Received July 10, 2019, accepted July 26, 2019, date of publication August 6, 2019, date of current version August 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933454

Efficient Training Techniques for Multi-Agent Reinforcement Learning in Combat Tasks

GUANYU ZHANG[1], YUAN LI[2], XINHAI XU[2], AND HUADONG DAI[2]
[1]Computer School, National University of Defense Technology, Changsha 410073, China
[2]Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing 100071, China

Corresponding author: Huadong Dai (hddai@vip.163.com)

This work was supported in part by the National Defense Science and Technology Foundation for Young Scientists of China under Grant 030403, and in part by the National Natural Science Foundation of China under Grant 11801563, Grant 91648204, and Grant 61532007.

ABSTRACT Multi-agent combat scenarios often appear in many real-time strategy games. Efficient learning for such scenarios is an indispensable step towards general artificial intelligence. Multi-agent reinforcement learning (MARL) algorithms have attracted much interests, but few of them have been shown effective for such scenarios. Most of previous research is focused on revising the learning mechanism of MARL algorithms, for example, trying different types of neural networks. The study of training techniques for improving the performance of MARL algorithms has not been paid much attention. In this paper we propose three efficient training techniques for a multi-agent combat problem which is originated from an unmanned aerial vehicle (UAV) combat scenario. The first one is the scenario-transfer training, which utilizes the experience obtained in simpler combat tasks to assist the training for complex tasks. The next one is the self-play training, which can continuously improve the performance by iteratively training agents and their counterparts. Finally, we consider using combat rules to assist the training, which is named as the rule-coupled training. We combine the three training techniques with two popular multi-agent reinforcement learning methods, multi-agent deep q-learning and multi-agent deep deterministic policy gradient (proposed by Open AI in 2017), respectively. The results show that both the converging speed and the performance of the two methods are significantly improved through the three training techniques.

INDEX TERMS Scenario-transfer training, self-play training, rule-coupled training.

I. INTRODUCTION

Reinforcement learning has gained great successes in many single-agent competitive games such as the Go game [1] and Atari games [2]. However, traditional reinforcement learning methods, such as Q-learning [3] and Policy Gradient Learning [4], are poorly suitable for multi-agent environments. Firstly, with the number of agents increasing, the state-action space expands exponentially. Secondly, in the multi-agent environment, all agents learn at the same time. When the strategy of one agent changes, the optimal strategy of other agents may also change, which will affect the convergence of the algorithm [5].

For multi-agent scenarios, reinforcement learning has been applied in robot control [6], communications [7], traffic light control [8], etc. However, the application of multi-agent reinforcement learning (MARL) in combat tasks is more difficult since interactions among agents include both cooperative and competitive behavior. Previous research on multi-agent reinforcement learning mainly focuses on either cooperative or competitive behaviors. For cooperative behaviors, MARL methods are used for multiple agents to cooperatively finish a joint task such as scheduling, coverage control and so on [9]–[12]. For competitive behaviors, many problems like pursuer-invader problems, multi-player online games and business competition problems have been studied with MARL methods, see [13]–[15]. There are also some works studying combat tasks with mixed behaviors. But many of these works take centralized methods to control all agents, which are hard to meet the real-time requirements of practical systems.

In this paper we study a combat task which is originated from the multi-UAV combat scenario. We define an UAV as an agent. There are two teams, and the UAVs in one team (red agents) need to work cooperatively to fight with UAVs in the other team (green agents). We propose three

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

efficient training techniques: scenario-transfer training, self-play training and rule-coupled training, which are used progressively to enhance the performance of MARL methods. The scenario-transfer training is to start the training from a simple task, since it is always hard to train a good model from the scratch. A trained model is in fact a combat policy for an agent. With an initial trained model, the self-play training is a way of improving the model by iteratively training the agent and its counterpart. The idea of the rule-coupled training is to incorporate some combat rules to reduce the exploration space of the problem. The three methods work in different aspects and can in fact be used in any combination for the training of MARL methods for some other tasks.

For the considered multi-UAV combat task, we adopt two popular MARL methods, i.e., multi-agent deep Q-learning (MADQN) and multi-agent deep deterministic policy gradient (MADDPG), which are extended for DQN and DDPG (famous single-agent reinforcement learning methods) respectively. The two methods and their variants have been used in many other problems [16]–[18]. The main idea of MADQN is to train each agent with DQN by setting the input as all observations of all agents. MADDPG is proposed by OpenAI researchers in 2017 [19], which extends DDPG by setting the input for the actor network as its own observations and the input for the critic network as observations of all agents. It realizes centralized learning and decentralized execution in multi-agent environments.

With MADQN and MADDPG as baselines, we make comprehensive experiments to illustrate the effectiveness of the proposed training techniques. The scenario-transfer training reuses the experience learned in the scenario when green agents do not move to train red agents for the scenario when green agents move randomly. The convergence speed of the two methods are all accelerated by around 50%. Consequently, with the self-play training, the win rate for both methods are improved by around 10%. Further, we equip the green agents with a fighting rule and find that red agents with previous trained model do not perform well. We use the rule-coupled training, for which the action of each red agent is chosen by a simpler rule (compared to that of green agents) under some cases while by the reinforcement learning otherwise. The win rate of red agents is greatly improved on both MADQN and MADDPG. It is surprising to see that red agents could get around 70% win rate when fight with the green agents equipped with combat rules. Besides, we find that MADDPG performs much better than MADQN for this kind of combat task through experiments, which is a useful conclusion for future studies.

The main novelties of this paper are summarized below:

- We formulate the multi-UAV combat task as a multi-agent reinforcement learning problem and adopt state-of-the-art MARL method to solve it.
- We propose three efficient training techniques which have greatly enhanced the performance of both MADQN and MADDPG on the multi-UAV combat task.
- We are the first to comprehensively study the training techniques for multi-agent reinforcement learning methods, which could be used for other scenarios.

The source code of this paper and combat videos of agents have been uploaded to Github. [1]

II. RELATED WORK

Multi-UAV combat tasks have been mainly studied with hand-crafted algorithms and non-learning strategies [20]. Reinforcement learning, which develops rapidly in recent years, brings new solutions to multi-UAV combat tasks [21]. With the emergence of deep reinforcement learning (DRL), artificial intelligence has surpassed humans in some areas. In 2015, Mnih *et al.* [2] proposed deep Q-network (DQN), which has the professional performance in 49 Atari games [22]. In 2016, AlphaGo, a Go-game agent created by DeepMind, won top professional players [23]. DRL also performs well in 3D maze games [24] and MuJuCo control problems [25].

Multi-UAV combat tasks are essentially multi-agent decision problems. In multi-agent systems, each agent not only acts individually but also cooperates with each other to get better joint rewards. For multi-agent decision problems, the direct strategy is to apply independent learning mechanism (e.g. Q-learning [26]) to each agent and consider other agents as a part of the environment. Tampuu *et al.* [14] used DQN to replace the Q-learning algorithm to train each agent individually, and proposed a DRL model that can cooperate and compete with each other by adjusting the reward dynamically according to different goals. The problem is that the policy of other agents is always changing, which results in a non-stationary environment. Some other researchers try to solve this problem by inputting other agent's policy parameters to the Q function [27], explicitly adding the iteration index to the replay buffer, or using importance sampling [28]. But the performance is not improved too much.

Many researches focus on the cooperation of multiple agents, see Panait *et al.* [29] for a review about cooperative multi-agent learning. Gupta *et al.* [30] extended three classes of single-agent RL algorithms based on policy gradient, temporal-difference error, and actor-critic methods for cooperative multi-agent problems. Sukhbaatar *et al.* [31] designed a communication neural network that allows agents to learn continuous communication dynamically along with their policies for fully cooperative tasks. For mixed cooperative and competitive problems, Shao *et al.* [13] raised a curriculum transfer RL method to control multiple units in Star Craft micro-management. Peng et al. [7] introduced a multi-agent bidirectionally-coordinated network with a vectored extension of actor-critic formulation, which can learn various types of cooperative and competitive strategies for the battle scenario of the Star Craft game. However these methods use a

[1]Code and videos of this paper can be found here in https://github.com/sanjinzhi/multiagent-confrontation.git.

single centralized critic network for all agents, differing with our method, which learns a critic network for each agent.

To improve the performance of reinforcement learning, people also study some methods to assist the learning process. Chen *et al.* [32] introduced a method that can transfer the weights of one neural network to another neural network, which can accelerate the training of the new neural network. Bianchi [33] *et al.* proposed a heuristic Q-learning method, which uses a heuristic function to help the agent to select actions. It can accelerate the convergence speed but does not help to improve the converged results. Busoniu *et al.* [34] used prior knowledge to accelerate the RL algorithm called online least-squares policy iteration(LSPI). The LSPI with prior knowledge learns much faster and more reliable than the original algorithm. Cutler *et al.* [35] proposed a method that uses simulator to generate simulated data as the prior knowledge for the learning algorithm that acts directly on the real-world robot platform. These methods are mainly used in robot control, which has not been applied to multi-agent reinforcement learning for combat tasks. Silver et al. [1] proposed AlphaZero, in which self-play is adopted in the reinforcement learning, which can achieve a superhuman performance in Go game. Heinrich and Silver [36] combined self-play with DRL, and developed a scalable end-to-end approach to learn approximate Nash equilibria without prior domain knowledge. These methods mainly focus on single-agent scenarios and the application for multi-agent scenarios remains to be explored.

III. PRELIMINARY

A. PROBLEM DESCRIPTION

In this paper we construct a multi-agent combat environment based on a multi-UAV combat scenario. In this combat scenario, red team and green team fight against with each other. The team that destroys more vehicles in the other team will win the game. Each agent (UAV) can be described with 4 properties: speed (v_x, v_y), attacking zone θ_1, unprotected zone θ_2, and position (x, y). The attacking zone is in front of the vehicle, covering a sector of θ_1. The unprotected zone is a θ_2 sector behind the agent.

At any time instance t, the relation of any attacker-target pair (i, j) can be characterized by a quaternion, $[\omega_{ij}(t), d_{ij}(t), \psi_{ij}(t), \delta_{ij}(t)]^T$. Fig.1 shows the relative relation between the attacker (red) and the target (green). The coordinate of the attacker i is (x_i, y_i), and that of the target is (x_j, y_j). $\omega_{ij}(t)$ is a distance vector between the two agents. The speed of the attacker i is (v_i^x, v_i^y), and that of the target is (v_j^x, v_j^y). $d_{ij}(t)$ is the Euclidean distance between the two agents. $\psi_{ij}(t)$ is the attacking angle of attacker i relative to target j and $\delta_{ij}(t)$ is unprotected angle of target j relative to attacker i. Each element of the quaternion is computed by formulations (1).

$$\omega_{ij}(t) = (\omega_{ij}^x(t), \omega_{ij}^y(t)) = (x_j - x_i, y_j - y_i) \tag{1a}$$

$$d_{ij}(t) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{1b}$$

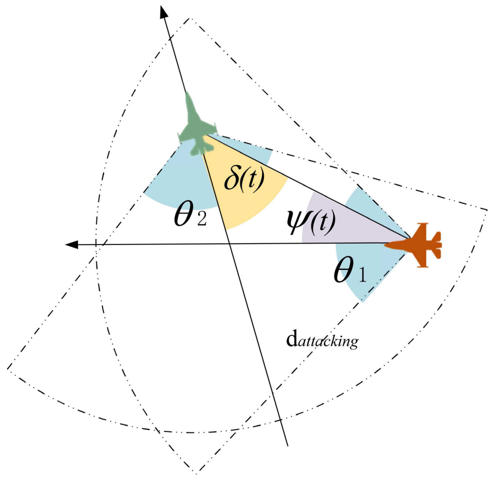


FIGURE 1. An illustration of an attacker-target pair.

$$\psi_{ij}(t) = \arg\cos \frac{v_i^x \omega_{ij}^x(t) + v_i^y \omega_{ij}^y(t)}{d_{ij}(t)\sqrt{v_i^{x2} + v_i^{y2}}} \tag{1c}$$

$$\delta_{ij}(t) = \arg\cos \frac{v_j^x \omega_{ij}^x(t) + v_j^y \omega_{ij}^y(t)}{d_{ij}(t)\sqrt{v_j^{x2} + v_j^{y2}}} \tag{1d}$$

In the considered scenario, each agent has the same size of attacking zone, unprotected zone, and attacking distance. An attacker i can destroy a target j if and only if the following three conditions are satisfied: a) the distance between the attacker and the target is smaller than the attacking range d_{attack}; b) the target is in the attacking zone of the attacker; c) the attacker is located in the unprotected area of the target. These conditions can be formulated as (2).

$$d_{ij}(t) \leq d_{attacking} \tag{2a}$$

$$\delta_{ij}(t) < \theta_2/2 \tag{2b}$$

$$\psi_{ij}(t) < \theta_1/2 \tag{2c}$$

B. SYSTEM MODEL

We consider modeling the combat scenario mentioned above as a multi-agent reinforcement learning problem. Each agent is modelled by an actor-critic framework, which includes an actor network and a critic network. The whole system is shown in Fig.2. The actor network is used to compute the action of the agent based on the observed state. The critic network is used to evaluate the action computed by the actor network, which helps to improve its performance. The experience replay buffer is used to collect experiences obtained from the environment. During the training, the input for the actor network of an agent is only the observation belonged to it. For the critic network, the input includes not only the observation of the corresponding agent but also the observations of all other agents. The critic network computes the Q value for the state-action pair of the actor network, which is used to update the parameters in the actor network. In this way, the actor-critic framework is trained with whole

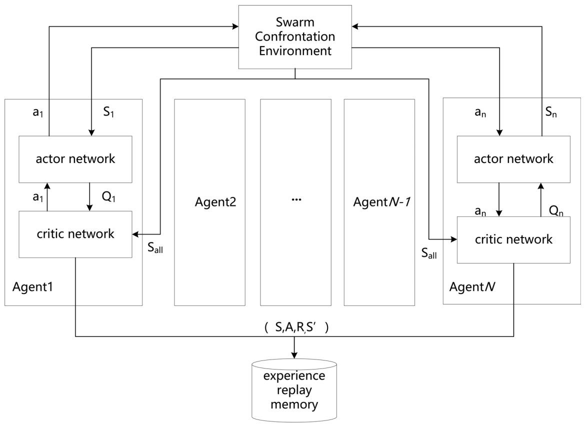


FIGURE 2. The training framework.

information (centralized training) and executed with its own observation (decentralized execution). Therefore, each actor network has a god-viewing instructor who can observe all the information to improve the strategy. With the trained model, each agent uses the actor network to interact with the environment. Even if an agent only has partial state information, it can still make a proper decision.

This multi-agent combat problem can be expressed as a Markov decision process of N agents. We define state $\mathcal{S}$ as all possible configurations of all agents, and action $\mathcal{A}$ as a set of actions $a_1, a_2, \ldots a_N$. The policy of each agent i is defined by π_i, which is used to compute the action based on the current state s_i. After executing the action a_i, agent i obtains its reward r_i. Each agent aims to maximize its total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ where γ is a discount factor and T is the maximal steps executed in an episode.

We use $\theta = \{\theta_1 \ldots \theta_n\}$ to parameterize policy $\pi = \{\pi_1, \ldots, \pi_n\}$. Then the gradient of the expected return for agent i is expressed by the equation (3), which is used to update the actor network.

$$\nabla_{\theta_i} U(\theta_i) = E_{a_i \sim \pi_i}\left[\nabla_{\theta_i} \sum_{t=0}^{\infty} \gamma^t r_i^t\right] = E_{a_i \sim \pi_i}\left[\nabla_{\theta_i} \log \pi_i(a_i|s_i) Q_i^{\pi}(x, a_1, \ldots a_N)\right] \quad (3)$$

Note that in equation (3), $Q_i^{\pi}(x, a_1, \ldots a_N)$ is the value function for agent i with states and actions of all agents as the input. x includes observations of all agents. This Q-value is computed by the critic network. Taking experiences from the experience replay buffer, the critic network is updated based on the loss function (4).

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}\left[\left(Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N) - y\right)^2\right],$$
$$where \quad y = r_i + \gamma Q_i^{\pi'}\left(\mathbf{x}', a_1', \ldots, a_N'\right)\Big|_{a'=\boldsymbol{\pi}_j'(s_j)} \quad (4)$$

IV. EFFICIENT TRAINING TECHNIQUES

A. SCENARIO-TRANSFER TRAINING

Reinforcement learning improves the intelligence of an agent through trial and error. However, a common problem is that the agent is hard to get effective rewards when learning from scratch in a complex scenario. Inadequate reward accumulation leads to slow convergence of learning and poor performance [37].

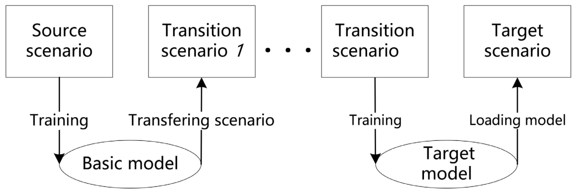


FIGURE 3. The schematic diagram of scenario-transfer training.

In fact, for the problem in this paper, it is hard to train a good model directly. We consider training a model for simple combat scenarios and then transfer the obtained experience to complex scenarios. We propose a scenario-transfer training method, for which the procedure is shown in Fig.3. The training for a scenario can be achieved by the training in several simple but similar transition scenarios, in which the first one is called the source scenario and the last one is called the target scenario. A reinforcement learning model could be firstly trained in the source scenario, then in several transition scenarios, and finally in the target scenario. The experience trained in each scenario is memorized in the model, which is used as the base model for subsequent training.

The scenario-transfer training is described in Algorithm 1. For clarification, a model is in fact a neural network model that is used to decide the next action that an agent should take. The training of a model is to update its parameters. Suppose there are N red agents in the combat task. We firstly initialize the parameters of all models randomly. Then we train the model for a set of pre-designed scenarios one by one. The parameters of the models will be updated through these training.

Algorithm 1 Scenario-Transfer Training

```
Input: A set of pre-designed scenarios.
Output: Generated models: $M_1, M_2, \ldots, M_N$
1: Initialize models $M_1, M_2, \ldots, M_N$ for all red agents
2: for each pre-designed scenario do
3:     Train the model for this scenario
4:     Update parameters of models $M_1, M_2, \ldots, M_N$
5: end for
```

B. SELF-PLAY TRAINING

With the scenario-transfer training, we could obtain basic trained models. It is hard to further improve the ability of the models since it is difficult to design good transition scenarios. Here we consider another training technique, self-play training, which has been successfully applied in as Go game, chess and shogi game [1]. The main idea of the self-play training is to copy the trained model for one agent to its opponent and then train the agent to fight with the opponent iteratively to improve the model. The scenario-transfer training and the self-play training are complementary. The former one could provide an initial model while the latter one could continuously improve the model.

Previously, the self-play training is mainly used in single-agent environments. In this paper, we propose a self-play training method for multi-agent problems, and it is described in Algorithm 2.

Algorithm 2 Self-Play Training

```
Input: Initial red models: $M_1, M_2, \dots, M_N$, self-play iteration number: $K$
Output: Generated model: $M_1^K, M_2^K, \dots, M_N^K$
1:  $M_i^1 = M_i, i = 1, \dots, N$
2:  for $k = 1; k <= K; k++$ do
3:      for $i = 1; i <= N; i++$ do
4:          Load $M_i^k$ to red agent $i$
5:          Load $M_i^k$ to green agent $i$
6:      end for
7:      while training not finished do
8:          for steps in each episode do
9:              for $i <= N$ do
10:                 Get observations $S$ for all agents
11:                 For each red agent $i$, compute action $a_i$
12:                 according to $M_i^k$
13:             end for
14:             Take the joint action $A = [a_1, a_2, \dots, a_n]$
15:             Execute action $A$ in the environment
16:             Get observation of next time $S'$ and reward $R$
17:             Store $(S,A,R,S')$ in experience buffer $G$
18:             Sample a minibatch of experiences from $G$
19:             Update parameters of $M_1^k, M_2^k, \dots, M_N^k$
20:         end for
21:     end while
22:     $M_i^{k+1} = M_i^k, i = 1, \dots, N$
23: end for
```

The input is the initial models for red agents and the output is the improved models. K is the number of self-play training times. We first load the initial model $M_1^1, M_2^1, \dots, M_N^1$ for both red and green agents and start one training. From the multi-agent combat environment, we could get the observations of red agents and green agents. For each red agent i, we compute its next action a_i based on its observation s_i with the model. We execute the joint action $A = [a_1, a_2, \dots, a_n]$ in the environment, and get the reward R and the next state S'. We store the experience (S,A,R,S') to the experience replay buffer. The models are updated with a sample of experiences from the experience replay buffer G. After the first round of training, we obtain updated models. Then we reload the new trained models and start a training round again.

C. RULE-COUPLED TRAINING

Combing rules into the training is another way of improving the models. Reinforcement learning is essentially learned by trial and error, which needs to explore in a large solution space. For agents in combat tasks, it is always very slow to learn a good combat strategy. A natural idea is to incorporate some priori knowledge into the learning process, which can reduce the ineffective exploration. The priori knowledge can be organized as a set of rules. At any time, if there is any state that satisfies the rules, the training will choose the corresponding action, rather than try other actions. In this way, the training will work efficiently to find good models.

Based on this idea, we propose a rule-coupled multi-agent reinforcement learning training technique, and its procedure is shown in Algorithm 3. The input for the rule-coupled training is a set of pre-defined rules, and the output is a set of trained models. With a set of initialized models $M_1, M_2, \dots, M_N$, the algorithm starts to interact with the environment, i.e., stating an episode. At each step of an episode, all agents get observations from the environment. For each agent, if a rule in the rule set is satisfied, its next-step action is computed by this rule. Otherwise, its action is computed by the reinforcement learning model. Then repeat the same steps as in Algorithm 2, and update parameters of all models. The algorithm will be stopped after training enough number of episodes.

Algorithm 3 Rule-Coupled Training

```
Input: A set of rules $\mathcal{R}$.
Output: Generated model: $M_1, M_2, \dots, M_N$.
1:  Initialized models $M_1, M_2, \dots, M_N$
2:  for each episode do
3:      Get observations $S$ for all agents
4:      for steps in each episode do
5:          $i = 1$
6:          for $i < N$ do
7:              if a rule $r \in \mathcal{R}$ is matched then
8:                  Get the action $a_i$ according to $r$
9:              else
10:                 Compute $a_i$ based on $M_i$
11:             end if
12:         end for
13:         step 14 - step 18 in Algorithm 1
14:         Update parameters of $M_1, M_2, \dots, M_N$
15:     end for
16: end for
```

In fact, the rules can be learned by the model implicitly through the rule-coupled training, since the actions delivered by rules are also used for the training.

V. EXPERIMENTS

We construct a combat scenario described in Section 3 by extending the multi-agent particle environment developed by Open AI [19]. There are three red agents and three green agents in this combat scenario. We combine the three training techniques with MADQN and MADDPG respectively, and make comprehensive experiments to illustrate the performance of the three techniques.

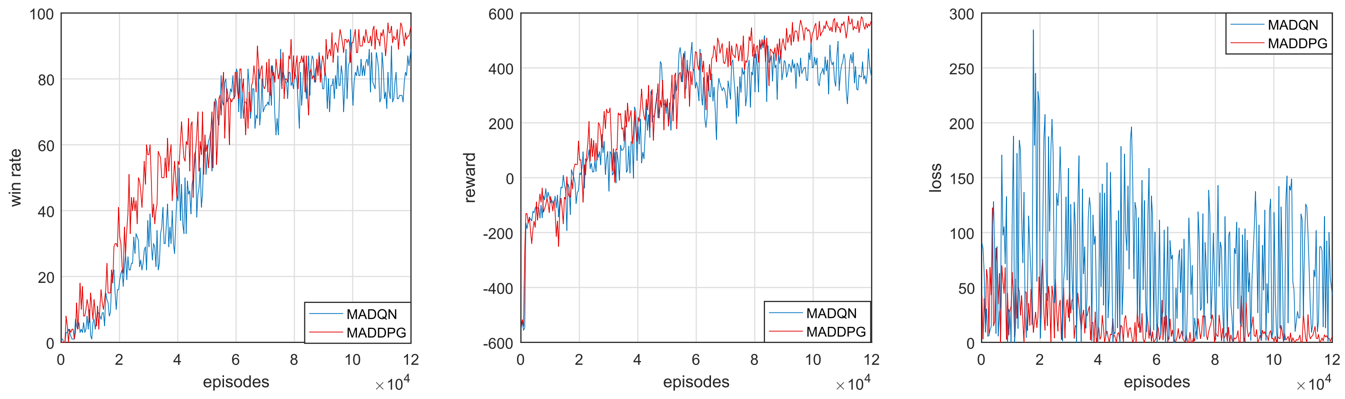


FIGURE 4. Comparison of MADQN and MADDPG for a simple combat task.

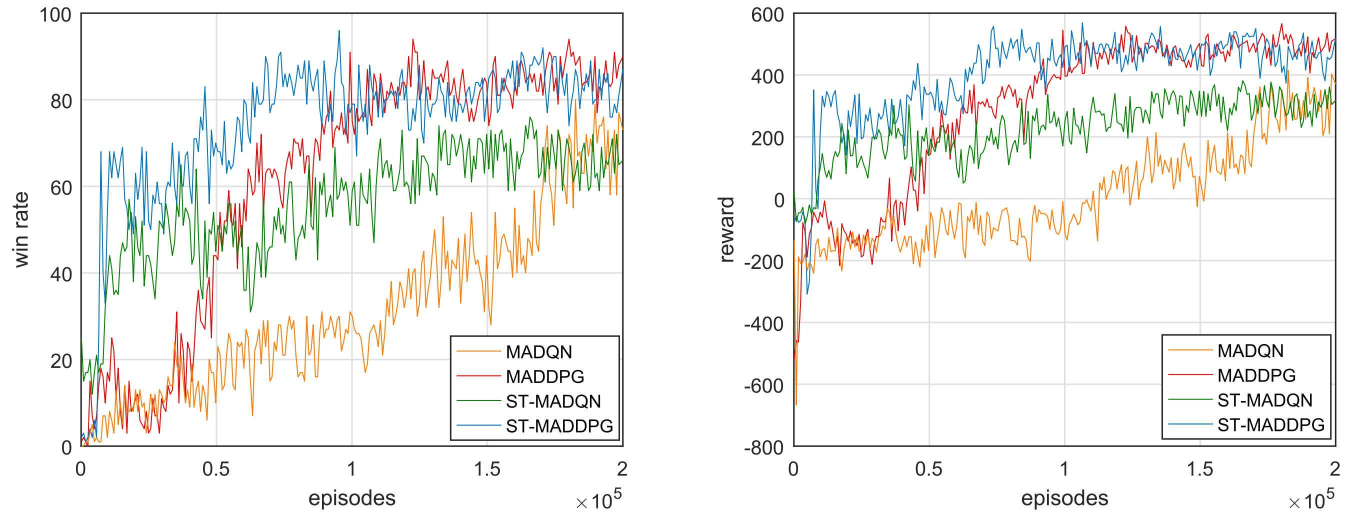


FIGURE 5. Comparison of scenario-transfer training with MADDPG and MADQN for the random combat task.

A. ENVIRONMENT SETUP

We firstly construct a simple combat task, in which green agents cannot move and attack. We train red agents with MADQN and MADDPG respectively. For MADQN, the neural network for each red agent includes 2 hidden layers. Each hidden layer contains 50 hidden units with ReLU activations. For MADDPG, the actor network contains 2 hidden layers with 50 hidden units each and ReLU activations. The critic network has the same structure. For the training of the neural network, we take the Adam optimizer. We set the learning rate to 0.01, the gamma value to 0.95 and the batch-size to 1024. During the training, the maximum steps for each episode were 150. The agent getting away from the border will receive a negative reward -60, and the agent that eliminates an enemy will receive a positive reward 40. Those parameters will be used in the following experiments if they are not explicitly specified.

The result of this experiment is shown in Fig.4. An episode is a combat round, which is terminated when one team wins the game, or the simulation step in a round exceeds the maximum step. The three figures in Fig.4 show the win rate, the reward and the loss of the two methods for the training of red agents,respectively. As we can see, both MADQN and MADDPG are convergent in this simple task. For the first subfigure, we carry out 100 combat tests every 100 training episodes, and compute the average win rate. The second subfigure shows the averaged reward over all red agents, and each value is the sum of rewards over all steps in an episode of the training. We can see that MADDPG outperforms MADQN both in the win rate (95% vs 85%) and the reward (580 vs 450). In the third subfigure, we can see that the loss value of MADQN fluctuates more violently than that of MADDPG, which indicates that the converged policy of MADDPG is more stable than that of MADQN in this combat task.

B. PERFORMANCE OF SCENARIO-TRANSFER TRAINING

Now we consider the combat scenario when green agents move randomly and can attack red agents, which is called randomly combat task. We consider using the scenario-transfer training, for which we treat the simple task in last section as the source scenario, and the randomly combat task as the target scenario. We load the trained model obtained in the source scenario for red agents and then start a new training with MADQN/MADDPG, which indicates that the experience

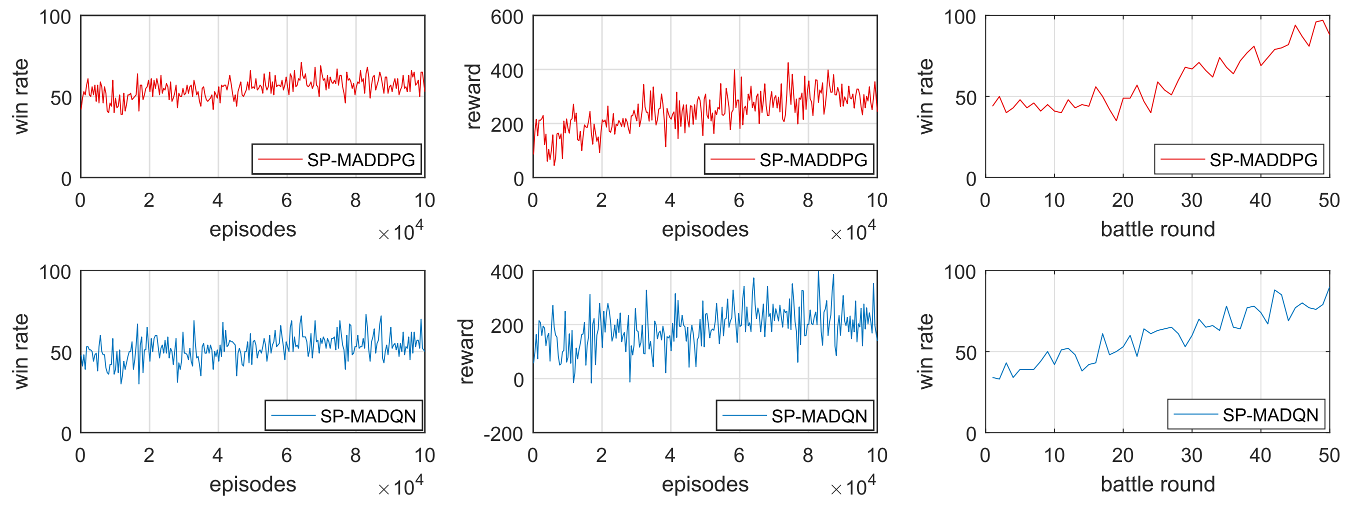


FIGURE 6. Performance of self-play training.

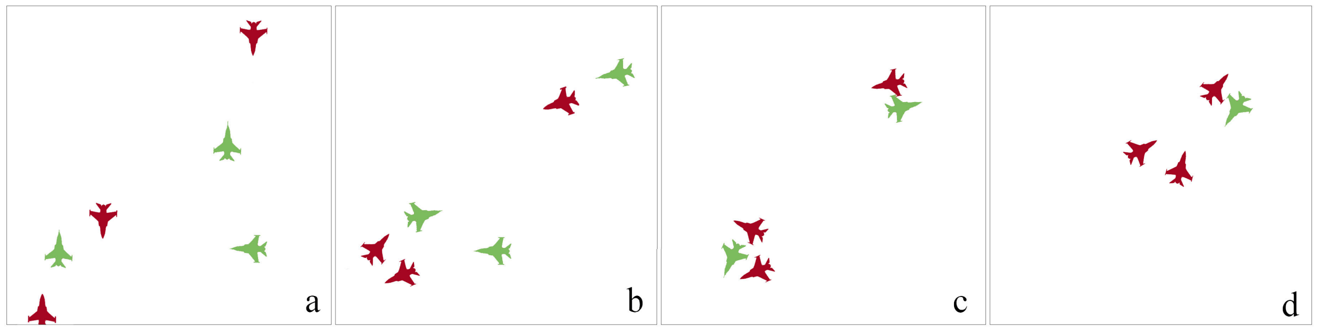


FIGURE 7. The cooperative behaviors learned by red agents to combat with green agents.

learned in the source scenario will be reused for learning new knowledge in the target scenario. We use ST-MADQN and ST-MADDPG to represent the two methods which combine the scenario-transfer training with MADQN and MADDPG respectively. The red agents are trained for 200, 000 episodes for the two methods. The comparison results of MADQN,ST-MADQN,MADDPG,and ST-MADDPG are shown in Fig.5. As we can see, all methods are convergent in this randomly combat task, in terms of the win rate and the reward. It is remarkable to see that for both MADQN and MADDPG, the scenario-transfer training can significantly accelerate the convergence speed. Take the training results of the win rate as the example, MADQN needs 180, 000 episodes to converge while ST-MADQN needs only 130, 000 episodes. MADDPG needs around 120, 000 episodes to converge while ST-MADDPG needs only 60, 000 episodes. The scenario-transfer training saves 50, 000 episodes for MADQN to converge, and speeds up the convergence speed of MADDPG for this task by around 50%. Besides, compared with MADQN, MADDPG has faster convergence speed and higher win rate, which again shows the superiority of MADDPG.

C. PERFORMANCE OF SELF-PLAY TRAINING

With trained models obtained by the scenario-transfer training as initial models, we consider continuously improving the performance of these models by the self-play training. Firstly, following the procedure of Algorithm 2, we make one round of self-play training with 100, 000 episodes for MADQN and MADDPG. We name them as SP-MADQN and SP-MADDPG respectively. The variation of the reward and the win rate during the training is shown in the left four sub-figures in Fig.6. As we can see, both the reward and the win rate are all improved through the self-play training. It is notable to see that the win rate for red agents is increased from around 50% to around 55% for SP-MADQN and from around 50% to around 60% for SP-MADDPG. This illustrates that the self-play training is a good way to improve the performance of the model for the combat task. Next, we carry out another experiment to demonstrate the superiority of the self-play training. We conduct 50 rounds of self-play ($K = 50$ in Algorithm 2). Each round of self-play includes 50, 000 episodes. For every 100 episodes, we make a test that the red agents loaded with the latest trained model fight with green agents equipped initial models for 100 times. The win rate of red agents can be seen in the right two sub-figures in Fig.6. With the increasing rounds of self-play, the performance of red agents becomes stronger gradually. It is surprising to see that red agents trained by SP-MADQN could achieve around 95% win rate when they fight with green agents loaded with models trained by ST-MADQN. Red agents trained

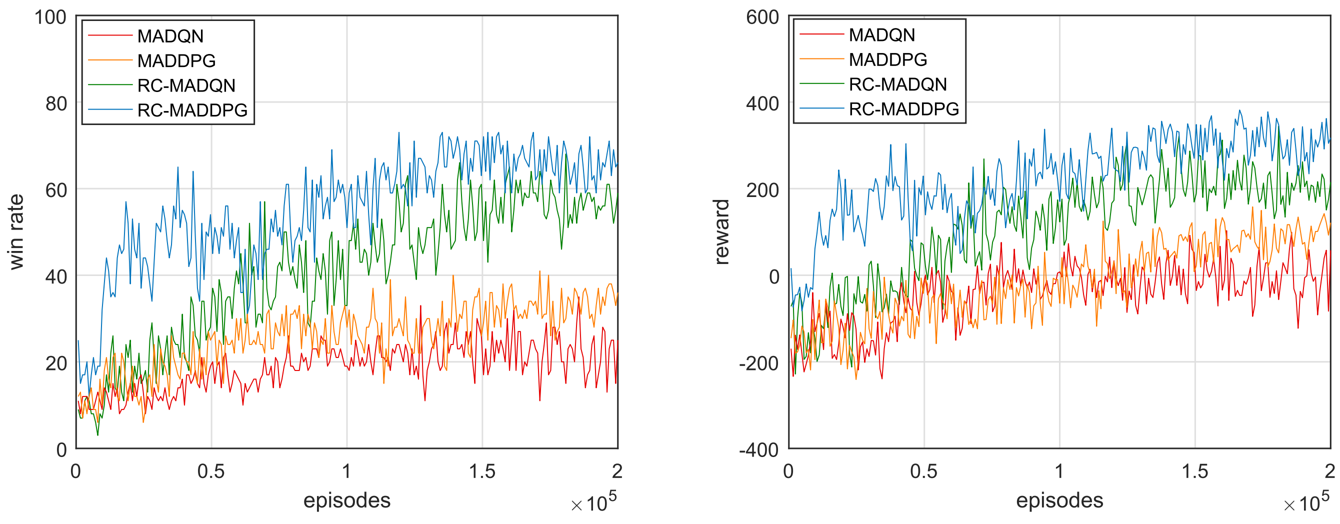


FIGURE 8. Performance of the rule-coupled training.

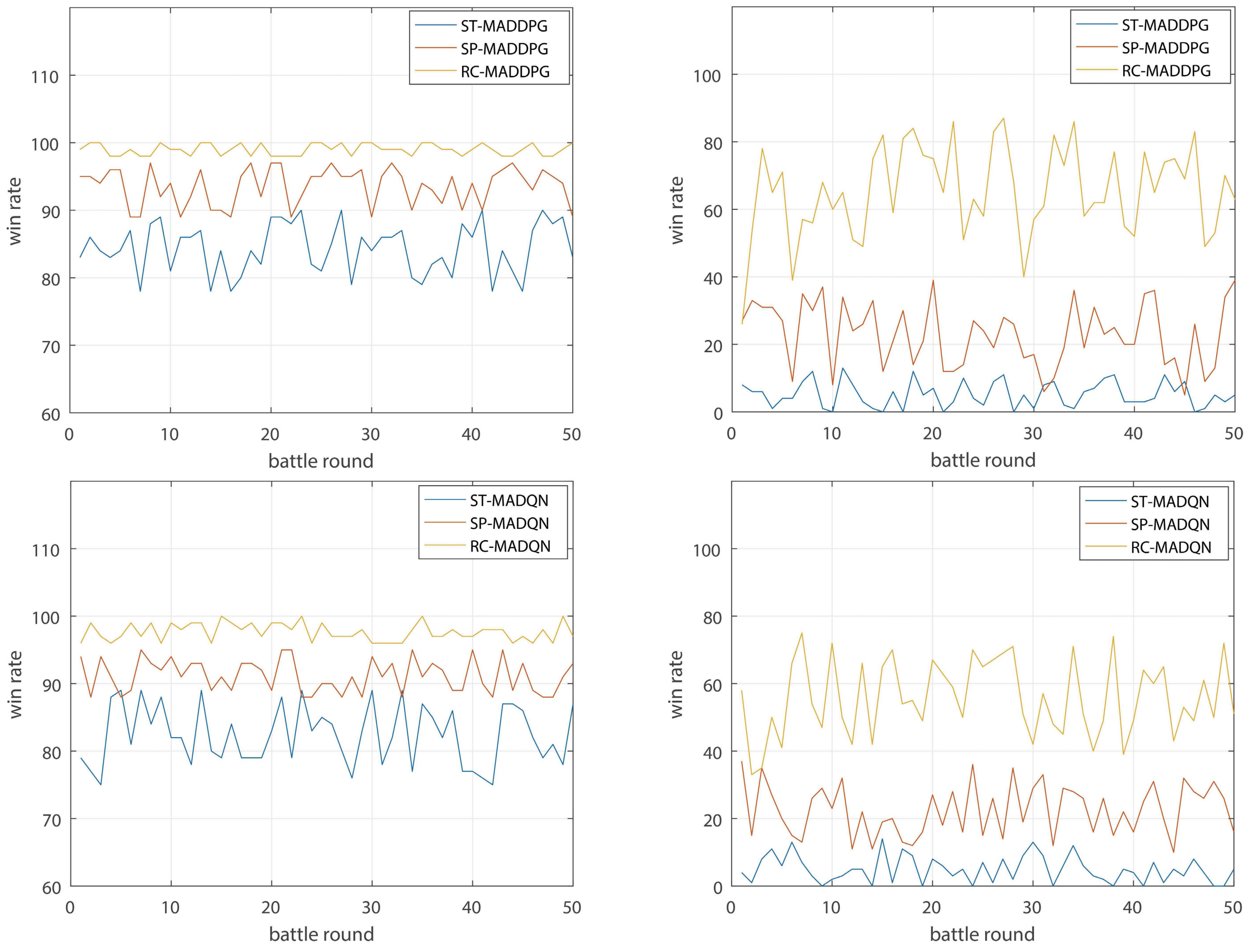


FIGURE 9. The comparisons for the three training techniques for the random task and the fix-rule task.

by SP-MADDPG could achieve around 96% win rate when they fight with green agents loaded with models trained by ST-MADDPG. Both methods enhance the performance of red agents by around 40%, which is a great improvement.

In addition, we find that though the self-play training, red agents learn some interesting behaviors as shown in Fig.7. Initially, three red agents are distributed at different locations in Fig.7(a). Then two red agents learn to cooperatively fight with green ones in Fig.7(b) and wipe out a green agent in Fig.7(c). At last, the three red agents work cooperatively to wipe out the last agent.

D. PERFORMANCE OF RULE-COUPLED TRAINING

Combing rules with the training is another way of improving the performance of agents by reducing the exploration space of the reinforcement learning. The performance of red agents has been greatly improved through self-play training. However, when we equip the green agents with a rule that a green agent always moves towards the nearest red agent (named as the green rule), the win rate of agents with the self-play training becomes very small. As we can see in Fig.8, red agents with models trained by the self-play training can only have a win rate of 10%. This value is improved to around 20% by the training of MADQN and to 38% by the training of MADDPG.

Therefore we adopt the rule-coupled training, i.e., training red agents with a simpler rule. When the distance between the red agent and the nearest green agent is smaller than $2d_{attacking}$, the red agent moves towards the green agent. Otherwise, the action of the red agent is computed by reinforcement learning. We apply the rule-coupled training to MADQN and MADDPG, which are noted as RC-MADQN and RC-MADDPG respectively. The results are shown in Fig.8. As we can see, both methods are converged. For RC-MADQN, the converged win rate is around 57%, which is improved by 37% compared to MADQN. For RC-MADDPG, the win rate is improved to 62%. The converged rewards are also improved through the rule-coupled training. Besides, the results also show that MADDPG outperforms MADQN in such combat scenarios.

Finally we make a comprehensive comparison for models delivered by the three training techniques of MADDPG and MADQN, which is shown in Fig.9. We compare these models for two tasks: green agents move randomly (random task) and green agents more following the green rule (fix-rule task). As we can see by consecutively using the three training techniques, the win rate of red agents with MADDPG is improved from around 85% to around 98% for the random task, while the win rate of MADQN is improved from 83% to around 94%. For the fix-rule task, the win rate of MADDPG is improved from 5% to 75%, while the win rate of MADQN is improved from 4% to around 68%.

VI. CONCLUSION

In this paper we study a multi-agent combat problem with multi-agent reinforcement learning methods. Much previous research is focused on devising different types of learning methods to deal with these tasks. We propose three efficient train techniques: scenario-transfer training, self-play training and rule-coupled training, which greatly improve the performance of two popular MARL methods (MADQN and MADDPG). The proposed methods achieve great performance for the considered combat problem. In fact, the three training techniques are general methods that can also be used for other MARL methods and also for other kinds of tasks. It would be interesting to explore the effectiveness of the three training techniques when they can be used solely or in any other combination for other problems.

REFERENCES

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and Y. Chen, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529, Feb. 2015.

[3] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[4] J. Peters and J. A. Bagnell, "Policy gradient methods," in *Encyclopedia of Machine Learning and Data Mining*. Boston, MA, USA: Springer, Jan. 2010, pp. 774–776.

[5] L. Busoniu, R. Babuska, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innov. Multi-Agent Syst. Appl.*, vol. 310, pp. 183–221, Jul. 2010.

[6] R. C. Arkin, "Cooperation without communication: Multiagent schema-based robot navigation," *J. Field Robot.*, vol. 9, no. 3, pp. 351–364, Jun. 2010.

[7] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," 2017, *arXiv:1703.10069*. [Online]. Available: https://arxiv.org/abs/1703.10069

[8] M. A. Khamis and W. Gomaa, "Enhanced multiagent multi-objective reinforcement learning for urban traffic light contro," in *Proc. 11th Int. Conf. Mach. Learn. Appl.*, vol. 1, Dec. 2012, pp. 586–591.

[9] J. Han, C.-H. Wang, and G.-X. Yi, "Cooperative control of UAV based on multi-agent system," in *Proc. IEEE 8th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2013, pp. 96–101.

[10] P. Frasca, R. Carli, and F. Bullo, "Multiagent coverage algorithms with gossip communication: Control systems on the space of partitions," in *Proc. Amer. Control Conf.*, Jun. 2009, pp. 2228–2235.

[11] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Hysteretic Q-learning : An algorithm for decentralized reinforcement learning in cooperative multi-agent teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Dec. 2007, pp. 64–69.

[12] E. Semsar-Kazerooni and K. Khorasani, "Multi-agent team cooperation: A game theory approach," *Automatica*, vol. 45, no. 10, pp. 2205–2213, 2009.

[13] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 1, pp. 73–84, Feb. 2018.

[14] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *Plos One*, vol. 12, Dec. 2015, Art. no. e0172395.

[15] H. Kebriaei, A. Tajeddini, and N. Rashedi, "Markov game approach for multi-agent competitive bidding strategies in electricity market," *IET Gener., Transmiss. Distrib.*, vol. 10, no. 15, pp. 3756–3763, Nov. 2016.

[16] K. Mateusz and J. Wojciech, "Heterogeneous team deep q-learning in low-dimensional multi-agent environments," in *Proc. IEEE Conf. Comput. Intell. Games*, Sep. 2016, pp. 1–8.

[17] L. Kaixiang, Z. Renyu, X. Zhe, and Z. Jiayu, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," 2017, *arXiv:1802.06444*. [Online]. Available: https://arxiv.org/abs/1802.06444

[18] M. Aleksandra, T. Tegg, S. Chae-Bong, K. Daniel, and S. Aleksei, "Deep multi-agent reinforcement learning with relevance graphs," 2018, *arXiv:1811.12557*. [Online]. Available: https://arxiv.org/abs/1811.12557

[19] 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 Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 6379–6390.

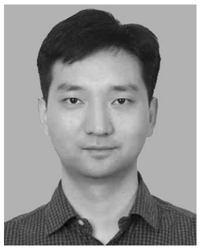
[20] Z. Yun, P. Yao, Y. Sun, and J. Yang, "Cooperative task allocation method of MCAV/UCAV formation," *Math. Problems Eng.*, vol. 11, p. 9, Jan. 2016.

[21] M. Xiaoteng, X. Li, and Z. Qianchuan, "Air-combat strategy using deep Q-learning," in *Proc. Chin. Automat. Congr. (CAC)*, Nov. 2018, pp. 3952–3957.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/abs/1312.5602
[23] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 1, Jun. 2014, pp. 387–395.
[24] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, and J. Schrittwieser, *Deepmind Lab*. Dec. 2016.
[25] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Int. Conf. Mach. Learn.*, May 2016, pp. 1329–1338.
[26] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, Jun. 1993, pp. 330–337.
[27] G. Tesauro, "Extending q-learning to general adaptive multi-agent systems," in *Proc. Adv. Neural Inf. Process. Syst.*, Mar. 2004, pp. 871–878.
[28] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1146–1155.
[29] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auton. Agents Multi-Agent Syst.*, vol. 11, no. 3, pp. 387–434, Nov. 2005.
[30] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Auto. Agents Multiagent Syst.*, Nov. 2017, pp. 66–83.
[31] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Inf. Process. Syst.*, May 2016, pp. 2244–2252.
[32] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," 2015, *arXiv:1511.05641*. [Online]. Available: https://arxiv.org/abs/1511.05641
[33] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristically accelerated Q–learning: A new approach to speed up reinforcement learning," in *Proc. Brazilian Symp. Artif. Intell.*, vol. 3171, Sep. 2004, pp. 245–254.
[34] L. Busoniu, B. De Schutter, R. Babuska, and D. Ernst, "Using prior knowledge to accelerate online least-squares policy iteration," in *Proc. IEEE Int. Conf. Automat., Qual. Testing, Robotics*, vol. 1, May 2010, pp. 1–6.
[35] M. Cutler and J. How, "Efficient reinforcement learning for robots using informative simulated priors," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2015, pp. 2605–2612.
[36] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," 2016, *arXiv:1603.01121*. [Online]. Available: https://arxiv.org/abs/1603.01121
[37] S. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, pp. 1345–1359, Nov. 2010.

GUANYU ZHANG received the B.S. degree in software engineering from the National University of Defense Technology, in 2017, where he is currently pursuing the master's degree. His research interests include multi-agent reinforcement learning algorithms and basic software.

YUAN LI received the B.Sc. degree in computer science from the National University of Defense Technology, China, and the Ph.D. degree in network optimization from Lund University, Sweden, in 2008 and 2015, respectively. He has published papers on top network journals such as the IEEE JSAC. He has published papers on conference such as the IEEE INFOCOM. Since 2018, his research interest has been focused on multi-agent reinforcement learning algorithms with applications in real-time strategy games. His research interests include network modeling, algorithm design, integer programming and other combinatorial methods with applications in communication networks. He has won the first place from the Multi-Agent Confrontation Competition 2019 that was held by China Electronics Technology Group Corporation.



XINHAI XU received the Ph.D. degree in computer science from the National University of Defense Technology, in 2012. He is currently an Associate Professor with the Artificial Intelligence Research Center, National Innovation Institute of Defense Technology, Beijing, China. His research interests include artificial intelligence algorithms, simulation, and parallel computing.

HUADONG DAI received the Ph.D. degree from the National University of Defense Technology, China, in 2002. He is currently a Professor with the National Innovation Institute of Defense Technology, Beijing, China. His main research interests include operating systems, computer architectures, and artificial intelligence.