PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Comprehensive cooperative deep deterministic policy gradients for multi-agent systems in unstable environment

Dong Xie, Xiangnan Zhong, Qing Yang, Yan Huang

Dong Xie, Xiangnan Zhong, Qing Yang, Yan Huang, "Comprehensive cooperative deep deterministic policy gradients for multi-agent systems in unstable environment," Proc. SPIE 11006, Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications, 110060K (10 May 2019); doi: 10.1117/12.2519153

SPIE.

Event: SPIE Defense + Commercial Sensing, 2019, Baltimore, Maryland, United States

Comprehensive Cooperative Deep Deterministic Policy Gradients for Multi-Agent Systems in Unstable Environment

Dong Xie[a], Xiangnan Zhong[a], Qing Yang[b], and Yan Huang[b]

[a]Department of Electrical Engineering, University of North Texas, Denton, TX, USA 76207
[b]Department of Computer Science and Engineering, University of North Texas, Denton, TX, USA 76207

ABSTRACT

Nowadays, intelligent unmanned vehicles, such as unmanned aircraft and tanks, are involved in many complex tasks in the modern battlefield. They compose the networked intelligent systems with varying degrees of operational autonomy, which will continue to be used increasingly on the future battlefield. To deal with such a highly unstable environment, intelligent agents need to collaborate to explore the information and achieve the entire goal. In this paper, we will establish a novel comprehensive cooperative deep deterministic policy gradients (C_2DDPG) algorithm by designing a special reward function for each agent to help collaboration and exploration. The agents will receive states information from their neighboring teammates to achieve better teamwork. The method is demonstrated in a real-time strategy game, StarCraft micromanagement, which is similar to a battlefield with two groups of units.

Keywords: multi-agent systems, comprehensive cooperative deep deterministic policy gradients, real-time strategy game, intelligent control.

1. INTRODUCTION

Recently, the actor-critic structures with deep neural networks are considered as one of the most important branches in reinforcement learning and have been developed rapidly.[1–4]

By combining the actor-critic structure and deep neural network, it is expanded into an algorithm combining value and policy ideas. Two neural networks are used in the actor-critic design. Specifically, the actor-network is responsible for generating actions, while the critic network is designed to estimate the corresponding value function to evaluate the performance of specific actions.[5,6]

Besides, the policy gradient (PG) method is proposed as a classic method for reinforcement learning to develop action strategies.[7] Through a probability distribution function, the optimal strategy of each step is expressed, and the sampling strategy is carried out according to the probability distribution in each step to obtain the current optimal action. Furthermore, deterministic policy gradient (DPG) is proposed with deterministic behavior strategy, and the action of each step can be directly obtained with the determined continuous value.[8]

Taking advantage of DPG, another advanced algorithm, deep deterministic policy gradients (DDPG), is designed by integrating deep neural network into the actor-critic structure to simulate policy function and value function to achieve improved convergence rate.[9,10] DDPG not only performs stably in a series of tasks in continuous action space but also takes fewer steps to obtain the optimal solution compared with other algorithms. Moreover, memory replay and target networks make training sample effective utilization to update the neural network. DDPG has attracted many attentions and been applied in various fields. For instance, DDPG was validated in process cooperative control tasks with multi-agent environment.[11] The authors developed a multi-agent deep deterministic policy gradient (MADDPG). Each agent has its independent actor, which inputs its

Further author information: (Send correspondence to Xiangnan Zhong)
Dong Xie: E-mail: dongxie@my.unt.edu
Xiangnan Zhong: E-mail: Xiangnan.Zhong@unt.edu
Qing Yang: E-mail: Qing.Yang@unt.edu
Yan Huang: E-mail: Yan.Huang@unt.edu

Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications
edited by Tien Pham, Proc. of SPIE Vol. 11006, 110060K · © 2019 SPIE
CCC code: 0277-786X/19/$18 · doi: 10.1117/12.2519153

states and outputs certain actions. Besides, each agent also corresponds to a critic, but the critic training data comes from all agents, which is optimizing the overall performance of all policies. Moreover, each agent is allowed to have its reward function to improve the stability and robustness of the method so that it can be used for cooperative and competitive tasks.[12]

Recently, DeepMind has published AlphaStar, which had been built for two years, onto the stage of history, creating the first artificial intelligence (AI) to defeat StarCraft II's top professional players.[13,14] This exciting news has redrawn public attention on AI's game challenges.[15–18] As one of the most famous real-time strategy game, StarCraft is produced by Blizzard Entertainment, which has rich multi-level gameplay and aims to challenge human intelligence. Due to its high complexity and strategy, this game has become one of the largest and most successful games in history, with players competing in e-sports competitions for more than 20 years. With the development of Brood War application programming interface (BWAPI) and StarCraft II learning environment (PySC2),[19,20] StarCraft has become the best platform for reinforcement learning research.[21–23]

Motivated by the above research, we developed a novel comprehensive cooperative deep deterministic policy gradients (C_2DDPG) algorithm to manipulate cooperation among different types of agents on the StarCraft game to achieve the ultimate team goal with a continuous strategy. The major contributions of this paper are summarized as follows: First, we design a separate actor-critic structure for each kind of units and provide cooperative strategies for the entire team. Second, different reward functions are shaped based on the agents' characteristics such that the agents can better cooperate to achieve the ultimate goal and defeat the enemies. Third, the proposed C_2DDPG method is validated on the starcraft strategy game with a complex scenario, which is similar to the battlefield environment.

The rest of this paper is organized as follows. In section 2, we provide the background of algorithm mathematical knowledge. Then, section 3 presents the designed C_2DDPG approach. Furthermore, section 4 shows the experiment design and StarCraft training configuration with detailed reward shaping under our algorithm. In section 5, experiment results are provided to demonstrate the effectiveness of our proposed method. Finally, section 6 concludes our work.

2. BACKGROUND

In this part, we provide the background of our algorithm. Markov decision processes, policy gradient, and deterministic policy gradient, deep deterministic policy gradient will mainly discuss.

2.1 Markov Decision Processes

In Markov decision processes (MDPs), agent and environment are the important and interrelated cores. Here are some basic concepts for understanding MDPs.

- State: s_t is the state of agent in environment at time t, which is including agent own information and observe surounding effective information.
- Action: a_t is the action of agent taking in environment at time t. After pocessing a_t, agent will obtain a new s_{t+1} from environment.
- Policy: π is the policy function of the agent, which tells the agent how to take the next action in the environment.
- Reward: r_t is a feedback value, which indicates agent performance level at step t in the environment.
- Return: R is viewed as a total discount reward, which is the weighted sum of all rewards from current to future time. At time t, R_t is generated in equation (1),

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \tag{1}$$

where $0 \leq \gamma \leq 1$ is defined as a determined parameter, which is called discount factor. $r(s_i, a_i)$ is the obtained reward after making action a_i in state s_i.

The basic idea of reinforcement learning is an MDPs in which agents learn by trial and error to determine the ideal behavior from its own experience with the environment. An agent in state s takes an action a with the policy π in environment, and then environment returns a new state s' and a reward r to the agent. The goal for the reinforcement learning method is to obtain an optimal policy π that determines the decision of choosing action a with state s.

2.2 Policy Gradient and Deterministic Policy Gradient

The key part for policy gradient is the probability distribution function $\pi_\theta(s_t|\theta^\pi)$, which is used to represent the optimal strategy for each step. Action decision is carried out according to the probability distribution in each step to obtain the current optimal action value.

$$a_t \sim \pi_\theta(s_t|\theta^\pi) \tag{2}$$

The process of generating action is essentially a random process, and the final learned strategy should be a stochastic policy.

Based on PG, DPG is proposed, which is embodied in outputting only the determined action a for the state s instead of randomly selecting the action a according to the probability distribution. The advantage of DPG lies in that when the dimension of value function is high, DPG uses fewer samples but outputs more gradient estimation efficiency. The behavior of each step directly obtains the determined value through the policy function.

$$a_t = \mu(s_t|\theta^\mu) \tag{3}$$

The function μ is an optimal behavior strategy, and is a deterministic strategy.

2.3 Deep Deterministic Policy Gradient

DDPG optimizes the actor-critic structure of DPG and improves the operation efficiency of the algorithm by referring to the target network and experience replay from deep Q-network (DQN) method.[24] In an actor, policy function is estimated by the neural network. The input of the neural network states s, and the output is action a. In addition, the critic part is regarded as part of the value function, which also uses neural networks to judge the quality of the action output by the strategy function. The value function takes actions and states as inputs (s, a), and $Q(s, a)$ as outputs.

Therefore, there are four networks in the DDPG method, two networks from actor and two networks from critic part. Assuming the actor-network is $\mu(s|\theta^\mu)$, and the critical network is $Q(s, a|\theta^Q)$. Their corresponding target actor-network and critic network are $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$, respectively. θ^μ and θ^Q are the weights for actor and critic networks, and $\theta^{\mu'}$ and $\theta^{Q'}$ are the weights for target actor and critic networks.

By updating the loss function with mean squared error, the estimation of value functions is generated by critic network.

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{4}$$

where y_i is the approximated real value which is formulated as,

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{5}$$

where $Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ is the output generated from critic target network, and $\mu'(s_{i+1}|\theta^{\mu'})$ is the actor target network produced output.

Furthermore, the actor-network is used for optimizing policies, by using a policy gradient method. A policy objective function is marked as $J(\theta^\mu)$ to evaluate a policy strategy. By searching appropriate θ^μ, the actor lets $J(\theta^\mu)$ to get the maximum value. Thus, the derivative of $J(\theta^\mu)$ to θ^μ is the policy gradient to optimize J.

Theoretically, the policy parameter θ^μ is updated in the direction of increasing the value $Q(s, a|\theta^Q)$ generating from critic network, so policy strategy is marked as $\mu(s|\theta^\mu)$, by modifying the policy gradient equation in the deterministic policy gradient, DDPG actor network parameters are updated by the following,[8]

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \cdot \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i} \tag{6}$$

where $Q(s, a|\theta^Q)$ is critic network output, and $\mu(s|\theta^\mu)$ is actor network output.

Besides, DDPG is off-policy method, and the policy used to generate behavior value is not the final strategy. Therefore, $\mu(s|\theta^\mu)$ is not agent taking action a_{t+1}, in order to ensure exploration, a policy of N is to add random noise $\mathcal{N}$ to $\mu(s|\theta^\mu)$ policy, using Ornstein-Uhlenbeck process, as shown in the following equation (7).[25]

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{7}$$

Based on the methods of DQN, DDPG perfectly integrates deep neural network, experience playback and target network with DPG. However, unlike the target update method in DQN, DDPG uses soft update method, which makes the target network take a small changes every step. The soft update of DDPG uses an average method that makes it easy to operate the changes of target network. The following equation gives the update of the target network.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{8}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{9}$$

where τ is defined as the update parameter.[9]

The connection between this actor-critic network is summarized as follows. The environment will first give a state, and then the agent will make a decisive action according to the participant network. Next, the environment will give a reward and a new state after receiving this action. In addition, DDPG will update the critic network according to the awards, and update the actor network based on the direction recommended by the critic network. After that, the agent continues to the next step. This cycle continues until DDPG has trained a satisfied actor network.

3. ALGORITHM INTRODUCTION

In this chapter, we will discuss the development of the comprehensive cooperative deep deterministic policy gradients method explicitly.

Because of the complexity of most of the battlefield environment, the traditional DDPG method is not insufficient to guarantee the performance of intelligent agents to reach the optimal goal independently. Therefore, we consider the cooperation among different agents based on their characteristics and design the C_2DDPG method by providing each kind of agents with its own decision-making strategies and achieve the ultimate goal as teamwork.

Specifically, each kind of agents has its own actor-critic structure with not only the information of themselves but also the state information of their teammates. This structure will help the agents understand the nearest surroundings and make more comprehensive decisions. Furthermore, the agents have memory buffers which can store the status information of the same kind of teammates, which will speed up the process of updating and optimizing actor-critic. Besides, the reward functions for different types of agents are different based on their characteristics, such that each kind of agents can act as a unique role in the combat and they can be cooperative with different types to achieve the ultimate goal.

Therefore, the proposed C_2DDPG method advances the traditional DDPG in two ways: First, the proposed method can provide different control strategies for different kinds of agents based on their reward function design. C_2DDPG will improve the teamwork of the agents by taking advantage of their own characteristics to achieve comprehensive cooperation. Second, we design different kinds of memory buffers for the agent to store the previously visited states. In this way, the updating process of neural network weights of each type of agents can use their memory buffer. Therefore, the method can improve the weights updating efficiency and reduce the convergence time.

The detailed algorithm of the proposed C_2DDPG method is provided in Algorithm 1.

Algorithm 1 C_2DDPG Algorithm

for agent type j = 1, K **do**
  Initialize critic network $Q_j(s, a|\theta_j^Q)$ and actor network $\mu_j(s|\theta_j^\mu)$ with weights θ_j^Q and θ_j^μ, respectively.
  Initialize target network Q'_j with weights $\theta_j^{Q'}$ and μ'_j with weights $\theta_j^{\mu'}$
  Initialize replay buffer R_j with previous model memory
end for
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    for agent type j = 1, K **do**
      Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action a_t at and observe reward r_t and observe new state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in R_j
      Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R_j
      Set $y_i = r_i + \gamma Q'_j(s_{i+1}, \mu'_j(s_{i+1}|\theta_j^{\mu'})|\theta_j^{Q'})$
      Update critic by minimizing the loss:
        $L = \frac{1}{N}\sum_i (y_i - Q_j(s_i, a_i|\theta_j^Q))^2$
      Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta_j^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q_j(s, a|\theta_j^Q) \cdot \nabla_{\theta_j^\mu}\mu_j(s|\theta_j^\mu)$
      Update the target networks:
        $\theta_j^{Q'} \leftarrow \tau\theta_j^Q + (1-\tau)\theta_j^{Q'}$
        $\theta_j^{\mu'} \leftarrow \tau\theta_j^\mu + (1-\tau)\theta_j^{\mu'}$
    end for
  end for
end for

4. EXPERIMENT SETTINGS

This section introduces the StarCraft scenario settings, the design of C_2DDPG with the comprehensive reward functions. The StarCraft scenario platform is designed on BWAPI and TorchCraft.[19,26]

4.1 StarCraft Scenario

There are two different units, Vulture and Zergling, in our experiment combat scenario. In StarCraft, the units are viewed as the agents in reinforcement learning. The detailed description of the two combat unit parameters is provided in Table 1.

Table 1. Vulture and Zergling units information.

Parameters	Vulture	Zergling
Attack Cooldown	1.26	0.336
Attack Damage	20	5
Attack Range	5	1
Health Value	80	35
Observable Range	8	5

Our combat scenario is set up with (3 Vultures and 10 Zerglings vs. 3 Vultures and 10 Zerglings), so the type and quantity of units are the same, in other words, there are three Vultures and ten Zerglings on each side. A traditional strategy like hit and run may not be useful for the same unit strategy. In order to win the game, the goal is set up to obtain the maximum reward as high as possible and make the number of steps as low as possible. Figure 1 is showing for the battlefield environment setting.

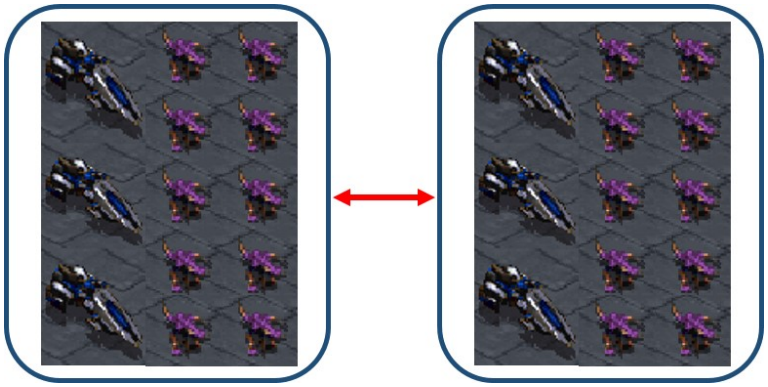

Figure 1. StarCraft scenario setting: 3 Vultures and 10 Zerglings vs. 3 Vultures and 10 Zerglings.

4.2 The Settings of C_2DDPG

Current state, current action, current reward, and next state are the four primary inputs for our C_2DDPG method. The number of variables of each unit state is 25, and Table 2 describes the detail information of the unit state. All state variables are using standard parameters, which is normalized within $[0, 1]$.

Table 2. The description of unit state.

Health	four variables which are the unit own current health value, health value level, and enemy current health value, enemy health value level
Number	four variables which are the number of the existing units, the number of the existing enemies, and the number of the unit's teammates and enemies within unit observable range
Attack	six variables which are unit attack cooldown value, enemy attack cooldown value, unit action, enemy action, unit under-attacking information, enemy under-attacking information
Position	nine variables which are unit map location (x, y), enemy map location (x, y), the nearest enemy distance coordinates (dx, dy, d), where dx, dy and d are the horizontal distance, the vertical distance and the minimum distance, respectively, and the enemy attack target map location (x, y)
Status	two variables including unit and enemy state of life or death

In the C_2DDPG network, each type of unit has its network of actor and critic. The number of nodes in the network hidden layer is set to 150. The participant network generates the next action strategy after inputting the current state. After the critic network obtains the new action strategy with the current state, it feeds back a value score to enable the actor to update its network. Besides, an activation function called rectifying linear unit is introduced to replace the traditional sigmoid function to process network weights. Besides, Adam optimizer is used to update the network weights.

Three-dimensional action is the output of C_2DDPG, which is a move or attack, location (x, y). Location determines the direction of movement. In this case, compared with other reinforcement learning algorithms with limited choices, the movement of each unit becomes a continuous way and increases the chance of adopting the best strategy. Moreover, the action is normalized within $[-1, 1]$ range. Thus, if the first action parameter is greater than 0, the unit will attack the nearest enemy. If the nearest enemy is not within the range of unit fire, the unit will move to the nearest enemy. If the first action parameter is less than 0, the unit will move to a certain position according to the second and third parameters. For example, if the action is $[-1, -0.5, 1]$, the unit will move in the $\tan^{-1}(\frac{-0.5}{1}) = -30°$ direction.

After each step, the current state, action, reward, and next state are stored as a batch in each unit type memory pool for memory replay training. When training the networks, each step randomly selects 160 connecting batches from their unit type memory pool to update the neural network. In our experiment, three Vultures use and synchronize the same memory pool, and ten Zerglings share and update the second memory pool.

4.3 Reward Function

In our experiment, we design the reward function into three parts: attack reward, position reward and destruction reward. The reward equation is shown with equation (10).

$$r = \alpha \times r_a + \beta \times r_p + \gamma \times r_d \tag{10}$$

where r_a, r_p and r_d are respectively scored with attacked reward, position reward and destruction reward. Futhermore, α, β and γ are respectively attacking reward weight, position reward weight and destroy reward weight. r_a is described the attack reward as

$$r_a = \rho_1 \times (e_t - e_{t-1}) - \rho_2 \times (u_t - u_{t-1}) \tag{11}$$

where e_t is the enemy health value at time t, and e_{t-1} is enemy health value at time $t-1$. Meanwhile, u_t and u_{t-1} are respectively unit health value at time t and $t-1$. Besides, ρ_1 is enemy health value weight, and ρ_2 unit health value weight.

The second reward part, position reward is shown with equation (12),

$$r_p(d) = \begin{cases} 0, & 0 \le d \le A \\ -0.1, & A < d \le B \\ -0.5, & B < d \le C \end{cases} \tag{12}$$

where d is the distance between own unit and the nearest enemy unit. Furthermore, area A, B, C respectively represent unit attach range, unit observable range and map boundaries.

Moreover, the destruction reward formula is defined as the following equation (13),

$$r_d = \tau_1 \times (n_0 - n_t) + \tau_2 \times h \tag{13}$$

where n_0 is the initial number of enemies, n_t is the current number of enemies, h is unit alive status indicating unit state of life or death, and τ_1 and τ_2 are the weights for destroy enemy factor and unit life status, respectively.

Because there are two types of units in our experiment, although the overall structure of the reward function is consistent, according to their parameters, the weight of the corresponding reward is different. Therefore, each type of unit will obtain different values and choose different actions according to its corresponding reward function to form a variety of different strategies.

5. RESULT ANALYSIS

Three kinds of data will be used to detect and analyze the effectiveness of our algorithm. We recorded 1,000 episodes of training with 3 Vultures and 10 Zerglings vs. 3 Vultures and 10 Zerglings. At the beginning of each episode, both sides have their own 3 Vultures and 10 Zerglings units. At the end of each episode, the outcome is decided, and either side loses all the controlled units. As the most import index, winning rates indicates the number of games our units have won every 100 episodes, for example, the winning rate is 0.4, which means we win 40 times in 100 episodes. Total steps indicate the average number of steps each unit takes in each episode, and it takes an average in every 100 episodes. Moreover, the calculation method of normalized rewards is similar to total steps record method, which is to take an average every 100 episodes showing the average reward obtained by each unit at the end of each round. Besides, the final rewards are normalized from 0 to 1.

Figure 2 shows the winning rates data. From the figure, we can identify that in the first 300 episodes, winning rates is slowly increasing, and especially in the first 100 episodes, the combat units under our control does not

gain a single victory. After 300 training times with episodes, the number of wins per 100 games reached 40. When it comes from 300 to 400 training episodes, the number of games winning times increases rapidly, and winning rates exceeds 0.8. After a huge jump for winning rates, the number of wins is not declined. In the next 600 training episodes, winning rates reaches a higher stable period, with the number of wins per 100 games maintained from 95 to 100.

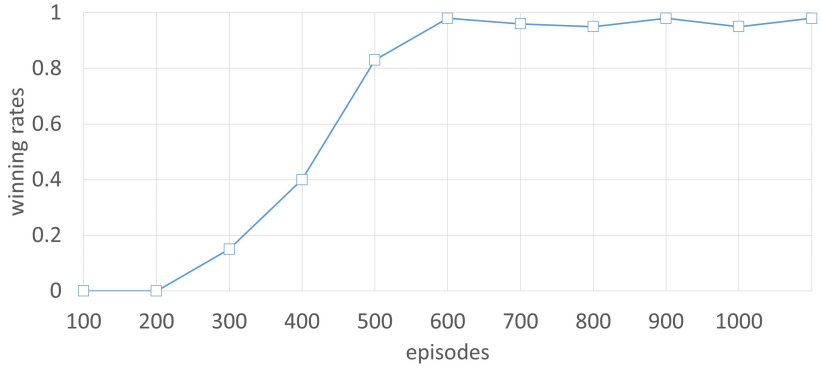

Figure 2. Winning rates for 3 Vultures and 10 Zerglings vs. 3 Vultures and 10 Zerglings.

The information of total steps is shown in Figure 3. After constant exploration and exploitation of the algorithm, the number of action steps for each combat unit in each battle decreases from more than 600 action decisions in the first 100 rounds to more than 300 in the fourth 100 rounds. At the same time, the corresponding winning rates are continuously increasing, which shows that our unit is steadily reducing additional actions and actively participating in the battle. From 500th to 1000th training episodes, although the winning rates have entered the stability, the total steps are still slowly decreasing, so we can conclude that the efficiency of combat units is continuously improving.

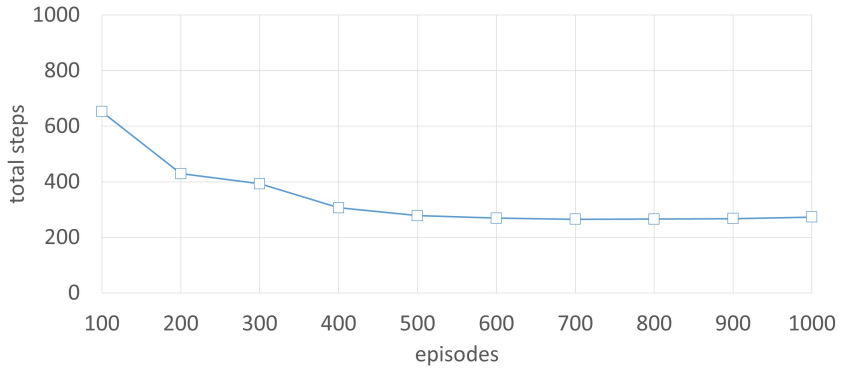

Figure 3. Total steps for 3 Vultures and 10 Zerglings vs. 3 Vultures and 10 Zerglings.

The result of normalized rewards is shown in Figure 4. Its general trend is similar to the outcome of winning rates. Although it did not have a period of rapid growth, its growth is very steady. Unit earns 0.2 effective normalized rewards in the first 100 training episodes, and then the normalized rewards rise to 0.85 after 400 training rounds with a rate of 0.2 per 100 training episodes. In the following 600 training episodes, the normalized rewards map is the same as the winning rates, reaching a stable period of 0.95 to 1.

By analyzing the three kinds of data, our algorithm can control two different types of units to carry out combat operations, and after a few learning steps, we can obtain a high winning rate. The winning rates and normalized rewards have been improving with the increasing number of training episodes, while total steps have been decreasing because of the reduction of ineffective actions, active participation in combat and the formation

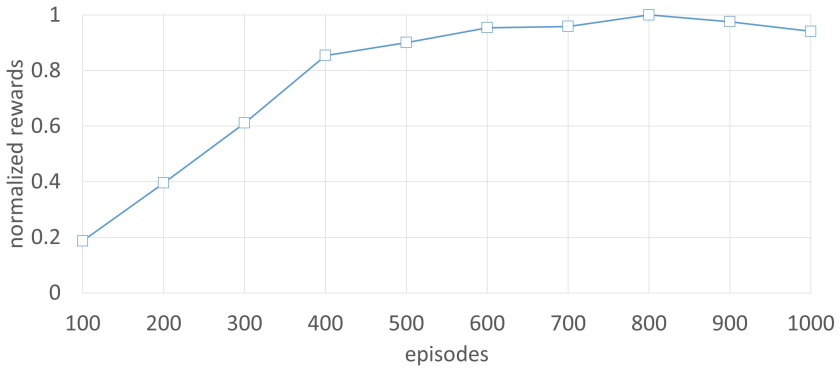

Figure 4. Normalized rewards for 3 Vultures vs. and 10 Zerglings vs. 3 Vultures and 10 Zerglings.

of relevant strategies. Also, with the process of continuous training, two different types of units have formed a variety of effective strategies. For example, units learn how to distract the enemy's attention and concentrate superior firepower to strike the enemy. The deployment of combat positions has also been learned. Zerglings can rush to the front of the battlefield to take part in the battle because of its fast attack speed and a large number of attacks. Furthermore, Zerglings can also be used as the bait to distract the enemy, and Vulture is at the combat rear because of its firepower advantage, using large-range firepower to focus on destroying the enemy who is attacking our unit. Through our C_2DDPG training, the two different units are responsible for each other, and at the same time they cooperate and fight together in the combat team.

6. CONCLUSION

This paper developed new comprehensive cooperation deep deterministic policy gradients algorithm for multi-agent systems to improve the cooperation among different kinds of agents. In the training process, different kinds of agents had their own particular control structures to generate distributed actions. The reward function was designed based on their characteristics to make the agents achieve comprehensive cooperation. The StarCraft game was applied to validate the proposed method. The experience results show that this method can make the agents cooperate to protect their teammates and defend the enemy at the same time with reduced training time.

REFERENCES

[1] Barto, A. G., Sutton, R. S., and Anderson, C. W., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics* (5), 834–846 (1983).

[2] Kaelbling, L. P., Littman, M. L., and Moore, A. W., "Reinforcement learning: A survey," *Journal of artificial intelligence research* **4**, 237–285 (1996).

[3] Potjans, W., Morrison, A., and Diesmann, M., "A spiking neural network model of an actor-critic learning agent," *Neural computation* **21**(2), 301–339 (2009).

[4] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R., "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42**(6), 1291–1307 (2012).

[5] Konda, V. R. and Tsitsiklis, J. N., "Actor-critic algorithms," in [*Advances in neural information processing systems*], 1008–1014 (2000).

[6] Bhatnagar, S., Ghavamzadeh, M., Lee, M., and Sutton, R. S., "Incremental natural actor-critic algorithms," in [*Advances in neural information processing systems*], 105–112 (2008).

[7] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y., "Policy gradient methods for reinforcement learning with function approximation," in [*Advances in neural information processing systems*], 1057–1063 (2000).

[8] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., "Deterministic policy gradient algorithms," in [*International Conference on Machine Learning*], (2014).

[9] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *International Conference on Learning Representations* (2016).

[10] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P., "Benchmarking deep reinforcement learning for continuous control," in [*International Conference on Machine Learning*], 1329–1338 (2016).

[11] Gupta, J. K., Egorov, M., and Kochenderfer, M., "Cooperative multi-agent control using deep reinforcement learning," in [*International Conference on Autonomous Agents and Multiagent Systems*], 66–83, Springer (2017).

[12] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I., "Multi-agent actor-critic for mixed cooperative-competitive environments," in [*Advances in Neural Information Processing Systems*], 6379–6390 (2017).

[13] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., et al., "AlphaStar: Mastering the real-time strategy game StarCraft II." https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ (2019).

[14] Arulkumaran, K., Cully, A., and Togelius, J., "Alphastar: An evolutionary computation perspective," *arXiv preprint arXiv:1902.01724* (2019).

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

[16] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al., "Mastering the game of go without human knowledge," *Nature* **550**(7676), 354 (2017).

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing atari with deep reinforcement learning," in [*NIPS Deep Learning Workshop*], (2013).

[18] Yannakakis, G. N. and Togelius, J., [*Artificial Intelligence and Games*], Springer (2018).

[19] Heinermann, A., "BWAPI: Brood war API, an API for interacting with starcraft: Broodwar (1.16.1)." https://bwapi.github.io (2015).

[20] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al., "Starcraft II: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782* (2017).

[21] Buro, M., "Real-time strategy games: A new ai research challenge," in [*IJCAI*], 1534–1535 (2003).

[22] Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M., "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games* **5**(4), 293–311 (2013).

[23] Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S., "The starcraft multi-agent challenge," *arXiv preprint arXiv:1902.04043* (2019).

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

[25] Uhlenbeck, G. E. and Ornstein, L. S., "On the theory of the brownian motion," *Physical review* **36**(5), 823 (1930).

[26] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N., "Torchcraft: a library for machine learning research on real-time strategy games," *arXiv preprint arXiv:1611.00625* (2016).