# BiC-DDPG: Bidirectionally-Coordinated Nets for Deep Multi-agent Reinforcement Learning

Gongju Wang[1], Dianxi Shi[1,2(✉)], Chao Xue[1,2(✉)], Hao Jiang[3], and Yajie Wang[3]

[1] Artificial Intelligence Research Center (AIRC), National Innovation Institute of Defense Technology (NIIDT), Beijing 100166, China
dxshi@nudt.edu.cn, xuec11@tsinghua.org.cn

[2] Tianjin Artificial Intelligence Innovation Center (TAIIC), Tianjin 300457, China

[3] College of Computer, National University of Defense Technology, Changsha 410073, China

**Abstract.** Multi-agent reinforcement learning (MARL) often faces the problem of policy learning under large action space. There are two reasons for the complex action space: first, the decision space of a single agent in a multi-agent system is huge. Second, the complexity of the joint action space caused by the combination of the action spaces of different agents increases exponentially from the increase in the number of agents. How to learn a robust policy in multi-agent cooperative scenarios is a challenge. To address this challenge we propose an algorithm called bidirectionally-coordinated Deep Deterministic Policy Gradient (BiC-DDPG). In BiC-DDPG three mechanisms were designed based on our insights against the challenge: we used a centralized training and decentralized execution architecture to ensure Markov property and thus ensure the convergence of the algorithm, then we used bi-directional rnn structures to achieve information communication when agents cooperate, finally we used a mapping method to map the continuous joint action space output to the discrete joint action space to solve the problem of agents' decision-making on large joint action space. A series of fine grained experiments in which include scenarios with cooperative and adversarial relationships between homogeneous agents were designed to evaluate our algorithm. The experiment results show that our algorithm out performing the baseline. -

**Keywords:** Multi-agent deep reinforcement learning · Large discrete joint action space · Cooperative · Mapping method

This work was supported in part by the Key Program of Tianjin Science and Technology Development Plan under Grant No. 18ZXZNGX00120 and in part by the China Postdoctoral Science Foundation under Grant No. 2018M643900.


H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 337–354, 2021.
https://doi.org/10.1007/978-3-030-67540-0_20

## 1 Introduction

Multi-agent problems appear common both in nature and human society, especially in computer science and robotics. The study of multi-agent problems is a hot issue both in academia and industry. A series of algorithms based on the mathematical basis of Markov decision process game theory and so on has been formed, such as communication neural network (CommNet) [21], bidirectionally-coordinated network (BiC-Net) [15], QMIX [16], Value-Decomposition Networks (VDN) [22], Counterfactual Multi-Agent Policy Gradients (COMA) [6], Multi-Agent Deep Deterministic Policy Gradient (MADPPG) [13].

In recent years, with the extensive application of deep neural networks (DNN) supported by the powerful computing capabilities, the combination of DNN and multi-agent reinforcement learning has provided new ideas for solving multi-agent cooperation and confrontation problems, among them AlphaStar [25] defeated professional StarCraft players and has exceeded 99.8% of active players online. Alphastar uses multi-agent reinforcement learning based methods to train its model leveling both the human game data and the agent game data. In particular, several league pools are designed to continuously learn better policies and countermeasures. The multi-agent reinforcement learning algorithms also performs well in autonomous cars [7] and the coordination of robot swarms [8].

The purpose of multi-agent reinforcement learning algorithms is to learn an optimal policy $\pi^*$ to get the maximum reward expectation as (1) based on the current state $s_t$.

$$V^*(s) = E_{\pi^*}[R_t|s_t = s] \tag{1}$$

The scenarios of multi-agent reinforcement learning (MARL) can be divided into three types: fully cooperative scenarios, fully competitive scenarios [12], mixed competition/cooperative scenarios. For fully cooperative scenarios, agents need to work together to maximize global reward expectation. The optimal policy can be expressed as (2) which means that every single agent selects its own optimal action under the assumption that other agents have already selected their optimal actions.

$$\pi^{i^*}(s) = \underset{u^i}{argmax} \underset{u^1,..,u^{i-1},u^{i+1},...,u^n}{max} Q^{\pi^*}(s,u) \tag{2}$$

On the contrary, for fully competitive scenarios [10], agent need to maximize their own reward expectation and minimize other agents' reward expectation. The optimal state-value function can be expressed as (3), which $-i$ means other agents except agent $i$.

$$\begin{aligned} V^{i^*}(s) = \underset{\pi^i(s,\cdot)}{max} \underset{u^{-i} \in u^{-i}}{min} \sum_{u^i \in U^i} Q^{i^*}\left(s, u^i, u^{-i}\right) \pi^i\left(s, u^i\right), \\ i = 1, 2... \end{aligned} \tag{3}$$

For the mixed scenarios [11], the learning of agent's policy must consider not only maximizing its own reward expectations, but also cooperating with other

agents to obtain the maximum global reward expectation, which will eventually form a game of Nash equilibrium. It can be expressed as (4).

$$V^{i^*}(s) = \max_{\pi^1(s,\cdot),\cdots,\pi^{n_1}(s,\cdot)} \min_{o^1,\cdots,o^{n_2} \in O^1 \times \cdots \times O^{n_2}} \sum_{u^1,\cdots,u^{n_1} \in U \times \cdots \times U^{n_1}} Q^i\left(s, u^1, \cdots, u^{n_1}, o^1, \cdots, o^{n_2}\right) \pi^1\left(s, u^1\right), \cdots, \pi^{n_1}\left(s, u^{n_1}\right) \tag{4}$$

Traditional single-agent reinforcement learning algorithms usually performs poorly in the scenarios mentioned above. This is due to the lack of Markov properties caused by partial observation of the environment, the neglect of other agent's action, and the large joint action space. The MADDPG algorithm gives each agent an actor-critic network, where the actor network relies on agent's own observation to make decisions, and the critic network will consider the actions of other agents when calculating the $Q$ value, which can be seen as an information sharing mechanism between agents, the algorithm can make the agents show a clear tendency to cooperate. As for COMA, each agent still has its own actor network, but it introduces a centralised critic network to uniformly calculate the $Q$ value of the state. The algorithm uses the global state provided by the environment in training, which can guarantee the Markov property and promote the convergence of the algorithm. However, in multi-agent cooperation scenarios, with the increase in the number of agents, the action space and overall state space of the cooperative agent will increase exponentially. How to learn an effective policy in fully competitive scenarios is a challenge.

In order to solve the problem of policy learning in fully cooperative scenarios, we propose a new multi-agent algorithm called Bidirectionally-Coordinated Deep Deterministic Policy Gradient (BiC-DDPG), we shed light on this problem by modifying it into decentralized partially observable Markov decision progress (Dec-POMDP) and used a policy-based model to deal with the value-based problem, in our algorithm we propose 3 novel methods to meet the challenge of information sharing between agents, lack of Markov property, and action space explosion respectively. First, we use the bidirectional rnn [18] to solve the problem of agents information sharing with each other in training, and it decouples the number of agents from the complexity of training. Then we use the global state training critic. Finally, we map the joint actions of the agents output by the actor network from continuous space to discrete space, so that the problem can be solved by using policy-based reinforcement learning algorithm deep deterministic policy gradient which is called DDPG for short [9].

The rest of the paper was arranged as follows: Sect. 2 first introduces the classification of reinforcement learning, then introduces the existing multi-agent reinforcement learning algorithm based on Dec-POMDP problem modeling. Section 3 undertakes the problem modeling of the Sect. 2 and introduces our algorithm from three aspects: neural network design, action generation process and training algorithm method. Section 4 introduces our experimental setup, including the introduction to the simulation environment, the setting of observations and rewards, and the experimental design of the evaluation algorithm.

Section 5 shows the experimental results of our algorithm compared with other baselines. Section 6 makes our conclusion.

## 2 Related Work and Preliminary

Significant progress has been made in research on reinforcement learning based on single agents, including the Go agent AlphaGo [20], the poker robot Libratus [1] that defeated the human master. Reinforcement learning algorithms can be divided into value-based reinforcement learning algorithms, policy-based reinforcement learning algorithms and search-supervision based reinforcement learning algorithms.

One of the representative algorithms of the value-based algorithm is the Deep Q-Network (DQN) [14]. Its performance in the experimental environment of various video games such as Atari video games even exceeds the level of human masters. The contribution of DQN lies in the combination of Convolutional Neural Network (CNN) and Q-learning algorithm [27] in traditional reinforcement learning, which is more focused on solving decision problems of small-scale discrete action space.

Policy-based reinforcement learning algorithm uses deep neural networks for parametric representation, and uses policy gradient algorithms to optimize its own behavior policy. The representative algorithms include actor-critic (AC) and its variant, DDPG is to deal with the problem of agent learning in continuous action space. It draws on the experience replay mechanism of DQN and draws on AC, The algorithm thought to establish a deterministic policy network and value function, and uses the gradient information on the value function of the action to update the policy network. The goal of optimization is to add up the cumulative rewards for discounts, it performs well in continuous control scenarios, and the convergence speed is much higher than the DQN algorithm.

The typical representative work of supervision based reinforcement learning algorithms is AlphaGo [20], It adopts the idea of combining human experience, Monte Carlo tree search (MCTS) [3] and the trial and error process of reinforcement learning to prove reinforcement learning has the potential to solve single agent control problems in complex state spaces.

Traditional single-agent reinforcement learning algorithm does not perform well in multi-agent scenarios, and most of the typical multi-agent algorithms are based on actor-critic to improve, so we first introduce the modeling of multi-agent problems, then introduce The characteristics of the AC algorithm, finally introduce the multi-agent algorithm in turn.

### 2.1 Decentralized Partially Observable Markov Decision Progress

Decentralised unit micromanagement scenarios refer to the low-level, short-term control of the combat unit during the battle against the enemy. For each battle, the agents on one side are fully cooperative, so each battle can be regarded as a confrontation between $N$ agents and $M$ enemies, in addition, the enemies usually

are built-in AI written by rules. This can be modeled as Dec-POMDP, $s \in S$ can describe the state of the current environment, every agent $a \in A \equiv \{1, \ldots, n\}$ in every game step choose an action $u^a \in U$ form joint action of agents $u \in U \equiv U^{\mathrm{n}}$, In addition, joint action $u \in U$ corresponds to an n-dimensional vector, such that $u \in R^n$,This vector provides information related to the joint action. It will transform the environment according to the state transition function $P\left(s'|s, \mathbf{u}\right) : S \times \mathbf{U} \times S \rightarrow [0, 1]$, all agents share global rewards $r(s, \mathbf{u}) : S \times \mathbf{U} \rightarrow R$, and $\gamma \in [0, 1)$ represents the discount factor.

We also need to describe the partially observable scenario of the agent $\mathrm{z} \in Z$ according to observation function $O(s, a) : S \times A \rightarrow Z$, every agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$ It represents the policy of the agent. $\pi^a(u^a|\tau^a) : T \times U \rightarrow [0, 1]$. The joint policy also has a joint action-value function $Q^\pi\left(s_t, \mathbf{u}_t\right) = {}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}\left[R_t|s_t, \mathbf{u}_t\right]$, the cumulative rewards can be expressed as $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

### 2.2 Actor-Critic and Deep Deterministic Policy Gradient

Both the Actor-Critic algorithm [23] and the Deep Deterministic Policy Gradient algorithm can be classified as policy-based algorithms. The main idea is to adjust the parameters $\xi$ the direction $\nabla_\xi J(\xi)$ to maximize the expected reward $J(\xi) = E_{s\sim p^\pi, u^a \sim \pi_\xi}[R]$. The gradient direction of the AC algorithm can be expressed as (5).

$$\nabla_\xi J(\xi) = \pi_\xi(u^a|s)_{s\sim p^\pi, \mathrm{u}^a \sim \pi_\xi}\left[\nabla_\xi \log \pi_\xi(u^a|s) Q^\pi(s, u^a)\right] \tag{5}$$

The AC algorithm can learn an approximate distribution of the value function $Q^\pi(s, \mathrm{u})$ through time difference error. This distribution is the critic net in AC algorithm. The DDPG algorithm combines the advantages of both the DQN and AC algorithms. It uses experience replay technology and dual neural network structure and AC algorithm to fit the idea of distribution for training. It has the advantages of both algorithms, but if the DDPG is directly deployed under the problem of multi-agent cooperative scenarios, it will also fail to ensure convergence due to the loss of Markovianness in the environment. It is also because the participation environment of multiple agents is unstable.

Inspired by the single agent reinforcement learning algorithm, the multi-agent reinforcement learning algorithm has also made good progress.

### 2.3 CommNet

The CommNet acquiesces agents use the full connection mechanism within a certain range to share information, using $N(j)$ to represent the set of agents $j$. Hidden state $h_j^i$ and exchange information $c_j^i$ are used to get the exchange information of next moment, the calculation of $h_j^i$ and $c_j^i$ are as (6) and (7).

$$c_j^{i+1} = \frac{1}{|N(j)|} \sum_{j' \in N(j)} h_{j'}^{i+1} \tag{6}$$

$$h_j^{i+1} = f^i\left(h_j^i, c_j^i, h_j^0\right) \tag{7}$$

We can find that the shared information is the average value of the hidden state of all agents at the last time, and the hidden state is also derived from the communication information about the last time and the hidden states of the last time. The hidden state output action of the last layer, but with the effect of the increasing number of agents, algorithm has not been demonstrated and is the goal of its next step. This learning algorithm with implicit communication is a good baseline to evaluate our algorithm.

### 2.4 Bidirectionally-Coordinated Network

The work of BiC-Net can be seen as an extension of CommnNet. It consists of a multi-agent actor network and a multi-agent critic network. Both the policy network (actor) and the $Q$ network (critic) is based on a bidirectional rnn structures [15] policy network to receive shared observations, return actions for each individual agent. Since the bidirectional recursive structure can not only be used as a communication channel, but also as a local memory protection program, each agent can maintain its own local state and share information about collaborators, and the structure of bidirectional rnn helps alleviate the increase in the number of agents and the training problems caused by.

### 2.5 QMIX and VDN

The QMIX and VDN algorithm is belonged to value function approximation algorithms, and the agents get a global reward. The algorithms adopt a framework of centralized learning and distributed execution of applications. VDN integrates the value function of each agent to obtain a joint action value function. Let $\tau = (\tau^1, ..., \tau^n)$ be the joint action-observation history, where $\tau^i = \left(u_0^i, o_1^i, \cdots, u_{t-1}^i, o_t^i\right)$ is the action-observation history, which $u = (u^1, ..., u^n)$be the joint action. $Q^{tot}$ is a joint action value function and $Q^i\left(\tau^i, u^i\right)$ is a local action-value function of agent $i$. The local value function only depends on the local observation of each agent. The joint action value function adopted by VDN is direct addition and summation as (8).

$$Q^{tot} = \sum_{i=1}^{n} Q^i\left(\tau^i, u^i\right) \tag{8}$$

QMIX and VDN are also committed to obtaining the joint action value function of all agents, and the expression of QMIX is more abundant. In order to get a decentralized policy that has the same effect as the centralized policy training, there is no need to add the $Q$ value like VDN, and the policy adopted is to make the global value equal to the argmax of each agent's decentralized $Q$ as (9).

$$\text{argmax}Q^{\text{tot}}(\tau, \mathbf{u}) = \begin{pmatrix} \text{argmax}_{u^1} Q^1\left(\tau^1, u^1\right) \\ \vdots \\ \text{argmax}_{u^n} Q^n\left(\tau^n, u^n\right) \end{pmatrix} \tag{9}$$

This allows each agent a to select the $Q$ value associated with it to select the action, and it is very easy to obtain the argmax of each local $Q$. QMIX proposed a mixed neural networks.The processing is through inputting each agent's action-observation history to achieve the agent's communication with an rnn network composed of GRU cells and finally selecting the local $Q$ of each agent. Then input the above $Q$ combination into a mixing network structure to get the $Q^{tot}$ the QMIX algorithm will be deployed in a larger number of agent confrontation environments as its next goal.

### 2.6 COMA

The COMA algorithm is also based on the AC algorithm. A centralized critic network is used to estimate the $Q$ value of the joint action $u$ in state $s$, and the $Q$ function is compared to each agent. The counterfactual baseline keep the actions of other agents $u^{-a}$ and marginalizes the action $u^a$ of agent $a$. The advantage function is defined as (10).

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum \pi^a \left(u'^a | \tau^a\right) Q\left(s, \left(\mathbf{u}^{-a}, u'^a\right)\right) \tag{10}$$

In order to solve the problem of multi-agent credit allocation, a separate baseline $A^a(s, \mathbf{u})$ is calculated for each agent. This process uses the counterfactuals inferred by the critic network to learn directly from the individual experience of the agent without relying on additional simulations, but it is not performed under the condition of a larger number of agents experiment verification record.

## 3 Bidirectionally-Coordinated Deep Deterministic Policy Gradient

We make a comprehensive elaboration of the algorithm BiC-DDPG we proposed, BiC-DDPG was purposed to solve the problem of policy learning in multi-agent scenarios. We will introduce the solutions proposed by the algorithm in order to overcome the three problems: environmental stability, information sharing between agents, and large joint action space.

### 3.1 Environmental Stability

The modeling of this work is based on Dec-POMDP, for each agent, due to the limitation of its own observation range, its observation of the environment is incomplete, resulting in the environment is unstable for a single agent. In order to solve this problem, the current mainstream is to adopt the idea of centralized training distribution execution to design the algorithm. In other words, the decentralized actor network makes actions based on part of the agent's observation, while the centralized critic network is based on the state output $Q$ value to update the actor network.

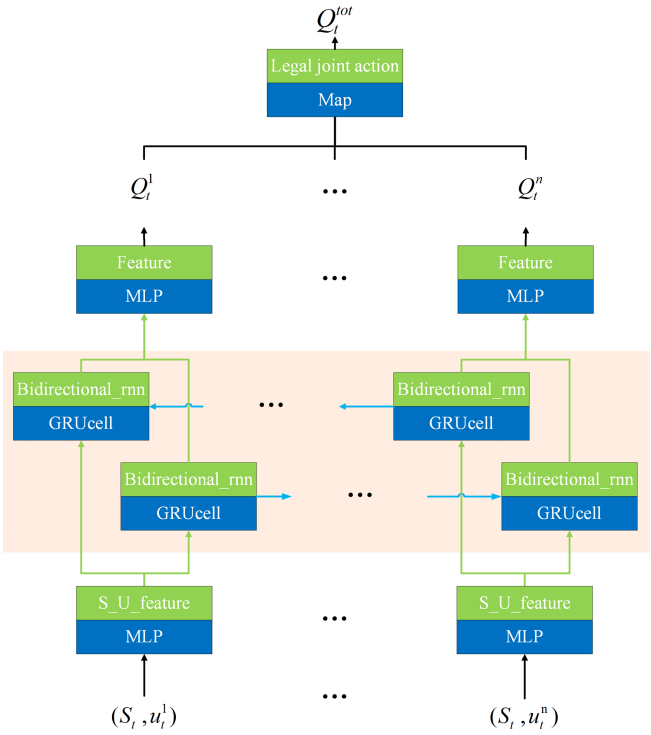



**Fig. 1.** Critic net of Bic-DDPG.

In the simulation environment of this paper, at each time step of training we can know the actions taken by all agents. So based on this fact we can assume that the simulation environment satisfies the Markov property as (11)

$$P_{ss'} = \mathbb{E}\left(s_{t+1} = s' \mid s_t = s, u_t = u\right) \tag{11}$$

Under assumption of (11), we designed the critic network as shown in the Fig. 1. The input data is global state and every agent's legal action in order to make the centralized critic function more quickly and efficiently fit the state transition function $P$. We use bidirectional rnn network structure to accept the last layer's data features. According to the features of bidirectional rnn, the output of each time step is not only related to the input of other locations but also related to the input of its corresponding position. So each output of the time step implies the evaluation of the joint action, we finally merge these outputs into $Q^{tot}$. In order to better approximate the state transition function $P$, we minimize the loss as follows:

$$L\left(\theta\right) = \frac{1}{N}\sum_{t}\left[y_t - Q_\theta\left(\mathbf{s}_t, \mathbf{u}_t\right)\right]^2 \tag{12}$$

$$y_t = r_t + \gamma Q_{\theta'}\left(s_{t+1}, \pi_{\xi'}\left(o_{t+1}\right)\right) \tag{13}$$

In addition, the calculation of the value function is designed into two steps. First, the local value function of each agent is calculated as the value $Q_t^n$, and then the values are used to calculate the $Q^{tot}$. In a scenario where only a global reward is given, $Q^{tot}$ can be used for calculation, and if the scene gives each agent's own local reward, the $Q$ value of each agent can be used to participate in the calculation and fitting.

### 3.2 Information Sharing

We inspired by the work of CommNet to add the bidirectional rnn structure to the actor network as shown in Fig. 2. Since the bidirectional rnn structure is an extended version of the rnn structure, its output combines the results of forward propagation and backward propagation in the structure, which enables the agent to implicitly consider the situation of other agents when making decisions. Equation (14) shows that the hidden state of the agent $i$ at step $t$ in the forward propagation structure depends on its own observation $o_i^t$, hidden state $fo_{i-1}^t$ calculated by forward propagation, and hidden state $bk_{i+1}^t$ calculated from backward propagation. And each agent also continues to pass forward $fo_i^t$ and backward hidden state $bk_i^t$ at the same time, providing a basis for other agents to make decisions. So we can think that the decision of each agent is based on the local observations of all agents. We use variable $U$ and $W$ to represent the parameters of the corresponding position in the process of forward propagation

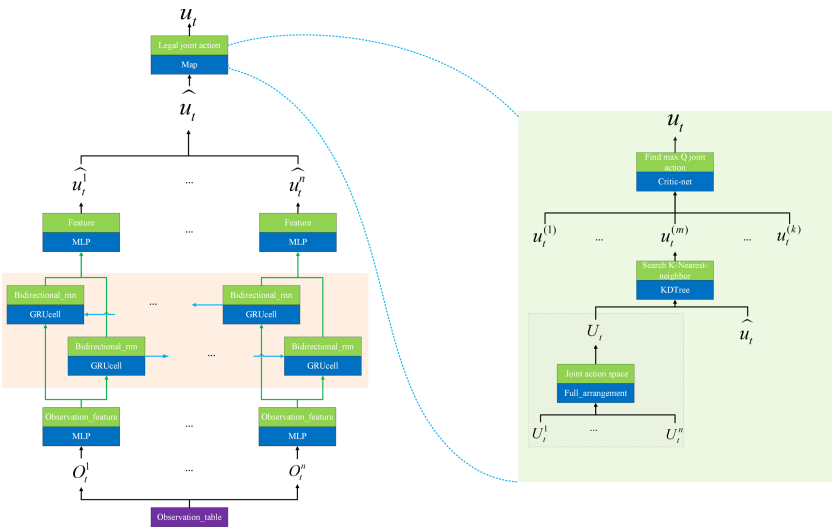



**Fig. 2.** Actor net of BiC-DDPG. The right part shows the mapping process from proto joint action to legal joint action.

and backward propagation, we use variable $b$ represents the bias and $\phi$ represents activation function. The calculation of the forward hidden state can be expressed as Eq. (14) and the calculation of the backward hidden state can be expressed as Eq. (15), both of them have the same structure but the propagation direction is opposite.

$$fo_i^t = \phi(U_i o_i^t + W_i(fo_{i-1}^t) + b_i) \tag{14}$$

$$bk_i^t = \phi(U_i^{'} o_i^t + W_i^{'}(bk_{i+1}^t) + b_i^{'}) \tag{15}$$

The final output is the combination of forward propagation output in (16) and the backward propagation output in (17), as shown in (18).

$$fo\_out_i^t = V_i * fo_i^t + c_i \tag{16}$$

$$bk\_out_i^t = V_i^{'} * bk_i^t + c_i^{'} \tag{17}$$

$$out_i^t = [fo\_out_i^t, bk\_out_i^t] \tag{18}$$

### 3.3 Large Joint Action Space

A method of using policy-based reinforcement learning algorithm DDPG to handle large-scale discrete actions when training single agents was proposed by G. Dulac-Arnold et al. [5]. It provides new ideas and algorithms for solving the problem of modeling based on value-based. In order to solve the problem of large joint action space, inspired by their work, we propose a method of action mapping. We adopt a deterministic policy and the agent's decision is based on their own observations, our policy for generating actions should be:

$$\pi_\xi(o) = \hat{u} \tag{19}$$

However, the proto joint action $\widehat{u}$ of the output agent may not be an executable joint action at this time, so $\widehat{u} \notin U$. Therefore, we need to map the proto joint action to a legal discrete joint action space $U$.

$$f_k(\widehat{u}) = \arg\min_{u \in U} |u - \widehat{u}|_2 \tag{20}$$

So we used an action mapping method as (20), Where $f_k$ is the $k$ nearest neighbors to $\widehat{u}$ in the discrete joint action space, the distance measure is the Euclidean distance, in order to improve the efficiency of searching, the paper [5] uses the K-Nearest Neighbor [4] algorithm to search. But the time complexity of knn querying neighbors is generally estimated to be $O(D * N * N)$, where $D$ represents the data dimension and $N$ represents the number of samples, In order to query of near legal joint actions more quickly, this paper uses the method of

KDTree to find it, and it can also find the result in logarithmic time which is $O(log(D * N * N))$. The proto joint action maps to the $k$ number of legal joint actions we specify. To ensure that the selected joint action always evaluates the highest profit, the candidate joint action set needs to be evaluated. And we choose the joint action with the highest possible return which is shown in Fig. 2.

$$u = argmaxQ(s,u)|_{u=[u^{(1)},u^{(2)},...,u^{(k)}]} \tag{21}$$

The goal of the training algorithm is to find an optimal policy $\pi_{\xi^*}$, and the most important thing of the work is to optimize the parameters $\xi$ so that the cumulative return obtained by it reaches the maximum value. The optimization goals for actor parameters are defined as (22).

$$\nabla_\xi J \approx \frac{1}{N}\sum_i \nabla_u Q_\theta(s,\hat{u})|_{s=s_i,\hat{u}=\pi(o_i)} \nabla_\xi \pi(o)|_{o=o_i} \tag{22}$$

We summarize the algorithm flow as the pseudo code shown in as follows.

**Algorithm 1.** BiC-DDPG algorithm

Initialise actor network and critic network with $\xi$ and $\theta$
Initialise target network and critic network with $\xi^{'} \longleftarrow \xi$ and $\theta^{'} \longleftarrow \theta$
Initialise replay buffer with buffer $R$
**for** $epsiod = 1, 2, ..., E$ **do**
  initialise a random process $\mu$ for action exploration
  receive initial local observation $o_1$ and global state $s_1$
  **for** $t = 1, 2, ..., T$ **do**
    for each agent $a$, get select action $\hat{u}_t^a = \pi_\xi(o_t^a) + \mu_t^a$
    all agent actions form a joint action $\hat{u}_t$
    get legal joint action set $[u_t^{(1)}, u_t^{(2)}, ..., u_t^{(k)}]$
    get the max $Q$ legal joint action
    $u_t = argmaxQ(s_t,u)|_{u=[u_t^{(1)},u_t^{(2)},...,u_t^{(k)}]}$ and excute
    receive reward $r_t$ and new local oberservation $o_t$ and global state $s_t$
    Store transition $([o_t, s_t], u_t, r_t, [o_{t+1}, s_{t+1}])$ in $R$
    Sample a random minibatch of $N$ transitions $([o_i, s_i], u_i, r_i, [o_{i+1}, s_{i+1}])$ from $R$
    Set $y_i = r_i + \gamma Q_\theta(s_{t+1}, \pi_{\xi'}(o_{t+1}))$
    Update the critic by minimizing the loss:
    $L(\theta) = \frac{1}{N}[\sum_i (y_i - Q_\theta(s_i, u_i))]^2$
    Update the actor using the sampled gradient:
    $\bigtriangledown_\xi J \approx \frac{1}{N}\sum_i \bigtriangledown_u Q_\theta(s,\hat{u})\ |_{s=s_i,\hat{u}=\pi(o_i)}\ \bigtriangledown_\xi \pi(o)\ |_{o=o_i}$
    Update the target networks:
    $\xi^{'} \longleftarrow \tau\xi + (1-\tau)\xi^{'}$
    $\theta^{'} \longleftarrow \tau\theta + (1-\tau)\theta^{'}$
  **end for**
**end for**

## 4 Experiments

In this section, we will describe the simulation environment for evaluating our algorithm and the arrangement of experiments.

### 4.1 StarCraft II Cooperative Scenarios

Real-time strategy (RTS) is the main research platform and tool for multi-agent reinforcement learning [2,19,24], StarCraft II is a typical RTS game, which contains the confrontation with a variety of heterogeneous agents, providing sufficient materials for simulation experiments, and existing researchers had developed API wrappers for researchers to carry out on this platform Reinforcement learning research, which is representative of pysc2 [26] and SMAC [17]. These efforts alleviate the pressure of researchers on the environment interface call work, allowing them to focus more on the optimization of algorithms.

We use SMAC as the interface for the algorithm to interact with the environment, It specifies the agent's discrete action space: move[direction], attack[enemy id], stop, and no-op. An agent can only move in four directions: east, west, south, and north. An agent is only allowed to attack when the enemy is within its range.

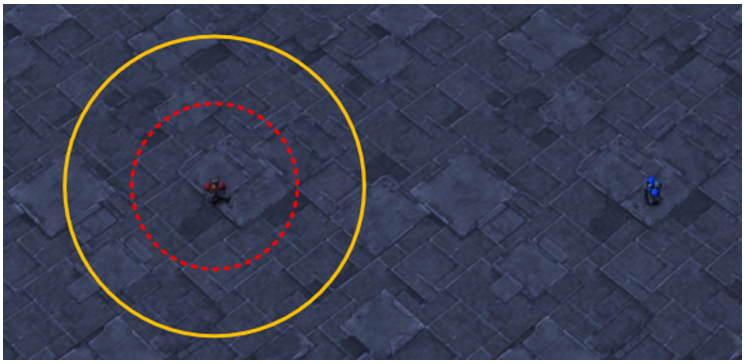


**Fig. 3.** Partial observation of the agent, where the red dotted circle represents the attack range of the agent, and the orange solid circle represents the observation range of the agent. (Color figure online)

Part of the observability is achieved by restricting the unit of view of the field of view, which limits the agent to observe the information about the enemy or friendly agent outside the field of view as shown in Fig. 3. In addition, the agent can only observe living enemy units, and cannot distinguish between units that are dead or units that are out of range.

In order to understand the properties and performance of our proposed BiC-DDPG algorithm, we conducted experiments with different settings in the cooperative scenarios of StarCraft II, Similar to Tabish Rashid's work [16].

BiC-DDPG controls a group of agents to try to defeat enemy units controlled by built-in AI. We use the winning rate and reward in training as the sole criterion for evaluating the algorithm.

### 4.2 State, Local Observation, Reward

The local observation value of each agent is a circular observation range centered on the unit it controls, and 10 is a radius. The observation contains information about all agents in the range: distance from the target, attribution of target, type of target, The relative $x$ coordinate of the target, the relative $y$ coordinate of the target, the health, the shield, and the cooling of the weapon. All attribute items are normalized and preprocessed according to their maximum values, which is convenient to maintain a stable amplitude when the neural network is learning.

The global state is used to train the critic network. It contains information about all units on the map. Specifically, the state vector includes the coordinates of all agents relative to the center of the map, and the unit features that appear in the observation.

As for reward calculation: at each time step, the agents receiving a global reward equal to the damage to the target unit plus minus half of the damage taken. Killing an enemy will get a 10 point reward, and winning the game will get a reward equal to the team's total remaining health plus 200. Compensation calculation followed Table 1. This damage-based reward signal uses the same mechanism as the QMIX algorithm. The benefit is that the results of joint action are evaluated using a shared reward. Since the absolute value of the maximum reward obtained in different scenarios is different, we finally limit the range of rewards to between 0 and 20.

**Table 1.** SMAC's rewards calculation

| Name | Description | Reward |
|---|---|---|
| Kill | Kill an enemy | 10 |
| Win | All enemies are eliminated | 200+remaining health |
| Attack | Attack an enemy once in a time step | Damage to target |
| Be attacked | Be attacked by an enemy once in a time step | Half of damage taken |

### 4.3 Homogeneous Agents Cooperative Scenarios

We have designed experiments with different features in order to evaluate the performance of various aspects of the algorithm. Our experiments can be divided into three categories, We will explain the purpose of the experiment settings.

**Cooperation with a Smaller Number of Homogeneous Agents.** In this type of experiment, we avoid the impact on the heterogeneity of the agent, and test the learning ability of the algorithm for fine-grained combat in a small number of cases. Compare its performance with the baseline algorithms. The test environment is selected as 3 marines (3m) (Fig. 4).


**Fig. 4.** n marines scenarios(n = 3, 5, 8) and red is our agents. In these scenarios, the number and types of our agents are equal to those of the enemy. The purpose of this is to evaluate the willingness to cooperate between agents. (Color figure online)

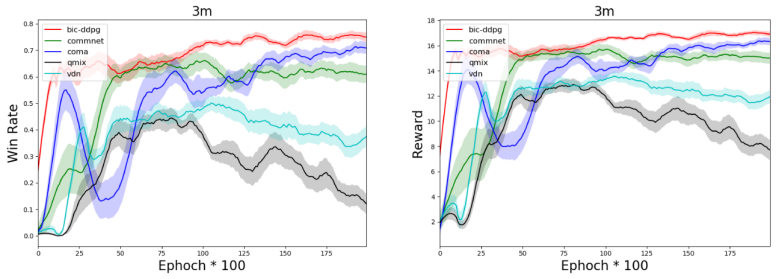
**Cooperation with a Large Number of Homogeneous Agents.** In this type of experiment, we also avoid the influence of the heterogeneity of the agents, but we increase the number of agents to make the confrontation complicated. This mainly evaluates the learning degree of the algorithm for the general fire policy, The test environment is selected as 5 marines (5m), 8 marines (8m).

**Self-contrast Experiment.** In the self-comparison experiment, we will compare the performance curve of the algorithm with the change of the number of homogeneous agents. The test environment is Selected as 3m, 5m, 8m.

All scenarios are run 20,000 epochs, and the reward and win rate are recorded every 100 epochs. The above process loops 10 times to evaluate the overall performance of the algorithm.

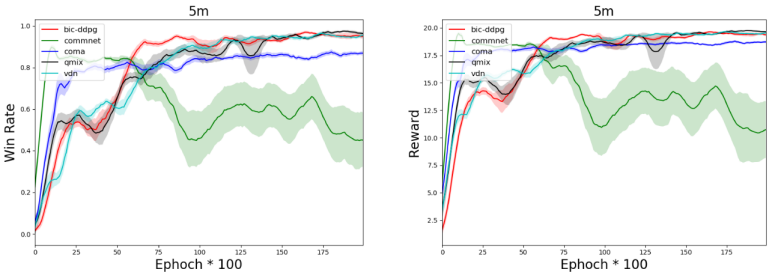
## 5 Results and Discussion

The 3m scenario's experiment results are as shown in Fig. 5, we can find that all algorithms can update their policies on the direction on increasing win rate and rewards. The BiC-DDPG algorithm can achieve win rate of about 60% in the first

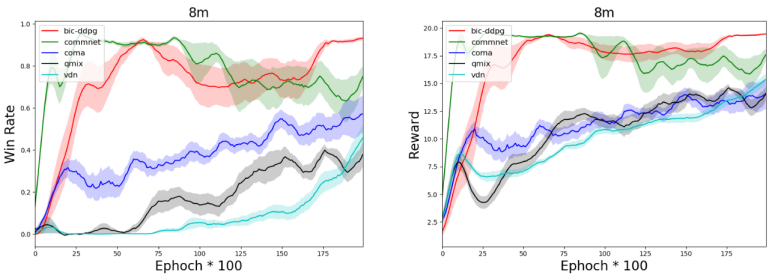



(a) Win rate of 3m. (b) Reward of 3m.

**Fig. 5.** Results of cooperation with a smaller number of homogeneous agents




(a) Win rate of 5m. (b) Reward of 5m.




(c) Win rate of 8m. (d) Reward of 8m.

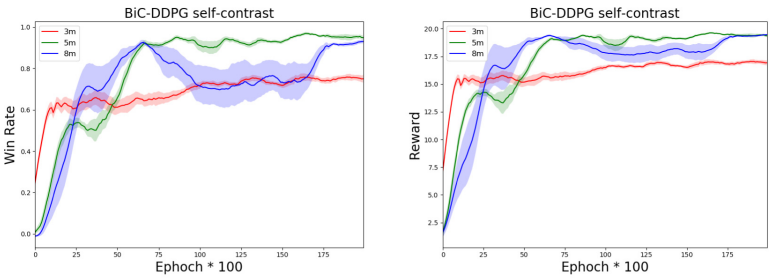
**Fig. 6.** Results of cooperation with a larger number of homogeneous agents.

800 epochs, baselines except COMA algorithm seems to find the right update direction on the gradient, other algorithms are still looking for a better gradient update direction. From the global observation of the convergence of the policies, BiC-DDPG can update the policy on a good direction, it is slightly better than COMA and CommNet. Although the COMA algorithm has experienced the wrong gradient update direction, it can correct the direction in time and its final win rate is concentrated on 70%. CommNet's convergence curve trend is similar to BiC-DDPG, its overall performance is weaker than BiC-DDPG algorithm and its final win rate is around 60%. The QMIX and VDN algorithms update parameters in the wrong direction and their final win rate are around 11% and 40%. The final results of CommNet, COMA and BiC-DDPG algorithms show that they have the ability to converge policy learning to a relatively high win rate in a short training time. QMIX and VDN are not good at handling the problem of fine-grain agents control in a short period of training time. The trend of the reward curve is like the trend of the win rate (Fig. 6).

In the 5m experiment, we can find that the performance of the COMA algorithm is still good, it can upgrade the policy on a higher level in the early stage of training, and its parameter optimization speed is second only to CommNet, its final win rate is stable at about 81%. The final results of both QMIX, VDN, and BiC-DDPG are close and their win rates are stable at 90%, The optimiza-

tion speeds of the three algorithm are similar. In the range of 5,000 to 9,000 epochs, the BiC-DDPG obviously surpasses other algorithms. CommNet's policy improvement speed in the early stage is the fastest, but the parameter update in the later training seems to be in the wrong direction, resulting in its worst performance, but still have a 50% win rate. The trend of the reward is similar to the winning percentage (Fig. 7).

With the increase in the number of agents again in 8m experiment, the final performance of baselines have decreased significantly relative to the 5m experiment. The VDN algorithm and QMIX algorithm have once again fallen to 30% and 43% win rate. Although the COMA algorithm has a relatively large drop rate, it eventually stabilized at 58% win rate, and in this scenario our algorithm's win rate suddenly rose to 90% win rate near the end of the experiment, and the second best performing CommNet algorithm was only about 78%.




(a) Win rate (b) Reward

**Fig. 7.** Self-Contrast, in order to observe the changes in the performance of the algorithm with the number of agents, it can be found that the performance is the best in the 5m scenario.

In the scenarios of self-contrast result, we found that the algorithm performs best in the 5m scenario, and the final performance in the 8m scenario is close to it. The performance in the 3m scenario is obviously not as good as the above two, which proves that the algorithm is not very good at policy learning in small-scale scenarios.

## 6 Conclusion

This paper proposes a deep multi-agent reinforcement learning algorithm BiC-DDPG, which allows for end-to-end learning of decentralized policy on a centralized environment and the effective use of additional state information.

BiC-DDPG allows the use of policy-based reinforcement learning algorithms to solve value-based decision-making problems. This is achieved by mapping the actions of the agent in the continuous space to the discrete space. The results

of our decentralized unit micromanagement scenarios of StarCraft II show that, compared with other multi-agent algorithms, BiC-DDPG improves the problem of the explosion of the decision space dimension caused by the increase in the number of agents.

This paper evaluates the effectiveness of the algorithm in a simulation environment where the number of agents is less than 8, because the simulation environment provides a variety of multi-agent cooperation scenarios. Therefore, In the following work, we also intend to verify the effect of the algorithm in the cooperation of heterogeneous agents in a similar way, and also verify the impact of changes in the number of agent types on the algorithm. Then we will verify the effectiveness of our algorithm in the scenarios of larger number of homogeneous agents and heterogeneous agents.

In addition, BiC-DDPG will be deployed in the experimental environment of the full-length confrontation with StarCraft II, as a micro-management module added to the algorithm of the full-length game.

## References

1. Brown, N., Sandholm, T.: Safe and nested endgame solving for imperfect-information games. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
2. Bubeck, S., Cesa-Bianchi, N.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found. Trends Mach. Learn. **5**(1), QT06, 1–7, 9–21, 23–43, 45–65, 67–105, 107–115, 117–127 (2012)
3. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75538-8_7
4. Dudani, S.A.: The distance-weighted k-nearest-neighbor rule. IEEE Trans. Syst. Man Cybern. **SMC-6**, 325–327 (1976)
5. Dulac-Arnold, G., et al.: Deep reinforcement learning in large discrete action spaces. http://arxiv.org/abs/ArtificialIntelligence (2015)
6. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients (2017)
7. Liu, J., Li, P., Chen, W., Qin, K., Qi, L.: Distributed formation control of fractional-order multi-agent systems with relative damping and nonuniform time-delays. ISA Trans. **93**, 189–198 (2019)
8. Kolokoltsov, V.N., Malafeyev, O.A.: Multi-agent interaction and nonlinear Markov games (2019)
9. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. Comput. Sci. **8**(6), A187 (2015)
10. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Machine Learning Proceedings 1994, pp. 157–163. Elsevier (1994)
11. Littman, M.L.: Friend-or-foe Q-learning in general-sum games. ICML **1**, 322–328 (2001)
12. Littman, M.L.: Value-function reinforcement learning in Markov games. Cogn. Syst. Res. **2**, 55–66 (2001)

13. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. ArXiv abs/1706.02275 (2017)
14. Mnih, V., et al.: Playing Atari with deep reinforcement learning. ArXiv abs/1312.5602 (2013)
15. Peng, P., et al.: Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play StarCraft combat games (2017)
16. Rashid, T., Samvelyan, M., de Witt, C.S., Farquhar, G., Foerster, J.N., Whiteson, S.: QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In: ICML (2018)
17. Samvelyan, M., et al.: The StarCraft multi-agent challenge. In: AAMAS (2019)
18. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. IEEE Trans. Sig. Process. **45**(11), 2673–2681 (1997)
19. Seabold, S., Perktold, J.: Statsmodels: econometric and statistical modeling with Python (2010)
20. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
21. Sukhbaatar, S., Szlam, A., Fergus, R.: Learning multiagent communication with backpropagation. ArXiv abs/1605.07736 (2016)
22. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning. In: AAMAS (2018)
23. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. IEEE Trans. Neural Netw. **16**, 285–286 (1998)
24. Tavares, A.R., Azpurua, H., Santos, A., Chaimowicz, L.: Rock, paper, StarCraft: strategy selection in real-time strategy games. In: AIIDE (2016)
25. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
26. Vinyals, O., et al.: StarCraft II: a new challenge for reinforcement learning (2017)
27. Watkins, C.J.C.H.: Learning from delayed reward. Ph.D. thesis, Kings College University of Cambridge (1989)