Cooperative Multi-Agent Deep Reinforcement Learning with Counterfactual Reward

Kun Shao[1,2], Yuanheng Zhu[1,2], Zhentao Tang[1,2], Dongbin Zhao[1,2]
[1]State Key Laboratory of Management and Control for Complex Systems,
Institute of Automation, Chinese Academy of Sciences. Beijing, China.
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China.
shaokun2014@ia.ac.cn, yuanheng.zhu@ia.ac.cn; tangzhentao2016@ia.ac.cn; dongbin.zhao@ia.ac.cn;

***Abstract*—In partially observable fully cooperative games, agents generally tend to maximize global rewards with joint actions, so it is difficult for each agent to deduce their own contribution. To address this credit assignment problem, we propose a multi-agent reinforcement learning algorithm with counterfactual reward mechanism, which is termed as CoRe algorithm. CoRe computes the global reward difference in condition that the agent does not take its actual action but takes other actions, while other agents fix their actual actions. This approach can determine each agent's contribution for the global reward. We evaluate CoRe in a simplified Pig Chase game with a decentralised Deep Q Network (DQN) framework. The proposed method helps agents learn end-to-end collaborative behaviors. Compared with other DQN variants with global reward, CoRe significantly improves learning efficiency and achieves better results. In addition, CoRe shows excellent performances in various size game environments.**

***Index Terms*—reinforcement learning, deep reinforcement learning, cooperative games, counterfactual reward**

I. Introduction

In the last few years, we have witnessed massive progresses of artificial intelligence (AI) in games with deep reinforcement learning (DRL) [1]–[3]. These DRL methods achieve impressive performances in various games, including Atari [4], Go [5] [6], Vizdoom [7]–[9] and StarCraft [10]–[13]. DRL has proven to be a general and effective method for game AI.

Many complicated decision-making processes in games can be modeled as multi-agent learning problems [14]–[17]. As a long-standing field of machine learning research, multi-agent learning systems have some own characteristics. First of all, as a result of involving multiple learning agents, the action space grows exponentially with the number of agents, and the search space is usually very large. Secondly, multi-agent system involves multiple learners, and every learner optimizes the policy under the influence of other agents. How to model other agents and update own policy properly is also a key problem [15].

Multi-agent games have several types, including cooperation, competition, and the mix of cooperation and competition. Here we focus on cooperative multi-agent games. Agents in cooperative games consider to play with multiple allies, so as to maximize the global reward. Generally, there are mainly two kinds of training schemes in collaborative multi-agent learning system. The first is the team learning, also known as centralised learning, that uses one learner to output the joint action of all agents in the team. This method suffers from the scalablity problem when agents' number increases. The second is the concurrent learning, also known as decentralized learning. This method uses multiple concurrent learners to update each agent's policy. Different from team learning, concurrent learning has one controller for each agent. Through mapping the joint space to multiple separated subspaces, concurrent learning can reduce the dimension of search space [14]. Decentralized learning helps to deal with the complexity in multi-agent system, and the limitation of the observability and communication make the decentralized policy more practical.

The simplest way to solve multi-agent task is supposing that each agent learns independently, and takes the action according to its local observation [18]. Independent learning is a widely in multi-agent games, and has achieved impressive performances [12] [19]. Independent learning is simple and direct, but suffers from the instability problem caused by multiple agents. It has troubles to learn multi-agent collaborative behaviors and evaluate each agent's contribution for the global reward. Parameter-sharing among multiple agents can accelerate the learning process in the multi-agent system. This method only learns one policy network, and applies it to all agents [20]. Due to different local observations, each agent can make different actions. Parameter-sharing method is suitable for homogeneous multi-agent problem.

However, these methods does not directly solve the multi-agent credit assignment problem. In collaborative tasks, joint action only produces a global reward. It is difficult to determine the contribution of each agent. Sometimes we suppose that each agent can obtain individual reward. However, these local rewards are difficult to design, and are unable to guarantee agents to maximize the global reward in collaborative scenarios [21]. To train multiple agents in fully cooperative games, the simplest method is giving each agent the same global reward or allocating it equally. This method has obvious disadvantage. It can't distinguish the contribution of different agents for the global reward. On the other hand, using the same reward will aggravate the homogeneity of agents, making each

This work was supported in part by the National Key Research and Development Program of China under Grants 2018AAA0101005 and 2018AAA0102404.

agent to learn similar behavior.

In this paper, we present counterfactual reward (CoRe) mechanism to solve the multi-agent credit assignment problem in fully cooperative game. Counterfactual reward is obtained by calculating the global reward difference when the agent's action is replaced by other available actions, while fixing other agents' actions. We assume that the agent has the same probability to choose other actions. Experimental results in a simplified Pig Chase game prove that this is an effective method to solve multi-agent credit assignment problem. In addition, we adopt the parameter-sharing method to accelerate the training process, and help to make collaborative behaviors.

The rest of the paper is organized as follows. In Section II, we introduce the related work of multi-agent credit assignment. Next section describes the background of Dec-POMDP and reinforcement learning. Then we present multi-agent deep reinforcement learning with counterfactual reward. In Section V, we present the experimental details and results. Finally, we draw a conclusion of our research.

II. Related Work

To determine each agent's importance in cooperative multi-agent games, the Shapley value is a widely used method [22]. It allocates a unique distribution among all agents of the total surplus. The characteristics of the Shapley value are a set of desirable attributes. While in cooperative games, we can't complete the task and receive the global reward with a subset of agents. Hence, the Shapley value can't handle the multi-agent credit assignment in our setting.

Counterfactual action is also used to tackle exploratory noise in multi-agent systems. In cooperative games, the global reward includes plenty of noise due to exploratory actions. Agents are unable to distinguish the global reward's ingredient from true environmental dynamics and other agents' exploratory action. Coordinated Learning without Exploratory Action Noise (CLEAN) rewards are designed to remove exploratory action noise [23] [24] [25]. At each training step, agents execute actions by following the greedy policy without exploration to produce a global reward. After that, each agent privately computes the global reward in condition that it executes an exploratory action, and the rest of the agents follow their greedy policies. CLEAN rewards are defined as follows:

$$r^i_{clean} = r(s, \mathbf{a} - a^i + a^i_{clean}) - r(s, \mathbf{a}). \tag{1}$$

where $\mathbf{a}$ is the joint action when all agents follow the greedy policies, a^i is the greedy action executed by agent i, a^i_{clean} is the counterfactual action following ϵ-greedy, $r(s,\mathbf{a})$ is the global reward when all agents execute their greedy policies and $r(s, \mathbf{a} - a^i + a^i_{clean})$ is a counterfactual reward of agent i.

The value decomposition network (VDN) [21] aims to learn an optimal linear value decomposition from the global reward, by back-propagating the total Q gradient through deep neural networks that approximate the individual value functions. This method is specifically motivated by avoiding the spurious rewards that emerge in purely independent learners. VDN can help to improve the coordination problem of independent learning with centralised training and decentralised execution [26]. Counterfactual multi-agent (COMA) policy gradient uses a centralised critic to train decentralised actors, and estimates a counterfactual advantage function for each agent in order to address the multi-agent credit assignment problem [27]. The centralised critic $Q(s,\mathbf{a})$ estimates Q-values for the joint action $\mathbf{a}$ and the central state s. For each agent, COMA computes an advantage function with a counterfactual baseline that marginalises out a^i, while keeping the other agents actions $\mathbf{a}^{-i}$ fixed. QMIX lies between independent Q learning and COMA, but has the ability to represent a much richer class of action-value functions [28]. Compared to VDN which is not necessary to extract decentralised policies, QMIX only needs to ensure that a global argmax performed on Q_{global} has the same result as a set of individual argmax operations performed on each Q_i. QMIX uses a mixing network to combine each Q_i into Q_{global}, not as a simple sum as in VDN. QMIX can represent complex centralised action-value functions with a factored representation that scales well in the number of agents and allows decentralised policies to be easily extracted via linear-time individual argmax operations.

III. Dec-POMDP and Reinforcement Learning

In this section, we will introduce the background of fully cooperative multi-agent game, and reinforcement learning.

A. Fully Cooperative Multi-Agent Game

A fully cooperative multi-agent game can be described as a decentralized partially observable Markov decision process (Dec-POMDP) [29]. $s \in S$ denotes the global state. At every time step, each agent $i \in I \equiv \{1, \cdots, n\}$ chooses an action $a^i \in A$, resulted in joint action $\mathbf{a} \in \mathbf{A} \equiv A^n$. The environment receives the joint action and transfers to the next state s' according to probability $P(s'|s,\mathbf{a})$. Agents receive the global reward $r(s,\mathbf{a}) : S \times \mathbf{A} \to \mathbb{R}$. The global action-state value function $Q^\pi(s_t, \mathbf{a}_t)$ under policy π is

$$Q^\pi(s_t, \mathbf{a}_t) = \mathbb{E}_{s_{t+1:T}, a_{t+1:T}}[R_t | s_t, \mathbf{a}_t]. \tag{2}$$

Because of Dec-POMDP, each agent can only receive its local observability $o \in O$, and $o(s,i) : S \times I \to O$. We define the individual action-state value function of agent i as $Q^\pi(o_t, a_t)$:

$$Q^{\pi^i}(o^i_t, a^i_t) := \mathbb{E}[R^i_t | o^i_t, a^i_t], \tag{3}$$

where $R^i_t = \sum_{l=0}^{T} \gamma^l r^i_{t+l}$ is the cumulative discount reward of agent i.

B. Reinforcement Learning

Reinforcement learning considers the paradigm that an agent learns by trial and error, and determines the ideal behavior from its own experiences with the environment [30]. As a classical RL algorithm, Q-learning is a widely-used off-policy method. The update process of Q learning is

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t), \tag{4a}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t, \tag{4b}$$

where δ_t is the temporal-difference (TD) error.

Deep Q learning uses a neural network to approximate action value function, and stores experiences into a replay buffer [4]. At each training step, DQN samples a minibatch of experiences to minimize the TD error, so as to update the network.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s') \in D}[(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{\theta}^-) - Q(s, a; \boldsymbol{\theta}))^2], \tag{5}$$

where $\hat{Q}$ is the target Q function. DQN uses the target network to stabilize the learning process, and $\boldsymbol{\theta}^-$ is the parameter of the target network. Experience replay improves the sample efficiency, and target network stabilizes the training process. After that, several variants of DQN have been proposed. Dueling architecture neural network is suitable for model-free deep reinforcement learning [31]. It has two different estimators, the state value function estimator $V(s)$, and the state-action advantage function estimator $A(s, a)$. For those actions with similar values, dueling DQN performs better on policy evaluation. DQN agent randomly samples experiences from the replay buffer to train model, while it does not take the importance of different experiences into account. Priority experience replay (PER) measures the importance according to the TD error δ of different experiences, and preferentially samples more important experiences [32]. The sampling probability for experience j is $P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$, where $p_j = |\delta_j| + \epsilon$ is the priority. α determines how much prioritization is used, and ϵ is a small positive constant. DQN uses the expectation of action value function to learn the optimal policy, while categorical DQN depends on the distribution of action value function [33]. Compared to the expectation of action value function, the distribution can provide more useful information for decision-making. For some cases, agents tend to choose actions with smaller variance, rather than blindly choosing those with higher mean values. For some MDPs or POMDPs, similar states may have completely different value functions. If we only consider the mean values, this part information will be completely confused. Quantile regression (QR) DQN performs distributional RL over the Wasserstein metric, and achieves better performance in some benchmark [34].

IV. Multi-Agent RL with Counterfactual Reward

Independent and parameter-sharing deep reinforcement learning methods have made some achievements in multi-agent tasks, while they are unable to deal with the credit assignment problem among agents in team games. We follow the idea behind CLEAN rewards, and put forward multi-agent reinforcement learning with counterfactual reward to solve this problem.

A. Counterfactual Reward

The cumulative discounted reward of multi-agent system is $R_t(s_t, \mathbf{a}_t) = \sum_{l=0}^{T} \gamma^l r_{t+l}(s_{t+l}, \mathbf{a}_{t+l})$, where $\mathbf{a}_t$ is the joint action of all agents, and s_t is the global state at time step t. $r_t(s_t, \mathbf{a}_t)$ is the one-step global reward depended on all agents.

Difference reward uses a default action to define each agent's counterfactual [35]. We can calculates each agent's local reward to get the counterfactual when the agent's action is changed. This is an effective method to deal with the credit assignment problem among multiple agents. Hence, we propose the counterfactual reward $c_t^i(s_t, \mathbf{a}_t)$ of agent i.

$$c_t^i(s_t, \mathbf{a}_t) = r_t(s_t, \mathbf{a}_t) - \mathbb{E}_{a_t^{i,c}} r_t(s_t, \mathbf{a}_t - a_t^i + a_t^{i,c}) \tag{6}$$

In collaborative multi-agent reinforcement learning tasks, when fixing other agents' actions, the current agent takes other actions so as to result in the global reward difference. We assume that the agent has the same probability to choose other actions, and we need to know the model of the environment to evaluate reward $r_t(s_t, \mathbf{a}_t - a_t^i + a_t^{i,c})$.. $\mathbf{a}_t - a_t^i + a_t^{i,c}$ denotes that at time step t, agent i does not take its action, but takes other actions $a_t^{i,c}$ equally. $a_t^{i,c} \in A$ and $a_t^{i,c} \neq a^i$.

The partial differential of counterfactual reward with respect to a^i is:

$$\begin{aligned}\frac{\partial}{\partial a^i} c_t^i(s_t, \mathbf{a}_t) &= \frac{\partial}{\partial a^i}(r_t(s_t, \mathbf{a}_t) - \mathbb{E}_{a_t^{i,c}} r_t(s_t, \mathbf{a}_t - a_t^i + a_t^{i,c})) \\ &= \frac{\partial}{\partial a^i} r_t(s_t, \mathbf{a}_t) - \frac{\partial}{\partial a^i} \mathbb{E}_{a_t^{i,c}} r_t(s_t, \mathbf{a}_t - a_t^i + a_t^{i,c}) \\ &= \frac{\partial}{\partial a^i} r_t(s_t, \mathbf{a}_t) - 0 \\ &= \frac{\partial}{\partial a^i} r_t(s_t, \mathbf{a}_t)\end{aligned} \tag{7}$$

With counterfactual reward has high consistency with the global learning tasks, because maximizing agent's local reward will also increase the global reward. Without the consistency, agents in multi-agent learning systems will tend to be lazy. High consistency is very important for multi-agent collaboration. In addition, counterfactual reward is sensitive to the action of each agent. Reward sensitivity means that compared with other agents' actions, each agent is more sensitive to its own action. By contrast, independent DQN and parameter-sharing DQN that distribute global reward equally to each agent have lower sensitivity to action. Counterfactual reward can improve the performance of current multi-agent deep reinforcement learning algorithms. This reward mechanism is more sensitive to each agent, and is simpler for the learning task.

B. Multi-Agent DQN with Counterfactual Reward

Multi-agent deep Q learning with counterfactual reward bases on the traditional deep Q learning algorithm, but use different reward mechanisms. This can converge faster in multi-agent games. The update rule of partially observable Q learning with counterfactual reward is

$$Q(o_t^i, a_t^i) \leftarrow Q(o_t^i, a_t^i) + \alpha(c_t^i + \gamma \max_{a^i} Q(o_{t+1}^i, a^i) - Q(o_t^i, a_t^i)). \tag{8}$$

Algorithm 1 Multi-Agent DQN with Counterfactual Reward

```
1:  Initialize experience replay buffer $D$ to capacity $N$, step counter $T = 0$
2:  Initialize action-value function $Q$ with random weight $\boldsymbol{\theta}$
3:  Initialize target action-value function $\hat{Q}$ with weight $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
4:  Get initial state $s$
5:  repeat
6:      for each agent $i$ do
7:          receive observation $o^i$, get $Q(o^i, a; \boldsymbol{\theta})$
8:          choose $a^i$ according to $\epsilon$-greedy policy
9:      end for
10:     execute $\mathbf{a}$, get global state $s'$ and global reward $r$
11:     for each agent $i$ do
12:         receive new observation $o^{i'}$
13:         calculate $c^i$ according to equation 1
14:         store transition $(o^i, a^i, c^i, o^{i'})$ in $D$
15:         randomly sample minibatch transitions from $D$
16:         $y = \begin{cases} c^i + \gamma max_{a^{i'}} \hat{Q}(o^{i'}, a^{i'}; \boldsymbol{\theta}^-), & \text{else} \\ c^i, & \text{if } s' \text{ is terminal} \end{cases}$
17:         update parameters $\boldsymbol{\theta}$ according to equation 9
18:         $o^i = o^{i'}$
19:     end for
20:     $T \leftarrow T + 1$
21:     $s = s'$
22:     if $T$ mod 1000 == 0 then
23:         update target network weight $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
24:     end if
25: until $T > T_{max}$
```

When we use deep neural network as function approximator, and use experience replay D and target action-value network $\hat{Q}$ introduced in DQN, the loss function of agent i is

$$\mathcal{L}^i(\boldsymbol{\theta}_t) = \mathbb{E}_{(o^i,a^i,c^i,o^{i'})\in D}[(c_t^i + \gamma \max_{a^{i'}} \hat{Q}(o^{i'}, a^{i'}; \boldsymbol{\theta}_t^-) - Q(o^i, a^i; \boldsymbol{\theta}_t))^2]. \tag{9}$$

We define $y = c^i + \gamma max_{a^{i'}} \hat{Q}(o^{i'}, a^{i'}; \boldsymbol{\theta}^-)$ if s' is not terminal. Otherwise, $y = c^i$. Algorithm 1 presents the details of Multi-Agent Deep Q Network with Counterfactual Reward (CoRe-DQN).

V. Experiments

We use a multi-agent game environment to evaluate the performance of counterfactual reward for multi-agent deep reinforcement learning algorithms.

A. Experimental Platform

Pig Chase game used in the experiment is shown in Figure 1. Agent 1, agent 2 and the pig are presented with red, blue and green respectively, with the form of T in the maze environment. Agents use the simplified first-person perspective visual images as input, resulting in high-dimensional state space. The fields behind agents are grey. The roadblocks are black and walkable areas are white. Agent's available actions are move forward, turn left, turn right and catch. At the beginning, two agents and the pig are located randomly with no overlap. When two agent catch the pig successfully, the environment will be reset, and three objects are reset randomly.

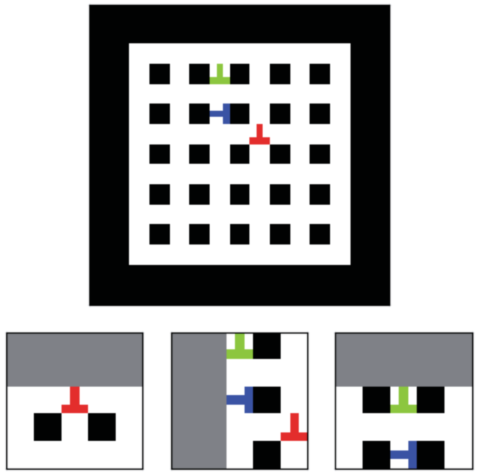


Fig. 1. Representation of Pig Chase. The top view is the global information. The bottom views are local observations of agent 1, agent 2 and the pig respectively.

When the agent move, it can only move to the front space. This action does not work if there are obstacles or agents in this space. The pig in the maze is controlled with built-in game AI. We assume that the pig is strong, that means the pig will not be arrested until two agents use the catch action at the same time. The catch action can only be used when agents and the pig are in the adjacent position, and the agent should face the pig. Each agent can only observe two neighbouring grid spaces. Direction 0, 1, 2, 3 represent the left side, the up side, the down side, and the right side respectively.

The observation sample is shown in Figure 1. The global view is on the top, but we don't use it for partially observable Markov decision process. The bottom views are the observations of agent 1, agent 2 and the pig, from left to right. This sample shows that agent 1 is toward down, agent 2 is toward right, and the pig is toward down.

Pig Chase is a fully cooperative multi-agent game. Agents will receive a global reward of +1 if the pig is caught successfully. At other time step, agents will receive a punishment of 0.001, as shown in equation (10). The maximal steps in a game is 1000, so the game score is in [-1, 1]. We train agents in maze scenarios with different sizes, such as 11×11, 13×13 and 15×15.

$$r_{global} = \begin{cases} +1, & \text{catch the prey successfully} \\ -0.001, & \text{every time step} \end{cases} \tag{10}$$

B. Learning Model

CoRe agents learn a action-value function $Q_\theta(o_t^i, a^i)$ for each agent. Agents receive state observation o_t^i, and use deep neural networks as the Q function approximator. The resolution of the original RGB visual image is 15×15×3

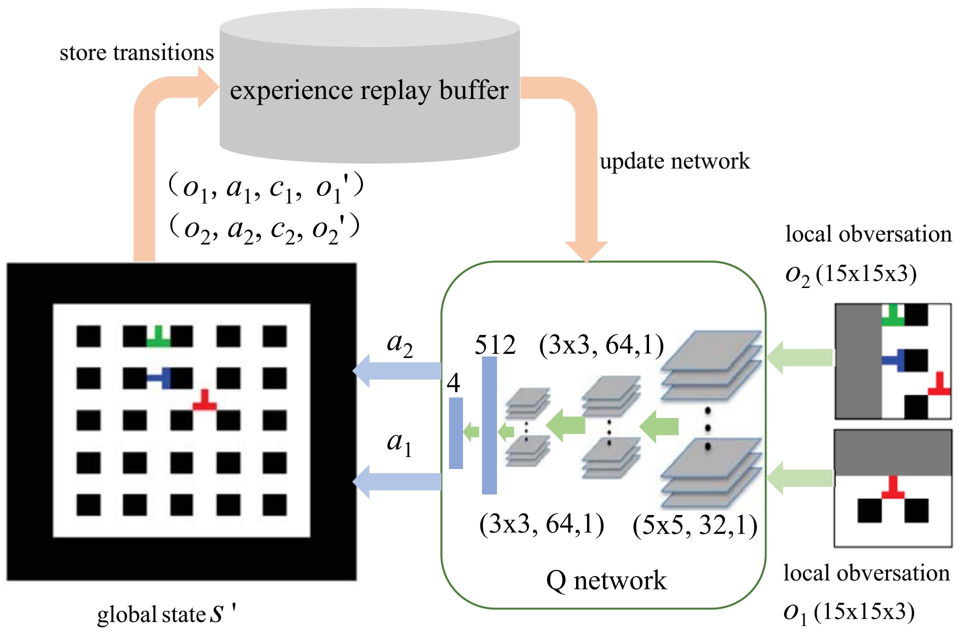


Fig. 2. The CoRe diagram in Pig Chase. Two agents receive local partially observable visual observations as input, and use deep convolutional neural network model to output Q values for each available action. The environment receives the joint action, and transits to the next state. Agents receive the global reward, and calculate individual counterfactual reward. These experiences are stored in a replay buffer, and used to update the Q network which is shared among agents.

(width: 15, height: 15, RGB channels: 3). The convolutional neural network in our model refers to the network architecture of DQN. The network uses a convolutional layer with 32 filters of size 5×5, followed by a convolutional layer with 64 filters of size 3×3, followed by a convolutional layer with 64 filters of size 3×3, followed by a fully connected layer with 512 hidden units. We set stride to 1 in all convolutional layers. All four hidden layers are followed by a rectifier nonlinearity. At last, we use a fully connected layer with 4 nodes to output the Q values of all the available actions. We use PyTorch to construct the neural network.

The details of the CoRe model for Catch the pig game are depicted in Figure 2. Two agents receive local first-person perspective visual observations as input, and use deep convolutional neural network model to output Q values for each available action. To balance exploration and exploitation in reinforcement learning, we use ϵ-greedy method to select actions. ϵ starts at 1, and reduces to 0.001 in the end, as $\epsilon = 0.001 + (1 - 0.001) \times e^{\frac{-step}{6000}}$. The environment receives each agent's action, output global reward and move to the next state. We calculate counterfactual reward for each agent according to the global reward, and storage experience into the replay buffer. To update Q network, we sample a minibatch of experience from the replay buffer in each update.

In the training process, the learning rate is set to 0.0001. Target network is updated every 1000 steps. The replay buffer size is 100000, and batch size is 64. In the first 10000 steps, agents take action randomly. After that, we update the Q network every step.

C. Comparison on Global Reward and Counterfactual Reward

First of all, we train agents in the 11×11 small game environment, and compare the performance of deep reinforcement learning with global reward and counterfactual reward. We use four kinds of deep Q learning algorithms to train agents: DQN, Dueling-DQN, PER-DQN, and QR-DQN.

We average the episode rewards of the last 100 episodes evey 10000 training steps. Figure 3 show the average rewards of various DRL agents with global reward independent learning (Ind), global reward parameter-sharing learning (PS) and counterfactual reward (CoRe). These rewards and steps are evaluated after every 10000 training steps within nearest 100 games. The curves of average reward increase fast in the first 200000 steps, and gradually converge after that. The average steps that agents needed to catch the pig gradually reduced during training, and eventually drop to a low level.

With different reward mechanism, average rewards and average steps of different deep reinforcement learning algorithms are presented in Table I. The result shows that our deep reinforcement learning agents can eventually learn effective strategies and catch the pig. Global reward independent DR-L provides a high-level baseline. Global reward parameter-sharing learning is superior to independent learning in learning speed. CoRe has better performances compared with global reward independent learning and parameter-sharing learning, especially on learning speeds and final scores.

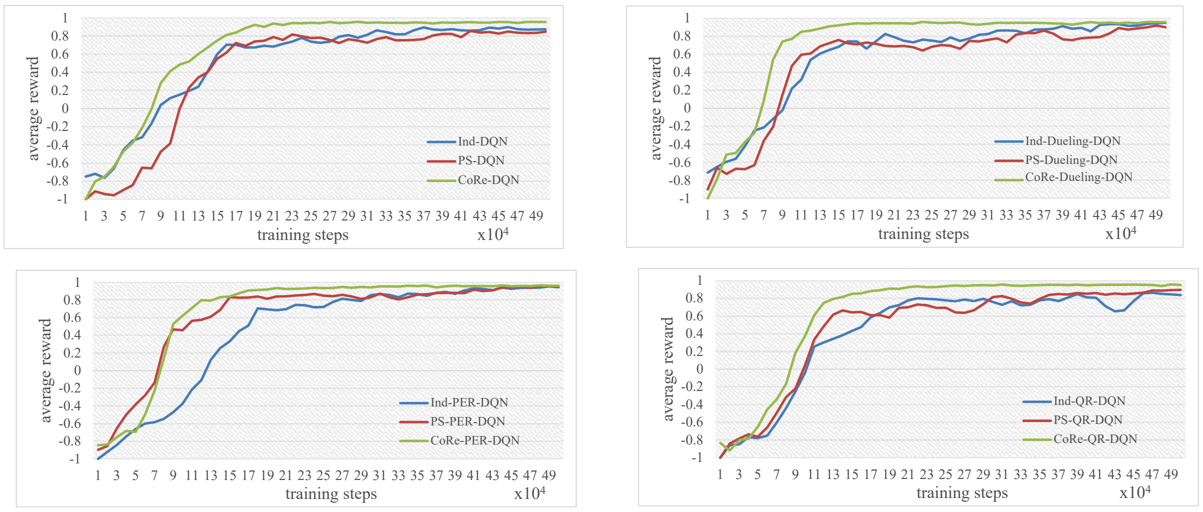


Fig. 3. Average rewards of DRL agents with various reward mechanisms in 11x11 game environment during training.

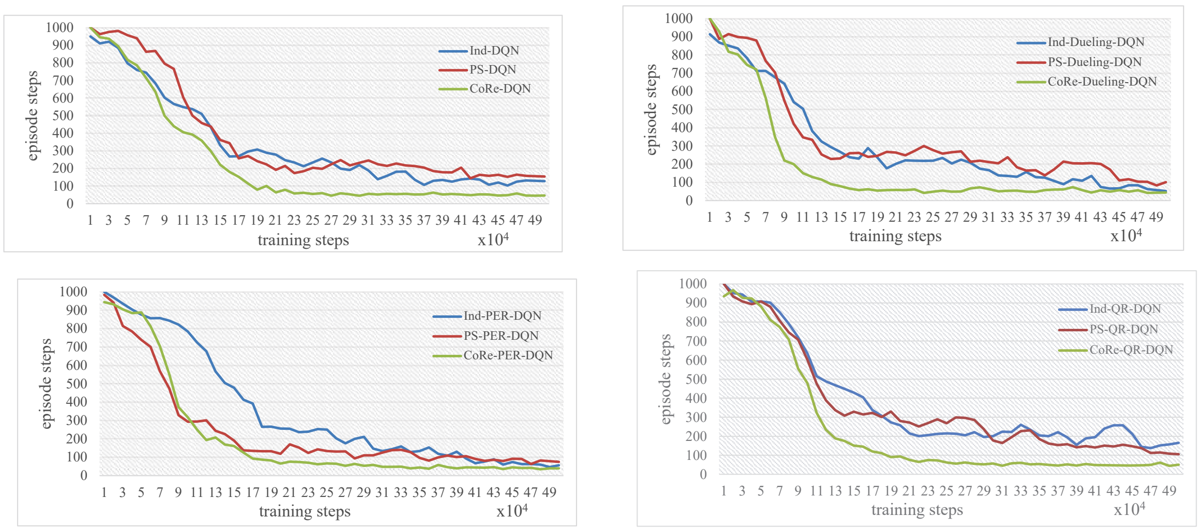


Fig. 4. Average steps of DRL agents with various reward mechanisms in 11x11 maze environment during training.

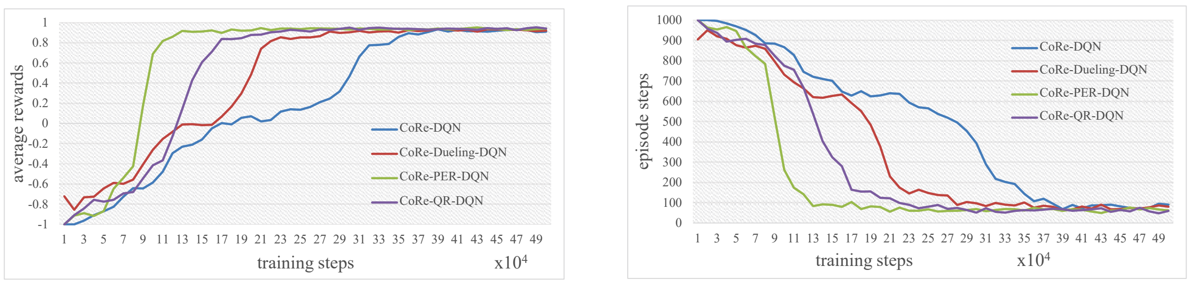
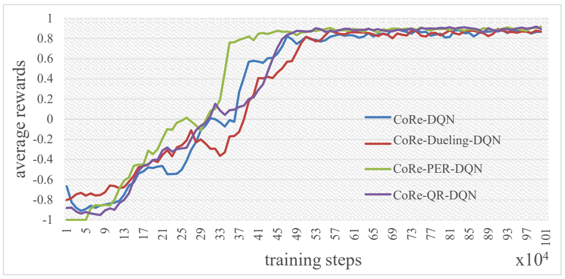


Fig. 5. Experimental results of CoRe agents in 13x13 game environment, including average rewards, and episode steps.



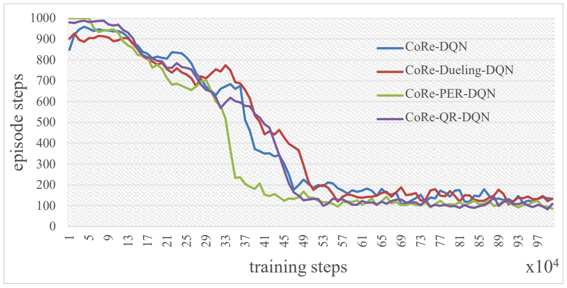


Fig. 6. Experimental results of CoRe agents in 15x15 game environment, including average rewards, and episode steps.

TABLE I
PERFORMANCE OF DRL AGENTS WITH VARIOUS REWARD MECHANISMS IN 11X11 GAME ENVIRONMENTS.

Algorithms	Metrics	DQN	Dueling-DQN	PER-DQN	QR-DQN
Global reward Ind	rewards	0.875	0.921	0.935	0.785
	steps	126.5	79.4	65.7	191.8
Global reward PS	rewards	0.835	0.855	0.937	0.868
	steps	162.2	140.3	82.1	131.7
CoRe	rewards	**0.951**	**0.951**	**0.960**	**0.952**
	steps	**49.8**	**50.1**	**41.3**	**48.9**

TABLE II
PERFORMANCE COMPARISON OF VARIOUS CORE AGENTS IN 13X13 AND 15X15 GAME SCENARIOS.

Maze size	Metrics	DQN	Dueling-DQN	PER-DQN	QR-DQN
13x13	rewards	0.919	0.924	0.937	**0.939**
	steps	81	76	63	**61**
15x15	rewards	0.870	0.869	**0.897**	0.897
	steps	130	131	**103**	104

D. *Comparison on Different Environments*

The experimental results in 11×11 game environment show that CoRe agents can achieve remarkable performance for multi-agent reinforcement learning. In the following experiments, we will explore whether CoRe can still learn effective policies in 13×13 and 15×15 maze environments.

Figure 5 shows the average rewards and average win rate of different CoRe agents in 13×13 game environment during training. In this scenario, our agents are trained for 500000 steps. CoRe agents can learn effective strategies, and catch the pig eventually. The curves of average rewards are similar to those in 11×11 game scenario, with a smooth rising trend. The curves of average win rate are similar to that of average rewards, and agents finally reach 100% win rate. The average steps are about 100 at last.

Figure 6 shows the average rewards and average win rate in 15×15 game environment during training. In this scenario, our agents are trained for 1000000 steps. Although this scenario is more difficult, the proposed CoRe agents can still catch the pig successfully. Compared with the former two scenarios, agents' performances have more fluctuations in the training process.

Table II presents average rewards, average win rate and average steps of different CoRe-DQNs in 13×13 and 15×15 game environments. These performance metrics are averaged in 1000 games. The results show that with maze size being larger, average rewards are reduced, and agents need more steps to catch the pig. In 13×13 game scenario, CoRe-QR-DQN has the best performance. While in 15×15 game scenario, CoRe-PER-DQN is the best.

VI. CONCLUSION

In this paper, we focus on fully cooperative games, and propose multi-agent deep reinforcement learning with counterfactual reward to solve the credit assignment problem. Counterfactual reward fixes other agents' actions, and calculates the global reward difference when the current agent executes other actions to reshape each agent's local reward. This method can help to measure each agent's contribution for the global reward, resulting in collaborative behavior among multiple agents. In Pig Chase game, CoRe agents have better performances and learning efficiency compared with the global reward independent learning and parameter-sharing multi-agent DRL algorithm.

At present, we need to know the model of the environment to calculate the counterfactual reward of each agent. In the future, we will combine the model-based method, and learn the model of the environment with data-driven method. In addition, we will extend our method to policy gradient reinforcement learning, with centralized critic and decentralized actor, and generalize to environment with large number of agents.

Acknowledgment

The authors would like to thank Qichao Zhang, Dong Li, Zhentao Tang, Weifan Li, Jun Wang, and Yuzhen Zhuang for the helpful comments and discussions about this work.

References

[1] D. Zhao, K. Shao, Y. Zhu, D. Li, Y. Chen, H. Wang, D. Liu, T. Zhou, and C. Wang. Review of deep reinforcement learning and discussions on the development of computer Go, Control Theory and Applications, 33(6), 701–717, 2016.

[2] Z. Tang, K. Shao, D. Zhao, and Y. Zhu. Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero, Control Theory and Applications, 34(12), 1529–1546, 2017.

[3] K. Shao, Z. Tang, Y. Zhu, N. Li and D. Zhao. A survey of deep reinforcement learning in video games, arXiv preprint arXiv:1912.10944, 2019.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski. Human-level control through deep reinforcement learning, Nature, 518(7540), 529–533, 2015.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot. Mastering the game of Go with deep neural networks and tree search, Nature, 529(7587), 484–489, 2016.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton. Mastering the game of Go without human knowledge, Nature, 550(7676), 354–359, 2017.

[7] G. Lample, D. S. Chaplot. Playing FPS games with deep reinforcement learning, 31st AAAI Conference on Artificial Intelligence, 2017.

[8] K. Shao, D. Zhao, N. Li and Y. Zhu. Learning battles in ViZDoom via deep reinforcement learning, IEEE Conference on Computational Intelligence and Games, 2018, 1–4.

[9] S. Huang, H. Su, J. Zhu, and T. Chen. Combo-Action: Training agent for FPS game with auxiliary tasks, 33rd AAAI Conference on Artificial Intelligence, 2019.

[10] Z. Tang, K. Shao, Y. Zhu, D. Li, D. Zhao and T. Huang. A review of computational intelligence for StarCraft AI, IEEE Symposium Series on Computational Intelligence, 2018, 1167–1173.

[11] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Kttler, J. Agapiou, and J. Schrittwieser. StarCraft II: A new challenge for reinforcement learning, arXiv preprint arXiv:1708.04782, 2017.

[12] O. Vinyals, I. Babuschkin, J. Chung, and et. al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350-354, 2019.

[13] Z. Tang, D. Zhao, Y. Zhu and P. Guo. Reinforcement learning for build-order production in StarCraft II, International Conference on Information Science and Technology, 2018.

[14] L. Panait, and S. Luke. Cooperative multi-agent learning: The state of the art, Autonomous Agents and Multi-Agent Systems, 11(3), 387–434, 2005.

[15] K. G. Jayesh, E. Maxim, and K. Mykel. Cooperative multi-agent control using deep reinforcement learning, AAMAS, 2017, 66–83.

[16] S. Sukhbaatar, A. Szlam, R. and Fergus. Learning multiagent communication with backpropagation, NIPS, 2016, 2244–2252.

[17] J. Foerster, I. A. Assael, N. de. Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning, NIPS, 2016, 2137–2145.

[18] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning, PLOS ONE, 12(4), 2017.

[19] C. Berner, G. Brockman, and et. al. Dota 2 with large scale deep reinforcement learning, arXiv preprint arXiv:1912.06680, 2019.

[20] K. Shao, Y. Zhu, and D. Zhao. StarCraft micromanagement with reinforcement learning and curriculum transfer learning, IEEE Transactions on Emerging Topics in Computational Intelligence, 3(1), 73-84, 2019.

[21] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls. Value-decomposition networks for cooperative multi-agent learning based on team reward, AAMAS, 2018, 2085–2087.

[22] S. Hart. Shapley value, Discussion Paper, 2006.

[23] C. Holmesparker, M. E. Taylor, A. K. Agogino, and K. Tumer. CLEAN rewards to improve coordination by removing exploratory action noise, in IEEE/WIC/ACM International Joint Conferences on Web Intelligence, 2014.

[24] K. Malialis, J. Wang, G. Brooks, and G. Frangou. Feature selection as a multiagent coordination problem, arXiv preprint arXiv:1603.05152, 2016.

[25] M. Colby, S. Kharaghani, C. H. Parker, and K. Turner. Counterfactual exploration for improving multiagent learning, AAMAS, 2015.

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, NIPS, 2017, 6379–6390.

[27] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients, AAAI, 2018, 2974–2982.

[28] T. Rashid, M. Samvelyan, C. S. D. Witt, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning, 35th International Conference on Machine Learning, 2018, 4295–4304.

[29] F. A. Oliehoek, and C. Amato. A Concise Introduction to Decentralized POMDPs, Springer International Publishing, 2016.

[30] R. S. Sutton, and A. G. Barto. Reinforcement Learning: an Introduction. MIT Press, 1998.

[31] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning, ICML, 2016, 1995–2003.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, ICLR, 2016.

[33] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning, ICML, 2017, 449–458.

[34] W. Dabney, M. Rowland, M. G. Bellemare, R. Munos. Distributional reinforcement learning with quantile regression, AAAI, 2018, 2892–2901.

[35] A. K. Agogino, and K. Tumer. QUICR-learning for multi-agent coordination, in Proceedings of the 21th National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, 2006.