Deep Deterministic Policy Gradients with Transfer Learning Framework in StarCraft Micromanagement

Dong Xie, Xiangnan Zhong
Department of Electrical Engineering
University of North Texas
Denton, TX, USA 76207
Email: dongxie@my.unt.edu, Xiangnan.Zhong@unt.edu

***Abstract*—This paper proposes an intelligent multi-agent approach in a real-time strategy game, StarCraft, based on the deep deterministic policy gradients (DDPG) techniques. An actor and a critic network are established to estimate the optimal control actions and corresponding value functions, respectively. A special reward function is designed based on the agents' own condition and enemies' information to help agents make intelligent control in the game. Furthermore, in order to accelerate the learning process, the transfer learning techniques are integrated into the training process. Specifically, the agents are trained initially in a simple task to learn the basic concept for the combat, such as detouring moving, avoiding and joining attacking. Then, we transfer this experience to the target task with a complex and difficult scenario. From the experiment, it is shown that our proposed algorithm with transfer learning can achieve better performance.**

***Index Terms*—multi-agent, deep deterministic policy gradients, strategy game, intelligent control, transfer learning.**

I. Introduction

Recently, reinforcement learning with deep neural networks has received increasing attention [1]. Deep reinforcement learning (DRL) provides an opportunity to train the agent to solve a series of human-level tasks by an end-to-end fashion [2]. For instance, deep Q-network (DQN) uses the experience replay technique and a target network to remove the correlations between samples and stabilize the training process [3]. Furthermore, policy gradients (PG) is proposed based on gradient descent to find the action with its probability [4]. An extension algorithm combining value and policy ideas is called actor-critic by applying two different neural networks as the actor and critic network [5]. The actor is designed to produce actions, while the critic network is developed to evaluate the performance of certain actions by generating the corresponding value functions [6]. One of the most successful actor-critic algorithms is deep deterministic policy gradients (DDPG), which uses a deep neural network for function approximation and can solve the problem of continuous motion space [7], [8]. Recently, DDPG has been introduced into multi-agent problems. DDPG was used to train intractable cooperative multi-agent control tasks [9]. A multi-agent deep deterministic policy gradient (MADDPG) was established to help agents not only cooperate with each other in communication and navigation but also build decision thinking including physical deception and predator-prey [10]. In [11], the authors designed the novel experimental method with MADDPG for multi-agent systems, and promising performance was achieved.

Moreover, DRL has been integrated into the game field, from classic Atari video games to traditional board games, and made huge progress in the field [12]–[16]. Since the players are required to take precisely operation for a game character in a real-time strategy game to accomplish a mission, multi-agent computer games have received increasing attention and become an excellent testbed to challenge various algorithms [17], [18]. Among most of the computer games, StarCraft, designed by Blizzard Entertainment, has been published more than 20 years, and it is still one of the most popular real-time strategy games. Because of the complex environment and diverse units, StarCraft provides an ideal research platform for multi-agent study [19]. Recently, due to the development of Brood War application programming interface (BWAPI) [20], the study of StarCraft control is making impressive progress [21], [22]. In recent years, many efficient programming applications including BWMirror and TorchCraft are developed to promote the development in this field [23]. Actually, many researchers develop new methods on StarCraft to overcome decision making in an uncertain environment. A Bayesian model for StarCraft agents was developed to predict opponent decision making [24]. Through improving Sarsa and neural networks, parameter sharing multi-agent gradient-descent Sarsa (PS-MAGDS) algorithm was designed to keep a high win rate in an uncertain environment with different units [25]. Furthermore, [26] developed a multi-agent bidirectionally-coordinated network with modified actor-critic structure. In [27], a master-slave architecture can help agent effectively study information from local agents in micromanagement tasks. Moreover, [28] developed a zero-order (ZO) backpropagation algorithm to enable the agents to achieve better performance with non-trivial strategies for scenarios. Besides, counterfactual multi-agent (COMA) policy gradients, is proposed with a centralized critic and decentralized actors to generate agents' action [29].

Motivated by the above research, in this paper, we designed the deep deterministic policy gradient algorithm for the StarCraft strategy game to achieve intelligent control. The major contributions of this paper are as follows. First, the DDPG algorithm is designed into the StarCraft game to enable the agents to move in a continuous fashion. This is critical in improving the control performance of a highly unstable environment on the battlefield. Second, the transfer learning techniques are integrated into the designed method to facilitate the learning process. This will make the agents obtain effective information from simple task and present a better outcome with a more complex environment. Third, a special reward function is designed for the StarCraft game based on the agents' own situation and enemies' information ensuring the agents to achieve steady high win rate with limited training times in complex environments.

The rest of this paper is organized as follows. In Section II, we formulate the StarCraft problem analyzed in this paper and provide the background of transfer learning method. Section III provides our designed DDPG approach with transfer learning method in real-time StarCraft micromanagement scenario. Furthermore, Section IV shows the experiment design, and the experiment results demonstrate the effectiveness of our proposed method. Section V concludes our work.

II. Problem Formulation

A. Reinforcement Learning in StarCraft

The StarCraft game can be viewed as a Markov decision process (MDP) in which the agent interacts with the environment continuously. Specifically, the agent receives a current state from the environment, based on it providing the control signal. Then the environment returns a new state to the agent. Meanwhile, the environment also provides a reward signal to determine the goal. Therefore, the agent updates its policy according to the reward given by the environment. So the main purpose of reinforcement learning is to get the optimal strategy for maximum reward. The following are some basic concepts for reinforcement learning connecting with StarCraft.

a) Policy: π is the behavior function of the agent, a mapping from state to action, which tells the agent how to select the next action in StarCraft.

b) Episode: an episode is one round for StarCraft, that consists of a series of state s_t, action a_t, reward r_t,

$$(s_1, a_1, r_1, s_2, a_2, r_2, ..., s_{n-1}, a_{n-1}, r_{n-1}, s_n, a_n, r_n) \quad (1)$$

which is from initial step to terminal step. After one episode, the outcome will be posted, and the setting of scenario will be initialized.

c) Reward: r_t is a feedback signal and a value indicating how well the agent did at step t in the game.

d) Return: G is defined as cumulative discount reward. Return at step t is formulated in equation (2),

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \quad (2)$$

where $0 \leq \gamma \leq 1$ is called discount factor.

The goal of reinforcement learning is to maximize the cumulative rewards it receives in the long run. However, based on equation (2), the return score and episode are uncertain, and there are many possible situations, where the return is a variable. Therefore, the task becomes to maximize the expectation of cumulative rewards with the following value function. The value function corresponding to a state s is

$$V_\pi(s) = E_\pi(G_t | s_t = s) \quad (3)$$

and this value function is also called the state value function. For a given s, $V_\pi(s)$ indicates the expected value of return when following the policy π starting from state s.

Besides, there is another kind of value function is called state action-value function,

$$Q_\pi(s, a) = E_\pi(G_t | s_t = s, a_t = a) \quad (4)$$

which indicates the expected value of return when taking action a from state s with policy π. After having the guide of the value function, agents will accumulate scores purposefully and make better performance in the game.

B. Transfer Learning

Transfer learning as a powerful tool in the AI field has made much progress in recent years [30], [31].

The key idea of transfer learning is to find the similarities between the new problem and the original problem, and make rational use of them. Given a marked source domain $D_s = \{x_i, y_i\}_{i=1}^{n}$ and an unmarked target domain $D_t = \{x_i\}_{j=n+1}^{n+m}$, the data distribution between marked source domain $P(x_s)$ and unmarked target domain $P(x_t)$ are different, i.e. $P(x_s) \neq P(x_t)$. The purpose of transfer learning is to learn the knowledge of the target domain D_t with the experience of source domain information D_s.

III. Learning Method for StarCraft

In this chapter, we develop the DDPG algorithm with transfer learning and reward design for the StarCraft computer game. The structure of the algorithm can be described in Fig. 1.The designed DDPG algorithm includes three parts: actor, critic, and experience replay.

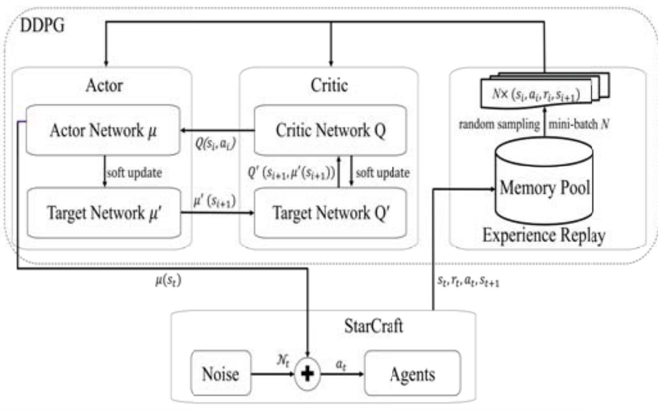

Fig. 1. The structure of our DDPG approach.

A. Reward Design

In order to improving the ability of learning strategies, we develop an unique reward function for StarCraft agents. It includes three parts: attack reward, position reward and destroy reward. The reward equation is written below,

$$r = \alpha \cdot r_a + \beta \cdot r_p + \gamma \cdot r_d \tag{5}$$

where r_a, r_p and r_d are attacked reward, position reward and destroy reward respectively. α, β and γ are attacking reward weight, position reward weight and destroy reward weight respectively. The attack reward r_a is described as

$$r_a = (ehv_t - ehv_{t-1}) - \rho \cdot (ahv_t - ahv_{t-1}) \tag{6}$$

where ehv_t and ehv_{t-1} are the enemy health value at time t and $t-1$, respectively. Also, ahv_t and ahv_{t-1} are agent health value at time t and $t-1$, respectively. Besides, ρ is the weight for agent health value ratio.

Moreover, the position reward is defined as equation (7),

$$r_p(d) = \begin{cases} 0, & 0 \le d \le A \\ -0.1, & A < d \le B \\ -0.5, & B < d \le O \end{cases} \tag{7}$$

where d is the distance between own agent and closest enemy. Besides, area A, B, O stand for agent fire range, battle field and other areas, respectively.

Furthermore, the destroy reward is formulated as the following equation (8),

$$r_d = en_0 - en_t + \tau \cdot o \tag{8}$$

where en_0 is the initial number of enemy, en_t is the current number of enemy, o is agent life status means whether agent is alive or not, and τ the weight for agent life status.

B. Neural Network Implementation

Our method uses DDPG neural networks to approximate the function, and the value function network called a critic network has action and state as the input (s, a), and $Q(s, a)$ as the output. In addition, another neural network is used to approximate the policy function which is called actor network that the input is state s, and output is action a. Moreover, the target networks are used in the learning method to ensure the convergence of parameters.

Suppose the critic network is $Q(s, a|\theta^Q)$, its corresponding target critic network is $Q'(s, a|\theta^{Q'})$. Actor network is $\mu(s|\theta^\mu)$, its corresponding target actor network is $\mu'(s|\theta^{\mu'})$. θ^Q and θ^μ are the weights for critic and actor networks, and $\theta^{Q'}$ and $\theta^{\mu'}$ are target network weights.

1) Critic Network: The critic network is used for approximation of value functions. Besides, critic network is updated by the loss function with mean squared error as below,

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{9}$$

where y_i is the estimated real value which can be written as,

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{10}$$

where $Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ is the output from critic target network, and $\mu'(s_{i+1}|\theta^{\mu'})$ is the actor target network output. Then the gradient descent method is used for updating. Besides, both actor and critic use the target network to calculate the target value.

2) Actor Network: Actor network is used for optimizing policies. In order to evaluate a strategy, there should have a goal called policy objective function at first, which is marked as $J(\theta^\mu)$. The actor wants $J(\theta^\mu)$ to get the maximum value by finding the appropriate θ^μ. Therefore, the derivative of $J(\theta^\mu)$ to θ^μ is the policy gradient.

The policy parameter θ^μ is updated in the direction of increasing the value of value function $Q(s, a|\theta^Q)$. The strategy is marked as $\mu(s|\theta^\mu)$, and the actor network parameter is updated by the following equation (11),

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \cdot \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i} \tag{11}$$

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

Our method is off-policy, and the policy used to generate behavior value and the policy used to evaluate are not the same policy. In other words, the action a_{t+1} actually taken by the agent is not generated by $\mu(s|\theta^\mu)$. In order to ensure exploration, a policy $\mathcal{N}$ is to add random noise with Ornstein-Uhlenbeck process to the $\mu(s|\theta^\mu)$ policy in equation (12) [32].

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{12}$$

Moreover, by using soft update, the target network is updated every step with small update amplitude. The following equations show the target networks updating,

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{13}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{14}$$

where τ is the update parameter [7].

In a nutshell, actor and critic estimate the policy and corresponding value function for StarCraft agents. Then, agents implement this policy with noise to deal with the unstable StarCraft environment. Furthermore, experience replay collects each agent's information, and stores in the memory pool for the following actor and critic training.

C. Transfer Learning Design in DDPG

In order to increase the learning rate, we integrate transfer learning into our DDPG method. Transfer learning is an optimal shortcut to save time and get better performance. A basic network $N_1$is trained with the initial data set to achieve an original task. After that, the network is transferred to a second target network N_2. Then the transferred network N_2 is trained with the target data set and the target task.

Therefore, instead of randomly initializing the neural network weights, our algorithm transfers the well-trained networks in the traditional DDPG method. Moreover, the experience memory pool is imported by well-trained model experience. The designed DDPG with transfer learning algorithm is presented below.

Algorithm 1 DDPG Algorithm with transfer learning

```
1:  Initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with transfer weights $\theta^Q$ and $\theta^\mu$ from previous actor and critic networks.
2:  Initialize target network $Q'$ and $\mu'$ with transfer weights from previous target networks
3:  Initialize replay buffer $R$ with previous model memory
4:  for episode=1, M do
5:      Initialize a random process $\mathcal{N}$ for action exploration
6:      Receive initial observation state $s_1$
7:      for t = 1, T do
8:          Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
9:          Execute action $a_t$ at and observe reward $r_t$ and observe new state $s_{t+1}$
10:         Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
11:         Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
12:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13:         Update critic by minimizing the loss:
                $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
14:         Update the actor policy using the sampled policy gradient:
                $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q) \cdot \nabla_{\theta^\mu}\mu(s|\theta^\mu)$
15:         Update the target networks:
                $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
                $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
16:     end for
17: end for
```

IV. Experiment Analysis

This section provides the experiment settings and experimental analysis of two StarCraft scenarios. This StarCraft scenario platform is similar to the one used in [26]. The code has been implemented with TorchCraft [23].

A. Experiment Settings

Our experiment mainly involves four different kinds of units, Dragoon, Vulture, Zergling and Zealot, in two different combat scenarios (3 Vultures vs. 20 Zerglings and 2 Dragoons with 3 Zealots vs. 2 Dragoons with 3 Zealots). TABLE I gives a detailed description of the parameters of four combat units. Here, we consider the units in StarCraft as the agents in reinforcement learning.

The first scenario, we will verify through a comparative experiment that the transfer could complete the unit training faster and better. In the second scenario, we will show that our DDPG with transfer learning method is effective in a more complex environment. In addition, the combined comprehensive power for the enemy side is equal to or greater than our controlled units. StarCraft AI takes a heuristic strategy to control the enemy, and these two scenes are difficult for ordinary Starcraft players to control and win the game. We conducted 1000 experiments in two scenarios and counted the win rates of each experimental groups to determine the experiment result. We record winning episodes divided by the total number of episodes trained as win rates that are calculated in every 100 episodes.

TABLE I
THE PARAMETERS OF FOUR UNITS.

Attributes	Dragoon	Vulture	Zergling	Zealot
Hitpoints	100	80	35	100
Range	4	5	1	1
Sight	8	8	5	7
Cooldown	1.26	1.26	0.336	0.924
Ground Damage	20	20	5	8

Our method has four main inputs - current state, current action, current reward, and next state. The dimension of the unit state is 25, and the detail information is shown as follows. All states are normalized into $[0, 1]$.

- Health: includes three variables which are the unit's own current health value, health value rank, and enemy current health value.
- Shield: includes two variables which are the unit's own current shield value and enemy current shield value.
- Number: includes 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 in its observation.
- Attack: includes five variables which are unit power cooldown value, unit attacking or moving, enemy attacking or moving, unit under-attacking status, enemy unit under-attacking status.
- Position: includes nine variables which are unit position coordinates (x, y), enemy unit position coordinates (x, y), the closest enemy distance coordinates (dx, dy, d), where dx, dy and d are horizontal distance, vertical distance and minimum distance, respectively, and enemy attacking target position coordinates (x, y).
- Status: there are two variables including own unit and enemy alive or dead status.

For DDPG networks, there are 150 nodes in the hidden layer for actor and critic networks, respectively. By using current state, actor network produces the next action policy. After determining the new policy with the current state, the critic network gives a value to assist the actor in updating its network. Rectified Linear Unit (ReLU) as a type of activation function is used to process network weights instead of the sigmoid function. Besides, Adam optimizer is imported to update network weights.

The output is an action with three dimensions, which are move or attack, position (x, y), which determines the direction for the move. In this way, the movement for each unit becomes a continuous fashion and increase the opportunity to move to a better position and form the best strategy compare to the other reinforcement learning algorithms with limit choices. We normalize the number in the range of $[-1, 1]$. Therefore, for the action first parameter, if it is greater than 0, the unit will attack the closest enemy. If not, the unit will move to somewhere depend on the second and third parameter. For example, action is $[-0.5, 0.2, 0.4]$, unit will move with $\tan^{-1}(\frac{0.2}{0.4}) = 27°$ direction.

In each step, the current state, action, reward and next state are stored as one batch in memory pool for replay training. Therefore, 160 batch size from memory is randomly selected for each step to update the neural networks. Besides, based on the attributes of four types of units, each type of units has its specific reward function weights.

B. Scenario One

The first combat scenario is shown in Fig. 2(b), we control 3 Vultures to fight against 20 Zerglings. The enemy has higher attack speed and quantity advantage, while we have a better attack range and attack capability, and we have only three units. In this scenario, we have conducted two groups of experiments. The first group is the DDPG training experiment, and the second group is DDPG with transfer learning method (Algorithm 1), whose network initialized parameters are migrated from the Fig. 2(a) 3 Vultures vs. 15 Zerglings with 200 episodes training.

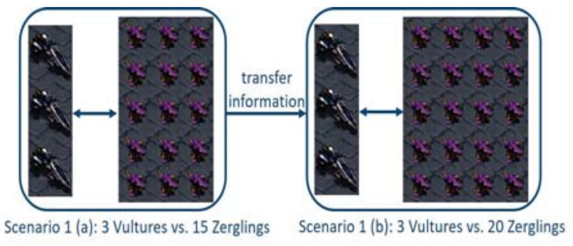
transfer
information
Scenario 1 (a): 3 Vultures vs. 15 Zerglings
Scenario 1 (b): 3 Vultures vs. 20 Zerglings

Fig. 2. StarCraft scenario 1 setting.

The first scenario has a larger number of enemies which take part in the fight, so the new task is more difficult for DDPG empowering agents to win the game. Therefore, transfer learning is imported to help agents take advantage of the game. Fig. 3 presents that the original DDPG and transfer learning DDPG method win rates after transferred 200 episodes knowledge from 3 Vultures vs. 15 Zerglings.

Our method has three main improvements comparing to original DDPG. Firstly, the initial starting point is higher than DDPG, which helps agents save strength for experience collection and action trails. Secondly, transfer learning method has a higher increasing rate than DDPG, from the diagram, transfer learning method achieves 0.7 win rates after 300 trails. However, DDPG still jumps into the stage of constantly testing

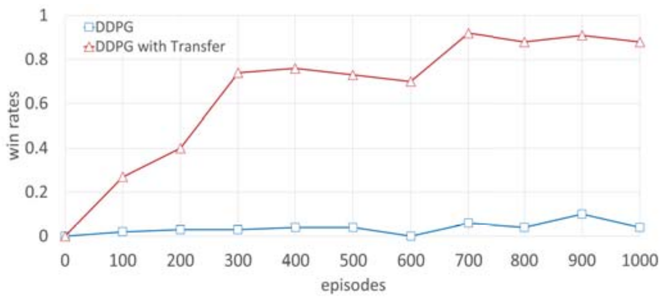

Fig. 3. Win rates for 3 Vultures vs. 20 Zerglings.

period and cannot have better win rates after 1000 episodes. Lastly, transfer learning reaches a higher platform after a short training, and it keeps more than 0.9 win rates after 700 episodes training. Nevertheless, DDPG stops at 0.1 win rates after 1000 episodes training.

C. Scenario Two

The second combat scenario is shown in Fig. 4(b), and we control 2 Dragoons with 3 Zealots to fight against 2 Dragoons with 3 Zealots. We have the same units and number with the enemy. In this scenario, we will also show the original DDPG and DDPG with transfer learning method result whose network is transferred from Fig. 4(a) 3 Zealots vs. 3 Zealots with 200 episodes training. Then after the same initial settings, we retrain 2 Dragoons with 3 Zealots vs. 2 Dragoons with 3 Zealots. Fig. 4 shows the units settings.

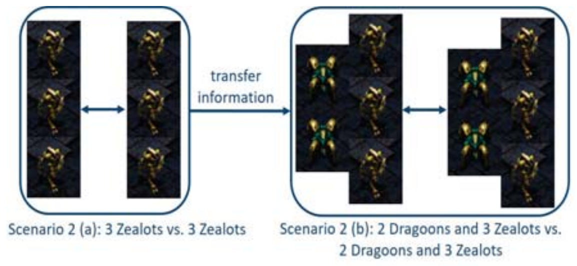
transfer
information
Scenario 2 (a): 3 Zealots vs. 3 Zealots
Scenario 2 (b): 2 Dragoons and 3 Zealots vs.
2 Dragoons and 3 Zealots

Fig. 4. StarCraft scenario 2 setting.

From Fig. 5, the 2 Dragoons and 3 Zealots vs. 2 Dragoons and 3 Zealots experiment result is obviously showed that the DDPG with transfer learning method could obtain a high growth trend of win rates at first. Then after 700 episodes training, our units are able to reach more than 98 number of wins in every 100 episodes. But the DDPG method only keeps 0.3 win rates after training 1000 episodes. Besides, we compare our result with DQN, PG, and ZO. In 2 Dragoons and 3 Zealots vs. 2 Dragoons and 3 Zealots scenario, DQN and PG only respectively achieve 0.61 and 0.69 win rates, and ZO has 0.9 win rates [28]. However, our DDPG with transfer learning method stabilizes at 0.98 win rates after 1000 episodes training. Thus, the result shows that our method is supporting multi-agents learning strategy quickly.

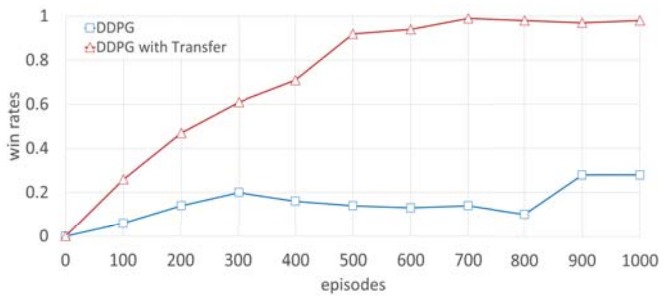

Fig. 5. Win rates for scenario 2. Both teams have 2 Dragoons and 3 Zealots joining the combat.

V. Conclusion

In this paper, we developed the DDPG with transfer learning framework for StarCraft micromanagement with multiple agents. In this way, we enabled the agents to move in a continuous fashion. We also designed the reward function considering the agents' own situation and enemies' information to ensure the agents achieving steady high win rate with limited training times. Two experiments were conducted with detailed analysis. From the experiment results, we know that the proposed method could achieve better performance with less training time in the target task.

Acknowledgment

This paper is supported by NSF grant 1850240.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[2] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[3] 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, p. 529, 2015.

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

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

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

[7] 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," in *International Conference on Learning Representations*, 2016.

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

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

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

[11] S. L. Barton, N. R. Waytowich, E. Zaroukian, and D. E. Asher, "Measuring collaborative emergent behavior in multi-agent reinforcement learning," in *International Conference on Human Systems Engineering and Design: Future Trends and Applications*. Springer, 2018, pp. 422–427.

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

[13] 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, p. 484, 2016.

[14] 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, p. 354, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

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

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

[18] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, "A review of computational intelligence in rts games," in *Foundations of Computational Intelligence (FOCI), 2013 IEEE Symposium on*. IEEE, 2013, pp. 114–121.

[19] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Magazine*, vol. 33, no. 3, p. 106, 2012.

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

[21] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[22] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontannón, and M. Čertickỳ, "Starcraft bots and competitions," *Encyclopedia of Computer Graphics and Games*, pp. 1–18, 2016.

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

[24] G. Synnaeve and P. Bessiere, "A bayesian model for opening prediction in rts games with application to starcraft," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 2011, pp. 281–288.

[25] K. Shao, Y. Zhu, and D. Zhao, "Starcraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

[26] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[27] X. Kong, B. Xin, F. Liu, and Y. Wang, "Revisiting the master-slave architecture in multi-agent deep reinforcement learning," *arXiv preprint arXiv:1712.07305*, 2017.

[28] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks," *arXiv preprint arXiv:1609.02993*, 2016.

[29] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] S. J. Pan, Q. Yang *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[31] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, 2016.

[32] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.