions. Later, they are extended to a multi-agent setting, called multi-agent hybrid Q network (MAHHQN) [17], and trained in a distribute paradigm with parallel curriculum experience replay [18]. Moreover, a novel multi-agent hierarchical policy gradient algorithm (MAHPG) [19] is proposed for parameterized actions, which is capable of learning various strategies and transcending expert cognition by adversarial self-play learning. However, MAHPG is evaluated in an air combat simulation environment, where two competitive agents end up with the same trained policy. It would be more challenging, if two competitive agents end up with different policies in the self-play setting, like the scoring policy for the offense agent and the defense policy for the goalkeeper in the scoring task of RoboCup soccer.

Notice that, a Q-value based method only satisfies the self-consistent equation by training the Q value, thereby indirectly

* The corresponding author.

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

optimizes the agent's performance. Meanwhile, a strategy optimization method can directly optimize the desired policy. Proximal policy optimization (PPO) [20] is such a popular policy gradient algorithm. Recently, hybrid proximal policy optimization (H-PPO) [21] extends PPO to handle parameterized actions. In specific, it applies PPO for the discrete policy and the continuous policy, and updates both policies by minimizing their clipped surrogate objectives, respectively. However, there is no direct relation between the discrete policy network and the continuous policy network. Then the discrete policy has to choose the high-level action only based on the state information without considering the corresponding parameters returned by the continuous policy. In this paper, we introduce another extension of PPO and propose a new hybrid actor-critic DRL framework in parameterized action space. In specific, we consider both the output of the continuous policy network and the state information as the input of the discrete policy network, which can help the discrete policy to learn a more accurate response to the state.

Meanwhile, distributed training architectures, that separate learning from acting and collect experiences from multiple actors running in parallel on separate environment instances, have become an important tool for DRL algorithms to improve the performance and reduce the training time [22], [23], [24], [25], [26], [27]. In this paper, we implement our extended PPO algorithm and the self-play paradigm in a distributed training framework.

We evaluate our approach in the HFO 1v1 task, i.e., the scoring task with an offense agent and a goalkeeper. Our approach learns both an offense policy and a defense policy in the self-play setting. We respectively evaluate their performance for the offense agent and the goalkeeper with multiple different policies. We compare our offense policy with Helios [28], a manually programed policy, and other trained policies by other DRL methods, like DQN, P-DQN, and MP-DQN. The experimental results show that our approach is effective and learns more robust policies against various opponents compared to other DRL methods.

Our main contributions are summarized as follows.

- We extend PPO and propose a new hybrid actor-critic DRL framework in parameterized action space.
- We introduce a distributed self-play training framework for the extended PPO that trains the policy by playing against a league and separates learning from acting.
- We implement a distributed DRL algorithm with self-play for the HFO 1v1 task.
  - Experiments show that the algorithm is effective and learns more robust policies compared to other DRL algorithms.
  - To the best of our knowledge, this algorithm is the first algorithm that succeeds in applying the self-play training paradigm in discrete-continuous (parameterized) hybrid action space for two competitive agents with different policies.

II. Approach

We first introduce the definition of parameterized action Markov decision process (PAMDP) and the corresponding extension of PPO. Then we propose the distributed architecture that uses Redis database to collect replay fragments for distributed training. We also specify the league self-play training paradigm. At last, we provide the entire distributed self-play training framework for the extended PPO algorithm in parameterized action space.

A. Parameterized Action MPDs

The parameterized action space [13] consists of a discrete action set and its corresponding parameter set, defined as follows. Each parameterized action is a pair (k, x_k), where a discrete high-level action $k \in A_d$, a continuous action-parameter $x_k \in X_k$, the discrete action set $A_d = \{1, \ldots, K\}$, and for each k, the value of its action-parameters are in the domain X_k and $X_k \subseteq \mathbb{R}^{m_k}$ for its dimension m_k. Then the action space is specified as follows:

$$A = \bigcup_{k \in A_d} \{\, (k, x_k) \mid x_k \in X_k \,\}.$$

A parameterized action Markov decision process (PAMDP) [13] is defined as a tuple $\langle S, A, P, R, \gamma \rangle$, where S is the set of all states, A is the parameterized action space, $P(s' \mid s, k, x_k)$ is the Markov state transition probability function, $R(s, k, x_k, s')$ is the reward function, and $\gamma \in [0, 1)$ is the reward discount coefficient of the reward function. The policy $\pi : S \times A \to [0, 1]$ aims to maximize the expected discounted rate of return.

B. Parameterized Proximal Policy Optimization

We first specify two policies $\pi_{\theta_x}(\vec{x}_t \mid s_t)$ and $\pi_{\theta_a}(k_t \mid s_t, \vec{x}_t)$ for the vector of continuous action-parameters $\vec{x}_t$ and the discrete high-level action k_t, respectively. In particular, $\pi_{\theta_x}(\vec{x}_t \mid s_t)$ maps a state to a vector of action-parameters for every high-level actions, i.e., $\vec{x}_t = (x_1, \ldots, x_K)$ and

$$\pi_{\theta_x}(\vec{x}_t \mid s_t) : S \to (X_1, \ldots, X_K).$$

$\pi_{\theta_a}(k_t \mid s_t, \vec{x}_t)$ is a mapping from the state space and vectors of action-parameters to a distribution over discrete high-level actions, i.e.,

$$\pi_{\theta_a}(k_t \mid s_t, \vec{x}_t) : S \times (X_1, \ldots, X_K) \times A_d \to [0, 1].$$

As shown in Fig. 1, we propose a new hybrid actor-critic DRL framework in parameterized action space here. Different from the framework in H-PPO, we consider both the result $\vec{x}_t$ of π_{θ_x} and the state s_t as the input of discrete policy network π_{θ_a}, which can help the discrete policy to learn a more accurate response to the state.

We extend PPO to maximize the expected value of π_{θ_x} and π_{θ_a} using the policy gradient method. We use stochastic gradient ascent to maximize the expected return $J(\pi_{\theta_x})$ and $J(\pi_{\theta_a})$, respectively. In specific,

$$\nabla_\theta J(\pi_{\theta_x}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta_x}, \pi_{\theta_a}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_{\theta_x}(\vec{x}_t \mid s_t) A^{\pi_{\theta_x}, \pi_{\theta_a}}(s_t, k_t, \vec{x}_t) \right],$$

$$\theta_{x_{k+1}} = \theta_{x_k} + \alpha \nabla_\theta J(\pi_{\theta_x}),$$

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

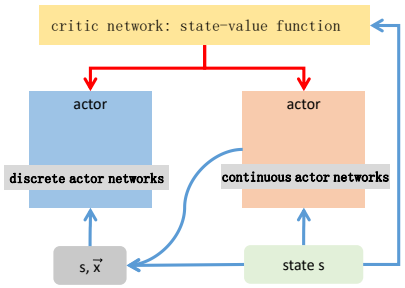


Fig. 1. Hybrid actor-critic DRL framework.

$$\nabla_\theta J(\pi_{\theta_a}) = \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_x},\pi_{\theta_a}}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_{\theta_a}(k_t \mid s_t, \vec{x}_t) A^{\pi_{\theta_x},\pi_{\theta_a}}(s_t, k_t, \vec{x}_t)\right],$$
$$\theta_{a_{k+1}} = \theta_{a_k} + \beta \nabla_\theta J(\pi_{\theta_a}),$$

where α, β are corresponding learning rates and $A^{\pi_{\theta_x},\pi_{\theta_a}}(s_t, k_t, \vec{x}_t)$ denotes the advantage function estimator defined below.

Note that, $A^{\pi_{\theta_x},\pi_{\theta_a}}$ is an estimator of the advantage function for both policy networks. We apply the multi-step optimization that uses the same trajectory τ for both $J(\pi_{\theta_x})$ and $J(\pi_{\theta_a})$. PPO imports importance sampling and the clip method to avoid disruptive and large-scale updates of the policy. Then we define

$$\theta_{x_{k+1}} = \arg\max_{\theta_x} \mathbb{E}\left[L_x(s_t, k_t, \vec{x}_t, \theta_{x_k}, \theta_x)\right], \quad (1)$$
$$L_x(s_t, k_t, \vec{x}_t, \theta_{x_k}, \theta_x) = \min\Big(\frac{\pi_{\theta_x}(\vec{x}_t \mid s_t)}{\pi_{\theta_{x_k}}(\vec{x}_t \mid s_t)} A^{\pi_{\theta_{x_k}},\pi_{\theta_{a_k}}}(s_t, k_t, \vec{x}_t), g(\epsilon, A^{\pi_{\theta_{x_k}},\pi_{\theta_{a_k}}}(s_t, k_t, \vec{x}_t))\Big),$$
$$\theta_{a_{k+1}} = \arg\max_{\theta_a} \mathbb{E}\left[L_a(s_t, k_t, \vec{x}_t, \theta_{a_k}, \theta_a)\right], \quad (2)$$
$$L_a(s_t, k_t, \vec{x}_t, \theta_{a_k}, \theta_a) = \min\Big(\frac{\pi_{\theta_a}(k_t \mid s_t)}{\pi_{\theta_{a_k}}(k_t \mid s_t)} A^{\pi_{\theta_{x_k}},\pi_{\theta_{a_k}}}(s_t, k_t, \vec{x}_t), g(\epsilon, A^{\pi_{\theta_{x_k}},\pi_{\theta_{a_k}}}(s_t, k_t, \vec{x}_t))\Big),$$

where

$$g(\epsilon, A) = \begin{cases}(1+\epsilon)A, & A \geq 0,\\ (1-\epsilon)A, & A < 0,\end{cases}$$

and ϵ is a hyper-parameter that limits the update rate.

We use another network to fit the state value estimation function $V_\phi(s)$. Then we construct the variance reduction advantage function estimator as follows:

$$A^{\pi_{\theta_x},\pi_{\theta_a}}(s_t, k_t, \vec{x}_t) = \delta_t + (\gamma\lambda)\delta_{t+1}, \quad (3)$$

where $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$ and the discount factor $0 \leq \gamma < 1$, $0 \leq \lambda < 1$.

C. Distributed Experience Replay

We introduce the distributed training architecture and the specific implementation of experience replay here. As shown in Fig. 2, the architecture is composed of multiple actor nodes, a learner node, and the shared replay buffer. In particular, multiple actor nodes run in parallel to generate experiences, which would be collected in the shared replay buffer. Then, the learner node learns from the experiences in the buffer and regularly updates the parameters of the networks.

In our implementation, two competitive agents need to be trained in the distributed architecture simultaneously. In particular, there is a training process for each agent, i.e., the offense agent and the defense agent. The training process contains two threads. The first thread collects experiences from each actor node and stores them in the Redis database. The second thread trains the network with the experiences extracted from the Redis database and updates corresponding parameters.

In each episode, the system starts a scoring game between the offense agent and the defense agent, which are driven by the current offense policy network and the defense policy network, respectively. The system also collects corresponding experiences into the buffer during the game. The game will end after a score is made, the ball goes out of bounds, or the goalkeeper catches the ball. With the help of the Redis database, we can implement the distributed training architecture on multiple computers using a Gigabit switch in a local area network (LAN).

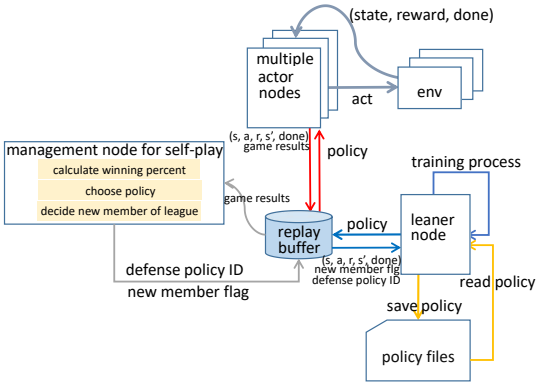


Fig. 2. The distributed self-play training framework for SDP-PPO.

D. League Self-Play

We intend to learn a robust offense policy from a proper training curriculum provided by self-play. During the training process, both the offense policy and the defense policy would be continuously improved. However, due to the catastrophic forgetting phenomenon of neural networks, the offense policy network would "forget" how to defeat early versions of the defense policy. Then we train the offense policy by playing against a group of historical versions of the defense policy, i.e., the league.

In our implementation, we first start multiple scoring games between the offense policy and every version i of the defense policy in the league. Then we calculate the winning rate p_i of the offense policy for each version i. We define $f_i = p_i \times (1 - p_i)$ and choose the version i with the maximum value of f_i as the opponent to compete with the offense policy in a certain number of scoring games to generate corresponding experiences. Note that, f_i achieves

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

its maximum value if $p_i = 0.5$. In our experiments, we find out that the offense policy can be greatly improved by learning from the experiences generated by competing with the defense policy whose p_i is close to 0.5. We also clear the memory and recalculate the winning rate p_i after certain steps of the training, as the early winning history may not be proper for the updated offense policy.

E. SDP-PPO Algorithm

Now we introduce the distributed DRL algorithm SDP-PPO with self-play in parameterized action space. Algorithm 1 specifies the algorithm for an actor node. Algorithm 2 specifies the algorithm for the learner node of the offense agent. Algorithm 3 specifies the algorithm for the learner model of the defense agent. Note that, it also stores multiple versions of the defense policy to maintain the league. Algorithm 4 specifies the algorithm for the management node for self-play, which selects proper versions of the the defense policy and construct corresponding scoring games.

Algorithm 1 Actor node

Initialize SDP-PPO with the experience buffer D
for *episode* $= 1, 2, \ldots, N$ **do**
  (π_{θ_x},π_{θ_a}) ←REDISCLUSTER.GetLearnerParameters();
  reset *env*
  for *step* $k = 1, 2, \ldots, K$ **do**
    $\vec{x}_t = \pi_{\theta_x}(\vec{x}_t \mid s_t)$
    a_t is the result of $\pi_{\theta_a}(a_t \mid s_t, \vec{x}_t)$
    $(r_t, s_{t+1}, d_t) =$ *env*$(a_t, \vec{x}_t)$
    $(s_t, r_t, a_t, \vec{x}_t, d_t) \rightarrow D$
    if d_t **then**
      REDISCLUSTER.Rpush(D)
      REDISCLUSTER.Rpush(*env.game_result*)
      clear D
      break
    end if
  end for
end for

III. EXPERIMENTS

We first evaluate our SDP-PPO on the Platform domain [13], which is based on a simulation environment as shown in Fig. 3 with an agent that has three high-level actions, i.e., run, hop, and leap, and each with a continuous action-parameter to control horizontal displacement. The Platform domain has been widely applied to evaluate the performance of DRL algorithms in parameterized action space. Fig. 4 shows the learning curves of SDP-PPO and other DRL algorithms. In particular, we use P-PPO to denote the ablation version of SDP-PPO without the distributed training. Note that, self-play is not used here. We use DQN to denote our implementation of the DQN algorithm [10] by discretizing the continuous parameters. We use P-DQN and MP-DQN to denote our implementations of the P-DQN algorithm [15] and the MP-DQN algorithm [16], respectively. Fig. 4 shows that both SDP-PPO and P-PPO outperform

Algorithm 2 Offense learner node

Randomly initialize both policy networks π_{θ_x}, π_{θ_a} and the value network V_ϕ
Initialize SDP-PPO with the experience buffer D
REDISCLUSTER.Set(π_{θ_x}, π_{θ_a})
for *epoche* $= 1, 2, \ldots, E$ **do**
  while *length of* D ¡ *epoch* **do**
    D.append(REDISCLUSTER.Lrange(D_i))
    REDISCLUSTER.Lpop(D_i)
  end while
  compute $A^{\pi_{\theta_x}, \pi_{\theta_a}}$ by Equation (3)
  $R_t = r_t + V_{\phi_k}(s_{t+1})$
  update θ_x by Equation (1)
  update θ_a by Equation (2)
  update ϕ by

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D|} \sum_{s^t \in D} \left(V_{\phi_k}(s_t) - R_t\right)^2$$

  clear D
  REDISCLUSTER.Set(π_{θ_x}, π_{θ_a})
end for

Algorithm 3 Defense learner node

defense_id=REDISCLUSTER.Get(*defense_id*)
Randomly initialize both policy networks π_{θ_x}, π_{θ_a} and the value network V_ϕ
Initialize SDP-PPO with the experience buffer D
REDISCLUSTER.Set(π_{θ_x},π_{θ_a})
for *epoche* $= 1, 2, \ldots, E$ **do**
  while *length of* D ¡ *epoch* **do**
    $D_i =$ REDISCLUSTER.Lrange(D_i)
    REDISCLUSTER.Lpop(D_i)
    id = REDISCLUSTER.Get(*defense_id*)
    if *defense_id* = *id* **then**
      D.append(D_i)
    end if
    if REDISCLUSTER.Get(*new defense_id*) **then**
      save file $\pi_{\theta_x}^{defense\_id}$, $\pi_{\theta_a}^{defense\_id}$
      defense_id = *new defense_id*
      REDISCLUSTER.Del(*new defense_id*)
    end if
    REDISCLUSTER.Set($\pi_{\theta_x}^{id}$, $\pi_{\theta_a}^{id}$)
  end while
  compute $A^{\pi_{\theta_x}, \pi_{\theta_a}}$ by Equation (3)
  $R_t = r_t + V_{\phi_k}(s_{t+1})$
  update θ_x by Equation (1)
  update θ_a by Equation (2)
  update ϕ by

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D|} \sum_{s^t \in D} \left(V_{\phi_k}(s_t) - R_t\right)^2$$

  clear D
  REDISCLUSTER.Set(π_{θ_x}, π_{θ_a})
end for

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

Algorithm 4 Management node for League self-play

Require: N_{CHOOSE}: the maximum number of unmatched games for the training model; N_{RESET}: the number of games when to reset the record; P_{RESET}: the winning probability when to reset the record; N_{CALM}: the number of games when the winning rate is not calculated

$defense\_id = 0$
$max\_id = 0$
Initialize the buffer *result* of game results
while *True* **do**
  $result_{defense\_id}$ = REDISCLUSTER.Lrange(*game_result*)
  REDISCLUSTER.Lpop(*game_result*)
  if $notchosen > N_{CHOOSE}$ **then**
    clear $result_{max\_id}$
  end if
  for $id = 0, 1, 2, \ldots,$ *max_id* **do**
    if $result_{max\_id}.sum > N_{RESET}$, $p_{max\_id} > P_{RESET}$ **then**
      $max\_id = max\_id + 1$
      REDISCLUSTER.Set(*new defense_id*, *max_id*)
    end if
    if $result_{id}.sum > N_{RESET}$, $p_{id} > P_{RESET}$ **then**
      clear $result_{id}$
    end if
    if $result_{id}.sum < N_{CALM}$ **then**
      $p_{id} = 0.5$
    else
      Calculate the winning rate p_{id} of $result_{id}$
    end if
    $f_{id} = p_{id} \times (1 - p_{id})$
  end for
  REDISCLUSTER.Set(*defense_id*, $softmax(f)$)
  if $softmax(f) = max\_id$ **then**
    $notchosen = 0$
  else
    $notchosen = notchosen + 1$
  end if
end while

others, and the distributed training paradigm can improve the performance of SDP-PPO.

In the rest of the section, we evaluate SDP-PPO on HFO.

A. Half Field Offense (HFO) Domain

As shown in Fig. 5, Half Field offense (HFO) [2] specifies the scoring task with goalkeeper for robots in RoboCup soccer. In particular, an offense agent is specified by a state with 9 elements, i.e., ego position (x_o, y_o), whether it is holding the ball h, ball's position (x_b, y_b), goal's position (x_g, y_g), and defense agent's position (x_d, y_d). An offense agent has 4 high-level actions, i.e., interception, shooting, dribbling backwards, and dribbling. The dribbling action has two continuous parameters, i.e., the court coordinate (x, y). A defense agent is specified by a state with 7 elements, i.e., ego position (x_d, y_d), ball's position (x_b, y_b), offense agent's position (x_o, y_o), and the speed of the ball v. As illustrated in Fig. 5, a defense agent also has 4 high-level actions, i.e., intercepting (in the large penalty area), running to the small penalty area defense point, running to the shooting defense point, and running in the small penalty area. There are two continuous parameters for the last action, i.e., the coordinate (x, y) in the small penalty area.

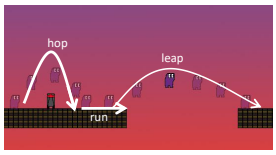


Fig. 3. Platform domain.

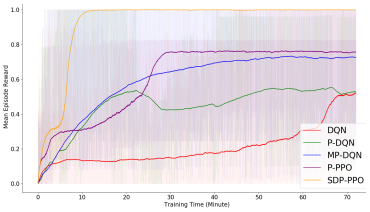


Fig. 4. Learning curves on Platform.

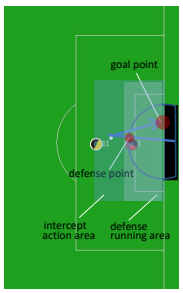


Fig. 5. HFO domain.

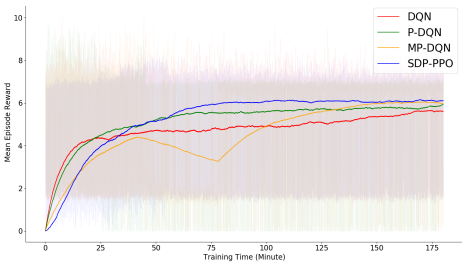


Fig. 6. Learning curves on HFO.

B. DRL Setup

We trained our offense agent and defense agent for the HFO 1v1 task following SDP-PPO on two computers. In specific, Computer 1 is used to train the networks, run the management node (Algorithm 4), and maintain the Redis database. Computer 2 is used to run multiple scoring games and generate corresponding experiences. The training hardware is specified in Table I and hyper-parameters for SDP-PPO are listed in Table II.

TABLE I
TRAINING HARDWARE

	CPU (Intel Core)	DRAM	GPU (GTX)
Computer 1	i5-9600	16GB	1660Ti
Computer 2	i5-9600	16GB	1660super

All networks used in our experiments are fully connected neural networks with 6 hidden layers, which has 128 neurons with the ReLU activation.

C. Reward Function

HFO is a challenging task with sparse rewards. Different reward shaping methods would greatly affect the performance of DRL algorithms. For a fair comparison, we continue to use the reward function proposed in [2] for the

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

TABLE II

HYPER-PARAMETERS FOR SDP-PPO

Parameter	Value
learning rate for parameter policy	3×10^{-4}
learning rate for discrete policy	1×10^{-3}
learning rate for value	3×10^{-3}
replay buffer size (D_{max})	2000
maximum episode length	1000
clip ratio (ε)	0.2
discount factor (γ)	0.99
λ	0.97
N_{CHOOSE}	20
N_{RESET}	120
P_{RESET}	0.5
N_{CALM}	20

offense agent, i.e.,

$$r_t = d_{t-1}(a,b) - d_t(a,b) + \mathbb{I}_t^{kick} + 3(d_{t-1}(b,g) - d_t(b,g)) + 5\mathbb{I}_t^{goal},$$

where $d(a,b)$ denotes the distance between the agent to the ball and $d(b,g)$ denotes the distance between the ball to the goal. Clearly, the reward function encourages the agent to approach the ball and dribble the ball close to the goal. Moreover, $\mathbb{I}_t^{kick}$ returns 1 if the agent kicks the ball for the first time and $\mathbb{I}_t^{goal}$ returns 1 if the agent scores the goal. Notice that, this reward function is still quite sparse and tells no information for the defense agent.

We specify the following reward function for the defense agent,

$$r_t = 100\mathbb{I}_t^{nogoal} - 100\mathbb{I}_t^{goal} - 5\mathbb{I}_t^{risk},$$

where $\mathbb{I}_t^{nogoal}$ returns 1 if the goalkeeper holds the ball or the ball is out of bounds, $\mathbb{I}_t^{goal}$ returns 1 if the offense agent scores the goal, $\mathbb{I}_t^{risk}$ returns 1 if the defense agent leaves the small penalty area and the offense agent is approaching. Note that, this reward function is also quite sparse and tells no information for the offense agent.

D. Experimental Results

We trained an offense agent and a defense agent by SDP-PPO using self-play for the HFO 1v1 task. We not only compare the performance our offense agent with the ones by DQN, P-DQN, and MP-DQN on different defense agents, but also with two manually programed policies, i.e., Helios Base[1] and Helios [28].

Fig. 6 shows the learning curves of SDP-PPO and other DRL algorithms for the offense agent, when the defense agent is driven by Helios Base. It shows that SDP-PPO outperforms other DRL algorithms in the task. Notice that, although the total reward for SDP-PPO is only slightly better than P-DQN and MP-DQN, the winning rate for SDP-PPO is 94.6% which is much better than the winning rate for the both, i.e., 85.0% and 84.2%. Fig. 7 shows the learning curves of 8 different versions of the defense agent in SDP-PPO by the self-play training. Note that, we intend to learn a better offense agent, then the total rewards decrease for every version of the defense agent during the training.

[1]Helios used to be the champion of RoboCup 2D soccer and Helios Base is the open source version of Helios at https://zh.osdn.net/projects/rctools/.

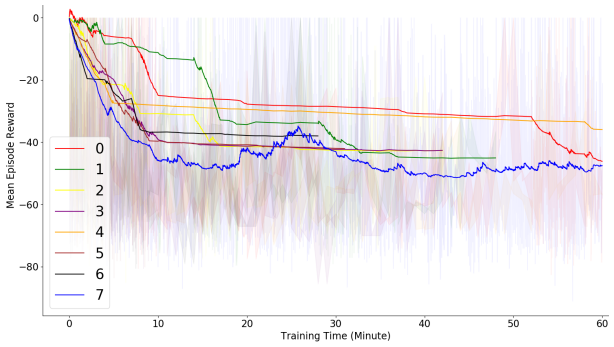


Fig. 7. Learning curves of 8 versions of the defense agent in SDP-PPO.

Table III summarizes the performance of all algorithms on the HFO 1v1 task in our experiments by 500 trials. In particular, Random denotes a random policy based on discretized actions as by DQN. Note that, both the offense agent and the defense agent in SDP-PPO are only trained by self-play, without using other defense policies in the training process. However, other DRL algorithms, i.e., DQN, P-DQN, MP-DQN, require themselves to be trained in environments with the defense agent driven by Helios Base.

Table III shows that the offense agent learned by SDP-PPO outperforms the one learned by other DRL algorithms in almost all cases, which shows that SDP-PPO is efficient and learns more robust policies. The only exception is the case when the defense agent is driven by Helios. This is mainly due to the fact that Helios has manually programed a useful policy against the shooting action of the offense agent. Notice that, the HFO platform and the corresponding actions for offense are constructed from the open source project of Helios Base. Then Helios Base, other DRL algorithms, and our SDP-PPO are based on similar sets of actions. Meanwhile, different from others, Helios can perform other fine-grained actions which can be used to generate useful policies against those actions applied by other methods.

TABLE III

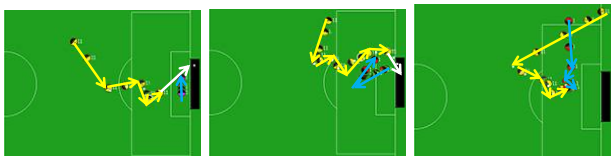
PERFORMANCE OF ALGORITHMS ON HFO 1V1

Offense \ Defense	Random	Helios Base	Helios	SDP-PPO
Random	75.8%	8.20%	4.6%	41.6%
Helios Base	98.2%	88.6%	91.2%	84.8%
Helios	**99.8%**	**99.6%**	**96.0%**	94.8%
DQN	24.0%	82.6%	85.6%	79.2%
P-DQN	83.0%	85.0%	79.4%	83.8%
MP-DQN	80.6%	84.2%	68.4%	87.8%
SDP-PPO	98.4%	94.6%	74.4%	**96.8%**

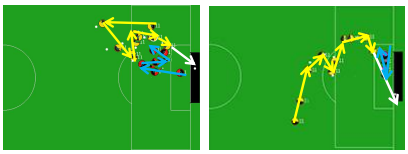
It also shows that the offense agent learned by DQN performs poorly when it competes with the Random defense. This is mainly due to the certain way that we have used to discretize corresponding actions for both the offense and defense agents. It is quite easy for the Random defense policy

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

(a) P-DQN offense vs. Helios Base defense (b) P-DQN offense vs. SDP-PPO defense (c) P-DQN offense vs. SDP-PPO defense



(d) MP-DQN offense vs. Helios Base defense (e) MP-DQN offense vs. SDP-PPO defense (f) MP-DQN offense vs. SDP-PPO defense



(g) SDP-PPO offense vs. SDP-PPO defense (h) SDP-PPO offense vs. Helios Base defense

Fig. 8. The trajectory of the players and the ball

to choose a discretized defense action to stop a discretized offense action generated by the DQN offense policy.

Fig. 8 illustrates typical scenarios for different offense and defense agents, where yellow arrows denote the attack of the offense, blue arrows denote the move of the defense, and white arrows denote the moving direction of the ball after the shooting action.

IV. Conclusions

In this paper, we introduce SDP-PPO, an extension of PPO in parameterized action space with the distributed self-play training. We first extend PPO and propose a new hybrid actor-critic DRL framework in parameterized action space. Then we introduce a distributed self-play training framework for the extended PPO that trains the policy by playing against a league and separates learning from acting.

We evaluate SDP-PPO in the HFO 1v1 task, i.e., the scoring task with an offense agent and a goalkeeper. We compare SDP-PPO with existing DRL algorithms, like DQN, P-DQN, and MP-DQN. Different from others, SDP-PPO trains both the offense agent and the defense agent by self-play, and does not need to use other defense policies during the training. The experimental results show that SDP-PPO is effective and learns more robust policies against various opponents compared to other DRL methods.

References

[1] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada, "The robocup synthetic agent challenge 97," in *Robot Soccer World Cup*. Springer, 1997, pp. 62–73.

[2] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone, "Half field offense: An environment for multiagent learning and ad hoc teamwork," in *Proceedings of the 15th AAMAS Adaptive Learning Agents Workshop (ALA-16)*. sn, 2016.

[3] 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*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[6] P. Sun, J. Xiong, L. Han, X. Sun, S. Li, J. Xu, M. Fang, and Z. Zhang, "Tleague: A framework for competitive self-play based distributed multi-agent reinforcement learning," *arXiv preprint arXiv:2011.12895*, 2020.

[7] D. Fudenberg and D. K. Levine, "Consistency and cautious fictitious play," *Levines Working Paper Archive*, 1996.

[8] E. Hendon, H. J. Jacobsen, and B. Sloth, "Fictitious play in extensive form games," *Games And Economic Behavior*, vol. 15, no. 2, pp. 177–202, 2013.

[9] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *arXiv preprint arXiv:1603.01121*, 2016.

[10] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *Proceedings of the 6th International Conference on Learning Representations (ICLR-2018)*, 2018.

[12] L. Han, J. Xiong, P. Sun, X. Sun, and Z. Zhang, "Tstarbot-x: An open-sourced and comprehensive study for efficient league training in starcraft ii full game," *arxiv-2011.13729*, 2020.

[13] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, vol. 30, no. 1, 2016.

[14] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *Proceedings of the 4th International Conference on Learning Representations (ICLR-2016)*, 2015.

[15] 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," *Proceedings of the 6th International Conference on Learning Representations (ICLR-2018)*, 2018.

[16] 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.

[17] H. Fu, H. Tang, J. Hao, Z. Lei, Y. Chen, and C. Fan, "Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces," *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

[18] Y. Li and J. Ji, "Parallel curriculum experience replay in distributed reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS-21)*, 2021.

[19] H. Piao, "Multi-agent hierarchical policy gradient for air combat tactics emergence via self-play," *Engineering Applications of Artificial Intelligence*, vol. 98, 2020.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR abs/1707.06347 (2017)*, 2017.

[21] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

[22] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[23] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–12, 2016.

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.

[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[25] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *Proceedings of the 6th International Conference on Learning Representations (ICLR-2018)*, 2018.

[26] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *Proceedings of the 6th International Conference on Learning Representations (ICLR-2018)*, 2018.

[27] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell *et al.*, "Alphastar: Mastering the real-time strategy game starcraft ii," *DeepMind blog*, vol. 2, 2019.

[28] H. Akiyama, T. Nakashima, T. Fukushima, J. Zhong, Y. Suzuki, and O. An, "Helios2018: Robocup 2018 soccer simulation 2d league champion," *Springer, Cham*, 2018.

Manuscript 324 submitted to 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Received April 8, 2021.