IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 28, 2019, accepted September 17, 2019, date of publication September 20, 2019, date of current version October 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2942635

A Novel Method for Improving the Training Efficiency of Deep Multi-Agent Reinforcement Learning

YAOZONG PAN, HAIYANG JIANG, HAITAO YANG, AND JIAN ZHANG
Space Engineering University, Beijing 101416, China
Corresponding author: Yaozong Pan (panyaozong1284@163.com)

ABSTRACT Deep reinforcement learning (RL) holds considerable promise to help address a variety of multi-agent problems in a dynamic and complex environment. In multi-agent scenarios, most tasks require multiple agents to cooperate and the number of agents has a negative impact on the training efficiency of reinforcement learning. To this end, we propose a novel method, which uses the framework of centralized training and distributed execution and uses parameter sharing among homogeneous agents to replace partial calculation of network parameters in policy evolution. The parameter asynchronous sharing mechanism and the soft sharing mechanism are used to balance the exploratory of agents and the consistency of homogenous agents' policy. We experimentally validated our approach in different types of multi-agent scenarios. The empirical results show that our method can significantly promote training efficiency in collaborative tasks, competitive tasks, and mixed tasks without affecting the performance.

INDEX TERMS Multi-agent, reinforcement learning, neural network, parameter sharing, MADDPG, training efficiency.

I. INTRODUCTION

In recent years, deep reinforcement learning (DRL) has been very active, which is selected as one of the MIT Technology Review 10 Breakthrough Technologies in 2017. The breakthroughs mainly include deep Q-network(DQN) for Atari games [1], [2] and strategic policies combined with tree search for the game of go [3], [4]. DQN solves problems with high-dimensional observation spaces. Deep deterministic policy gradient(DDPG) [5] addresses problems with continue observation and action spaces. Other notable examples of utilising DRL include learning robot control strategy from video [6] playing video games [7], indoor navigation [8] et al. Most of these studies belong to single agent reinforcement learning.

Nevertheless, in reality, multi-agent systems(MAS) are applied in a variety of fields such as robotic teams [9], distributed control [10], collaborative decision [11], etc. [12] and [13] have done some impressive work in the research of high-order MAS.

In multi-agent scenarios, it is very difficult to pre-design behaviors for agents when the environment is complex and changing over time. The agents learn new policies online, which is helpful to improve the performance of the agents gradually [14]. Reinforcement learning can realize the evolution of policy through interaction between agents and the environment. So far, the multi-agent reinforcement learning community presented perhaps the most expressive progress towards autonomous learning in MAS [15]. To extend reinforcement learning to MAS, the core challenge is to specify a multi-agent learning goal [16], because the return of an agent is influenced by other agents, and cannot be maximized independently. By combining reinforcement learning with game theory, many algorithms of multi-agent reinforcement learning (MARL) are formed, for example, Team-Q [17], Distributed-Q [18], [19] et al for fully cooperative tasks; Minimax-Q for fully competitive tasks; Nash-Q [20], [21], WoLF-PHC [22], [23] for mixed tasks. But these algorithms cannot address the problem of multi-agent credit assignment. Agents cannot use their own reward function separately.

Recently, deep learning has been applied to MARL, which leads to a crossed area—deep MARL. This area integrates the development of deep learning, game theory, and reinforcement learning(RL). Recent works focus on solving non-stationarity [24], communication problems [25]–[27] and credit assignment [28], [29]. It's remarkable that recent works

The associate editor coordinating the review of this manuscript and approving it for publication was Jianxiang Xi.

have proposed a centralized critic decentralized actor model, in which critic networks update their parameters by joint actions and states, but each agent acts according to local observations and receives returns based on its own reward function separately [24], [30].

However, in the course of training, It is necessary to compute the network parameters for each agent at every time step, which reduces the training efficiency of the method. In this work, we combine the existing work [24] with parameter sharing, and propose a new method to improve the training efficiency. In our method, firstly, we classify agents according to their reward functions. Secondly, only one agent's parameters are calculated for each policy evolution in the same class of agents. Finally, other agents in the same class share parameters to realize policy evolution. In order to solve the problem of exploration-exploitation trade-off, asynchronous parameter sharing and parameter soft sharing are adopted in our method to control the degree of parameter consistency among agents in the same class. We evaluate our method in the testbed of multi-agent particle environment, which has been used in works [24]. The empirical results show that in cooperative scenario, cooperative and competitive scenario and scenario mixed with cooperative, competitive and communication, our method significantly improves the training efficiency without affecting the performance of agents.

The rest of the paper is organized as follows. In section II related work is discussed, followed by some background knowledge in section III. We present the main approach in section IV and report experimental results in section V. Conclusion is put in section VI.

II. RELATED WORK

In the domain of MARL, there is much work on how to improve the efficiency of agent training. By combining transfer learning(TL) with reinforcement learning, knowledge reuse can be realized to accelerate agent learning process, such as inter-agent learning through the teacher-student model [31] and introducing human knowledge [32] in the training process. Some work improves the training efficiency of agents by sharing parameters or gradients [25], [33] among agents. But these algorithms are different from our methods: (1) they do not use centralized critic decentralized actor model; (2) they do not solve the problem of multi-agent credit assignment.

In recent work, the framework of centralized training and decentralized execution is adopted. In [34], the actor-critic methods are investigated for decentralized execution with centralized training. However their critic condition on local observations and single-agent actions. Markov property of MAS is difficult to guarantee.

Multi-agent DDPG(MADDPG) [20] adopts centralized critic decentralized actor model, which trains a separate centralized critic for each agent via joint observations and actions. So the Markov property of MAS is guaranteed. Each agent has its own reward function by decentralized actor, which addressed the multi-agent credit assignment. We combine parameter sharing with MADDPG and propose MADDPG-PS algorithm.

Our approach is based on MADDPG, but the differences are that, (1) we do not need to calculate the network parameters of all agents in each policy evolution, (2) we use parameter sharing among similar agents to update network parameters, (3) we use asynchronous parameter sharing and soft parameter sharing to encourage agents to explore in the early stages of training.

III. BACKGROUND

A. MULTI-AGENT REINFORCEMENT LEARNING (MARL)

Using reinforcement learning to solve multi-agent problems can avoid the difficulties brought by the pre-design of agents' behaviors, and can realize the evolution of policies through the interaction between agents and environment, so that the performance of agents and MAS can be improved gradually.

In multi-agent scenarios, the policies of all agents are evolving. In the perspective of any agent, the environment is unstable. So MARL faces not only the dimension disaster of traditional reinforcement learning and the problem of exploration-exploitation trade-off, but also the challenges of learning goals setting and learning instability. As there is no limit on the number of agents in MAS, improving the training efficiency of agents is also an important field in the research of multi-agent problems.

The Markov decision process(MDPs) of single agent can be generalized to multi-agent scenarios. The standard MDPs is defined as $< S, A_1, \ldots, A_n, R_1, \ldots R_n, T, \gamma >$, where n is the number of agents, S is the set of multi-agent environment states, A_i, $i = 1, \ldots, n$ are the sets of actions available to the agents, yielding the joint action set $A = A_1 \times \cdots \times A_n$, $T : S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, and $R_i : S \times A \times S \rightarrow \mathbb{R}, i = 1, \ldots, n$ are the reward functions of the agents. Since we are using a model-free MDPs, the transfer function is unknown.

B. MADDPG

Multi-agent DDPG(MADDPG) method is a generalization of DDPG to MARL. Similar to DDPG, agent$_i$ takes action a_i based on their own observations s_i of the environment and calculate returns based on its own reward function $R_i(s, a)$. This decentralized execution of MADDPG enables agents to set different reward functions according to tasks, thus effectively solving the problem of specifying learning goals in multi-agent scenarios.

In MADDPG, critic is trained with joint actions $\{a_1, \ldots, a_n\}$ and joint observations $\{s_1, \ldots, s_n\}$, which guarantees the markov property of MAS and the convergence of policy evolution.

MADDPG ignores the existence of homogeneous agents. In the training process, every training step needs to calculate the network parameters of each agent, which reduces the training efficiency.

IV. OUR APPROACH

A. MADDPG-PS

In the training process of MADDPG method, every policy evolution needs to calculate the derivation and gradient descent to update all each agent's network, which consumes computing resources, reduces the training efficiency of agents and increases the training time.

In this part, a new method MADDPG-PS is proposed, which can effectively improve the training efficiency while maintaining the same performance as MADDPG method. Our MADDPG-PS method combines the MADDPG method with the parameter sharing mechanism to reduce the computational complexity of the network parameters of the agents in the policy evolution, thereby reducing the training time and improving the training efficiency of the agent.

In most multi-agent environments, the number of agents is much larger than the number of tasks, and usually one task corresponds to multiple agents. In reinforcement learning, different tasks represent different training goals of agents which is expressed in the form of reward function. Agents with the same reward function are homogeneous agents. When all agents are homogenous agents, we can directly use MADDPG-PS; when there are heterogeneous agents, we can first classify the agents and then use MADDPG-PS in each class. The homogeneous agents still have different behaviors because each agent receives different observations.

For each policy evolution, only one agent's network parameters are calculated during homogeneous agents. Other homogeneous agents update the network parameters by sharing parameters to avoid calculating the parameters of each agent's network, so as to improve the training efficiency and shorten the training time.

Fig. 1 is the consumption time of MADDPG and MDDPG-PS method in a network parameter update. In Fig. 1 agent_{ni} represents class n and number i agent in the same multi-agent scenario; t_n represents the time required to complete the calculation of updating the parameters of a single agent's network in the n class.

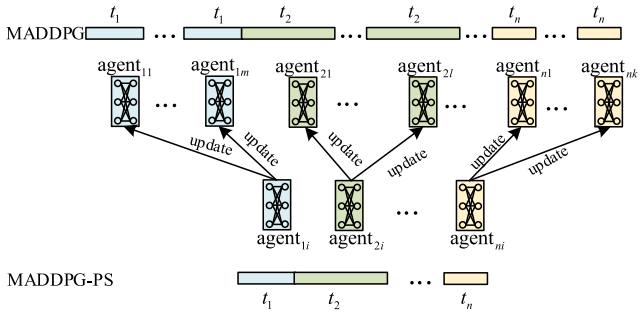


FIGURE 1. Time Consumption in a policy evolution in MADDPG and MADDPG-PS.

When using MADDPG, the time spent in a policy evolution is

$$t_{maddpg} = mt_1 + lt_2 + \cdots + kt_n \tag{1}$$

When using MADDPG-PS, the time spent in a policy evolution is

$$t_{maddpg-ps} = \sum_{i=1}^{n} t_i, \tag{2}$$

where, m, l, k represents the number of agents in different classes. We can conclude that

$$t_{maddpg-ps} < t_{maddpg}. \tag{3}$$

In the early stage of training, the agents is exploring the environment, and has not yet formed an effective policy. At this time, if the policies of the homogeneous agents are too consistent, it will limit the exploratory ability of the agents, thus affecting the training effect.

So in the initial stage of training, parameter asynchronous sharing mechanism and parameter soft update mechanism are introduced to encourage agents to explore in the initial stage. With the evolution of policies, the consistency of network parameters of homogeneous agents is gradually enhanced.

Fig. 2 is the framework of MADDPG-PS. MADDPG-PS adopts the framework of centralized training and decentralized execution. Centralized training ensures the markov property of the whole multi-agent environment and the convergence of method. Decentralized execution enables us to design reward functions for different agents in the same scenario.

The array (s, a, r, s') is stored in the experience replay buffer, where $s = \{s_{11}, \ldots, s_{nk}\}$ represents a set of local states that each agent faces; $a = \{a_{11}, \ldots, a_{nk}\}$ represents a set of actions taken by each agent according to the local state it faces; $r = \{r_{11}, \ldots, r_{nk}\}$ represents a set of rewards for each agent acting according to the local state it faces; $s' = \{s'_{11}, \ldots, s'_{nk}\}$ represents a set of the next local states that each agent faces after taking action; $a' = \{a'_{11}, \ldots, a'_{nk}\}$ is generated by each agent's target actor network according to s' in order to calculate Q_{target}.

When training, agent_{ij} is randomly selected from agents in the same class. Critic network of agent_{ij} uses observations set s_{11}, ..., s_{nk} and actions set a_{11}, ..., a_{nk} generated by all agents to fit Q value, and updates the parameters of actor network by equation (4). $\boldsymbol{\mu}_{ij}(a_{ij}|o_{ij})$ represents the deterministic strategy of using actor network fitting.

$$\nabla_{\theta_{ij}} J(\boldsymbol{\mu}_{ij}) = \mathbb{E}_{s,a\sim\mathcal{D}}[\nabla_{\theta_{ij}} \boldsymbol{\mu}_{ij}(a_{ij}|o_{ij}) \nabla_{a_{ij}} Q_i^{\boldsymbol{\mu}}(s,a)|_{a_{ij}=\boldsymbol{\mu}_{ij}(o_{ij})}] \tag{4}$$

The critic network parameters are updated by equation(5).

$$\nabla_{\theta'_{ij}} \mathcal{L}(\theta'_{ij}) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}[(Q_i^{\boldsymbol{\mu}}(s', a|\theta'_{ij}) - Q_{target})^2 \frac{\partial Q_i^{\boldsymbol{\mu}}}{\partial \theta'_{ij}}] \tag{5}$$

$$Q_{target} = r_{ij} + \gamma_{ij} Q^{\mu'}(s'_{11}, \ldots, s'_{nk}, a'_{11}, ..., a'_{nk})|_{a'_{ij}=\mu'_{ij}(o'_{ij})} \tag{6}$$

where $\mu' = \{\mu_{\theta''_{11}}, \ldots, \mu_{\theta''_{nk}}\}$, θ''_{ij} represents the parameters of the target critic network. The pseudo-code is shown in appendix.

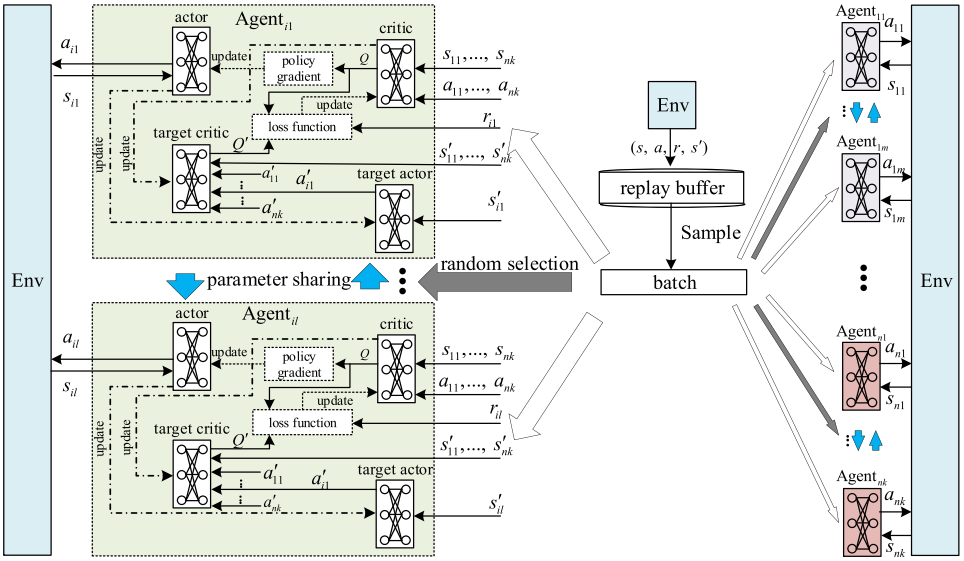


FIGURE 2. The framework of MADDPG-PS.

B. PARAMETERS ASYNCHRONOUS SHARING AND SOFT SHARING MECHANISM

In the DQN method, the target network is set up to break the correlation during training data. The target network has the same structure as the main network, but the parameters are different. The parameters of the main network are assigned to the target network after updating n steps, or the parameters of the main network are assigned to the target network with a certain weight.

In our MDDPG-PS method, we adopt a similar parameter sharing mechanism for homogeneous agents. The main purpose is to solve the problem of exploration-exploitation trade-off, which is different from the purpose of updating the parameters of the target network in the DQN method.

In the last section, we theoretically analyze the convergence of homogeneous agents to the same policy. Therefore, we encourage agents to fully explore the environment in the early stage of training, and guide agents to converge to the same strategy in the later stage of training.

For this purpose, we adopt parameters asynchronous sharing mechanism and parameters soft sharing mechanism in training.

In the training process, parameters are shared to other homogeneous agents according to the weight W in N timesteps at intervals, and exploration-exploitation trade-off is addressed by adjusting N and W.

V. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) MULTI-AGENT ENVIRONMENTS

We introduced three multi-agent environments shown in Fig. 3 including collaborative tasks, competitive tasks, and mixed tasks. These three scenarios are used in [24], [35], which are multi-agent environments exposed on github by OpenAI [36].

The first scenario is cooperative navigation, which is a collaborative task. In this scenario, including N agents and N landmarks, N agents must occupy all landmarks without avoiding collisions.

The second scenario is predator-prey, which includes competitive tasks and cooperative tasks, including N good agents, M adversarial agents, and two obstacles. The good agent moves faster and avoids the impact of adversarial agent. Adversarial agent moves slower and always tries to hit the good agent. The obstacle blocks the path of the agents.

The third scenario is cooperative-competitive chasing, which includes collaborative task, confrontation task, and communications. This scenario includes N good agents, M adversarial agents, a food, and a foggy area. Good agents move faster and hunt foods and avoid being bumped by adversarial agents. The adversarial agent closely follows the good agent. When the agent enters the foggy area, other agents cannot obtain its information. There is a leader in the adversarial agents that can get information about all agents and can communicate with other adversarial agents to help chase good agents.

2) HYPERPARAMETER

Four networks of the same structure are set up for each agent, which are actor, critic, target-actor, and target-critic. Each network has 2 fully connected layers with 64 units per layer. The learning rate l is 0.01 and the discount factor γ is 0.95. The target network parameters are updated once every 100 steps of training, and the target network soft update factor p is 0.01.

Parameter sharing between homogenous agents, using parameter asynchronous sharing mechanism and parameter soft sharing mechanism. During the initial stage of training, the parameters are shared to other homogenous agents once every 100 steps, and gradually decrease with the increase

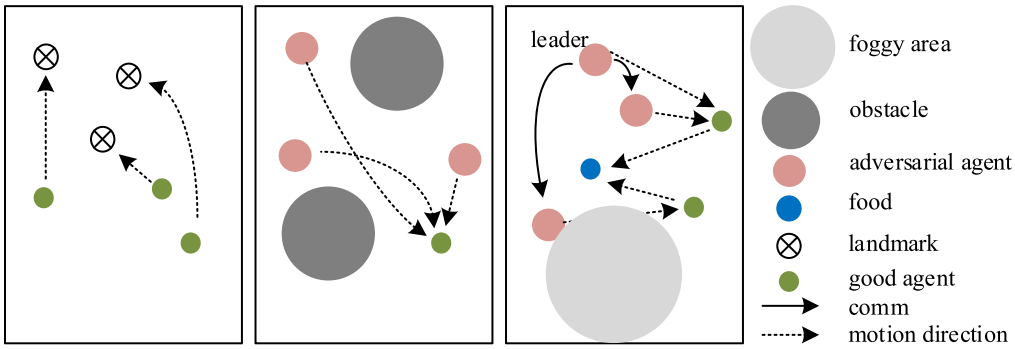


FIGURE 3. (Left)Cooperative navigation (middle) predator-prey (Right)cooperative-competitive chasing.

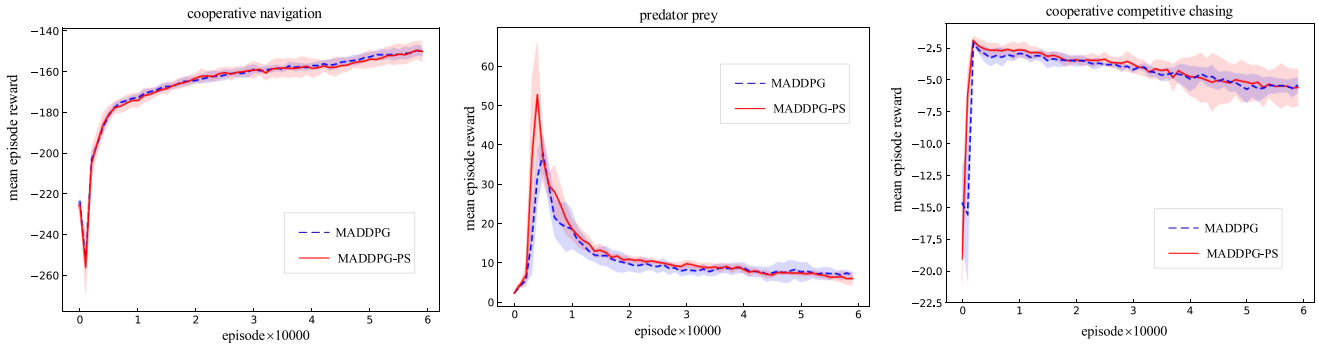


FIGURE 4. Averaged episode reward of MADDPG and MADDPG-PS in cooperative navigation(left), predator prey(middle), and cooperative competitive chasing(right).

of training episode. The soft parameters sharing factor α between homogeneous agents is 0.95.

B. RESULT AND ANALYSIS

In order to evaluate the performance and training efficiency of the MADDPG-PS algorithm, we tested it in three different scenarios: cooperative-navigation, predator-prey and cooperative-competitive chasing. The computer we used for experiments whose cache is 8G, CPU is core i7, and operating system is Ubuntu 16.04. The code is implemented based on python3.5 with Tensorflow 1.12.0 [37], Gym 0.10.9 [38] and multi-agent particle environment.

In each scenario, we trained agents using the MADDPG and MADDPG-PS methods, respectively. Every averaged reward curve is computed 6 times with a continuous error bar, illustrated in Fig. 4 and Fig. 7. To evaluate the training efficiency, we calculated the average time spent per 1×10^3 episodes based on the total time of training, as illustrated in Fig. 5 and Fig. 7.

1) WE EVALUATE THE PERFORMANCE OF MADDPG-PS

We set up these three scenarios. In cooperative-navigation, we set three agents and three landmarks, and use the MADDPG method and the MADDPG-PS method to train the agents respectively. In predator-prey, there are three adversarial agents and one good agent. The good agent always uses MADDPG, and the three adversarial agents use MADDPG

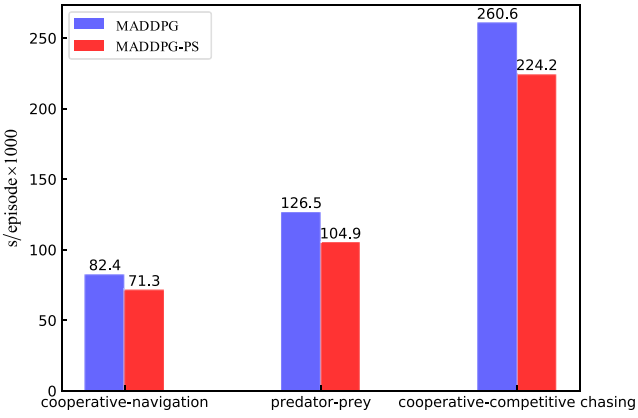


FIGURE 5. The average consumption time of MADDPG and MADDPG-PS in cooperative-navigation(left), predator-prey(middle), and cooperative-competitive chasing(right).

and MADDPG-PS separately. In cooperation competition chase, there are two good agents and four adversarial agents. The adversarial agents have always used MADDPG and the good agents use MADDPG and MADDPG-PS separately.

Fig. 4(left) illustrated that the MADDPG-PS and MADDPG methods have the same performance in cooperative-navigation. Fig. 4(mid) illustrated that the two different methods have no impact on performance of adversarial agents in predator-prey. As illustrated in the

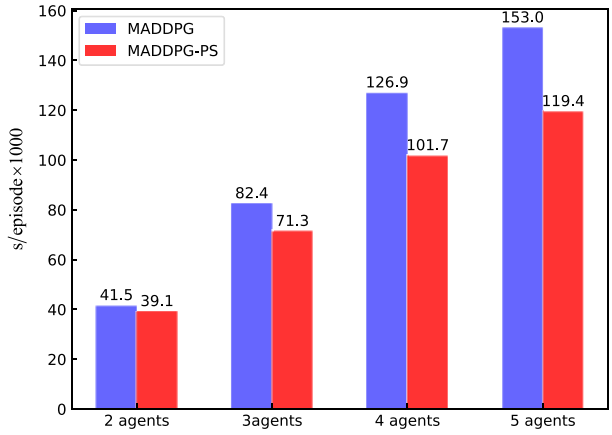


FIGURE 6. The average consumption time of MADDPG and MADDPG-PS in cooperative-navigation with different number of agents.

Fig. 4(right), there is no difference in the performance of the two methods in cooperation competition chase.

In the above three scenarios of different tasks, the performance of the MADDPG-PS method is no different from MADDPG method. The main reason is that in a multi-agent environment, homogenous agents have the same reward function which directs agents to evolve the same policy. Parameter sharing between homogenous agents does not affect the evolution of agents' policy, so the performance of MADDPG-PS after final convergence is almost the same as the MADDPG method.

2) WE EVALUATE THE TRAINING EFFICIENCY OF MADDPG-PS

In the above three scenarios, we calculated the average time consumed by training per 1000 episodes. As shown in Fig. 5, the average training time of MADDPG-PS in cooperative-navigation is reduced by 13.5%, in predator-prey by 17.1%, and in cooperative-competitive chasing by 14%. Compared with MADDPG, MADDPG-PS has higher training efficiency. The main reason is that without violating the principle of convergence of homogeneous agents to the same strategy, the MADDPG-PS method utilizes parameter sharing among homogeneous agents to replace the calculation of network parameter updating for each agent in MADDPG, which saves computing resources and improves training efficiency.

3) WE VERIFY THE EFFECTIVENESS OF MADDPG-PS WHEN THE NUMBER OF AGENTS CHANGES

In cooperative navigation, the number of agents is set to 2, 4 and 5. MADDPG and MADDPG-PS are used for training separately. Fig. 4(left) illustrated that the performance of MADDPG-PS is basically the same as that of MADDPG. However, when the number of MADDPG-PS agents is 2,3,4 and 5, the average training time per 1000 episodes is reduced by 5.8%, 13.5%, 19.9% and 22%, as shown in Figure 6. In cooperative navigation, there is only one class of agents, and all agents are homogeneous agents. MADDPG-PS algorithm mainly relies on parameter sharing among homogeneous agents to improve training efficiency, so its effectiveness is mainly related to the number of homogeneous agents. The more homogeneous agents are, the more efficient the training of MADDPG-PS is compared with that of MADDPG. Conversely, if there is only one agent in each class, the training efficiency of MADDPG-PS is exactly the same as that of MADDPG.

VI. DISCUSSION

We have shown that in multi-agent scenarios with different relationships (e.g., cooperation, competition, etc), MADDPG-PS method can reduce the training time of agents under the premise of ensuring the training effect, through parameter sharing among homogeneous agents. And its effectiveness is also mainly related to the number of homogeneous agents. In the same scenario, the same reward function will guide the homogenous agents to finally converge to the same strategy. The parameter asynchronous sharing mechanism can avoid the network parameter update calculation for each agent under the premise of ensuring the agent exploration, which is the theoretical guarantee for improving the training efficiency. The sharing of parameters among homogeneous agents does not change the evolution direction of their strategies. This is the theoretical guarantee of MADDPG-PS performance. The strategy fitting ability of the neural network is proportional to the task complexity of the agent. The more the number of hidden layers and nodes per layer, the stronger

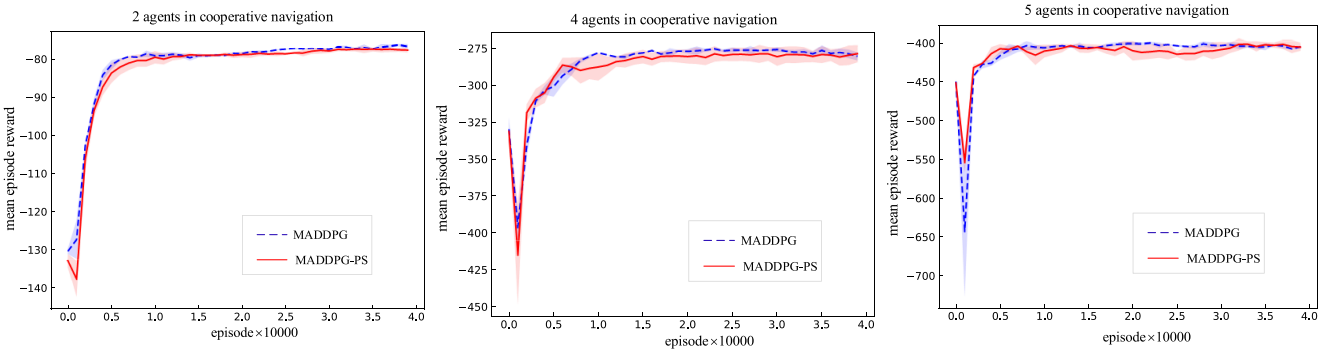


FIGURE 7. Averaged episode reward of MADDPG and MADDPG-PS in cooperative-navigation with different number of agents.

the strategy fitting ability of the neural network, but the computational load of updating the network parameters also increases. In the multi-agent scenarios with complex tasks, it is significance to study the training efficiency of agents. MADDPG-PS theoretically guarantees that the number of networks requiring parameter update calculation is less than MADDPG. So our method is more efficient than MADDPG under the same hardware condition.

VII. CONCLUSION

Based on experimental results we claim that in multi-agent environment, MADDPG-PS method can improve training efficiency by sharing parameters among homogeneous agents without affecting training performance. According to the network structure of DDPG, each agent has a target-critic network, a target-actor network, a critic network, and an actor network. In our view, there are two future work: (1) analysis of the impact of sharing only part of the network parameters (e.g., the target network) on the agent; (2) analysis of the influence of adding noise to the transmitted parameters on agent exploration is analyzed.

APPENDIX

See Algorithm 1.

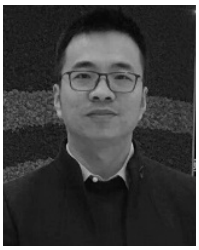
Algorithm 1 MADDPG via Parmeters Sharing

Initialize environment, agents network parmeters
for episode = 1 to max episode **do**
  forstep = 1 to max step **do**
    each agent k, $a_k = \mu x_k$
    $x', r \leftarrow (a_1 \ldots, a_n)$ atstatex
    replay buffer $D \leftarrow$ (x,a,r,x')
    for class = 1 to sum class **do**
      for each class, agent i = random(agent$_{\text{class}}$)
      sample minibatch *(x,a,r,x')* from D
      update the critic network $\nabla_{\theta_{ij}} \mathcal{L}(\theta'_{ij}) = \mathbb{E}_{s,a,r,s'-\mathcal{D}}[(Q_i^{\boldsymbol{\mu}}(s'_{11}, \ldots, s'_{nk}, a_{11}, \ldots, a_{nk}/\theta'_{ij}) - Q_{truget})^2 \frac{\partial Q^{\boldsymbol{\mu}}}{\partial \theta'_{ij}}]$
      update the actor network $\nabla_{\theta_{ij}} J(\boldsymbol{\mu}_{ij}) = \mathbb{E}_{s,a\sim\mathcal{D}}[\nabla_{\theta_{ij}} \boldsymbol{\mu}_{ij}(a_{ij}|o_{ij}) \nabla_{a_{ij}} Q_i^{\boldsymbol{\mu}}(s_{11}, \ldots, s_{nk}, a_{11}, \ldots, a_{nk})|_{a_{ij}} = \boldsymbol{\mu}_j(o_j)_{a_{ij}=\mu_j(o_j)}]$
      if every n episodes **then**
        for agent in class **do**
          net var(agent) = net_parmeters(agent /) $*a$ + net parmeters(agent)$*$(l-a)
        end for
      end if
    end for
  end for
end for

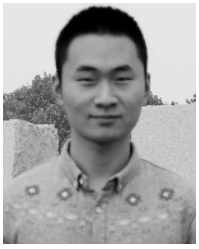
REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Dec. 2013.
[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[5] 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," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Sep. 2015.
[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2015.
[7] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2863–2871.
[8] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May/Jun. 2017, pp. 3357–3364.
[9] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, vol. 5. Belmont, MA, USA: Athena Scientific, 1996.
[10] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, nos. 2–3, pp. 235–262, 1998.
[11] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Auto. Robots*, vol. 8, no. 3, pp. 345–383, Jun. 2000.
[12] J. Xi, M. He, H. Liu, and J. Zheng, "Admissible output consensualization control for singular multi-agent systems with time delays," *J. Franklin Inst.*, vol. 353, no. 16, pp. 4074–4090, Nov. 2016.
[13] J. Xi, C. Wang, H. Liu, and L. Wang, "Completely distributed guaranteed-performance consensualization for high-order multiagent systems with switching topologies," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 7, pp. 1338–1348, Jul. 2019.
[14] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.
[15] A. L. C. Bazzan, "Beyond reinforcement learning and local view in multiagent systems," *KI-Künstliche Intelligenz*, vol. 28, no. 3, pp. 179–189, 2014.
[16] L Buşoniu, R Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications-1*. Springer, 2010, pp. 183–221.
[17] M. L. Littman, "Value-function reinforcement learning in Markov games," *Cogn. Syst. Res.*, vol. 2, no. 1, pp. 55–66, 2001.
[18] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 1–8.
[19] F. Li, J. Qin, and W. X. Zheng, "Distributed q-learning-based online optimization algorithm for unit commitment and dispatch in smart grid," *IEEE Trans. Cybern.*, to be published.
[20] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proc. ICML*, vol. 98, 1998, pp. 242–250.
[21] L. Yang, Q. Sun, D. Ma, and Q. Wei, "Nash Q-learning based equilibrium transfer for integrated energy management game with We-Energy," *Neurocomputing*, to be published.
[22] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artif. Intell.*, vol. 136, no. 2, pp. 215–250, Apr. 2002.
[23] H. Yao, T. Mai, J. Wang, Z. Ji, C. Jiang, and Y. Qian, "Resource trading in blockchain-based industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3602–3609, Jun. 2019.
[24] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[25] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. NIPS*, 2016, pp. 1–9.
[26] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7254–7264.
[27] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent q-networks," Feb. 2016, *arXiv:1602.02672*. [Online]. Available: https://arxiv.org/abs/1602.02672
[28] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst.*, 2018, pp. 2085–2087.
[29] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," Mar. 2018, *arXiv:1803.11485*. [Online]. Available: https://arxiv.org/abs/1803.11485
[30] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 22nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–9.
[31] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," Jul. 2017, *arXiv:1707.00183*. [Online]. Available: https://arxiv.org/abs/1707.00183
[32] A. Rosenfeld, M. Cohen, M. E. Taylor, and S. Kraus, "Leveraging human knowledge in tabular reinforcement learning: A study of human subjects," *Knowl. Eng. Rev.*, vol. 33, p. e14, Sep. 2018.
[33] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2137–2145.
[34] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2017, pp. 66–83.
[35] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.
[36] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. San Francisco, CA, USA: OpenAI, 2019.
[37] *An Open Source Machine Learning Framework for Everyone: Tensorflow/Tensorflow*. TensorFlow, 2019.
[38] *A Toolkit for Developing and Comparing Reinforcement Learning Algorithms: Openai/Gym*. San Francisco, CA, USA: OpenAI, 2019.



YAOZONG PAN received the bachelor's and master's degree in communication engineering from Aviation University, Yantai, China. He is currently pursuing the Ph.D. degree in information system with Space Engineering University, Beijing, China. His research interests include multi-agent planning using deep learning and reinforcement learning.



HAIYANG JIANG received the bachelor's degree in communication engineering from Liaocheng University, Liaocheng, China. He is currently pursuing the M.S. degree in information system with Space Engineering University, Beijing, China. His research interests include situational awareness and behavioral recognition.

HAITAO YANG received the bachelor's and master's degree in communication engineering and Ph.D. degree in information system from Space Engineering University, Beijing, China. He is currently an Associate Professor of communication and information system. He published his book *Design of Complex Information Network Performance*. His research interests include image processing and decision making.

JIAN ZHANG received the bachelor's degree in electrical and information engineering from Xidian University, Xi'an, China, in 2012, and the master's degree in computer science from Space Engineering University, Beijing, China, in 2015, where he is currently pursuing the Ph.D. degree in information system. His research interests include multi-agent decision making under uncertainty and deep reinforcement learning.