IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 5, 2020, accepted May 23, 2020, date of publication May 27, 2020, date of current version June 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2997899

Adaptive Learning: A New Decentralized Reinforcement Learning Approach for Cooperative Multiagent Systems

MENG-LIN LI, SHAOFEI CHEN, AND JING CHEN
College of Intelligence Science and Technology, National University of Defence Technology, Changsha 410073, China
Corresponding authors: Shaofei Chen (chensf005@163.com) and Jing Chen (chenjing001@vip.sina.com)

This work was supported by the National Natural Science Foundation of China under Grant 61702528.

ABSTRACT Multiagent systems (MASs) have received extensive attention in a variety of domains, such as robotics and distributed control. This paper focuses on how independent learners (ILs, structures used in decentralized reinforcement learning) decide on their individual behaviors to achieve coherent joint behavior. To date, Reinforcement learning(RL) approaches for ILs have not guaranteed convergence to the optimal joint policy in scenarios in which communication is difficult. Especially in a decentralized algorithm, the proportion of credit for a single agent's action in a multiagent system is not distinguished, which can lead to miscoordination of joint actions. Therefore, it is highly significant to study the mechanisms of coordination between agents in MASs. Most previous coordination mechanisms have been carried out by modeling the communication mechanism and other agent policies. These methods are applicable only to a particular system, so such algorithms do not offer generalizability, especially when there are dozens or more agents. Therefore, this paper mainly focuses on the MAS contains more than a dozen agents. By combining the method of parallel computation, the experimental environment is closer to the application scene. By studying the paradigm of centralized training and decentralized execution(CTDE), a multi-agent reinforcement learning algorithm for implicit coordination based on TD error is proposed. The new algorithm can dynamically adjust the learning rate by deeply analyzing the dissonance problem in the matrix game and combining it with a multiagent environment. By adjusting the dynamic learning rate between agents, coordination of the agents' strategies can be achieved. Experimental results show that the proposed algorithm can effectively improve the coordination ability of a MAS. Moreover, the variance of the training results is more stable than that of the hysteretic Q learning(HQL) algorithm. Hence, the problem of miscoordination in a MAS can be avoided to some extent without additional communication. Our work provides a new way to solve the miscoordination problem for reinforcement learning algorithms in the scale of dozens or more number of agents. As a new IL structure algorithm, our results should be extended and further studied.

INDEX TERMS Reinforcement learning, multiagent system, intelligent control.

I. INTRODUCTION

In the past decade, multiagent systems (MASs) have attracted considerable attention in many fields, especially for intelligent multirobot systems, road traffic signal control, distributed system control [1], etc. Hence, MASs are very convenient for practical applications. Indeed, a decentralized MAS point of view offers several potential advantages, such as increased speed, scalability and robustness [2]. In this paper, we focus on the coordination mechanism in a fully cooperative multiagent reinforcement learning algorithm.

The associate editor coordinating the review of this manuscript and approving it for publication was Mostafa M. Fouda.

Recently, with the development of machine learning, a branch called reinforcement learning has become increasingly important in the field of control. In particular, in the control of physical models [3], reinforcement learning has reached amazing levels. Therefore, the researchers focus on how to use the advantage of the reinforcement learning algorithm to solve the challenges of the MAS.

To date, behavior coordination between cooperative agents has remained a difficult problem to solve in the MAS. In centralized learning, joint action learners (JALs) [4] and centralized learning strategies can theoretically fit the value function of all joint actions and find the optimal joint behavior. However, this approach also creates serious problems.

As the number of agents increases, the curse of dimensionality emerges. This makes such algorithms difficult to adapt to the actual requirements of current multiagent systems. A reinforcement learning algorithm with a structure based on independent learners (ILs) [4] can avoid the curse of dimensionality while satisfying current computing power requirements. Therefore, this decentralized structure has become the dominant structure in use today [5]. The fundamental goal in such a system is to fit a joint value function by coordinating the value functions maintained by all of the decentralized agents. In this context, one of the main problems to be solved for cooperative MASs is how to coordinate the behavior between agents such that the joint action can overcome the limitations of individual behaviors.

To this end, there are many methods of combining the centralized and decentralized approaches, such as QMIX [6] and QTran [7]. In essence, the goal of these methods is to fit a centralized joint Q-value by summing the local Q-values of the decentralized agents. These are improved methods based on value decomposition networks [8]. Similar to the loss of partial information, these methods somewhat alleviate the computational pressure imposed by centralized algorithms. However, in the face of a system with a large number of agents, these methods cannot completely solve the problem of excessive computation. Although, the paradigm of centralized training and decentralized execution(CTDE) has worked well in the small-scale MAS, such as PySC2 [9] and SC2LE [10]. In large scale MAS, explicit centralization training is bound to increase the computation and affect the practicability of the algorithm. Therefore, it is particularly important to weaken the centralization among agents while preserving coorperation. The CTDE paradigm with parallel algorithm proposed in this paper can effectively solve the computational difficulties caused by the curse of dimensionality. While weakening the centralization of the algorithm, it can effectively improve the coordination performance of agents in large-scale MAS.

Another solution for achieving such centralization in a MAS is to add a communication mechanism between agents, as in the methods proposed in [11], [12]. Indeed, this is an intuitive way to solve the problem of collaboration between agents. However, it is necessary to consider the possibility of channel obstruction in the real world. At the same time, because this kind of algorithm needs a special network structure, its generalizability is not strong. In addition, in the face of the increasing demand for a larger number of agents, it is particularly important to study how to improve the coordination of such algorithms. No agent communication mechanism is explicitly added in this article. Instead, by analyzing the joint-action information contained in TD error in each agent of a cooperative MAS, each agent can cooperate more effectively. Because we think that the TD error contains the influence of a single action on the joint action, which can be used as the evaluation standard of agent action reliability distribution. In this way, the generalization performance of the algorithm in large-scale system is guaranteed, and conditions are provided for the practical application of the algorithm.

To meet the requirements of large-scale and practicability of the algorithm in practice, this paper studies a reinforcement learning algorithm for a collaborative multiagent system consisting of ILs. Based on game theory, we propose a method of simplifying a multiagent system in section 3. Through this simplification, the reasons for the unsatisfactory results obtained by the existing algorithms with an IL structure are analyzed. By analyzing the causes of many kinds of miscoordination, we propose a standard dynamic evaluation function A that is similar to a membership function [13] in section 4. Using function A, agents can assess the effects of their actions on joint actions. We combine the CTDE paradigm with parallel computing and propose a new algorithm. By combining the training of the shared model and A function, we can weaken the centralization requirement and improve the coordination performance of the algorithm among agents. The structure of the new algorithm is more close to the current large-scale MAS application scenario in the experimental environment. Meanwhile, this enhances an agent's ability to identify the effects of its actions on the system. With no additional communication between agents, the improved performance of the proposed algorithm is verified by experiments.

Through our research results, we hope to provide a new way to coordinate the joint actions of multiple agents(Dozens of agents). Different from the perceptual cognition of enhanced communication mechanisms, we propose a mechanism of quantizing agent strategies to dynamically adjust the agent's actions on the basis of the joint action reliability. Our research provides a new way to avoid miscoordination in cooperative multiagent systems, which require high reliability in practical applications. At present, considerable research prospects still remain for quantitative evaluation criteria of this kind.

II. RELATED WORKS

In this section, related works addressing RL algorithms in cooperative MASs are reviewed, with an emphasis on research related to Q-learning [14] and Q-learning variants for ILs. The research focuses in this paper is put on the coordination of movement between agents. In this part, we summarize the algorithms to solve the agent coordination problem and their disadvantages.

So far, the multi-agent reinforcement learning algorithm can be divided into three paradigms according to the way of training and execution:1. Centralized training and centralized execution(CTCE); 2. Centralized training and decentralized execution(CTDE); 3. Decentralized training and execution(DTDE). We sorted out the recently popular algorithms according to this classification and select some important algorithms to detail.

In [15], the author directly applied a deep Q-network to the gym-pong [16] environment with two agents. The author discusses the tasks of implementing zero-sum games

with different return values under different types of reports and agent interactions. This laid the foundation for the use of the deep Q reinforcement learning (RL) algorithm in a multi-agent environment. Moreover, the two-agent environment can be simplified as a game matrix to be solved. As an early algorithm used in a multiagent environment, it was not discussed in relation to the ''curse of dimensionality'' faced with an increasing number of agents, nor did the author study whether a Q-learning algorithm can converge in an uncertain environment. Nevertheless, this study undeniably laid the foundation for the application of deep reinforcement learning in a multiagent environment and it was fallen in CTDE paradigm.

Another natural way to address cooperative MARL problem is the centralized approach, which views the MAS as a whole and solves it as a single-agent learning task [17]. These methods are also fallen in the CTCE paradigm. In such settings, existing reinforcement learning (RL)techniques can be leveraged to learn joint optimal policies based on agents joint observations and common rewards [18]. However, the centralized approach usually scales not well, since the joint action space of agents grows exponentially as the increase of the number of agents. Furthermore, partial observation limitation and communication constraints also necessitate the learning of decentralised policies, which condition only on the local action-observation history of each agent [19]. Commnet [20], and Bicnet [21] can be fallen in this paradigm. This algorithm is a good solution to the problem of full collaboration. However, with the increase of the number of agents required by the application of UAVs in MAS algorithm [22], this algorithm is faced with the shortcoming of poor practicability.

CEDT is a paradigm that combines the advantages of the first two paradigms. The MADDPG [23] is the basis of these algorithms. Thus, the MADDPG algorithm can simultaneously solve the multi-agent problem in the cooperative environment or competitive environment, and a mixed environment. Another classical algorithm under the CEDT paradigm called actor-critic. The actor-critic assumes that each agent has its independent critic network and actor network, and each agent has its independent reward function. The value decomposition network(VDN) [8] can be also fallen in this paradigm. At the same time, this method can obtain better results in MAS than the previous two traditional methods. The basic idea of the VDN is to train a federated Q network centrally, which is obtained by adding up the local Q networks of all agents. In this way, not only the problems caused by non-stationary environment can be dealt with by centralized training, but also the complex interrelationships between agents can be decoupled because the local model of each agent is learned. The QMIX algorithm [6] is the follow-up work of the VDN algorithm. Its motivation is to address the issue that the joint q-value decomposition of the VDN is just a simple sum, which will make the local Q function expression ability of the learned limited, and there is no way to capture the more complex interaction between agents. The QTRAN algorithm [7] is a further improvement of the VDN and the QMIX algorithm. The whole approximation process is divided into two steps: Firstly, the local Q function of the sum is obtained by using VDN as the approximation of joint Q function. Then the difference between the local Q function and the joint Q function is fitted. Paper theoretically derives a linear decomposing formation from Q_{tot} to each Q_i.

To date, Starcraft 2 is one of the main platforms for verifying algorithms as a complex environment that can control multi-agent microoperations. In this environment, another method that can achieve good results is q-value path decomposition (QPD) [24], which is also based on the idea of value decomposition. This algorithm leverage the integrated gradient attribution technique into deep MARL to directly decompose global Q-values along trajectory paths to assign credits for agents. Then, the COMA proposed in this paper [19] aims to solve the problem of multi-agent credit assignment in the decentralized partially observable Markov decision process(Dec-POMDP) problem, i.e., the problem of multi-agent credit assignment. However, the fully centralized critic of COMA suffers difficulty in evaluating global Q-values from the joint state-action space especially when there are more than a dozen agents, and it is hard to give an appropriate multiagent baseline. By summarizing the above algorithms, it is not difficult to see that the CEDT paradigm has achieved good results in multi-agent reinforcement learning. It is worth noting that the centralization conditions of these algorithms are very strong, and centralization in a real MASs may be unstable. Therefore, the CEDT paradigm algorithm that can be applied to the real environment should weaken the centralization condition appropriately.

With the expansion of the MAS, the instability of a multiagent environment has gradually become a major obstacle for such algorithms. In [25], the question of ''relative overgeneralization'' was raised. Relative overgeneralization occurs when a suboptimal Nash equilibrium in the joint space of actions is preferred over an optimal Nash equilibrium because each agent's action in that suboptimal equilibrium is individually a better choice when matched with arbitrary actions from the collaborating agents [25]. For instance, consider the continuous game depicted in Figure 1. The i and j axes represent the various actions that agents A_i and A_j may perform (we assume that the agents are performing deterministic actions), and the reward(i, j) axis represents the joint reward received by the agents from a given joint action (h_i, j_i). Joint action M has a higher reward than joint action N.

To date, the CTDE paradigm with good results has strong centralization conditions. At the same time, as the number of agents increases, this centralization brings heavy pressure on computing. On the other hand, the algorithm has to solve the problem of environmental instability introduced by decentralization. We want to use an algorithm that avoids algorithm instability and inherits the advantages of the CTDE paradigm. Therefore, we study the method of weakening the centralization in ILs to meet the large scale MAS. In this case, it is very important that the algorithm does not depend on

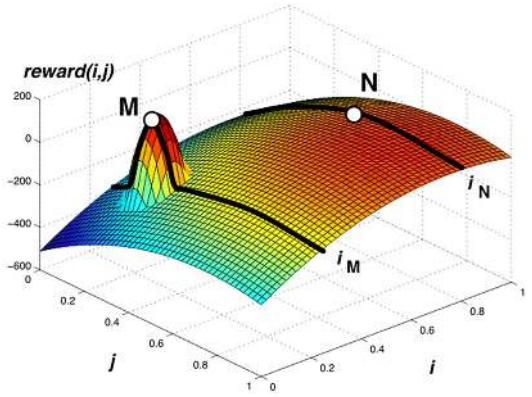


FIGURE 1. The relative overgeneralization pathology in multiagent learning. The axes i and j are the various actions that agents A_i and A_j may perform, and the axis reward(i, j) is the joint reward received by the agents from a given joint action $\langle i, j \rangle$. Higher rewards are better. Joint action M has a higher reward than joint action N. However, the average (or sum) of all possible rewards for action i_M, of agent A_i is lower than the average of all possible rewards for action i_N. To illustrate this, bold lines show the set of outcomes from pairing i_M or i_N instead with other possible actions in j [25].

the centralization of its own coordination performance. So we looked at the following two approaches.

An algorithm called hysteretic Q-learning was presented in [26]. This algorithm is a form of optimistic learning with a strong empirical track record in fully observable multiagent reinforcement learning. Originally introduced to prevent the overestimation of Q-values in stochastic games, hysteretic learners use two learning rates: a learning rate α for updates that increase the value estimate (Q-value) for a state-action pair and a smaller learning rate β for updates that decrease the Q-value [27]. Hysteretic learners' struggles in these domains have been attributed to the interdependency of the learning rate β with the exploration strategies of other agents [28].

Lenient learners offer an alternative to the hysteretic approach and have been empirically shown to converge towards superior policies in stochastic games with a small state space [25]. Lenient methods have received criticism in the past for the time they require to converge [25], the difficulty involved in selecting the correct hyperparameters, the additional overhead required for storing the temperature values, and the fact that they were originally proposed only for matrix games [28].

To date, many researchers have focused on improving the network structure [11], improving the agent communication mechanism [29] and modeling the adversarial strategy [30]. To meet the demand of large-scale MAS, this paper mainly analyzes how to improve the coordination performance between agents in the cluster environment (more than 10 agents) by improving the algorithm itself while preserving the traditional network structure and weaken centralization requirements.

III. THE GAMES AND MULTIAGENT REINFORCEMENT LEARNING

Research on learning algorithms in MASs is based on game theory and, in particular, on repeated games. In this section, we first establish this framework. Then, we simplify the problem of influencing the convergence results in a multiagent reinforcement learning environment to a problem of repeated matrix games. The feasibility and effectiveness of the new algorithm are explained by analyzing this problem.

A. THE GAMES AND THE MARKOV DECISION PROCESS(MDP)

1) THE GAMES DEFINITIONS AND REINFORCEMENT LEARNING

A *matrixgame*, also called a strategy game, is a multiagent, single-state framework. It can be defined as a tuple $\langle n, A_1, \ldots, A_n, R_1, \ldots, R_n \rangle$, where n is the number of players, A_i is the set of actions available to player i and A is the joint action space $A_1, \ldots, A_n$ [27].

Stochastic games (SGs) can be seen as the extension of matrix games to a multistate framework [31]. Specifically, each state of an SG can be viewed as a matrix game. SGs were examined first in the field of game theory and more recently in the field of multiagent RL [27]. A stochastic game is defined as a tuple $\langle n, S, A_1, \ldots, A_n, T, R_1, \ldots, R_n \rangle$. Unlike in repeated games, state transitions occur in stochastic games. Here, T is a transition function that defines the transition probabilities between states.

An important problem to be solved in reinforcement learning is the sequential execution of strategies in an environment with state transitions. Only static and random reward functions are allowed in standard games. At the same time, the standard game system cannot carry out state transition and there is no such concept. This is the core problem of the concept of the MDP. We used ILs for the MDP. An MDP can be represented as a tuple $(n, S, A_1, \cdots, A_n, R_1, \cdots R_n, T)$. Where n is the number of agents in the system. $S = \{s^1, \cdots, s^N\}$ represents a collection of system states. A_k is the action set of learner k. R_k is the agent's reward. T is the transition function.

The straightforward extension of centralized Q-learning to the MDP considers joint actions for the computation of Q-values. Thus, the update equation from a centralized perspective is:

$$Q(s, a_1, \cdots, a_n) \leftarrow (1-\alpha)\, Q(s, a_1, \cdots, a_n) + \alpha \left[r + \gamma \max_{a_1, \ldots, a_n} Q(s', a'_1, \cdots, a'_n) \right] \tag{1}$$

where s' is the new state, α is the learning rate and $\gamma \in [0,1]$ is the discount factor. This algorithm model is classified as an algorithm based on joint action learners (JALs). The problem with this kind of algorithm is that its spatial complexity greatly increases as the scale of the problem increases. This makes it difficult for the algorithm to cope with a large number of agents.

Another class of algorithms is based on independent learners (ILs). Such an algorithm is a decentralized algorithm.

In this framework, the Q-learning update equation is:

$$Q(s, a_i) \leftarrow (1-\alpha)\, Q(s, a_i) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right] \quad (2)$$

The Q_i tables for ILs are smaller. However, each agent has only a local view because it has no access to the actions of the other agents [27]. Since we wish to address applications involving a large number of agents, the algorithm structure used in this paper is based on ILs.

B. ENVIRONMENT SIMPLIFICATION

Reinforcement learning is a process in which an agent selects an action acting on the environment through an appropriate strategy and the environment provides a reinforcement signal as feedback to the agent to optimize its strategy [32]. In a MAS, the actions acting on the environment are joint actions between agents. Based on the selected action, the environment transitions to the next state and rewards the agent in accordance with the rationality of the selected action selection with respect to a set benchmark. To more intuitively describe the ideas applied in this paper, we consider two agents as an example. Under this simplification, the joint action returns can be thought of as a matrix. Unlike in repeated games operating under the same matrix, the state of the environment will change after the agents select actions. The structure of the whole game is similar to the tree structure shown in Figure 2.

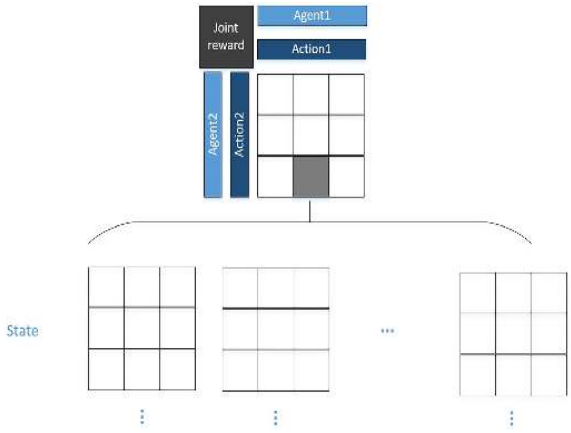


FIGURE 2. Simplify MAS environment and environment transfer examples. The gray box is a joint action selected by multiple agents. The environment rewards the agent after the action is performed. Due to environmental and observational uncertainties, the agent's selection of the same joint return will still result in different state transitions, resulting in a tree structure.

Indeed, standard games cannot be modeled in MAS. But it is not hard to see that the standard game is the basis of MDP. The reason that affects the performance of reinforcement learning in MAS is largely attributed to the results of the algorithm in the standard game. Therefore, improving the ability of the algorithm in the standard game becomes a sufficient condition to solve the poor performance of reinforcement learning in large-scale MAS. We summarized the ideas to solve the problem in the Figure 3:

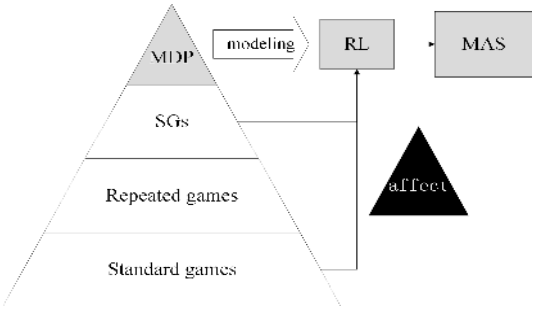


FIGURE 3. Effects of game theory on RL algorithm for MAS.

By analyzing the games and the MDP, we can see the essential features of this complex environment more clearly. By solving the problem of reinforcement learning in a matrix game, we wish to optimize the convergence performance of reinforcement learning in a large-scale environment.

In a repeated game, two agents perform repeated operations on the same game matrix. In the repetition process, the agents constantly improve their strategies in accordance with the benefits they receive. Although no state transitions occur, it is nevertheless difficult to learn joint actions due to the unknown nature of the opponent's strategy. This is similar to the IL-based structure that we use in our algorithm. In addition, this game is also similar to the incomplete observation of the environment in the MAS at the beginning of each game, which causes the action selection process to be completely dependent on the initially defined strategy. Therefore, we believe that the typical characteristics of repeated games may also be one of the keys to solving the coordination problem for ILs.

The ultimate goal of repeated games is to determine the optimal Nash equilibrium by collecting a large number of samples. Reinforcement learning follows a similar pattern. However, due to the occurrence of state transitions and the large state space, the convergence speed and quality of the algorithm must be improved for the case of finite-state sampling. Some environments will have multiple Nash equilibria. However, a traditional IL-structured algorithm can easily fall into suboptimal or worse Nash equilibria. If the convergence speed and quality of the reinforcement learning algorithm for matrix games can be improved, the problem of the difficulty of coordination in multiagent reinforcement learning may be alleviated to some extent.

Based on the current research, multiagent reinforcement learning algorithms still face the following key problems:

i) In a multiagent environment with heterogeneous agents, the existence of an optimal Nash equilibrium cannot be accurately prove.

ii) In a collaborative multiagent system, miscoordination may result from the existence of multiple suboptimal solutions, which will ultimately affect the quality of coordination.

iii) How to obtain a better and more stable convergence result in limited training time is also a problem to be solved.

To optimize the existing algorithm, we propose a method of simplifying the problem in this section. We wish to optimize the multiagent reinforcement learning algorithm by mitigating the miscoordination problems that arise in matrix games.

C. MISCOORDINATION ANALYSIS

There are many reasons for miscoordination. In multiagent reinforcement learning, the reasons for miscoordination may arise from the instability of the environment and the timeliness of the agents' strategies [33]. This scenario may even lead to miscoordination of the agents' strategies because of invalid experience gained through the experience playback mechanism [34]. In this paper, we focus on how to coordinate the system itself more effectively, while downplaying the impact of outdated experience on the system. In this analysis, we first discuss the causes of miscoordination mainly from the perspectives of strategy and reward. By analyzing these underlying causes, the structure of the basic algorithm can be optimized, and an improved method approach to this kind of algorithm can be provided, which is of great practical significance.

First, we discuss the problem that the algorithm does not optimally converge in a special case of the reward values in a repeated game: *relative overgeneralization* [25]. This problem is clearly illustrated in Figure 1. Relative overgeneralization occurs when a suboptimal Nash equilibrium in the joint space of actions is preferred over an optimal Nash equilibrium because each agent's action in the suboptimal equilibrium is individually a better choice when matched with arbitrary actions of the collaborating agents [25].

The most important reason for this is that when the average value of the suboptimal Nash equilibrium is significantly higher than that of the optimal Nash equilibrium, the algorithm will eventually tend to converge to the suboptimal Nash equilibrium [25]. This situation can be converted into a mathematical expression as follows. Agent A_i's portion of the joint action N (denoted by i_N) is individually preferred over its portion of the joint action M (denoted by i_M). That is, $quality(i_M) = Pjreward(i_M, j) < quality(i_N) = Pjreward(i_N, j)$. This problem will occur in an environment where there are more than 2 agents and the action space is large [25]. It can be considered to be widely present in cooperative, repeated games for independent learners. Table 1 presents an example of relative overgeneralization in a repeated game.

TABLE 1. The relative overgeneralization matrix.

Reward		Agent1		
		a	b	c
Agent2	a	10	0	0
	b	0	5	6
	c	0	6	7

In Table 1, we can see that $\langle a, a\rangle$ is the best joint move. However, if the agents sample their actions randomly and base their decisions on the average reward, action a will have the worst score, action c will have the best score, and b will be in the middle. Thus, the agents will tend to converge to $\langle c, c\rangle$.

The reason for this problem is that the reinforcement learning algorithm does not distinguish the effects of self-behavior on the reinforcement signal when updating the value function. In other words, the TD-error-based update policy is rarely fully used to distinguish the contribution of a single agent to a joint action. Sometimes, one agent's strategy is better but is forced to be adjusted because of the quality of the other agent's strategy. This is the reason for the convergence performance of the algorithm. In the IL-based framework, if measurement factors could be added to infer whether a worse joint action is chosen because of a particular agent's strategy, the algorithm would no longer be prone to completely random updates. This method would disrupt the updating strategy based on the calculation of mathematical expectations and make the algorithm more likely to converge to the optimal Nash equilibrium.

On the other hand, even with full cooperation, there is potential competition between agents. We illustrate this with the classic prisoners' dilemma from game theory. In a system of ILs, reinforcement learning relies on global rewards to update the agents' strategies. However, each agent maintains its own Q-value and adopts the maximization strategy to update its Q-value. This is analogous to the classic prisoners' dilemma problem in a repeated game. The payoff matrix can be regarded as representing the Q-values of the agents' respective optimization problems. The matrix is shown in Table 2.

TABLE 2. The prisoners' dilemma matrix.

Prisoners' Dilemma		Agent1			
		nice		rat	
Agent2	nice	Agent1 reward	Agent2 reward	Agent1 reward	Agent2 reward
		1	1	-5	0
	rat	Agent1 reward	Agent2 reward	Agent1 reward	Agent2 reward
		0	-5	-2	-2

When selecting strategies, the reinforcement learning algorithm will attempt to make the Q-value maintained by each agent as large as possible. However, as is the case in the prisoners' dilemma, even if both sides adopt what seems to be the best strategy, they can never achieve the Pareto-optimal solution. We consider cooperative actions that are numerically scattered in the range [0,1]. The total revenue of the game can be seen in Figure 4.

We can see in Figure 4 that the total benefits at the Nash point are significantly higher than those at the other points. However, due to the updating strategy of the IL agents, the optimization of the algorithm in the joint action rewards plane ultimately moves in the direction of a secondary advantage.

First, from the perspective of the mathematical expectation of the outcome of the game between the two sides, the profit expectation of choosing betrayal is higher than that

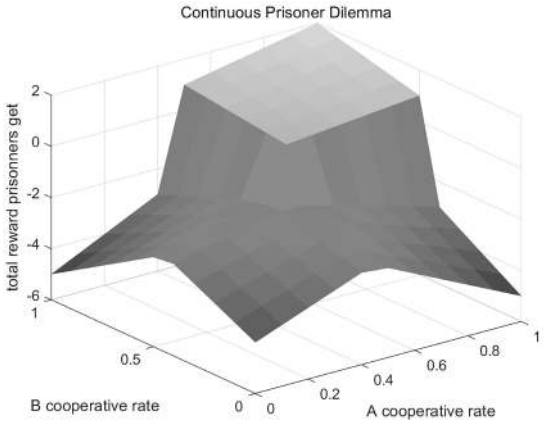


FIGURE 4. Total reward in the prisoners' dilemma.

of choosing cooperation. A theoretical assumption is adopted here, that is, if both sides know that the number of games is infinite, i.e., both sides of the business are endless, then both strategies will continue to choose cooperation, and the final game will be fixed at (1, 1), which is a Nash equilibrium. Unfortunately, however, our algorithm needs to find the correct solution in finite steps. Especially in multiagent reinforcement learning, the occurrence of state transitions limits the number of times that a particular state is traversed. The payoff in this problem can be thought of as the Q-value maintained by each agent among the ILs. Often, with a large number of samples, agents tend to be rational in their mathematical expectations. By optimizing their respective Q-value functions in the direction of their respective rational Q-values, the agents will cause the final joint return to converge to the secondary advantage at the intersection between the light and dark regions in Figure 4. This is why it is worthwhile to affect the algorithm's ultimate rewards.

A multiagent sequential decision problem is usually composed of such game matrices (tensors,when there are more than 3 players). This is an important reason for the final ''miscoordination'' of multiple agents in an IL-based structure. In other words, in a multiagent environment, each agent generates samples through its interactions with the environment, but the time available to reach ergodicity for each environment is limited. If the reinforcement learning algorithm can converge to the Nash equilibrium (even in the absence of the multiagent case, it can converge to a better solution) more quickly and stably, it can converge to the Nash equilibrium more quickly with fewer visits to the prisoner environment. Finally, it is possible to obtain better convergence results when the algorithm is applied to a MAS.

IV. ADAPTIVE REINFORCEMENT LEARNING

A. Q-LEARNING

In the traditional reinforcement learning algorithm, the value function of each state–action pair is recorded in the Q-table. Taking a single state as an example, the formula of the algorithm for updating the value function is shown as follows:

$$Q(s, a_i) \leftarrow (1-\alpha)\, Q(s, a_i) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) \right] \tag{3}$$

where a_i is the chosen action of agent i, r is the reward received, $Q_i(a_i)$ is the value of action a_i for agent i and $\alpha \in [0, 1]$ is the learning rate. It can be seen from the update strategy of the Q-learning algorithm that the updating and convergence performance of the algorithm mainly depends on the choice of the learning rate. Therefore, we consider whether the algorithm can adjust its learning rate based on its state environment to optimize its convergence performance. The idea of TD-error evaluation is that when the algorithm is negatively updated, if the overall error is small, it can be considered that the direction of agent a itself (X-axis) needs to be adjusted by means of a small gradient on the tensor. At the same time, other agents also use this principle as the basis for dynamic adjustment. Through this evaluation mechanism, we hope to increase the number of times the optimal Nash equilibrium is traversed. Finally, the convergence quality can be improved, thereby improving the performance of the algorithm.

1) HYSTERETIC Q-LEARNING

In a MAS with an IL-based structure, the reinforcement received by an agent relies on the actions chosen by the team. Thus, an agent can be punished because of a poor choice by another member of the team even if the agent itself has chosen an optimal action. Therefore, it would be preferable for each agent to be able to distinguish whether an instance of worse behavior was caused by its own action. A common idea for this purpose is distributed Q-learning, with the following equation:

$$\begin{aligned} \delta &\leftarrow r - Q_i(a_i) \\ Q_i(a_i) &\leftarrow \begin{cases} Q_i(a_i) + \alpha\delta & if\ \delta \geqslant 0 \\ Q_i(a_i) & else \end{cases} \end{aligned} \tag{4}$$

However, optimistic agents do not achieve coordination among multiple optimal joint actions [35]. Due to this key issue, [27] proposed a new update strategy for the algorithm. Agents must not be altogether blind to penalties, at the risk of remaining in suboptimal equilibria or continuing to miscoordinate on the same optimal joint action. The update equation is modified as follows:

$$\begin{aligned} \delta &\leftarrow r - Q_i(a_i) \\ Q_i(a_i) &\leftarrow \begin{cases} Q_i(a_i) + \alpha\delta & if\ \delta \geqslant 0 \\ Q_i(a_i) + \beta\delta & else \end{cases} \end{aligned} \tag{5}$$

where α and β are the rates of increase and decrease, respectively, for the Q-values. Hysteretic Q-learning (HQL) is a decentralized process: each IL builds its own Q-table, whose size is independent of the number of agents and linear in the number of the agent's own actions. We find that when the HQL algorithm is used, the results are greatly affected by the chosen rate of decrease, and the convergence speed

and outcome of the algorithm are both strongly dependent on the chosen value of β. This affects the generalizability of the algorithm. Moreover, the algorithm distinguishes only between positive and negative TD errors, which does not solve the problem of reliability allocation introduced by agents with negative updates. This will affect the convergence rate of the algorithm in a MAS. Therefore, we propose an adaptive Q-learning (AQL) method with an adaptive evaluation function to determine the learning rate and apply it in a deep Q-network.

2) LEINENT Q-LEARNING

Lenient Q-learning(LQL) was originally introduced by Potter and De Jong [36] and has later been applied to multi-agent learning as well [37]. It was designed to prevent relative overgeneralization [38], which occurs when agents gravitate towards a robust but sub-optimal joint policy due to noise induced by the mutual influence of each agent's exploration strategy on others' learning updates.

It has been shown that leniency can increase the likelihood of convergence towards the globally optimal solution in stateless coordination games for reinforcement learning agents. Lenient learners achieves this by effectively forgiving (ignoring) sub-optimal actions by teammates that lead to low rewards during the initial exploration phase. While initially adopting an optimistic disposition, the amount of leniency displayed is typically decayed each time a state-action pair is visited. As a result, the agents become less lenient over time for frequently visited state-action pairs while remaining optimistic within unexplored areas. This transition to average reward learners helps lenient agents avoid sub-optimal joint policies in environments that yield stochastic rewards

During training the frequency with which lenient reinforcement learning agents perform updates that result in lowering the Qvalue of a state action pair (s, a) is determined by leniency and temperature functions, $l(s_t, a_t)$ and $T_t(s_t, a_t)$ respectively [35]. The relation of the temperature function is one to one, with each state action pair being assigned a temperature value that is initially set to a defined maximum temperature value, before being decayed each time the pair is visited. The leniency function:

$$l(s_t, a_t) = 1 - e^{-K \times T_t(s_t, a_t)} \tag{6}$$

uses a constant K as a leniency moderation factor to determine how the temperature value affects the drop-off in lenience. Following the update, $T_t(s_t, a_t)$ is decayed using a discount factor $\beta \in [0,1]$ such that $T_{t+1}(s_t, a_t) = \beta T_t(s_t, a_t)$.

Given a TD-Error θ, where $\delta = Y_t - Q(s_t, a_t; \theta_t)$, leniency is applied to a Q-value update as follows:

$$Q(s_t, a_t) = \begin{cases} Q(s_t, a_t) + \alpha\delta & if\ \delta > 0\ or\ x > l(s_t, a_t) \\ Q(s_t, a_t) & if\ \delta \leqslant 0\ or\ x \leqslant l(s_t, a_t) \end{cases} \tag{7}$$

The random variable $x \sim U(0,1)$ is used to ensure that an update on a negative θ is executed with a probability $1 - l(s_t, a_t)$.

3) ADAPTIVE Q-LEARNING

In this paper, we are inspired by fuzzy control theory and the concept of a membership function [39]. Based on this, an algorithm that can dynamically adjust the learning rate is proposed, which is called the adaptive Q-learning algorithm. An appropriate evaluation function is designed that can be used in combination with the currently widely used double deep Q networks(DDQN) algorithm [40], [41]. The effectiveness of the algorithm is verified based on a single-state matrix game and the classic multi-intelligence predator-prey environment. In this section, we will introduce the principles and design concept of this algorithm.

Above, we mentioned 3 methods of reinforcement learning based on an IL architecture: distributed Q-learning, HQL and LQL. From the ideas for the improvement of distributed q-learning offered by HQL and LQL, it can be seen that an adjusted learning rate can have an impact on the results of the algorithm. The fundamental reason is that reinforcement learning is a learning algorithm based on reinforcement signals. However, in the IL framework, the reinforcement signal depends not simply on an agent's own actions but on the joint actions generated by multiple agents. When an agent receives a negative signal, the source of this negative signal does not necessarily lie in its own separately maintained Q-table or network. If the reinforcement signal is received without distinction in this respect, an agent may avoid choosing its optimal action and cause the system to fall into a poor solution because of the detrimental influence of other agents. The HQL algorithm is designed with two different learning rates and adjusts the reverse update strategy in the IL algorithm but does not distinguish this strategy as suggested above. In other words, the strategy for distinguishing between positive and negative learning causes the negative updates of the algorithm to be smaller than the positive updates, thereby reducing the convergence rate of the algorithm. From the results of [27], it can be seen that when the algorithm is negatively updated, a smaller learning rate is uniformly adopted, which will indeed cause such problems. This is undesirable in cases of increasing numbers of agents and more complex environments. As the complexity of the environment increases, the probability that each environment can be sufficiently sampled decreases. The new algorithm must converge to a better solution more quickly with a finite number of environment traversals. In other words, the algorithm should be able to escape from suboptimal Nash equilibrium faster and more efficiently. Only in this case will the algorithm have the possibility of approaching Pareto optimality.

B. ADAPTIVE FUNCTION AND AQL

In this paper, we propose a new algorithm to dynamically adjust the learning rate during a negative update, called adaptive Q-learning. The idea of dynamic regulation comes from the concept of a membership function in fuzzy control theory [42].

When a negative update of the Q-value is small, we have great reason to believe that the negative update associated

with the current action is likely to be caused by unreasonable actions of other agents. Taking Figure 1 as an example, when agent j selects a better action but the Q-value update of this agent is still negative, the action selection of agent i should instead be more strongly adjusted. This is the main way in which the algorithm can be optimized. To realize dynamic adjustment, we introduce an evaluation function similar to a membership function for a negative update. We define a function A, which outputs the learning rate to be used based on the input δ, i.e., the TD error.

Firstly, function A characterizes the mapping relationship between the feedback of the environment and td-error to learning rate generated in the agent network. We believe that the reward from the environment is generated by joint action. The Reward feedback on a single agent will contain information about the promotion of rational joint action. The purpose of designing A function is to find the relationship between td-error and the reliability of action chosen by single agent. The ultimate goal is to improve the coordination ability on the premise of weakening the centralized control.

1) DEGREE OF MEMBERSHIP FUNCTION

Consider a set A of numbers $A(x) \in [0,1]$ corresponding to any element x in the theoretical field (the scope of study) U. A is called a fuzzy set on U, and $A(x)$ is called the membership degree of x in A. When x changes in U, the function $A(x)$, called the membership function of A, shows how the degree of membership of x in A changes. The closer $A(x)$ is to 1, the higher the degree to which x belongs to A, and the closer $A(x)$ is to 0, the lower the degree to which x belongs to A [42].

The degree of membership is conceptualized as a fuzzy evaluation function. Fuzzy comprehensive evaluation is a very effective multifactor decision-making method for conducting a comprehensive evaluation of things affected by many factors. Its characteristic is that the evaluation result is not simply a positive or negative value but a fuzzy set. Using a similar idea, we define an adaptive function to allow an agent to dynamically adjust its learning rate.

2) ADAPTIVE FUNCTION

In the case of continuous actions of an agent or a large action space such that the actions can be considered continuous, rewards can be thought of as continuous planes or tensors [43]. It can be seen in Figure 4 that when a negative update of the Q-value is small, we have great reason to believe that the negative update associated with the current action is likely to be caused by unreasonable actions of other agents. Therefore, the negative update should be appropriately reduced, that is, the existing policy should be maintained in the next iteration. It can also be considered that the sensitivity of the upper saddle points of tensors should be improved when the algorithm is updated.

We use ILs for distributed learning to avoid many of the problems caused by centralization. However, at the same time, it is necessary to avoid the problem of miscoordination. Through analysis, we can see that although the joint return is the highest at Nash equilibrium points, the final optimization direction will often be toward a secondary advantage in the IL case. According to the above analysis of the prisoners' dilemma and relative overgeneralization, the fundamental reason for this behavior lies in the nondiscriminative update strategy. This strategy, which approximates the expected return, causes the algorithm to easily fall into suboptimal solutions. Therefore, we consider dynamically adjusting the learning parameters. In this way, updating in the direction of the mathematical expectation can be avoided. At the same time, we believe that if the TD error of an agent when updating is small, to a great extent, we can assume that this agent's own strategy had less impact on the improvement of the joint return and that more focus should be placed on the adjustment of the strategies of other agents. By combining these two ideas, we need to establish a reasonable adaptive function.

We define a function A to dynamically determine the value of the learning rate based on the TD error. In practice, we should set the learning rate of the Q-learning algorithm in a manner that is applicable to practical problems.

3) PARAMETER DESIGN

There is a coupling relationship between TD-error interval and the learning rate we need to determine. Therefore, we need to adopt a reasonable design method to determine the appropriate mapping relationship between them. In this paper, uniform design table [44], [45] is used to set reasonable parameter mapping. Uniform design tables are usually expressed in terms of $U_x(m^n)$. Where x represents the number of tests per test group, m is the number of proficiency tests in our experiment and n is the number of coupling factors that the experiment can accommodate. Two and three TD-error segments were used, so a $U_{12}\left(12^{10}\right)$ uniform design table was applied. Detailed use method can be seen in the appendix.

C. ADAPTIVE Q ALGORITHM

In this section, we introduce the corresponding mathematical formulas for updating the Q-table and updating the strategy of the Q-network. The function model and specific parameters used in our experiments will be specified in the section titled *Experiments*.

1) Q-TABLE UPDATE

In the model of the matrix game problem, due to the small state space, each agent maintains a Q-table to update its strategy for action selection. Accordingly, the update formula in our algorithm is:

$$\begin{aligned} \delta &\leftarrow r - Q_i(a_i) \\ Q_i(a_i) &\leftarrow \begin{cases} Q_i(a_i) + \alpha\delta \ \ if \ \ \delta \geqslant 0 \\ Q_i(a_i) + A(\delta)\,\delta \ \ else \end{cases} \end{aligned} \tag{8}$$

where $Q_i(a_i)$ represents agent i's Q-table for a repeated game. Therefore, the state s is not used in this equation. δ represents the TD error of the IL. Adaptive Q-learning is a decentralized

process: each IL builds its own Q-table, whose size is independent of the number of agents and linear in the number of the agent's own actions.

2) NETWORK STRUCTURE OF THE ALGORITHM

To ensure that the algorithm can meet the requirements for actual use, we study the algorithm using the proposed strategy under the condition of using deep neural networks. At present, the deep Q-network (DQN) framework is a mature algorithm framework with a high frequency of use [46]. A deep Q-network (DQN) is a multilayered neural network that outputs a vector of action values $Q(s, ; \theta)$ for a given state s, where θ denotes the parameters of the network. The Q-learning update for the parameters after action A_t is taken in state S_t and the immediate reward R_{t+1} and resulting state S_{t+1} are observed is expressed as:

$$\theta_{t+1} = \theta_t + \alpha \left(Y^{Q_t} - Q(S_t, A_t; \theta_t) \right) \nabla_{\theta_t} Q(S_t, A_t; \theta_t) \quad (9)$$

where α is the learning rate and the target is defined as:

$$\begin{aligned} Y_t^Q &= Y_t^{DoubleQ} \\ &= R_{t+1} + \gamma Q \left(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \theta_t) ; \theta_t' \right) \end{aligned} \quad (10)$$

where we use a double deep Q-network (DDQN) [40] for our multiagent experiment. A DDQN consists of two networks. The target network, with parameters θ^-, is the same as the online network except that its parameters are copied from the online network every τ steps, at which times $\theta_t^- = \theta^-$, and are kept fixed in all other steps. Note that the action selection process, based on argmax, still depends on the online weights θ_t. This means that, as in Q-learning, we are still estimating the value of the greedy policy in accordance with the current values, as defined by θ_t [40]. However, we use the second set of weights θ_t' to fairly evaluate the value of this policy. This second set of weights can be symmetrically updated by switching the roles of θ and θ'.

In summary, we modify the gradient update strategy of the algorithm to:

$$\begin{aligned} \delta &\leftarrow Y^{Q_t} - Q(S_t, A_t; \theta_t) \\ \theta_{t+1} &\leftarrow \begin{cases} \theta_t + \alpha \delta \nabla_{\theta_t} Q(S_t, A_t; \theta_t) & if\ \delta \geqslant 0 \\ \theta_t + A(\delta)\, \delta \nabla_{\theta_t} Q(S_t, A_t; \theta_t) & else \end{cases} \end{aligned} \quad (11)$$

where A is the function that takes the TD error as input and outputs the learning rate of the network that is used to update θ. Similar to Q-learning, the evaluation function A determines whether an agent's strategy needs to be adjusted by judging the value of the TD error. The structure of AQL can be seen in Figure 5.

We establish an implicit relation between agents through the basic geometric relation of the joint return function. Unlike in the case of explicit channel establishment, the condition for establishing such an association requires only that the joint reward is accepted by the agent and that an appropriate dynamic evaluation function A is determined as the standard. This is the core idea that we apply to optimize the algorithm. Moreover, as the scale of the system increases, the algorithm can be implemented by means of parallel computation. Since the function A is embedded in the structure of the algorithm, the neural network that serves as the basis of the algorithm does not need to be changed. This endows the algorithm with good generalizability.

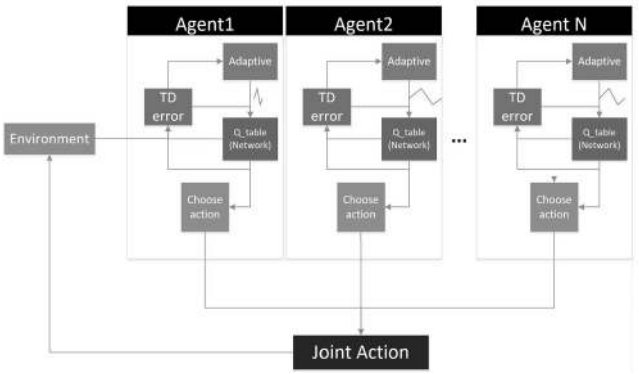


FIGURE 5. The structure of the AQL applied for parallel computation. The algorithm adopts the architecture of parallel computation, and the agent interacts with the environment simultaneously. By evaluating the value of td-error, different learning rates are generated to update the estimator parameters.

V. EXPERIMENTS

In this section, we explain the reasons for the miscoordination in current cooperative multiagent systems based on the classic "prisoners' dilemma" problem. Then, we prove that our method can solve the "relative overgeneralization" problem, in which the algorithm falls into a suboptimal solution in the repeated game environment. We test a variety of multiagent environments. These include the discrete environments of the prisoners' dilemma game and the relative overgeneralization matrix as well as the large-scale continuous action environment of the pursuit domain.

We test various algorithms on both cooperative repeated games (prisoner games): distributed Q-learning, hysteretic Q-learning and adaptive Q-learning. Then, we apply our approach to the pursuit domain. This is a popular multiagent domain in which several agents, called predators, seek to capture another agent, called the prey [27]. The environment we use was provided by [47].

Through the matrix game experiments, we prove the ability of the proposed algorithm to solve a cooperative problem with two agents. After that, a multiagent environment is considered to verify the improvement of the cooperative performance of the intelligent system in the case of a large number of agents.

A. PRISONERS' DILEMMA

The prisoners' dilemma (SG) can be understood as a matrix game in the multiagent framework. Specifically, each state of a repeated game can be viewed as its own matrix game. These concepts were first examined in the field of game theory. We wish to use them to explain the reasons for miscoordination between agents in an IL-based structure.

By solving this problem, the feasibility of proposed improved algorithm is proven.

In this section, we consider the classical discrete game matrix in game theory. This game matrix can be seen in Table 2.

1) ANALYSIS

From the perspective of a rational analyst, both sides of the game can obtain a positive reward by choosing silence. In this case, we set the reward to a positive number $\langle 1, 1\rangle$, which is the Nash equilibrium for this game. Unlike traditional game theory, in which prison time is used as a reinforcement signal, this setting is more in line with the concept of reward in reinforcement learning.

The problem of interest can be found either in the Q-function (the network) maintained by an agent or in the real effect of collaboration. From the point of view of the value function, it is easy to compare with the numerical values in the matrix game. An attempt is made to greedily optimize the value functions maintained by the agents to achieve better rewards. However, because the self-reward is equal to the joint action reward, even the final direction of optimization is towards the suboptimal Nash equilibrium. This is also an important factor affecting the convergence performance of multiagent reinforcement learning. On the other hand, from the perspective of actual collaboration, the agents treat the remaining agents as part of their environment. Therefore, the choices of the actions of the other agents are not considered when selecting actions. Consequently, it is difficult for the agents to make choices so as to avoid conflict. This is also something that needs to be addressed in the coordination strategy.

Unfortunately, due to the uncertainty of the environment, the agents can maximize their benefits only by choosing rational behaviors without prior knowledge. From each agent's own point of view, to avoid being betrayed by the other side and obtaining the worst gain of -5, an agent will often choose to betray to obtain a less negative gain.

2) PARAMETER SETTINGS

A trial consists of 10000 repetitions of the game. At the beginning of a trial, the Q-tables are initialized to 0. At the end of each trial, we determine whether the last chosen joint action is optimal. We set the learning rate $\alpha = 0.8$ for both methods and $\beta = 0.1$ for HQL. The adaptive function used in AQL can be seen in Figure 6.

Where the function A is considered to be a piecewise function: when the TD error is less than -6, the learning rate is the same as α, whereas when the TD error is greater than -6, the slope of the function is -0.004.

3) RESULTS

The rewards of the three algorithms from 10,000 repeated experiments can be seen in Figure 7.

The decentralized Q-learning algorithm performs the worst. When the algorithm converges, it finally converges to the suboptimal Nash equilibrium $\langle -2, -2\rangle$. By contrast, the HQL and AQL algorithms both converge to an optimal Nash equilibrium. It can be seen in Figure 7 that there is little difference between the outcomes of the latter two algorithms in the case of a small action space.

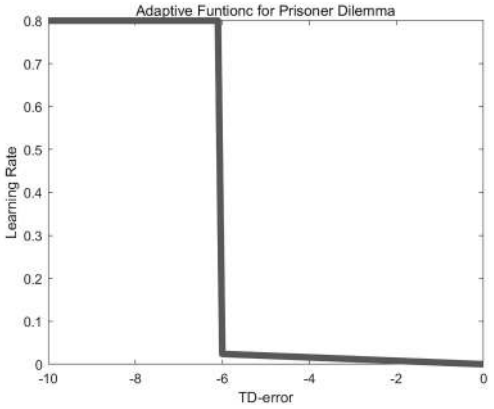
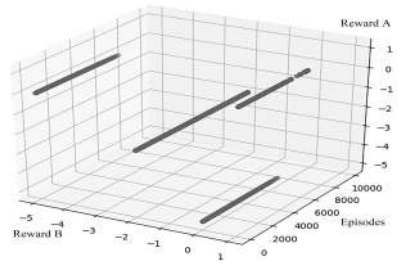
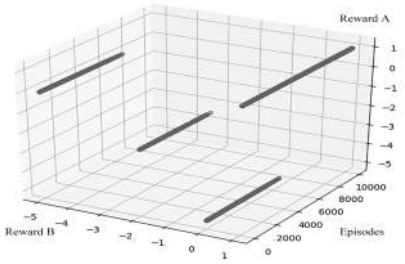


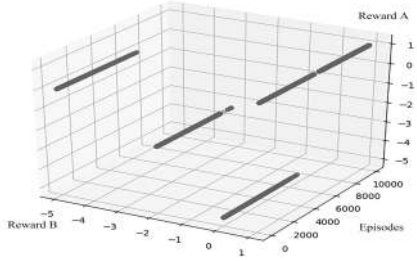
FIGURE 6. The linear evaluation function used in experiment 1.



(a) Q-learning rewards in 10000 episodes



(b) HQL rewards in 10000 episodes



(c) AQL rewards in 10000 episodes

FIGURE 7. Rewards obtained by the 3 algorithms in 10000 episodes.

4) DISCUSSION

The experimental results show that the traditional Q-learning algorithm is unable to solve such problems. At the same time, the final convergence results of HQL and AQL show that dynamic adjustment of the learning rate can give

decentralized agents the ability to distinguish the merits of their actions.

The reason why the results of the latter two algorithms are similar is that when the action and state spaces are small, the learning rate of the algorithm may not require much differentiation. In fact, we find that using a completely continuous function A to determine the learning rate will cause the algorithm to fail to converge. In this scenario, we can regard the HQL algorithm as a special case of AQL. With the addition of the negative update evaluation function A, the AQL algorithm can distinguish the quality of cooperation for different actions of a single agent. We can see from the results that when the environmental action space is small, the effect of the function A is also close to that of the piecewise function.

In summary, these results indicate that in the case of multiagent reinforcement learning, the traditional decentralized algorithm may fail to converge to a good result due to the trapping of the algorithm at a near-optimal solution.

B. RELATIVE OVERGENERALIZATION

1) ANALYSIS

Relative overgeneralization is a specific and problematic subcase of the slightly more general notion of action shadowing, occasionally used by the MARL community. To date, this issue has also caused wide concern. Relative overgeneralization can only arise if each agent has at least three actions available, ideally many more. In contrast, in CCEAs, the number of ''actions'' is very high and often infinite, so the pathology is a common occurrence in that context [25].

In this section, we study the problem of relative overgeneralization in discrete repetitive games. The game matrix is shown in Table 1.

2) PARAMETER SETTINGS

A trial consists of 10000 repetitions of the game. At the beginning of a trial, the Q-tables are initialized to 0. At the end of each trial, we determine whether the last chosen joint action is optimal. We set the learning rate $\alpha = 0.8$ for both methods and $\beta = 0.1$ for HQL. The adaptive function used in AQL can be seen in Figure 8.

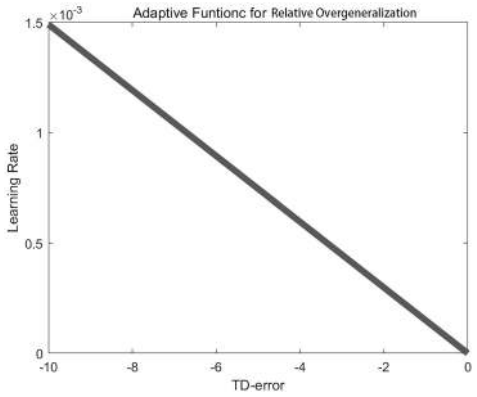


FIGURE 8. The linear evaluation function used in experiment 2.

As seen in Figure 8, the slope of the function is -0.025. The piecewise function is partitioned at -10. In this experiment, we appropriately raise the part of the function A that is below the cutoff point, that is, when TD-error less than -10, the learning rate is 1.1. The purpose of this improvement is to make the algorithm adjust faster when its behavior needs to be adjusted and ultimately accelerate the convergence performance. In this section, we illustrate the improved convergence performance of the algorithm by comparing HQL with AQL with two different parameters.

3) RESULTS

Figure 9 shows the final convergence results of the three algorithms.

It can be seen from the results that the Q-learning algorithm cannot overcome the influence of relative overgeneralization. By contrast, both the HQL and AQL algorithms can solve such problems. In addition, we compare the TD-error differences between the HQL algorithm and the AQL algorithm when different parameters are used in the learning process.

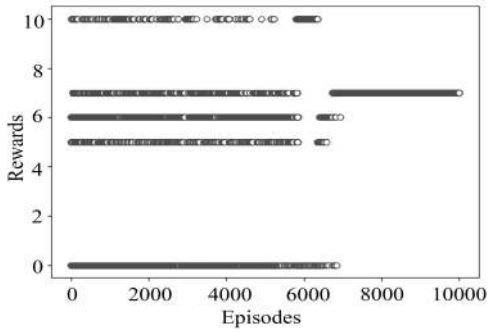
We can see from Figure 10 that when the maximum value of learning rate in the AQL algorithm is the same as that in the HQL algorithm, the fluctuations in the loss of the AQL algorithm will be greater than those for HQL. When the maximum learning rate for AQL is set to a larger value, the loss fluctuations in the later period will decrease.

4) DISCUSSION

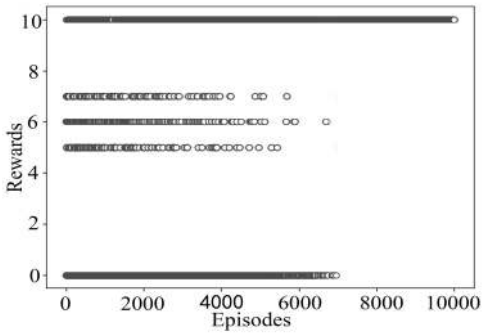
From this experiment, we can see that decentralized Q-learning cannot solve the relative overgeneralization problem. By contrast, both HQL and AQL can solve the relative overgeneralization problem in a 3D action space.

In addition, we have varied the range of the function A in the AQL algorithm. The analysis shows that when the reverse update of the TD error is large, the fluctuation of the algorithm should be increased when adjusting the action strategy. The purpose of doing this is to avoid the dilemma of optimization to the expected value rather than global optimization, with the same cause as in the original algorithm. To ensure a larger difference between the TD errors, we adjust the learning rate to be greater than one. This is done very rarely in reinforcement learning algorithms. Usually, this value is set to a positive number less than 1 to ensure that the algorithm will converge. From the final loss value, it can be seen that the algorithm can provide more stable convergence in the later period. This shows that the role of the learning rate in this problem is consistent with our expectations. On the other hand, it also reflects the influence of the actions of single agents on the performance of the joint actions through the evaluation of the TD error. These findings further illustrate the feasibility of using this method in RL for a MAS.

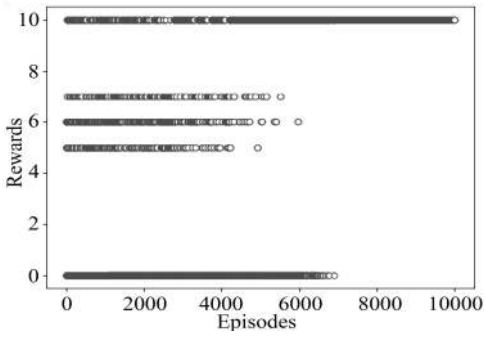
In addition, HQL differentiates between positive and negative updates. The performance of the HQL algorithm is no different from that of the AQL algorithm for systems with fewer agents and smaller action spaces. However, when the return space changes from a two-dimensional space to



(a) Q-learning rewards in 10000 episodes

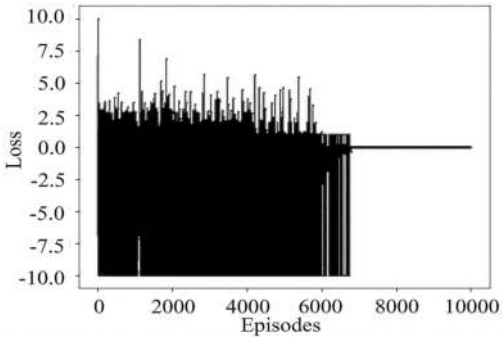


(b) HQL rewards in 10000 episodes

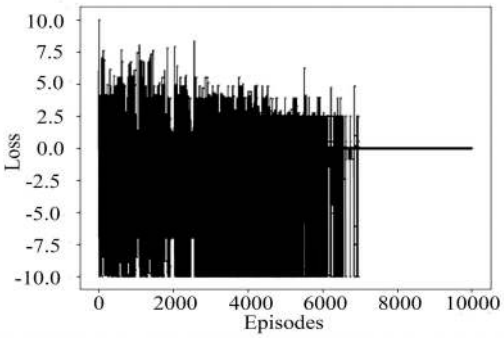


(c) AQL rewards in 10000 episodes

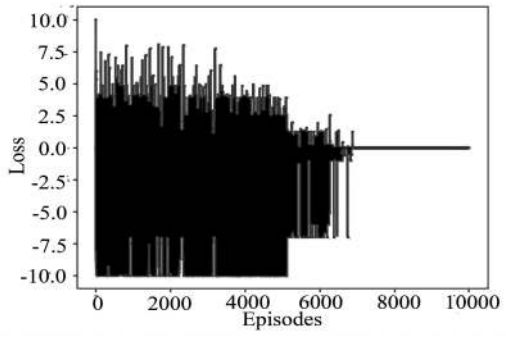
FIGURE 9. Rewards obtained by the 3 algorithms in 10000 episodes.



(a) HQL rewards in 10000 episodes



(b) AQL rewards (with a maximum learning rate of 1.1) in 10000 episodes



(c) AQL rewards (with a maximum learning rate of 1.1) in 10000 episodes

FIGURE 10. Losses obtained by 2 algorithms in 10000 episodes (with different A parameters in AQL).

a multidimensional tensor, this kind of distinction often fails. Therefore, choosing an appropriate dynamic adjustment function A is the key to the AQL algorithm and the most important parameter to ensure the performance of the algorithm.

C. PURSUIT DOMAIN

1) ANALYSIS

In this section, we apply the proposed algorithm to the ''pursuit domain''. As a relatively practical environment, this problem has become known as a benchmark for testing algorithms [27]. The pursuit problem is a popular multiagent domain in which several agents, called predators, seek to capture another agent, called the prey. Unlike the traditional pursuit domain, this environment contains far more agents than the four preys in the traditional pursuit domain. This can effectively simulate the agent collaboration problem existing in the large-scale system.

In this experiment, we use the ''MAgent'' environment to test the algorithm [47]. A large-scale world grid is used as the fundamental environment for the large population of agents. Each agent has a rectangular body with a local detailed perspective. Each predator's possible actions are to move, turn or attack. The prey's only action is to move. In addition, the speed of the prey is faster than that of the predators. Therefore, if the predators cannot learn how to cooperate, they cannot obtain the reward. Both types of agents should be trained for beneficial behavior in obtaining a good reward. We use a DQN for training the prey as a benchmark.

We use a discrete 50 $\times$ 50 toroidal grid environment in which 50 predators must explicitly coordinate their actions to attack 20 prey. The actions of the agents are discrete actions. They can move, attack or turn, as shown in Figure 11.

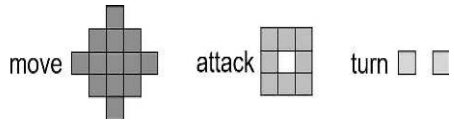


FIGURE 11. Possible actions of a predator showed in grids.

Where each point represents a valid action of the predator. By contrast, the prey can only move and turn. The complete state–action space accordingly consists of all combinations of the predators' and preys' relative positions. This yields either a Q-table with dimensions of $49 \times 48 \times 13^{50}$ in a centralized view or two Q-tables with dimensions of $49 \times 48 \times 13$ in a decentralized view. Obviously, it is difficult to update the Q-table in a centralized algorithm. This shows the necessity of using a decentralized IL-based algorithm structure. During training, we collect all trajectories and train a shared network model in each agent, so the training is centralized.

The reinforcement signal in this environment is for a predator to attack prey within the attack range. When a predator attacks prey, the predators receive a $+1$ reward. The prey, when attacked, will receive a -1 reward. When the predators attack incorrectly, they receive a punishment of -0.2.

To demonstrate the performance of AQL, we compare the new algorithm to a suitable benchmark in this environment. Three benchmark algorithms are provided in the literature: A2C [48], [49], Deep Recurrent Q Network(DRQN) [46], [50] and Deep Q Network(DQN) [31]. Among these benchmark algorithms provided in [47], DQN has the best performance; therefore, we take DQN as the benchmark for comparison. Then, we convert HQL into HQN. By comparing these two algorithms, the results of the Adaptive Q Network(AQN) algorithm are proven.

2) PARAMETER SETTINGS

In the neural network, we directly take the observation as the input and output the state value function Q of the optional action. The actions are selected by the Q-network using the epsilon policy, where the network we use has the structure of DDQN [41]. The parameters of the three algorithms are shown in Table 3.

TABLE 3. Algorithm parameters.

Algorithms	DDQN	HQN	AQN
Memory_size	2^{17}	2^{17}	2^{17}
Batch_size	256	256	256
Learning_rate	10^{-4}	α/β	A Function
Epsilon	decrease	decrease	decrease
Tar_update_freq	4000	4000	4000

Where the memory size represents the capacity of the experience memory replay (EMR) used for sample storage in the algorithm [51]. After several preliminary adjustments, we determined that the convergence speed and outcome of the neural network are optimal with a batch size of 256. The learning rates are different among the three algorithms. DDQN uses a fixed learning rate with a value of 10^{-4}. In HQN, α is 0.8, and β is 0.1. The A used in the AQN algorithm in this experiment is a discrete piecewise function rather than a continuous linear function. This function is shown in Figure 12. The epsilon strategy is used as the action selection strategy, in which the choice is between following the existing strategy or performing random selection. We set its rate of descent to be the same. "Tar update freq" is the parameter update frequency of the target network.

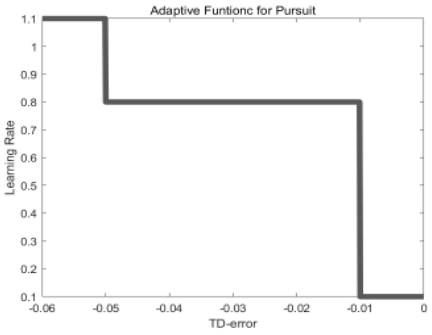


FIGURE 12. The linear evaluation function used in experiment 3.

In accordance with the concept described above, when the TD error is less than 0, we evaluate it using the function A. For this experiment, we define the evaluation function to have three segments. The domain of the first segment is $[0, -0.01]$. The domain of the second segment is $[-0.05, -0.01]$. The domain of the third segment is less than -0.05.

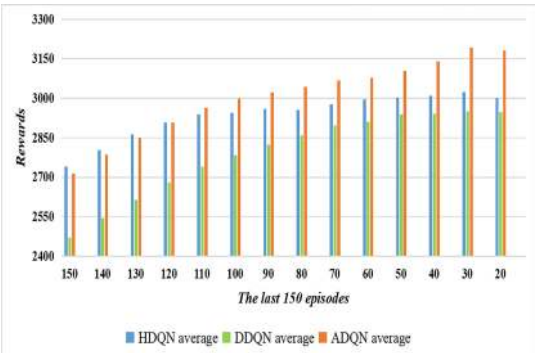
3) RESULTS

In this section, we compare the new algorithm with the two previous algorithms with the best results in this experimental environment. Due to the randomness of the experimental environment and the uncertainty of the gradient descent method, we trained each algorithm 10 times, with each training run consisting of 500 episodes. The means and standard deviations of the rewards obtained by different algorithms were calculated to show the convergence performance and stability of the algorithms. The performance of the agents in the actual test indicates their ultimate coordination ability.

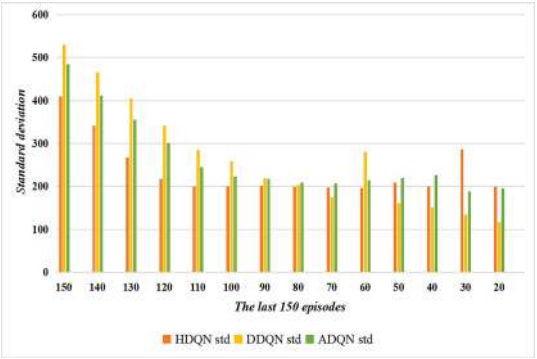
We show the standard deviations and average rewards in the last 20 to 150 episodes to show the rates of algorithm convergence and the results. We can see in Figure 13(a) that DDQN earns the worst reward among the three algorithms and that its increase is relatively gradual. For the first 120 of these episodes, the trends and values of the HDQN and AQN algorithms are similar. Then, HDQN's rewards level off, while those of AQN continue to rise.

Figure 13(b) shows the trends of variation of the standard deviation from 150 episodes to 20 episodes. Where the DDQN algorithm has the best stability. The AQN algorithm is more stable than the DDQN algorithm.

To more clearly compare the algorithms, we show the complete rewards in the last 30 and 100 episodes for the different algorithms. We also use the averages and standard deviations in the last 30 and 100 episodes to illustrate the effectiveness and stability, respectively, of the algorithms.

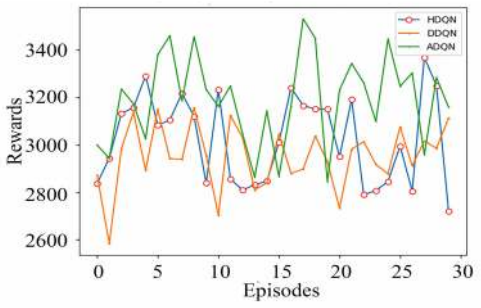


(a) Rewards in the last 20 to 150 episodes

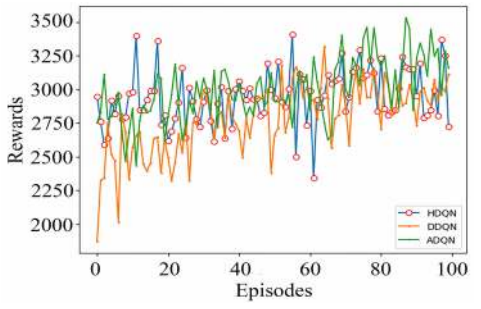


(b) STD values in the last 20 to 150 episodes

FIGURE 13. The rewards and STD values of the algorithms in the last 20 to 150 episodes.



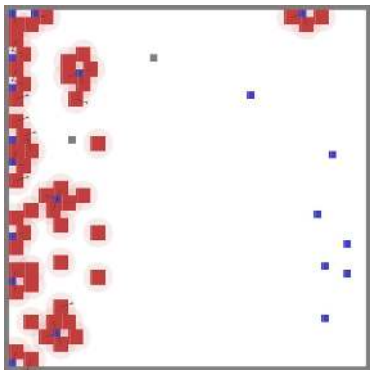
(a) Q-learning rewards in the last 30 episodes



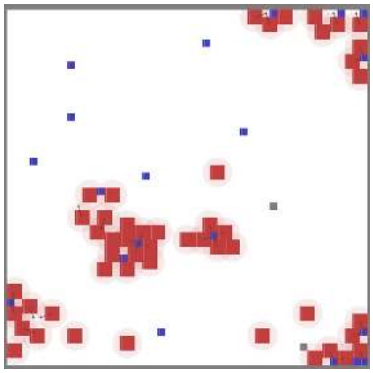
(b) The last 30 episodes rewards got by algorithms

FIGURE 14. Reward the algorithms got in the last 100 episodes.

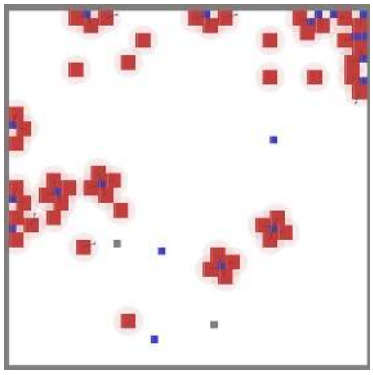
It can be seen in Figure 13. In the last 30 episodes, the average rewards of DDQN, HQN and AQN are 2951, 3024 and 3191, respectively, and the corresponding standard deviations are 134.68, 185.56 and 189.08. In the last 100 episodes, the average rewards of DDQN, HQN and AQN are 2782, 2945 and 2998, respectively, and the corresponding standard deviations are 258.63, 200.00 and 223.41.



(a) Q-learning rewards in 10000 episodes



(b) HQL rewards in 10000 episodes



(c) AQL rewards in 10000 episodes

FIGURE 15. Three different algorithms act as estimators in the real world. The red square represents the predator. The blue square represents the prey. The gray squares represent obstacles.

In Figure 15, the red points are predators, and the blue points are prey. To increase the uncertainty of the environment without excessively increasing the impact of the environment on the performance of the algorithm, we randomly added a few obstacles, represented by gray dots. The predators can use these obstacles to help catch prey if necessary. However, it can bee seen that in this experiment, because we set up relatively few barriers, few predators learned the strategy of using obstacles to capture prey.

4) DISCUSSION

It is not difficult to see from the data that the speed of the AQN algorithm in obtaining results is better than that of DDQN

or HQN, and the final results of the new algorithm in this realistic environment are superior to those of the former two. In addition, AQN is slightly more stable than HDQN. We can see from Figure 13 and Figure 14 that each algorithm will fluctuate around a baseline after convergence. This baseline can be thought of as the average of the results. Moreover, we can see that the fluctuations of the AQN algorithm are similar to those of HQN before the last 120 episodes. Thereafter, the HDQN algorithm fluctuates more smoothly. By contrast, AQN still shows the same rate of growth until the last 30 episodes. Among the three algorithms, DDQN has the best stability, but its convergence results are not satisfactory. It can be clearly seen that using AQN for multiagent cooperative tasks can significantly improve the final convergence results.

At the same time, we can also see from the experimental results that the AQN algorithm can be more efficient in the same environment. In the AQN algorithm, the agents can find the optimal cooperation strategy faster and more stably. With limited environment interactions, the AQN algorithm can find a better collaboration strategy. Therefore, it can be seen from this simulation environment that the AQN algorithm significantly improves the coordination ability of intelligent agents. Although HQL can also somewhat address these problems in a low-dimensional environment, its stability is worse than that of AQL due to its low TD-error discrimination capability. Therefore, its effect is not as good as that of AQL when dealing with high-dimensional problems, and the cooperative effect is not well optimized in a real environment. In the new algorithm, we distinguish among different TD errors, which is equivalent to adding an implicit coordination mechanism. This allows the algorithm to find significantly better joint actions than the previous algorithms. This is why the new algorithm can ultimately earn greater rewards.

Under the same structure, we tested the Advantage Actor Critic(a2c) algorithm [52] and the Deep Recurrent Q network (DRQN) algorithm [53]. The results are shown in appendix. It can be seen that the results obtained by DAQN algorithm are obviously superior to the other two algorithms. Similar to the MADDPG algorithm, the a2c algorithm is not suitable for the large-scale systems. When there are too many agents, the a2c algorithm will not converge. However, the DRQN algorithm converges too early to achieve the effect of multi-agent coordination.

VI. CONCLUSION

To mitigate the problem of the poor result quality caused by miscoordination in ILs for large-scale MAS, this paper analyzes two main reasons for this miscoordination. Considering the two kinds of miscoordination that are likely to occur in a multiagent cooperative environment, this paper proposes an algorithm called adaptive Q-learning to judge whether a particular agent's strategy affects the benefit of the corresponding joint action according to the TD error. By analyzing the differences in the quality of joint actions, we find that to a large extent, it can be assumed that a small TD error in the network maintained by a given agent may indicate irrational actions of other agents. Therefore, we use the TD error as an evaluation standard and propose the AQL algorithm based on a membership function. This algorithm is different from previous algorithms that have incorporated communication mechanisms or adopted a JAL-based architecture, and it inherits many advantages of IL-based algorithm. We use parallel computation to train the same network model to achieve the purpose of weakening the centralized control. This method enhances the adaptability of the algorithm in the large-scale MAS. At the same time, it offers improved joint action coordination.

Through application to matrix games, we prove that the proposed method can solve the problem of miscoordination in the traditional IL-based framework. The convergence performance of the new algorithm is obviously improved without increasing its time complexity. At the same time, it can be seen from its ability to solve the prisoners' dilemma and relative overgeneralization problems that the proposed algorithm has a strong ability to overcome the tendency to converge to suboptimal solutions. This ability provides a foundation for the algorithm to achieve desirable performance in a large-scale multiagent reinforcement learning environment. The enhancement of the coordination of the algorithm is a sufficient condition for the application in large-scale MAS. So, we apply the proposed algorithm to a classic multiagent RL benchmark environment known as the ''pursuit domain''. Compared to the traditional ''pursuit domain'', our environment contains more agents and has a larger action space. This environment is more likely to give rise the miscoordination problem of interest here. At the same time, the agents' partial observations of the environment and the instability of the communication channel make the environment closer to a practical application. Thus, it is demonstrated that the adaptive Q-learning algorithm successfully achieves the goal of coordination in various IL-based multiagent scenarios.

In fact, in the large-scale pursuit domain, we can also see that the AQL algorithm offers an improved convergence speed. With less training, this algorithm can achieve better results than other algorithms with an IL-based architecture. This is one of the advantages of the proposed algorithm. More importantly, the algorithm provides a new way to improve the performance of multiagent reinforcement learning without modifying the network structure. Weaken centralized control endows the method with a strong generalization ability and introduces a new means of improving the performance of collaborative reinforcement learning. As a method of improving collaborative performance in large-scale MAS, the proposed algorithm can be combined with decentralized heterogeneous multiagent systems. In the absence of reliable communication channels, the new algorithm can be used to guarantee the coordination performance in a MAS. Therefore, it is of great significance for applications of multiagent reinforcement learning in practical systems [43], such as cooperative systems of UAVs [22] under combat conditions.

In the future, we intend to develop a more convenient method of defining the function A for the proposed algo-

rithm. In this paper, to make the neural network more stable, we designed the function A as a piecewise function. Although this improves the stability of the algorithm under the existing conditions, it also reduces the differentiation ability of the function A with respect to the TD error. Moreover, in this paper, the method used to determine the function parameters was the uniform design table [13]. This approach is expensive and requires testing a large amount of data. In subsequent work, we intend to use a neural network to fit the function A to obtain a more effective function to improve the performance of AQL.

APPENDIX A
ADDITIONAL EXPERIMENTAL RESULTS

See Fig. 16.

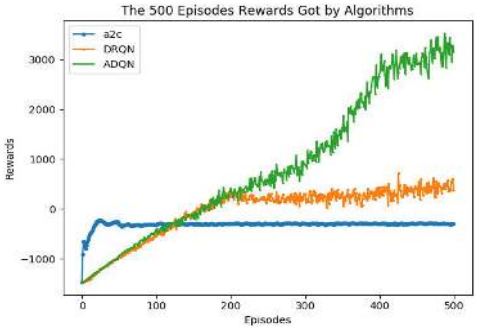


FIGURE 16. The rewards got by the DAQN compared with the a2c and the DRQN.

APPENDIX B
THE UNIFORM DESIGN TABLE FOR SETTING FUNCTION A PARAMETERS

See Tables 4 and 5.

TABLE 4. $U_{12}(12^{10})$.

Groups	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	3	5	7	11
3	3	6	9	12	2	5	11	1	4	10
4	4	8	12	3	7	11	6	10	1	9
5	5	10	2	7	12	4	1	6	11	8
6	6	12	5	11	4	10	9	2	8	7
7	7	1	8	2	9	3	4	11	5	6
8	8	3	11	6	1	9	12	7	2	5
9	9	5	1	10	6	2	7	3	12	4
10	10	7	4	1	11	8	2	12	9	3
11	11	9	7	5	3	1	10	8	6	2
12	12	11	10	9	8	7	5	4	3	1

TABLE 5. The application table of $U_{12}(12^{10})$.

S	column						
2	1	5					
3	1	6	9				
4	1	6	7	9			
5	1	3	4	8	10		
6	1	2	6	7	8	9	
7	1	2	6	7	8	9	10

REFERENCES

[1] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," Dept. Comput. Sci., Univ. Essex, Colchester, U.K., Tech. Rep. CSM-404, 2004, pp. 1–23.
[2] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Auto. Robots*, vol. 8, no. 3, pp. 345–383, Jun. 2000.
[3] Y. Guo, J. Oh, S. Singh, and H. Lee, "Generative adversarial self-imitation learning," 2018, *arXiv:1812.00950*. [Online]. Available: https://arxiv.org/abs/1812.00950
[4] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proc. Nat. Conf. Artif. Intell.*, 1998, pp. 746–752.
[5] Y. Zhong and G.-C. Gu, "Survey of distributed reinforcement learning algorithms in multi-agent systems," *Control Theory Appl.*, vol. 20, no. 3, pp. 317–322, 2003.
[6] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," 2018, *arXiv:1803.11485*. [Online]. Available: https://arxiv.org/abs/1803.11485
[7] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," 2019, *arXiv:1905.05408*. [Online]. Available: https://arxiv.org/abs/1905.05408
[8] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv:1706.05296*. [Online]. Available: https://arxiv.org/abs/1706.05296
[9] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," 2019, *arXiv:1902.04043*. [Online]. Available: https://arxiv.org/abs/1902.04043
[10] O. Vinyals *et al.*, "StarCraft II: A new challenge for reinforcement learning," 2017, *arXiv:1708.04782*. [Online]. Available: https://arxiv.org/abs/1708.04782
[11] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," 2016, *arXiv:1605.07736*. [Online]. Available: http://arxiv.org/abs/1605.07736
[12] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games," 2017, *arXiv:1703.10069*. [Online]. Available: http://arxiv.org/abs/1703.10069
[13] M. Li, X. Yu, D. Wu, and X. Xu, "Path planning based on ant colony tsp algorithm," (in Chinese), in *Proc. 6th China Command Control Conf.*, 2018, pp. 259–265.
[14] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992, doi: 10.1023/A:1022676722315.
[15] T. Ardi, M. Tambet, K. Dorian, K. Ilya, K. Kristjan, A. Juhan, A. Jaan, and V. Raul, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, pp. 172–395, 2017.
[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI GYM," 2016, *arXiv:1606.01540*. [Online]. Available: https://arxiv.org/abs/1606.01540
[17] Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, and H. Tang, "Qatten: A general framework for cooperative multiagent reinforcement learning," 2020, *arXiv:2002.03939*. [Online]. Available: https://arxiv.org/abs/2002.03939
[18] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 1081–1090.
[19] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," 2017, *arXiv:1705.08926*. [Online]. Available: https://arxiv.org/abs/1705.08926
[20] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," 2016, *arXiv:1605.07736*. [Online]. Available: http://arxiv.org/abs/1605.07736
[21] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games," 2017, *arXiv:1703.10069*. [Online]. Available: http://arxiv.org/abs/1703.10069
[22] W. J. W. Jun, X. Xin, and H. Hangen, "A review of the research progress of reinforcement learning for multi-robot systems," (in Chinese), *Control Decis.*, vol. 26, no. 11, pp. 1601–1610, 2011.
[23] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[24] Y. Yang, J. Hao, G. Chen, H. Tang, Y. Chen, Y. Hu, C. Fan, and Z. Wei, "Q-value path decomposition for deep multiagent reinforcement learning," 2020, *arXiv:2002.03950*. [Online]. Available: https://arxiv.org/abs/2002.03950
[25] E. Wei and S. Luke, "Lenient learning in independent-learner stochastic cooperative games," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2914–2955, 2016.
[26] N. Barbalios and P. Tzionas, "A robust approach for multi-agent natural resource allocation based on stochastic optimization algorithms," *Appl. Soft Comput.*, vol. 18, pp. 12–24, May 2014.
[27] L. Matignon, G. J. Laurent, and N. L. Fort-Piat, "Hysteretic Q-learning: An algorithm for decentralized reinforcement learning in cooperative multi-agent teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct./Nov. 2007, pp. 64–69.
[28] L. Matignon, G. Laurent, and N. Fort-Piat, "Coordination of independent learners in cooperative Markov games," Univ. Bourgogne Franche-Comté, Besançon, France, Tech. Rep., 2009.
[29] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," 2016, *arXiv:1605.06676*. [Online]. Available: http://arxiv.org/abs/1605.06676
[30] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," 2017, *arXiv:1702.08887*. [Online]. Available: https://arxiv.org/abs/1702.08887
[31] M. Song, G. Gu, and G. Y. Zhang, "Survey of multi-agent reinforcement learning in Markov games," (in Chinese), *Control Decis.*, vol. 20, no. 10, pp. 1081–1090, 2005.
[32] Q. Liu, J.-W. Zhai, Z.-Z. Zhang, S. Zhong, Q. Zhou, P. Zhang, and J. Xu, "A survey on deep reinforcement learning," *Chin. J. Comput.*, vol. 41, no. 1, pp. 1–27, 2018.
[33] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," 2019, *arXiv:1908.03963*. [Online]. Available: https://arxiv.org/abs/1908.03963
[34] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," 2017, *arXiv:1702.08887*. [Online]. Available: https://arxiv.org/abs/1702.08887
[35] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. 17th Int. Conf. Mach. Learn.*, Jul. 2000, pp. 1–8.
[36] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 1994, pp. 249–257.
[37] L. Panait, K. Tuyls, and S. Luke, "Theoretical advantages of lenient learners: An evolutionary game theoretic perspective," *J. Mach. Learn. Res.*, vol. 9, pp. 423–457, Mar. 2008.
[38] R. P. Wiegand, "An analysis of cooperative coevolutionary algorithms," George Mason Univ., Fairfax, VA, USA, Tech. Rep., 2004, p. 178.
[39] R. Hampel, M. Wagenknecht, and N. Chaker, "Fuzzy control: Theory and practice," *Adv. Intell. Soft Comput.*, 2000.
[40] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," 2015, *arXiv:1509.06461*. [Online]. Available: https://arxiv.org/abs/1509.06461
[41] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Red Hook, NY, USA: Curran Associates, 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf
[42] A. J. H. Al Gizi *et al.*, "Fuzzy control system review," *Int. J. Sci. Eng. Res.*, 2013.
[43] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*. [Online]. Available: https://arxiv.org/abs/1506.02438
[44] Z. Yishu, "Uniformly design the construction of the table and its use table," (in Chinese), *Tactical Missile Technol.*, vol. 4, no. 4, pp. 55–60, 1988.
[45] F. Kaitai, *Uniform Design Uniform Design Table*. Beijing, China: Institute of Applied Mathematics, Chinese Academy of Sciences, 1994.
[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[47] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu, "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence," 2017, *arXiv:1712.00600*. [Online]. Available: http://arxiv.org/abs/1712.00600
[48] R. Song, F. L. Lewis, Q. Wei, and H. Zhang, "Off-policy actor-critic structure for optimal control of unknown systems with disturbances," *IEEE Trans. Cybern.*, vol. 46, no. 5, pp. 1041–1050, May 2016.
[49] M. Babaeizadeh *et al.*, "GA3C: GPU-based A3C for deep reinforcement learning," Dept. Comput. Sci., Univ. Illinois Urbana-Champaign, Champaign, IL, USA, Tech. Rep., 2016.
[50] M. Hausknecht and P. Stone, "Deep recurrent Q-Learning for partially observable MDPs," 2015, *arXiv:1507.06527*. [Online]. Available: http://arxiv.org/abs/1507.06527
[51] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–6.
[52] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[53] M. Hausknecht and P. Stone, "Deep recurrent Q-Learning for partially observable MDPs," 2015, *arXiv:1507.06527*. [Online]. Available: http://arxiv.org/abs/1507.06527

MENG-LIN LI received the B.S. degree in electrical engineering from Xidian University, Xi'an, China, in 2018. He is currently pursuing the master's degree with the School of Intelligent Sciences, National University of Defence Technology.

His research interests include artificial intelligence, machine learning, and multiagent systems.

SHAOFEI CHEN received the B.S. degree in electrical engineering from the Harbin Institute of Technology (HIT), Harbin, China, in 2009, and the M.S. and Ph.D. degrees in automatic control from the National University of Defense Technology (NUDT), Changsha, China, in 2011 and 2016 respectively.

He was a Visiting Student with the Electronics and Computer Science, University of Southampton, U.K., from September 2013 to March 2015. He is currently an Associate Professor with the College of Intelligence Science and Technology, NUDT. His current research interests include multiagent learning and game theory.

JING CHEN received the B.E. and Ph.D. degrees in control science from the National University of Defense Technology (NUDT), Changsha, China, in 1993 and 1999, respectively.

He is currently a Full Professor with the College of Intelligence Science and Technology, NUDT. He has coauthored 2 books and published more than 100 articles in refereed international journals and academic conferences proceedings. His research interests include artificial intelligence, robotics, and automated planning.