

Article

Reducing Q-Value Estimation Bias via Mutual Estimation and Softmax Operation in MADRL

Zheng Li ¹, Xinkai Chen ¹, Jiaqing Fu ¹, Ning Xie ^{1,*} and Tingting Zhao ^{2,3}

¹ Center for Future Media, the School of Computer Science and Engineering, and Yibin Park, University of Electronic Science and Technology of China, Chengdu 611731, China; 202152080128@std.uestc.edu.cn (Z.L.); 202152011708@std.uestc.edu.cn (X.C.); 17713626261@163.com (J.F.)

² School of Computer Science and Technology, Tianjin University of Science and Technology, Tianjin 300457, China; tingting@tust.edu.cn

³ RIKEN Center for Advanced Intelligence Project (AIP), Tokyo 103-0027, Japan

* Correspondence: xiening@uestc.edu.cn or seanxiening@gmail.com; Tel.: +86-1779-640-6627

Abstract: With the development of electronic game technology, the content of electronic games presents a larger number of units, richer unit attributes, more complex game mechanisms, and more diverse team strategies. Multi-agent deep reinforcement learning shines brightly in this type of team electronic game, achieving results that surpass professional human players. Reinforcement learning algorithms based on Q-value estimation often suffer from Q-value overestimation, which may seriously affect the performance of AI in multi-agent scenarios. We propose a multi-agent mutual evaluation method and a multi-agent softmax method to reduce the estimation bias of Q values in multi-agent scenarios, and have tested them in both the particle multi-agent environment and the multi-agent tank environment we constructed. The multi-agent tank environment we have built has achieved a good balance between experimental verification efficiency and multi-agent game task simulation. It can be easily extended for different multi-agent cooperation or competition tasks. We hope that it can be promoted in the research of multi-agent deep reinforcement learning.

Keywords: reinforcement learning; game AI; multi-agent Q-network mutual estimation; softmax bellman operation; reinforcement learning environment



Citation: Li, Z.; Chen, X.; Fu, J.; Xie, N.; Zhao, T. Reducing Q-Value Estimation Bias via Mutual Estimation and Softmax Operation in MADRL. *Algorithms* **2024**, *17*, 36. <https://doi.org/10.3390/a17010036>

Academic Editor: Frank Werner

Received: 14 December 2023

Revised: 7 January 2024

Accepted: 11 January 2024

Published: 16 January 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The emergence and development of Artificial Intelligence in various key domains has intrinsic ties with games. The earliest AI program was a draughts game written by Christopher Strachey in 1951. It successfully ran for the first time on the Ferranti Mark I universal electronic computer at the University of Manchester in the UK. Impressed and influenced by Strachey and Turing, Arthur Samuel took on the draughts project's key points initiated by Strachey in 1952 and expanded it considerably over the years, allowing the program to learn from experience. As one of the earliest cases of Artificial Intelligence search technology, these advancements enabled the program to win a match against a former draughts champion from Connecticut, USA. Samuel's experiments and accumulation in the game technology field helped him propose the concept of machine learning in 1959.

In March 2016, Google's Deep Mind developed the Alpha Go system, defeating the then human world champion of Go, Lee Sedol, thus creating a global sensation. The Go board is composed of 19 horizontal lines and 19 vertical lines, with a total of $19 \times 19 = 361$ intersections. Each intersection on the Go board has three possibilities: a black stone, a white stone, or empty. Therefore, Go has a staggering number of possible positions: 3 to the power of 361. Each turn in Go has 250 possibilities, and a game can last up to 150 turns. As a result, the computational complexity of Go is 250 to the power of 150, which is approximately 10 to the power of 170. However, the observable number of atoms in the entire universe is

only 10 to the power of 80. This clearly demonstrates the complexity and variability of the game of Go. In principle, AlphaGo uses a modified version of the Monte Carlo algorithm, but more importantly, it enhances its learning effect by leveraging deep reinforcement learning [1].

Reinforcement learning is a machine learning method that utilizes sequence information consisting of actions from agents, states from the environment, and rewards through interaction between agents and the environment [2]. The Q function is a fundamental concept in reinforcement learning, which can output the value of action a in state s . Learning an accurate Q function can help policy functions update in the correct direction. Due to the influence of updated formulas, inflexible function estimation methods, and noise, the Q function often has bias in estimating the true Q value [3,4]. The deviation in Q-value estimation may lead to the algorithm converging to a suboptimal policy, resulting in a decrease in algorithm performance.

Bright possibilities also exist for the application of deep reinforcement learning in multi-agent systems like drone formation control, cooperative-control multi-robots, and traffic vehicle control, all of which can significantly improve collective gains and enhance the quality of human life. Multi-agent reinforcement learning can be seen as the expansion of single-agent reinforcement learning algorithms in a multi-agent scenario. The migration of policy gradient algorithms based on Q-function learning to multi-agent scenarios will bring more severe impact due to the bias in Q-function estimation. The bias in the estimation of the Q-function will increase the variance of the network gradient and affect the update process of the policy network. In a multi-agent environment, with the increase in agents, the difficulty of updating the policy network in the correct direction increases exponentially, so such a bias will bring greater difficulties to policy learning in multi-agent scenarios.

This article primarily revolves around the research of the Q-value estimation bias problem in Multi-Agent Deep Reinforcement Learning, and constructs a new multi-agent experimental environment. The detailed research objectives are as follows:

1. The method of combining multiple Q networks to reduce the Q-value estimation bias is further studied, according to the feature of all agents having the same structured Q network in the architecture of centralized training with distributed execution in multi-agent systems. In this research objective, we aim not to increase the network parameters of the model, but to alleviate the problem of Q-value underestimation caused by dual Q networks by organizing multi-agent systems.
2. The method of using softmax operation to reduce Q-value estimation bias is further studied, and the softmax operation method is applied to multi-agent algorithms. This research aims to utilize the characteristics of softmax to reduce the overestimation bias of Q-value, and to simplify the action space used for computation in multi-agent systems, thereby reducing computational load.
3. For scientific research on multi-agent reinforcement learning, we have developed the multi-agent tank environment of the University of Electronic Science and Technology of China (UESTC-tank). This is a tank simulation game environment with multiple units and scenes, which can generate cooperative, adversarial, and mixed task scenarios. It can achieve competitions of player vs. player, player vs. AI, AI vs. AI types, and collect battle data for model training. It can support unit editing, map editing, and rule editing, and has good scalability. We tested our proposed algorithm on UESTC-tank and achieved good experimental results.

The rest of this paper is arranged as follows: Firstly, we introduced the research context in Section 2. Then, in Sections 3 and 4, we introduced the multi-agent twin-delayed deep deterministic policy gradient and multi-agent soft-maximization twin-delayed deep deterministic policy gradient algorithms. In Sections 5 and 6, we introduced the multi-agent particle environment and multi-agent tank environment, conducted experiments, and analyzed the results. After summarizing our work so far, Section 7 provides limitations and conclusions of the paper.

2. Research Background

2.1. Deep Reinforcement Learning

Reinforcement learning can be modeled as an agent is faced with a sequential decision making problem. At time t , the agent observes state s_t from the environment, and adopts action a_t based on its policy. Due to the interaction of the agent, the environment transitions from state s_t to the next state s_{t+1} and returns a reward r to the agent. If the environment transitions to a termination state, we call it the end of an episode. The goal of reinforcement learning is to enable agents to learn an optimal policy to obtain the maximum cumulative reward (called return) from the entire episode.

Reinforcement learning algorithms can be classified into policy function-based [5] and value function-based [6]. A common value function is the action value function, also known as the Q function, which inputs the current state of the environment and the action taken by the agent, and outputs an estimate of the expected return for that state–action pair. When the agent learns the optimal Q function, the optimal policy can be to select the action that can maximize the output value of the optimal Q-function in any state. In deep reinforcement learning, we use deep neural networks to fit the policy function, value function, and Q function, hence also known as policy networks, value networks, and Q networks.

2.2. Deep Q-Learning Algorithms

Mnih and others [6] were the first to integrate deep learning and traditional reinforcement learning algorithm, thus proposing the DQN algorithm. This pioneering piece of work marked the first end-to-end deep reinforcement learning algorithm using only Atari game pixel data and game scores for reinforcement learning state information. Finally, the trained network could output action-value functions or Q-values for each action. From this, the decision with the highest corresponding Q-value is selected. The entire structure of the DQN algorithm is shown in Figure 1.

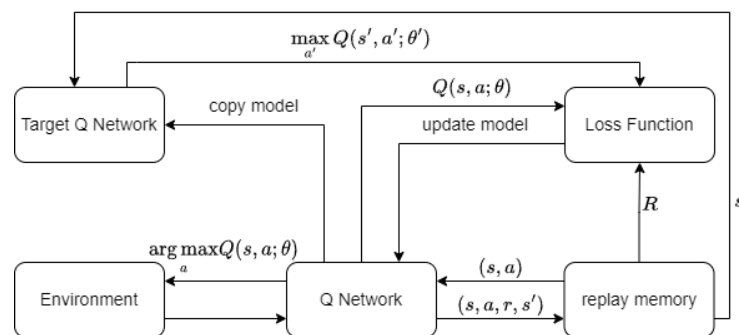


Figure 1. DQN Algorithm Flowchart. DQN uses deep neural networks to fit the Q function, selects the action that maximizes the Q value through a ϵ -greedy policy, interacts with the environment, and stores the obtained samples in replay memory. DQN continuously extracts samples from replay memory, performs gradient descent based on the loss function, and updates the parameters of the Q-network. The parameters of the Q-network are periodically copied to the target Q-network, which participates in the calculation of the loss function and makes Q-learning more stable.

The DQN algorithm introduces the experience replay mechanism, which stores the sample tuple (s, a, r, s') of the agent at each time step into an experience replay pool. During reinforcement learning training, a batch of samples is sampled from the experience replay pool and the neural network is updated using gradient descent. The use of experience replay mechanism greatly improves the data utilization of reinforcement learning algorithms and to some extent reduces the correlation between training data and sequences.

Another important contribution of the DQN algorithm is the concept of the target network, which achieves stable value estimation. The target value y for updating the Q network by computing the loss function is represented as:

$$y = r + \gamma \max_{\hat{a}} Q_{\theta'}(s', \hat{a}) \quad (1)$$

where $Q_{\theta'}$ represents the target network. γ is a discount factor ($0 < \gamma < 1$), used to give a preference for rewards reaped sooner in time. $\hat{a}$ is the action that allows the Q function to take the maximum value. The structure of the target network is the same as the Q network, but its parameters are only copied from the Q network every certain number of iterations.

The Q network updates its parameters in each round by minimizing the loss function defined in Equation (2):

$$L(\theta) = E_{(s,a,r,s')} [(y - Q_{\theta}(s, a))^2] \quad (2)$$

E represents mathematical expectation, which is calculated using Monte Carlo estimation method in practice, taking the average value of batch samples to estimate. The neural network is updated to make the Q network approximate the target value y , which essentially applies the Bellman equation. The use of target network helps improve the stability of the algorithm, and experiments have shown that NPC controlled by DQN algorithm can reach the level of professional players in many Atari games.

The DDPG [7] algorithm can be seen as a combination of Deterministic Policy Gradient (DPG) algorithm and deep neural networks, and it can also be seen as an extension of DQN algorithm in a continuous action space. It can solve the problem of the DQN algorithm's inability to directly apply to continuous action space, but there is still the problem of overestimation of Q values. The DDPG algorithm simultaneously establishes a Q network (critic) and a policy network (actor) [8]. The Q network is the same as the DQN algorithm, updated using a temporal difference method. The policy network utilizes the estimation of the Q network and is updated using policy gradient method.

In the deep deterministic policy gradient algorithm, the policy network is a deterministic policy function represented as $\pi_{\phi}(s)$, with the learning parameters represented as ϕ . Each action is directly computed by $a = \pi_{\phi}(s)$, without sampling from a random policy.

2.3. Estimation Bias of Q-Learning Algorithms

When an agent executes action a in state s , the environment will transition to state s' and return an immediate reward r . In the standard deep Q-learning algorithm, we update the Q-function using Equation (1). When using neural networks and other tools as function approximators to handle complex problems, there can be errors in the estimation of Q values, which means:

$$Q(s', \hat{a}) = Q^*(s', \hat{a}) + Y_{s'}^{\hat{a}} \quad (3)$$

where $Y_{s'}^{\hat{a}}$ is a zero-mean noise term, Q^* is a Q function that can output the real Q value, or the optimal Q function.

However, when using the maximum operation to select actions, there can be errors between the estimated Q values and the real Q values. The error is represented as Z_S :

$$Z_S \stackrel{\text{def}}{=} \gamma \left(\max_{\hat{a}} Q(s', \hat{a}) - \max_{\hat{a}} Q^*(s', \hat{a}) \right) \quad (4)$$

Considering the noise term $Y_{s'}^{\hat{a}}$, some Q values may be underestimated, while others may be overestimated. The maximization operation always selects the maximum Q value for each state, which makes the algorithm overly sensitive to the high estimated Q values of the corresponding actions. In this case, the noise term causes $\mathbb{E}[Z_S] > 0$, resulting in the problem of overestimation of Q values.

The TD3 [9] algorithm introduces clipped Double Q-Learning on the basis of the DDPG algorithm, by establishing two Q networks $Q_{\theta_1}, Q_{\theta_2}$ to estimate the value of the next state, and using the minimum estimated Q-value of two target networks to calculate the Bellman equation:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_\phi(s')) \quad (5)$$

where y is the update target for Q_{θ_1} and Q_{θ_2} , and π_ϕ is the policy network.

Using clipped Double Q-Learning, the value estimation of target networks will not bring excessive estimation errors to the Q-Learning target. However, this update rule may cause underestimation. Unlike actions with overestimated Q values, actions with underestimated Q values will not be explicitly updated.

Here is proof of the underestimation of the clipped Double Q-Learning, the two outputs values (Q_1 and Q_2) of the Q network, and that Q_i^* is the real Q value. The errors brought by the fitting of the two networks can be defined as:

$$Y^i = Q_i(s', \pi_\phi(s')) - Q^*(s', \pi_\phi(s')) \quad (6)$$

Assuming errors Y^1 and Y^2 are two independent random variables with nontrivial zero-mean distributions, the minimization operation of the TD3 algorithm introduces an error. The error denoted as D is as follows:

$$D = r + \gamma \min_{i=1,2} Q_i(s', \pi_\phi(s')) - (r + \gamma Q^*(s', \pi_\phi(s'))) = \gamma \min(Y^1, Y^2) \quad (7)$$

where r is the reward and γ is the discount factor. Let Z be minimum value between Y^1 , Y^2 , and we have:

$$E[Z] = E[Y^1] + E[Y^2] - E[|Y^1 - Y^2|]/2 \quad (8)$$

Obviously, there is $E(Y^1) = 0$, $E(Y^2) = 0$ and $E[|Y^1 - Y^2|] \geq 0$. Therefore, we can deduce that $E(D) \leq 0$, which means that the clipped Double Q-Learning usually has underestimation.

Specifically, in order to have a more intuitive perception of underestimation, we can assume that the error follows an independent and uniform distribution in the range $[-\varepsilon, \varepsilon]$. Since the estimated Q values and the true Q values have error Y^i that follows a uniform distribution, the probability density function and the probability distribution are represented by $f(x)$ and $P(Y^i > x)$, respectively:

$$f(x) = \begin{cases} \frac{1}{2\varepsilon}, & x \in [-\varepsilon, \varepsilon] \\ 0, & x \in \text{else} \end{cases} \quad (9)$$

$$P(Y^i > x) = \begin{cases} 1, & x \leq -\varepsilon \\ \frac{\varepsilon - x}{2\varepsilon}, & x \in (-\varepsilon, \varepsilon) \\ 0, & x \geq \varepsilon \end{cases}$$

Also, since Y^1 and Y^2 are independent, it follows that:

$$P(\min(Y^1, Y^2) > x) = \prod_{i=1,2} P(Y^i > x) \quad (10)$$

$$= \begin{cases} 1, & x \leq -\varepsilon \\ \left(\frac{\varepsilon - x}{2\varepsilon}\right)^2, & x \in (-\varepsilon, \varepsilon) \\ 0, & x \geq \varepsilon \end{cases}$$

Let Z be the minimum value between Y^1 , Y^2 . The probability density function and cumulative distribution function of Z are represented as $f_Z(x)$ and $F_Z(x)$, respectively:

$$f_Z(x) = \begin{cases} \frac{\varepsilon - x}{2\varepsilon^2}, & x \in [-\varepsilon, \varepsilon] \\ 0, & x \in \text{else} \end{cases} \quad (11)$$

$$F_Z(x) = \begin{cases} 0, & x \leq -\varepsilon \\ 1 - \left(\frac{\varepsilon - x}{2\varepsilon}\right)^2, & x \in (-\varepsilon, \varepsilon) \\ 1, & x \geq \varepsilon \end{cases}$$

Therefore, the expected value of Z can be calculated as follows:

$$E[Z] = \int_{-\varepsilon}^{\varepsilon} x f_Z(x) dx = \int_{-\varepsilon}^{\varepsilon} x \left(\frac{\varepsilon - x}{2\varepsilon^2} \right) dx = -\frac{1}{3}\varepsilon \quad (12)$$

Thus, it has been theoretically proven that the Q network update in the TD3 algorithm will result in an underestimation error for the true Q values.

2.4. Multi-Agent Twin Delayed Deep Deterministic Policy Gradient (Matd3)

The training modes of multi-agent deep reinforcement learning include Decentralized Training Decentralized Execution (DTDE), Centralized Training Centralized Execution (CTCE), and Centralized Training Decentralized Execution (CTDE), as shown in Figure 2.

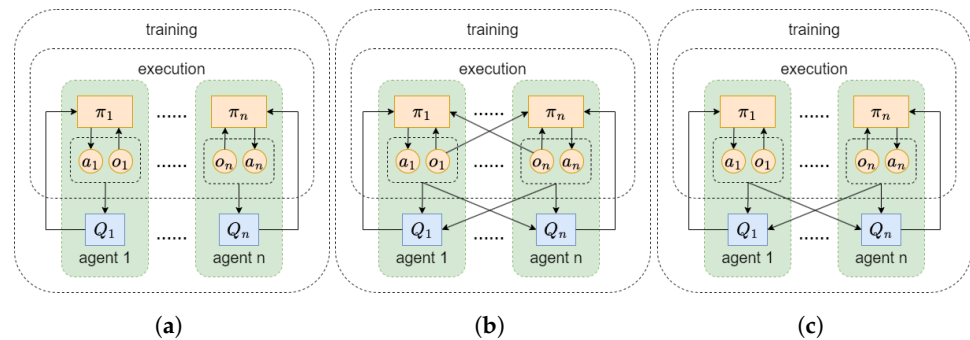


Figure 2. Multi-agent reinforcement learning training modes: (a) DTDE, each agent i makes decisions based on its own observation information, and each agent is relatively independent, with no information sharing between agents; (b) CTCE, which receives observation information from all agents through a centralized policy function and outputs the joint probability distribution of all agent action spaces; (c) CTDE, during the execution process of agents, each agent inputs its own local observation information into the policy function and outputs the probability distribution of the action space. However, during the training process, the agent can obtain additional information from other agents.

The MATD3 [10] algorithm extends the single-agent TD3 algorithm to a multi-agent environment, adopting the centralized training decentralized execution reinforcement learning framework. During the behavior execution process of the agents, each agent inputs their local observation information into the policy network, outputs deterministic actions, and during the training process, the agents can obtain additional information from other agents to evaluate their own actions.

The problem of Q network estimation bias will have a more severe impact when transferring policy gradient algorithms with Q-functions to a multi-agent scene, because the estimation bias of the Q network can increase the variance of the neural network gradient and affect the update process of the policy network neural network. In a multi-agent environment, as the number of agents increases, the difficulty of updating the policy function in the correct direction increases exponentially. The MATD3 algorithm simply extends the TD3 algorithm to a multi-agent environment, and there are still problems with the underestimation of Q values. In a multi-agent environment, this bias will have a greater impact on the agent's strategy learning.

3. Multi-Agent Mutual Twin Delayed Deep Deterministic Policy Gradient (M2ATD3)

In multi-agent scenarios, single Q network algorithms such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) have problems of overestimation. However, twin Q network's MATD3 algorithm, while resolving the overestimation of Q value, inevitably leads to underestimation. The Multi-Agent Three Delayed Deep Deterministic Policy Gradient (MATHD3) algorithm is proposed to create a third Q network. The overestimated results from it, and the underestimated results from the minimization of the first two Q networks, are added together by adjusting the weight hyperparameter α , and an unbiased estimate of the actual Q value can be obtained. In multi-agent algorithms, we denote the

observation received at runtime by agent i as o_i , and the full state information as s , from which the observations o_i are derived. The Q network update target y_i of the MATHD3 algorithm is as follows:

$$y_i = r_i + \gamma \alpha \min_{j=1,2} Q_{i,j}^{\pi'}(s', a'_1, \dots, a'_n) + \gamma(1 - \alpha) Q_{i,3}^{\pi'}(s', a'_1, \dots, a'_n) \quad (13)$$

where a'_i is obtained by adding noise ϵ to the output of the target policy network $\pi_{\phi'_i}$ of agent i based on its observation o'_i , that is: $a'_i = \pi_{\phi'_i}(o'_i) + \epsilon$.

While the MATHD3 algorithm effectively reduces bias in Q value estimates, it requires introducing a new Q network, leading to an increase in model parameters and increased model training difficulty.

Psychological studies show that humans exhibit overconfidence, typically estimating themselves beyond their actual abilities [11]. Introducing external evaluations can mitigate this phenomenon. Weinstein suggests that people are unrealistically optimistic because they focus on factors that improve their own chances of achieving desirable outcomes and fail to realize that others may have just as many factors in their favor [12]. Inspired by this, we propose the Multi-Agent Mutual Twin Delayed Deep Deterministic Policy Gradient (M2ATD3) algorithm. The training mode of M2ATD3 algorithm is shown in Figure 3.

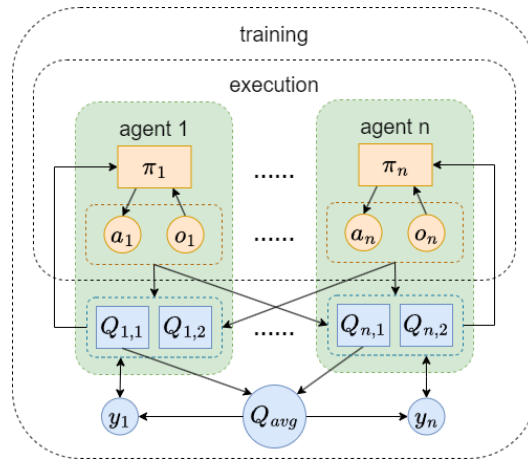


Figure 3. The training mode of M2ATD3. In this figure, rectangles represent networks, circles represent values, and dashed boxes represent sets or processes. Under the CTDE framework, each agent, equipped with Q networks of the same structure, can obtain shared information and make estimates of the current action-state value. The Q network of other agents can act as external critics, estimating the Q value of agent i and taking the average, denoted as the Q_{avg} . Agent i obtains the update objective y_i of the Q network through weighted sum of $\min(Q_{i,1}, Q_{i,2})$ and Q_{avg} .

Single Q networks, affected by noise, always overestimate Q values. Averaging the estimated Q values from each agent's Q network results in a slightly overestimated mutual Q value estimate. The mutual estimate of the Q value and the minimum value of two Q network estimates for agent i are weighted together to obtain the Q network update target y_i of the M2ATD3 algorithm:

$$Q_{avg}^{\pi'}(s', a'_1, \dots, a'_n) = \frac{1}{n} \sum_{i=1, \dots, n} Q_{i,1}^{\pi'}(s', a'_1, \dots, a'_n) \quad (14)$$

$$y_i = r_i + \gamma \alpha \min_{j=1,2} Q_{i,j}^{\pi'}(s', a'_1, \dots, a'_n) + \gamma(1 - \alpha) Q_{avg}^{\pi'}(s', a'_1, \dots, a'_n)$$

The mutual estimate of the Q value is overestimated, and the minimization of the twin Q network will lead to underestimation. By adjusting the weight hyperparameter α , in theory, an unbiased estimate of the true Q value of agent i can be obtained. The M2ATD3 algorithm reduces the Q value estimation bias without needing to introduce

a new Q network. Moreover, the operation of taking the average of Q value estimates can also reduce the variance of Q value estimate errors [13], making the training process more stable.

We have maintained the delayed policy updates used in the TD3 algorithm. The TD3 algorithm proposes that the policy network should be updated at a lower frequency than the value network, to first minimize error before introducing a policy update. This method requires updating the policy and target networks only after a fixed number of updates d to the critic.

The M2ATD3 algorithm pseudo code can be found in Algorithm 1.

Algorithm 1 Multi-Agent Mutual Twin Delayed Deep Deterministic policy gradient

initialize Replay buffer D and network parameters

for $t = 1$ to T **do**

 Select action $a_i \sim \pi_i(o_i) + \epsilon$

 Execute action $a = (a_1, \dots, a_n)$ and get reward r_i , transition to next state s'

 Store the state transition tuple $(s, a, r_1, \dots, r_N, s')$ into D

$s \leftarrow s'$

for agent $i = 1$ to N **do**

 A sample batch of size S is sampled from $D \left(s^b, a^b, r^b, s'^b \right)$

$Q_{avg}^{\pi'} = \frac{1}{n} \sum_{i=1, \dots, n} Q_{i,1}^{\pi'} \left(s^b, a_1, \dots, a_n \right) \Big|_{a_k = \pi'_k(o_k^b) + \epsilon}$

$y^b = r_i^b + \gamma \alpha \min_{j=1,2} Q_{i,j}^{\pi'} \left(s^b, a_1, \dots, a_n \right) \Big|_{a_k = \pi'_k(o_k^b) + \epsilon} + \gamma(1 - \alpha) Q_{avg}^{\pi'}$

 Minimize the loss of two Q-functions, $j = 1, 2$:

$L(\theta_j) = \frac{1}{S} \sum_b \left(Q_{i,j}^{\pi} \left(s^b, a_1^b, \dots, a_n^b \right) - y^b \right)^2$

if $t \bmod d = 0$ (d is the policy delay update parameter) **then**

 update π_i :

$\nabla_{\phi_i} J \approx \frac{1}{S} \sum_b \nabla_{\phi} \pi_{\phi_i} \left(o_i^b \right) \nabla_{a_i} Q_{i,1}^{\pi} \left(s^b, a_1^b, \dots, \pi_{\phi_i} \left(o_i^b \right), \dots, a_n^b \right)$

 Update target network:

$\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$

$\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$

 where τ is the proportion of target network soft updates

end if

end for

end for

4. Multi-Agent Softmax Twin Delayed Deep Deterministic Policy Gradient (MASTD3)

4.1. Theory Related to Softmax Bellman Operation

The standard Bellman operator is defined as:

$$\mathcal{T}Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a') \quad (15)$$

We define the softmax Bellman operation symbol as:

$$\mathcal{T}_{\text{soft}} Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \frac{\exp(\tau Q(s', a'))}{\sum_{\bar{a}} \exp(\tau Q(s', \bar{a}))} Q(s', a') \quad (16)$$

The optimal action-value function $Q^*(s, a)$ outputs the maximum cumulative return that can be obtained after choosing action a at state s . Q^* is the fixed point of the standard Bellman operator $\mathcal{T}Q$; that is, starting from any starting Q_0 , we have:

$$\lim_{k \rightarrow \infty} \mathcal{T}^k Q_0 = Q^* \quad (17)$$

However, due to the noise of the function approximator, the maximization operation of the standard Bellman operator always selects the maximum Q value for each state. This makes the algorithm exceptionally sensitive to the overestimated Q value of the action, and the overestimated Q value will be explicitly updated, which leads the Q value to the overestimation problem.

Softmax operation is generally considered to result in suboptimal action-value function and interfere with the convergence of the Bellman operator. Zhao Song et al. [14] proved that the softmax Bellman operator can quickly converge to the standard Bellman operator, and for any (s, a) , when k approaches positive infinity, the difference between $\mathcal{T}_{\text{soft}}^k Q_0(s, a)$ and $Q^*(s, a)$ is bounded:

$$\begin{aligned} \limsup_{k \rightarrow \infty} \mathcal{T}_{\text{soft}}^k Q_0(s, a) &\leq Q^*(s, a) \\ \liminf_{k \rightarrow \infty} \mathcal{T}_{\text{soft}}^k Q_0(s, a) &\geq Q^*(s, a) - \frac{\gamma(m-1)}{(1-\gamma)} \max \left\{ \frac{1}{\tau+2}, \frac{2Q_{\max}}{1+\exp(\tau)} \right\} \end{aligned} \quad (18)$$

Through experiments, Zhao Song et al. discovered that the combination of softmax Bellman operator and Deep Q Network yields better results than the Double Q-learning algorithm. They proved that the softmax Bellman operator can reduce the high estimation bias of Q value and believe this is the key reason for the superior performance of this method.

4.2. Multi-Agent Softmax Operation

The method of utilizing softmax operator to reduce overestimation of Q value can also be applied in multi-agent deep reinforcement learning algorithms. However, as the number of agents increases, the action space of the task will increase exponentially. It is impractical to simply apply the softmax operator to every possible combination of multi-agent joint actions, and the estimated Q value given by the Q network for the joint action, which is rarely sampled, is unreliable.

Ling Pan and others proposed an approximate softmax operator in Regularized Softmax Deep Multi-Agent Q-Learning (RES) [15]. Firstly, find the joint action $\hat{a} = \arg \max_{\hat{a}} Q_{\text{tot}}(s, \hat{a})$ that makes the joint Q value the greatest. For the agent i , fix the actions of other agents $\hat{a}_{-i}$ and only change the action of the agent a to obtain the corresponding action subset $A_i = \{(a_i, \hat{a}_{-i}) \mid a_i \in A\}$. The approximate action space $\hat{A} = A_1 \cup \dots \cup A_n$ is obtained by combining the action subsets of all agents. The approximate softmax operator used in the RES algorithm is to use this approximate action space rather than the complete action space for Q value softmax operation calculation. Ling Pan and others proved that the difference brought by the calculation using the approximate action space and the complete action space is bounded, and because the approximate softmax operator reduces the dependence on unreliable joint Q values, this method performs better than the softmax operator using the complete action space.

Inspired by the Softmax Deep Double Deterministic Policy Gradients (SD3) [16] and RES, this paper applies the softmax operator to multi-agent algorithms and proposes the MASTD3 algorithm. Firstly, the MASTD3 algorithm further reduces the action space used for Q value softmax operation calculation in the choice of action space.

Suppose there are N agents in a multi-agent task, each agent action space is of size K , the complete action space includes all permutations and combinations of the actions, and the softmax operation requires computing K to the power of N Q values. The approximate action space proposed in the RES algorithm, while traversing the action space of a single agent, will fix the optimal action of the other agents, and then merge each agent's sub action space as the approximate action space, the softmax operation needs to calculate K times N Q values. This article believes, within the structure of the MASTD3 algorithm, the Q network of the agent i can make a reliable estimate of the $(s, a) \mid a \in A_i$ state-action pair, but the estimate of the $(s, a) \mid a \in A_{-i}$ state-action pair formed through changes in the actions of other agents is not accurate. Therefore, in the simple softmax operator proposed in this

article, when agent i performs the softmax operation calculation on the Q value, only the sub-action space A_i containing all of its own actions, while the actions of other agents are fixed, is considered, and each agent only needs to calculate Q values K times:

$$\text{softmax}_{\tau,i}(Q_i(s, \cdot)) = \sum_{a \in A_i} \frac{\exp(\tau Q(s, a))}{\sum_{a' \in A_i} \exp(\tau Q(s, a'))} Q_i(s, a) \quad (19)$$

where τ is the temperature parameter in the softmax operation, controls the level of exploration versus exploitation in decision-making.

The softmax operator will lead to underestimation of the Q value, and the high estimate Q value obtained by using the twin Q network maximization operation to perform the softmax operation can alleviate the impact of underestimation. Therefore, the Q network update target y_i of the MASTD3 algorithm is as follows:

$$A'_i = \{(a'_i, \pi(s'_{-i})) \mid a'_i \in A\},$$

$$y_i = r_i + \gamma \text{softmax}_{\tau,i} \left(\max_{j=1,2} Q_{i,j}^{\pi}(s', a') \mid a' \in A'_i \right) \quad (20)$$

5. The Multi-Agent Particle Environment Experiment

The multi-agent particle environment was used as the experimental environment in this study, proposed by Lowe et al. [17], and used for the verification of MADDPG algorithm. The multi-agent particle environment is a recognized multi-agent reinforcement learning environment [10,14,17], including different tasks, covering competition, cooperation, and communication scenarios in multi-agent problems. The particle environment is composed of a two-dimensional continuous state space, in which agents move to fulfill specific tasks. The state information includes the agent's coordinates, direction, speed, etc. The shape of the agent is spherical, and actual collision between agents is simulated through rigid body collision. Figure 4 displays the cooperative communication task and the cooperative navigation task for the particle experiment.

Cooperative communication task: This task involves two types of cooperative agents who play the roles of speaker and listener, respectively, and there are three landmarks with different colors in the environment. In each round of the game, the listener's objective is to reach the designated color landmark, and the reward obtained is inversely proportional to the distance from the correct landmark. The listener can obtain the relative position and color information of the landmarks, but they cannot identify the target landmark on their own. The observation information for the speaker is the color information of the target landmark, and they send messages at every moment to help the listener reach the designated landmark.

Cooperative navigation task: This task involves N cooperative types of agents and N landmarks. The aim of the task is for the N agents to move to different N landmarks, respectively, while at the same time avoiding collisions with other agents. Each agent's observation information comprises its own and the landmarks' relative position information and the relative position information with other agents. The reward function is calculated based on the closeness of each agent to its landmark—the closer it is, the higher the reward. Additionally, each agent actually occupies a certain physical volume and will be punished when it collides with other agents.

Under the multi-agent particle environment, the M2ATD3 algorithm proposed in this paper was compared with the MASTD3 algorithm, the previous MATD3 algorithm, and the MADDPG algorithm. Below is a presentation of the hyperparameter settings for this study. To ensure the fairness of the experiment, general parameters were set uniformly, learning rate $\alpha = 0.01$, batch sample size $N = 1024$. All algorithms share the same neural network structure, i.e., two fully connected layers with a width of 64 serving as the hidden layer of the policy network and Q network. The activation function of the neural network used Rectified Linear Units (Relu), and all experiments employed Adam as the optimizer of the neural network.

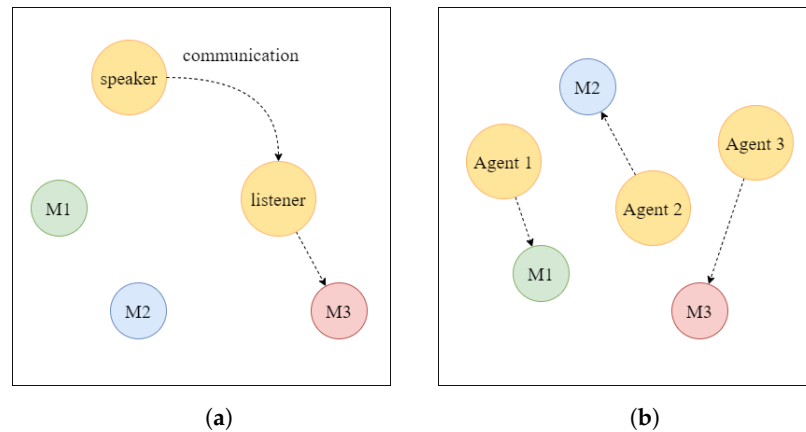


Figure 4. The Multi-Agent Particle Environment: (a) Cooperation Communication Task. In this example, the speaker needs to inform the listener that the target landmark is red landmark M3, and the listener needs to approach the correct landmark based on the speaker’s information. (b) Cooperation Navigation Task. In this example, agent 1 needs to approach green landmark M1, agent 2 needs to approach blue landmark M2, agent 3 needs to approach red landmark M3, and they should try to avoid colliding with each other while moving.

5.1. Q Value Estimation Bias Experiment

This article conducted a Q-value estimation bias experiment based on a cooperative navigation task in the particulate environment. By sampling from the experience replay pool to obtain samples of actions, states, and reward information to calculate the real Q-value and the estimated Q-value. In state s and action a , the average of 200 rounds of 100 steps’ cumulative rewards calculation is taken as the estimated value of the real Q-value. The comparison of the approximate value obtained through the Q network output with the real Q-value can reflect the situation of Q-function estimation bias of different algorithms.

As shown in Figure 5, the lines marked with crosses represent the Q-value estimated by the neural network, and the lines marked with triangles represent the real Q-value. In the MADDPG algorithm, the blue line with the triangle logo’s Q-value is higher than the orange line, implying that the MADDPG algorithm has an over-estimation, which is more apparent in the early training; under the MATD3 algorithm, the blue line with the triangle logo’s Q-value is lower than the orange line, indicating that the MATD3 algorithm actually has an under-estimation. Under the M2ATD3, MASTD3, and other algorithms, the blue line with the triangle logo’s Q-value is lower than the orange line, which means that M2ATD3, MASTD3, and other algorithms actually have an under-estimation, and the degree of underestimation is similar to that of the MATD3 algorithm, and it can be observed that the real Q-value of MATD3, M2ATD3, MASTD3, and other algorithms are higher than the real Q-value of the MADDPG algorithm, reflecting from the side that these three kinds of algorithms’ policies are better.

The Q-value estimation bias comparative experiment can qualitatively reflect whether different algorithms have overestimation or underestimation. However, due to the complexity, randomness, and instability of multi-agent games, it is difficult to accurately calculate the true Q-values. Therefore, it is difficult for us to reliably compare the degree of Q-value underestimation of MATD3, M2ATD3, and MASTD3 algorithms, and further research is needed through performance experiments.

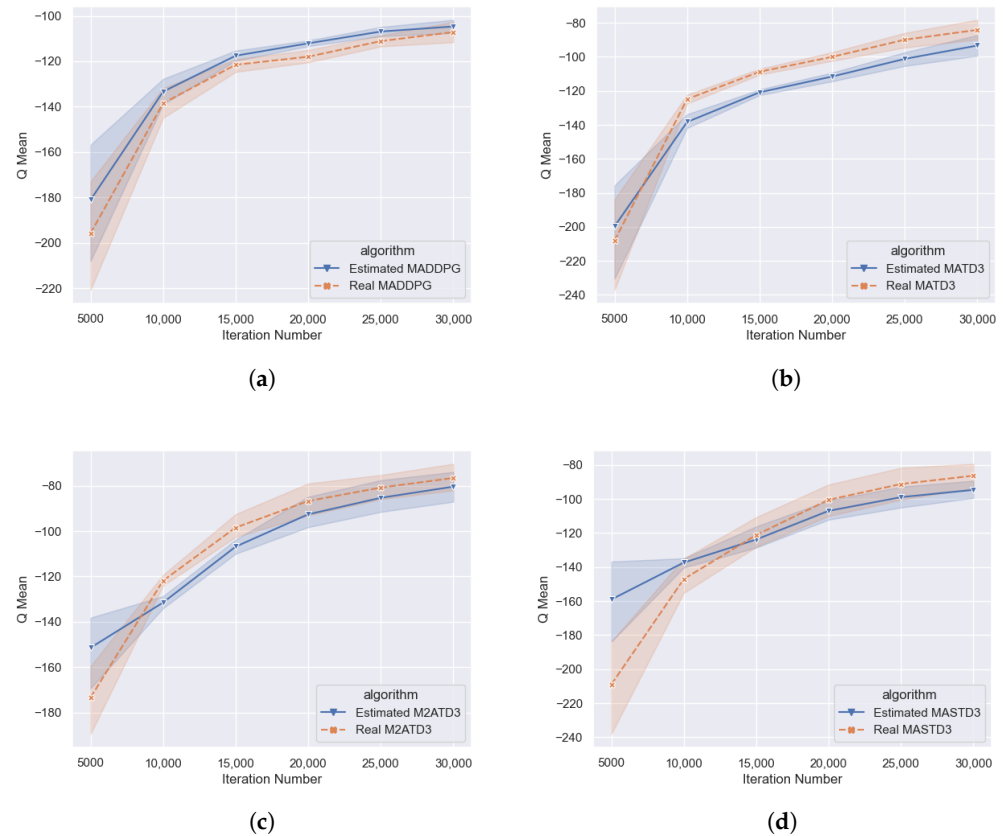


Figure 5. Q-value estimation bias comparative experiment based on cooperative navigation task in MPE. The Q-values estimated by the Q-network and the true Q-values are shown. The results are averaged across 5 runs and 95% CIs of the mean are shown for the estimated values. (a) DDPG, (b) MATD3, (c) M2ATD3, (d) MASTD3.

5.2. Algorithm Performance Experiment

In this section, the M2ATD3 algorithm and the MASTD3 algorithm are verified with the cooperative communication task and the cooperative navigation task in the particulate environment, and compared with the MADDPG algorithm and the MATD3 algorithm. All the experiments were trained for 60,000 steps. The experiment results are as shown in Figure 6.

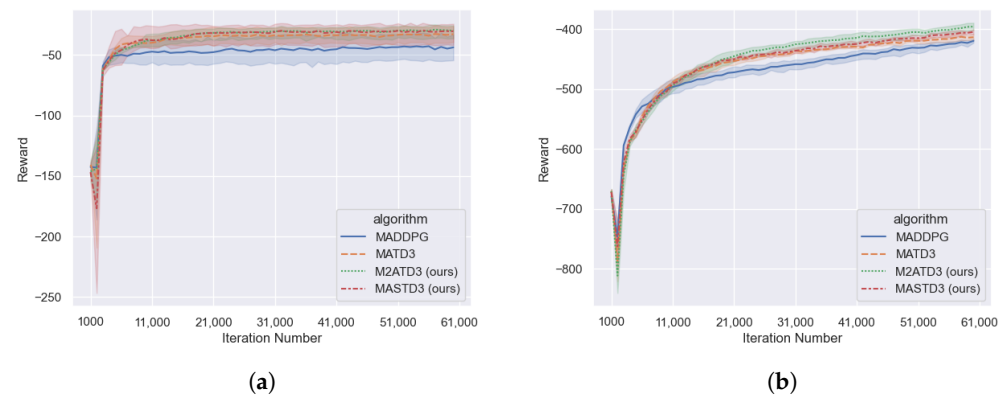


Figure 6. Comparative experiments using different Q-based multi-agent algorithms in MPE. Shown is the mean episodic reward over the last 1000 episodes, shaded areas are the 95% confidence intervals of the mean, averaged across 20 trials. (a) Cooperative Communication Task, (b) Cooperative Navigation Task.

As shown in Figure 6, in the Cooperative Communication Task and the Cooperative Navigation Task, the performance of the MATD3 algorithm is superior to the MADDPG algorithm, and both the M2ATD3 algorithm and the MASTD3 algorithm have achieved higher total rewards than the MATD3 algorithm.

As shown in Equation (14), hyperparameters α determine the proportion of the clipped double Q value to the average Q value when calculating the updated target value y_i , and select the appropriate hyperparameter α being able to balance the impact of underestimation of clipped double Q and overestimation of average Q.

As shown in Figure 7, in the cooperative communication task and the cooperative navigation task, the M2ATD3 algorithm receives lower rewards when the α is set to 0.2 or 0.8, indicating that the low estimation of Q value caused by the clipped Double Q-Learning or the high estimation of Q value caused by the average Q value estimates will have a negative impact on the algorithm. However, the M2ATD3 algorithm receives the highest reward when the α is set to 0.5, indicating that by balancing the two Q-value estimation methods, reducing the bias in Q-value estimation can improve the performance of the algorithm.

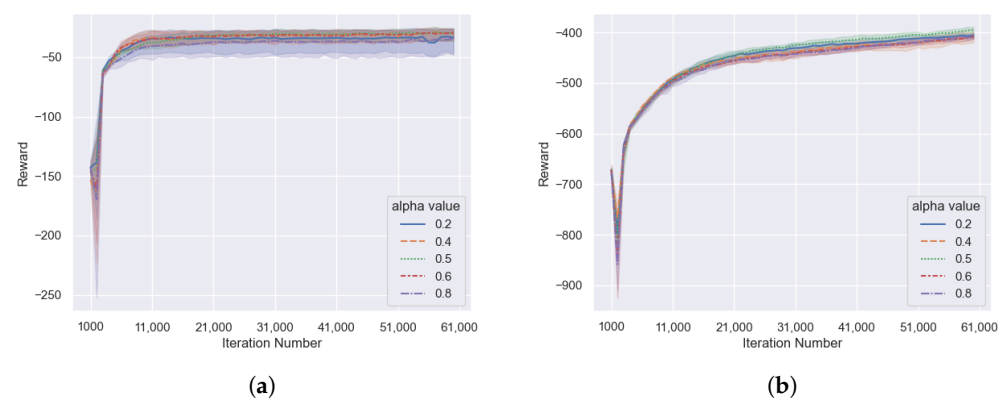


Figure 7. Comparative experiments using different hyperparameter α value on the M2ATD3 algorithm in MPE. Shown is the mean episodic reward over the last 1000 episodes, shaded areas are the 95% confidence intervals of the mean, averaged across 20 trials. (a) Cooperative Communication Task, (b) Cooperative Navigation Task.

6. Multi-Agent Tank Environment

Games have long been benchmarks and testbeds for AI research. Yunlong Lu et al. [18] held two AI competitions of Official International Mahjong, and claim that Mahjong can be a new benchmark for AI research. In order to further investigate the performance of M2ATD3 and MASTD3 algorithms, we searched for existing reinforcement learning game environments. The Atari gaming environment [19] is rich in content, but it is mainly a single-agent gaming environment; StarCraft [20] and Honor of Kings [21] are multi-agent environments, but the game state space is too large, the game randomness is too much, and the game running mechanism is protected. This places high demands on training resources and makes it difficult to set specific challenge scenarios according to research needs. Therefore, this paper referred to the classic “Tank Battle” game content and developed a multi-agent tank environment and its multi-agent reinforcement learning framework based on pygame and pytorch. This experimental environment can reflect the performance of different algorithms in real games, with suitable training scales, and can easily create different types of challenge scenarios according to research needs.

6.1. Introduction to the Multi-Agent Tank Environment

The Multi-agent Tank Environment is a game that simulates a tank battle scenario. As shown in Figure 8, the environment is divided into two types: the scoring task and the battle task. In the scoring task, the goal is for the AI-controlled tanks to collide with as

many scoring points as possible within a certain number of steps. In the battle task, the two teams of tanks win by firing bullets to destroy enemy tanks or the enemy base.

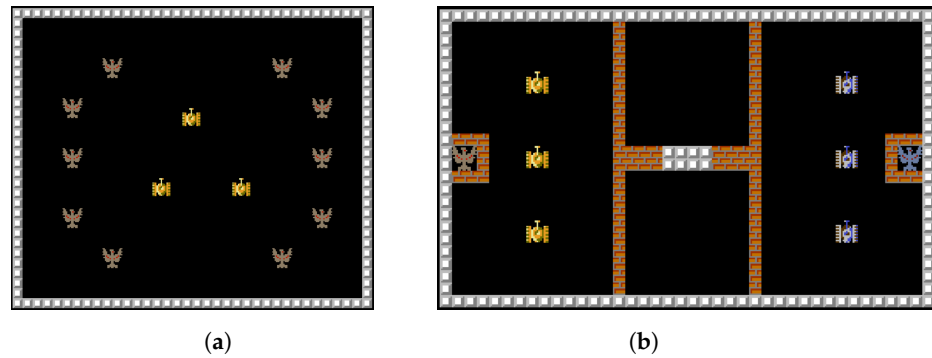


Figure 8. Multi-agent Tank Environment. (a) Scoring task, (b) battle task.

Common units in the game include: tanks, which have a certain amount of hit points (HP). When hit by a bullet, the tank will lose HP, and when the HP drops to 0, it is considered destroyed. Tanks can move and shoot. The shooting action will fire a bullet in the current direction of the tank. The bullet will be destroyed when it collides with an object or flies out of the map boundary. Bases also have a certain amount of HP. When hit by a bullet, the base will lose HP, and when the HP drops to 0, it is considered destroyed. The base cannot move. There are brick walls that can be destroyed by bullets, and stone walls that cannot be destroyed by bullets. The game map can be created and edited using *xlsx* format files.

Each agent in the game can obtain local observations, which include unit features and image features. The unit features are given in the order of friendly base, friendly tank, enemy base, and enemy tank, including each unit's survival, HP, absolute position, relative position, orientation, etc. These features are normalized. The image features use a six-channel binary image to represent whether there are stone walls, brick walls, friendly units, enemy units, friendly bullets, and enemy bullets within a 25×25 range near the agent. If the corresponding unit exists, the value is assigned to 1; if not, the value is assigned to 0. By using the six-channel binary image, the agent's complete nearby situation can be depicted. Figure 9 shows an example of graphical features of multi-agent tank environments. If the positions where the channel value is 1 are drawn as the channel index, and the positions where the value is 0 are not drawn, a complete image feature can be drawn as shown in Figure 9b.

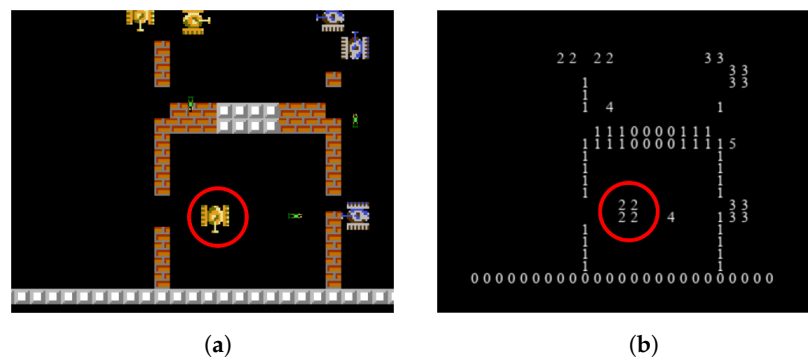


Figure 9. Example of image features in Multi-agent Tank Environment. (a) Original image, (b) image features. Draw the positions with a value of 1 for each channel as the channel index, and do not draw the positions with a value of 0 to obtain a diagram of 6 channel combinations.

The deterministic policy of the reinforcement learning algorithm enables the agent to make decisions in a continuous action space, which is more intuitive in video games. For the deterministic policy of the reinforcement learning algorithm, we design the agent's

action space as shown in Figure 10. We received inspiration from the way characters are controlled by the joystick in video games to design our action space for moving, as shown in Figure 10a, where we use a 2D action to control the tank's movement. Each movement action ranges from -1 to 1 , corresponding to the joystick's movement from left to right or from bottom to top. We determine the direction of the tank's movement based on the maximum absolute value of the action in the horizontal or vertical direction. If the absolute value of the action in both the horizontal and vertical directions is less than 0.3 , which is analogous to the joystick being in the blind area, the tank will not move. Similarly, we use a 1D action to control the tank's shooting, as shown in Figure 10b. The shooting action ranges from -1 to 1 . When the value of the shooting action is greater than 0 , the tank will shoot; otherwise, it will not shoot. The tank itself has a shooting cooldown. During the shooting cooldown, even if the agent issues a shooting action, the tank still cannot fire.

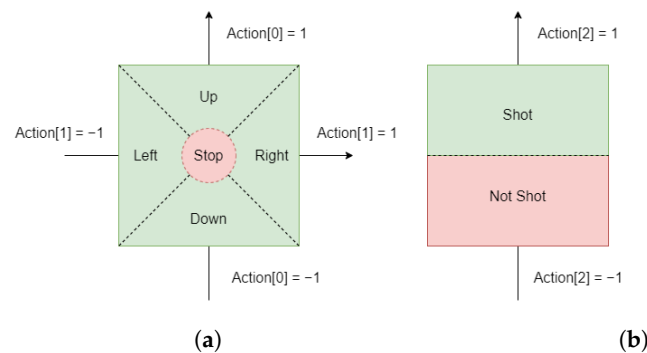


Figure 10. Example of action space in Multi-agent Tank Environment. (a) Move action, (b) shot action.

6.2. Multi-Agent Tank Environment Experiment

The process of deep reinforcement learning training using the Multi-agent Tank Environment is as follows: First, a corresponding model is created for each agent based on the selected algorithm. The model obtains actions through the policy network and interacts with the Multi-agent Tank Environment. The sequence samples obtained from the interaction are sent to the sample pool. When the samples stored in the sample pool reach a certain number, the training of the policy network and Q network of the model will start, and the model parameters will be updated at certain step intervals.

All algorithms in the experiment use the same neural network structure. A two-layer full connection layer with a width of 64 is used to process the unit features, and a convolutional neural network (CNN) is used to process the image features. The processed unit features and image features are concatenated and used as the input of the policy network and Q network. A two-layer full connection layer with a width of 64 is used as the hidden layer of the policy network and Q network. The network structure is shown in Figure 11.

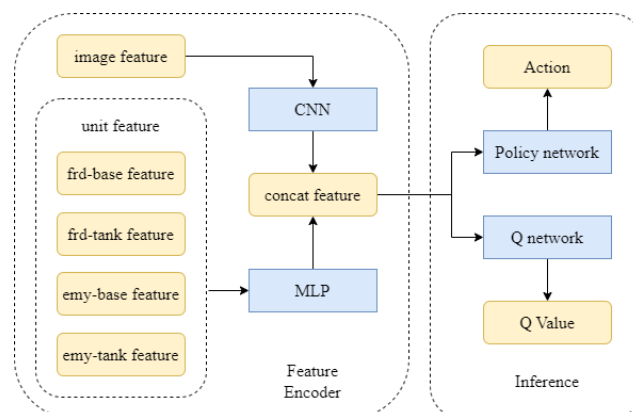


Figure 11. Neural network structure diagram of the Multi-agent Tank Environment experiment.

We conducted an experiment on the performance of multi-agent deep reinforcement learning algorithms in the scoring task. This task requires the AI to control three tanks to collide with as many scoring points as possible within 500 steps. There are a total of 10 optional scoring point birth positions on the map, and three positions are randomly selected each time to create scoring points. After the tank collides with a scoring point, the tank will receive a reward of +2, and a new scoring point will be created at a random other birth position. Since the reward for colliding with a scoring point is a sparse reward, we also set a dense reward according to the change in the distance from the tank to the nearest scoring point. In addition, in order to reduce the ineffective samples of the tank still moving towards the wall after colliding with the wall, we set an action mask, and set the action value in the direction of the wall to 0. The action mask effectively improves the efficiency of training.

The hyperparameters of the scoring task experiment are set as follows: learning rate $\alpha = 0.0005$, discount factor $\gamma = 0.95$, batch sample size $N = 512$. All algorithms use the same neural network structure, with the hidden layers of the policy network and Q network using two full connection layers with a width of 128. Training is conducted every 5000 steps using the current model to complete 20 evaluation rounds, and the average value of the evaluation round rewards is calculated. A total of 120,000 steps are trained, and the results are shown in Figure 12.

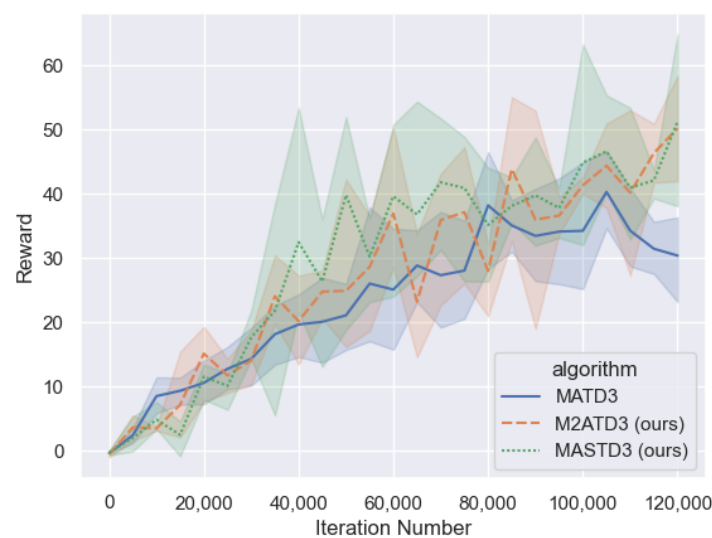


Figure 12. Comparative experiments using different Q-based multi-agent algorithms in the scoring task of UESTC-tank. Shown is the mean episodic reward over the last 50 evaluation episodes, shaded areas are the 95% confidence intervals of the mean, averaged across 20 trials.

The performance of the M2ATD3 algorithm and MASTD3 algorithm is better than that of the MATD3 algorithm, and they can achieve higher rewards than the MATD3 algorithm.

7. Limitations and Conclusions

7.1. Limitations

As the policy function of the agent is even more difficult to update in the correct gradient direction in the multi-agent scenario, the bias in the Q-function estimation in the multi-agent scenario can have a greater impact on policy learning compared to the single-agent scenario. Therefore, studying the bias in Q-function estimation in multi-agent scenarios is very meaningful. The M2ATD3 and MASTD3 algorithms presented in this study reduce the bias in Q-function estimation, but both algorithms have certain limitations in their application scenarios. In the M2ATD3 algorithm, each agent should have similar functions to ensure the reliability of the joint Q-value, while the application of MASTD3 in continuous action space tasks has not been deeply explored. Expanding the application

scenarios of these two methods to reduce Q-function estimation bias is a direction worth further study. In addition, the unit types and task types provided by the multi-agent tank environment are relatively simple, and the content of this environment can be further enriched, providing a good experimental environment for more experiments, such as multi-agent algorithm experiments, agent cooperation with human players experiments, and so on.

7.2. Conclusions

This paper primarily studies the Q-learning of agents based on deep reinforcement learning in multi-agent scenarios, and explores methods to reduce Q-value estimation bias in multi-agent scenarios, from multi-agent reinforcement learning algorithm theory to practical model deployment.

In the theory of multi-agent reinforcement learning algorithms, the focus is on solving the bias problem of the Q-function estimation of the MADDPG algorithm and its derivative algorithm, MATD3. In the MADDPG algorithm, the Q-function evaluates the policy function, so an unbiased estimation of the Q-function helps to learn better agent strategies. This research empirically demonstrates that the MADDPG algorithm overestimates the Q-function, while the MATD3 algorithm underestimates the Q-function, and theoretically proves that bias in the MATD3 algorithm. This is followed by the introduction of the M2ATD3 and MASTD3 algorithms presented in this study to solve this problem. The M2ATD3 algorithm combines the overestimation of the Q-value brought about by the joint estimation and the underestimation brought about by the minimization operation to reduce the bias in the Q-function estimation. The MASTD3 algorithm reduces the bias in Q-function estimation by combining softmax operation with maximization operation. The two algorithms have been tested in the multi-agent particle environment and the multi-agent tank environment, proving that they can indeed solve the estimation bias problem and that agents can learn better strategies to obtain higher rewards.

In the practice of multi-agent reinforcement learning, we constructed the UESTC-tank environment. This is a tank simulation game environment with multiple units and scenes, which can generate cooperative, adversarial, and mixed mission scenarios. We focused on its scalability when building the environment, so it can support unit editing, map editing, and rule editing. We hope it can become an open and effective benchmark in the field of multi-agent reinforcement learning.

Author Contributions: All authors contributed to the study conception and design. Methodology, software, validation, investigation, data analysis, and model verification: Z.L.; Investigation, resources, data curation, visualization, writing—review and editing, X.C.; Methodology, investigation, writing—original draft preparation, J.F.; Supervision, project administration, funding acquisition: N.X. and T.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key R&D Program of China (Grant No. 2022YFB3104600), and by Chengdu Science and Technology Project (2019-YF08-00285-GX), and by the National Natural Science Foundation of China under Grant No. 61976156, and by Intelligent Terminal Key Laboratory of SiChuan Province under Grant SCITLAB-30005.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset available on request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [[CrossRef](#)] [[PubMed](#)]
2. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
3. Hasselt, H. Double Q-learning. *Adv. Neural Inf. Process. Syst.* **2010**, *23*, 2613–2621.
4. van Hasselt, H. Insight in Reinforcement Learning. Formal Analysis and Empirical Evaluation of Temporal-Difference Algorithms. Ph.D. Thesis, Utrecht University, Utrecht, The Netherlands, 2011; ISBN 978-90-39354964.
5. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
6. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
7. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
8. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; PMLR; pp. 1928–1937.
9. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
10. Ackermann, J.; Gabler, V.; Osa, T.; Sugiyama, M. Reducing Overestimation Bias in Multi-Agent Domains Using Double Centralized Critics. *arXiv* **2019**, arXiv:1910.01465.
11. Polivy, J.; Herman, C.P. If at First You don't Succeed: False Hopes of Self-change. *Am. Psychol.* **2002**, *57*, 677–689. [[CrossRef](#)] [[PubMed](#)]
12. Weinstein, N.D. Unrealistic optimism about future life events. *J. Personal. Soc. Psychol.* **1980**, *39*, 806–820. [[CrossRef](#)]
13. Anschel, O.; Baram, N.; Shimkin, N. Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning. In Proceedings of the International Conference on Machine Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 176–185.
14. Song, Z.; Parr, R.; Carin, L. Revisiting the softmax bellman operator: New benefits and new perspective. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; pp. 5916–5925.
15. Pan, L.; Rashid, T.; Peng, B.; Huang, L.; Whiteson, S. Regularized softmax deep multi-agent q-learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 1365–1377.
16. Pan, L.; Cai, Q.; Huang, L. Softmax deep double deterministic policy gradients. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11767–11777.
17. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6379–6390.
18. Lu, Y.; Li, W.; Li, W. Official International Mahjong: A New Playground for AI Research. *Algorithms* **2023**, *16*, 235. [[CrossRef](#)]
19. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [[CrossRef](#)]
20. Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv* **2017**, arXiv:1708.04782.
21. Wei, H.; Chen, J.; Ji, X.; Qin, H.; Deng, M.; Li, S.; Wang, L.; Zhang, W.; Yu, Y.; Linc, L.; et al. Honor of kings arena: An environment for generalization in competitive reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 11881–11892.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.