Deep Reinforcement Learning with Fuse Adaptive Weighted Demonstration Data

Baofu Fang and Taifeng Guo(✉)

School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China
1136062482@qq.com

Abstract. Traditional multi-agent deep reinforcement learning has difficulty obtaining rewards, slow convergence, and effective cooperation among agents in the pretraining period due to the large joint state space and sparse rewards for action. Therefore, this paper discusses the role of demonstration data in multiagent systems and proposes a multi-agent deep reinforcement learning algorithm from fuse adaptive weight fusion demonstration data. The algorithm sets the weights according to the performance and uses the importance sampling method to bridge the deviation in the mixed sampled data to combine the expert data obtained in the simulation environment with the distributed multi-agent reinforcement learning algorithm to solve the difficult problem. The problem of global exploration improves the convergence speed of the algorithm. The results in the RoboCup2D soccer simulation environment show that the algorithm improves the ability of the agent to hold and shoot the ball, enabling the agent to achieve a higher goal scoring rate and convergence speed relative to demonstration policies and mainstream multi-agent reinforcement learning algorithms.

Keywords: Multiagent deep reinforcement learning · Exploration · Offline reinforcement learning · Importance sampling

1 Introduction

Reinforcement learning improves by interacting with the environment through rewards and punishments to solve sequential decisions for optimal returns to produce a general-purpose agent that solves complex problems [1]. Reinforcement learning has been combined with deep learning in recent years and, combined with the representational power of deep learning, is used to solve many challenging problems, such as IoT security [2], traffic control [3], and resource scheduling [4], reaching and exceeding human capabilities on these tasks.

Multi-Agent Reinforcement Learning (MARL) [5] is a new field formed by combining reinforcement learning with multi-agent systems, which divide the system into several self-consistent and agent subsystems, each of which can perform tasks independently while communicating and coordinating with each other to accomplish tasks together. Multiagent reinforcement learning has already solved many problems that cannot be solved by single agents in areas such as fleet scheduling management problems [6] and massively multiplayer online role-playing games [7]. However, multiagent reinforcement learning explores potential policies through iterative trials, requiring the agents to continuously interact with the simulation environment and learn from their collective traces, leading to low sample efficiency of these multi-agent deep reinforcement learning methods. Due to the huge state space and action space, this random initialization algorithm makes it difficult to obtain rewards in the early stages of training due to the high cost of global exploration, and its slow boosting policy is not suitable for complex environments such as RoboCup2D.

Offline reinforcement learning is inspired by deep learning in large datasets to learn policies from fixed datasets [8], which can use large amounts of existing recorded interaction data to solve real-world decision problems. In offline reinforcement learning, the agent does not receive feedback from the online environment, improving the sample utilization. However, because the policy generating the sample is different from the optimized policy (off-policy), there is a bias in the sampling of the two policies, which makes it difficult for ordinary reinforcement learning methods to cope with the new state space and can even harm the performance of the reinforcement learning algorithm. The policies in the collected demonstration dataset are often suboptimal. It also limits the upper limit of offline reinforcement learning.

The main contributions of this paper are as follows:

1. We propose the distributed combining of demands actor-critic (DCDAC) algorithm. using a scalable distributed actor-critic architecture, which can accelerate the data throughput and improve the learning speed while using policy gradients for learning on demonstration data to give a good initial guide to the policy network.
2. We propose adaptive weighting parameters generated based on the evaluation of performance in the training environment to adjust the weights of the demonstration data to eliminate the detrimental effects of the demonstration data on the policies of the agents.

2 Related Work

2.1 Deep Reinforcement Learning

Reinforcement learning means learning by interacting with the environment and receiving feedback. The balance between exploration and exploitation is a major issue with reinforcement learning [1]. Exploration means that the agent tries some new actions that may result in higher rewards or nothing; exploitation means that the agent repeatedly adopts actions that are known to maximize rewards because the agent already knows that a certain number of rewards can be obtained. Some of the commonly used methods for exploration include $\varepsilon -$ greedy methods with decay (1–0) values to return random actions (exploration if the random number is less than ε) or greedy actions (exploitation if the random number is greater than ε). Stochastic sampling models output random samples of action distributions (e.g., actions and variances) for exploration based on Gaussian distributions and OU (Ornstein-Uhlenbeck) noise to add noise to the model output actions. In real environments, the above methods of exploration are difficult to exploit because in real environments, exploration can damage expensive objects such as robotic arms and cars, while offline reinforcement learning only uses demonstration data for learning, improving the applicability of deep reinforcement learning in many real-world tasks. Its representative method is imitation learning, where successful demonstration trajectories are provided to the agent and the agent learns accordingly [9], and a simple approach to imitation learning is behavior cloning (BC) [10], which uses data from demonstrations as labels and trains the agent using supervised learning. Thus, pure imitation learning methods cannot go beyond the capabilities of demonstration data.

2.2 Multiagent Reinforcement Learning

Multiagent reinforcement learning is an emerging field where deep reinforcement learning (DRL) and multiagent systems are combined. There are problems with nonstationary environments, global exploration, and relative generalization caused by the increase in the number of agents [11]. Learning in a multiagent environment is fundamentally more difficult than in a single-agent environment.

The current methods commonly used for multiagent can be divided into three categories: decentralized methods, centralized methods, and value decomposition methods. The decentralized approach, also called independent learning (IL) [13], is an early method of multiagent reinforcement learning in which each agent learns only the individual action-value function and considers other agents as part of the environment, avoiding the curse of dimensionality. However, because the policies of other agents are constantly changing, the policies learned by the agents will also constantly change, resulting in nonstationary problems. The centralized approach learns the joint action-value function directly by considering all the agents' information, which can mitigate the adverse effects of nonstationarity. However, as the number of agents increases, the parameter space grows exponentially, and the joint action-value function is difficult to use in an environment with a large number of agents due to the problem of scalability. The value decomposition method [12] first "decentralizes" the learning of the individual action-value function of each agent and then "centralizes" the fitting of the joint action-value function using the individual action-values. In this approach, the computational complexity of the joint action-value function grows linearly with the number of agents, and the information of all agents is taken into account. However, the value decomposition approach is prone to the problem of relative overgeneralization [14], which means that the policy network incorrectly converges to suboptimal joint actions. Relative overgeneralization occurs when multiple agents must coordinate their actions but receive negative rewards if only some of them adopt the wrong behavior. The academic community then turned its attention back to decentralized learning approaches. Recently, there has also been independent proximal policy optimization (IPPO) [15] combined with proximal policy optimization (PPO) [16], which states the use of gradient clipping to control the magnitude of differences between old and new policies, thus mitigating nonstationarity in a multiagent environment, and experimental results also show that the use of independent learning in multiple agents can also achieve better results than value decomposition methods. However, the high cost of global exploration due to the huge state and action space leads to its slow convergence in complex environments.

Several approaches have combined reinforcement learning with demonstrative data to achieve faster learning, hopefully with good results in terms of obtaining policies that outperform demonstrators. For example, Deep Q-learning from Demonstrations (DQfD) [17], which combines behavioral cloning from imitation learning with deep q-learning, learns a policy for Atari games by using a loss function that combines a large number of supervised learning loss functions with an n-step Q-learning loss function, which helps ensure that the network satisfies the Bellman equation. However, its training speed is too slow, and in practice, behavioral cloning does not guarantee that the cloned policy will be effective due to the distribution bias between the states in the demonstration data and the policy's states [9]. It is also mentioned in some recent studies that using behavioral cloning on demonstration data does not show better results than using the policy gradient directly [18].

3 Methods

The framework of a distributed actor-critic algorithm with joint demonstration data is shown in Fig. 1, which includes a learner and multiple workers for a given agent i. The workers synchronize their parameters with the learner at regular intervals and interact with the environment to generate trajectories. Multiple workers can greatly increase

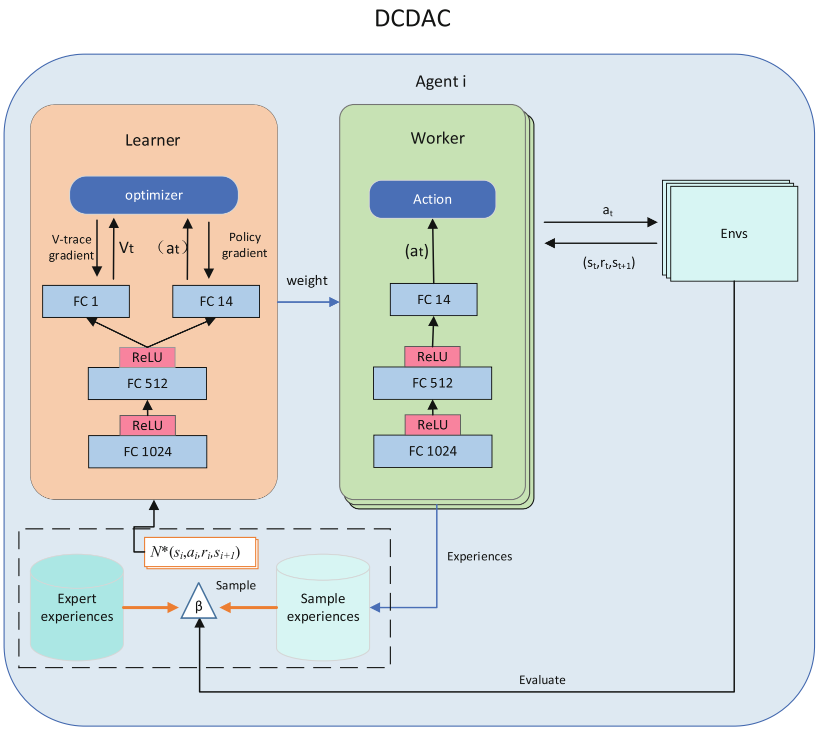

Fig. 1. Distributed actor critic algorithm for joint demonstration data

the sample generation speed and improve learning efficiency. The learner decides to use interaction-generated trajectories or demonstration data for learning based on the parameter β, which is generated by the performance metric of the training environment on the current worker.

The learner uses a fully connected (FC) layer to map the raw data to the hidden feature space, as shown in Fig. 1, and uses the ReLU activation function to improve the model representation. Both actors and critics use states as input, so we make actors and critics share the representation of states, and one head outputs action probabilities (n action types output n-dimensional vectors), and the other head outputs the value of the state. Whereas in workers, the neural network is similar to the learner but only uses the actor function, the trajectories for each learner's training are collected from many workers. In the case of a worker, for example, it updates its local policy μ to the latest learning policy π at the beginning of the trajectory generated by interacting with the environment and runs n steps in its environment. After running n-steps, the worker puts the trajectories $\text{trace}_i = (s_0, a_0, r_0, s_1, a_1, r_1, , \ldots, s_T, a_T, r_T) \in \text{TRACE}$ of state, action, and reward. After running n-steps, the trajectory generator saves the state, action, and reward trajectories along with the corresponding policy distribution $\mu(a_t|s_t)$ and puts them in a queue, while the demonstration data need to be collected in advance and sent to the learner with the trajectories in the trajectory generator according to the weight in learning. Then, the learner continuously updates the trajectory of its policy batches. However, when parameters are updated, the learner's policy π may be updated several times earlier than the worker's policy μ, so there is a policy lag between the worker and the learner. For this reason, we use V-trace to relieve this lag.

We consider the infinite discount reward model optimization problem in a Markov decision process (MDP) [1], where the objective is to find a policy that maximizes the expected sum of future discount rewards: $\text{argmax}_\pi \text{V}^\pi(s) = \text{E}_\pi\left[\sum_{t\geq 0} \gamma^t r_t\right]$, where $\gamma \in [0, 1)$ is the discount factor, $r_t = r(s_t, a_t)$ is the reward at time t, s is the state at time t (initialized $s = s_0$), and $a_t \sim \pi(\text{s}_t)$ is the action generated by following some policy.

The goal of the off-policy reinforcement learning algorithm is to use the trajectory generated by a certain policy μ, called the behavioral policy, to learn the value function V_π of another policy π (which may be different from μ), called the target policy. In the trajectory $(s_t, a_t, r_t)_{t=s}^{t=s+n}$ generated by the agent according to the given policy μ, the n-step V-trace goal is defined as follows:

$$v_t = V(s) + \sum_{t=t_0}^{t_0+n-1} \gamma^{t-t_0} \left(\prod_{i=t_0}^{t-1} c_i \right) \delta_t V \tag{1}$$

where $\delta_t V = \min\left(\overline{\rho}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}\right) * (r_t + \gamma V(s_{t+1}) - V(s_t))$ is the TD-target of the value function, $c_i = \min\left(\overline{c}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}\right)$ is the truncated importance sampling, and $\overline{\rho}$ and $\overline{c}$ are the truncation parameters. Using this approach can effectively reduce the variance in the evaluation of the value function.

There is a policy lag between worker and learner. We update the gradient parameters using importance sampling for the behavioral policy μ and the target policy π:

$$l_{pg} = \rho_t \log \pi_\theta(a_t, s_t)(r_t + \gamma v_{t+1} - V_\theta(s_t)) \tag{2}$$

Similar to PPO, where $\rho_t = \min\left(\overline{\rho}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}\right)$ is truncated importance sampling, limiting the difference between the old and new policies both safeguards the results of importance sampling and reduces the influence of other agents on the optimal policy of the current agent because the agent has fewer differences between the new and original policies. v_{s+1} is the value of the next state.

Define the set of demonstration data D for successive states s^D and actions a^D and returns r^{D} generated over time steps during task execution according to a fixed rule of demonstration policy:

$$D = \left\{s_0^D, a_0^D, r_0^D, s_1^D, a_1^D, r_1^D, , \ldots, s_T^D, a_T^D, r_T^D\right\} \tag{3}$$

Using the policy gradient to train on the demonstration data, the loss function of the demonstration data can be defined as:

$$l_D = -\left[log\pi_\theta(a_t|s_t)A(s_t, a_t)\right] \tag{4}$$

The actor network is initialized by θ and fed with a given state s_t. The output action distribution $log\pi_\theta(a_t|s_t)$ represents the direction of the policy gradient. $A(s_t, a_t)$ is the advantage function, which represents the difference between the performance of the current action a_t and the mean of the performance of all possible actions and is used to determine the direction of the policy update—a positive value increases the probability of this action, while a negative value decreases the probability of this action. The advantage of using the policy gradient for training the demonstration data is that it is similar to the way the collected trajectories are trained. This makes it possible to make the best use of the demonstration data.

The V-trace objective is defined by (1). Gradient descent is used to reduce the value parameter θ to the target value of *V(s)* during training.

$$l_{V(s)} = \frac{1}{2}(v_t - V_\theta(s_t))^2 \tag{5}$$

To prevent premature convergence, add an entropy reward to the loss function:

$$l_c = -\pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \tag{6}$$

However, the demonstration data are not valid for all training phases because the demonstration data are not perfect, and there is room for improvement and a negative impact on reinforcement learning in the second stage of training. Therefore, we introduce β to reduce the impact of the demo data when the performance reaches a stable level. Connect the above loss function as follows:

$$l_{DCD} = \beta(l_D) + (1 - \beta)\left(l_{pg} + l_{vs} + l_c\right) \tag{7}$$

In the selection of the weight parameter, we found that it is not the higher weight (0.5) of demonstration data that is more beneficial to the performance. Instead, higher weighted demonstration data can negatively affect reinforcement learning due to large policy differences. We experimentally observe that using a β value of 0.1 minimizes the negative impact of large policy differences on the offline data. When the performance of the demonstration data with 0.1 weight is unimproved, the β value is corrected to 0, and the next stage of training is performed entirely using reinforcement learning.

Algorithm 1 Distributed Actor Critic Algorithm for Combined Demonstration

Input: training count T, demonstration data D ,maximum trajectory length L_{tarce} , paramter β, batch size N.
Output: Neural network of actors and critics after training.

```
Randomly initialize the parameters of the actors and critics network.
1: def worker:
2:   for episode = 0 to T do
3:     Initialize the environment and get the initial state $s_0$
4:     Synchronize neural network parameters from the learner
5:     for t = 1 to $L_{tarce}$ do
6:       select action according to current policy $a_t = \pi(s_t | \theta_\pi)$
7:       Execution of action $a_t$ yields a return $r_t$ and the next state $s_{t+1}$
8:       Storage Status, action, distribution of action, next action
9:       if end of current trajectory then
10:        Quit the current for loop and proceed to the next epsiode.
11:      end if
12:    end for
13:    Send data to trainer
14:  end for

1: def learner:
2:   for episode = 1 to T do
3:     Randomly select sampling from the demonstration data and the worker's data according $\beta$ Calculate the loss function based on the sampled data
4:     Generate the gradient update model according to Equation (7).
5:   end for
```

4 Experimental Results and Analysis

4.1 Experimental Environment and Data

In this paper, the RoboCup2D multiagent soccer environment Fig. 2 is used as the simulation experiment environment, and the opponent is an agent2d standard soccer player, which has been used as the base for the RoboCup2D World Cup championship for the last 5 years. The task scenario is described in detail as follows: The abstract concept of soccer is used on a two-dimensional plane, where the players, the ball, and the field are all two-dimensional objects. In this, each agent receives its state perception and must independently choose its actions. At the beginning of each episode, the agents and the ball are randomly placed in the attacking half of the field. The episode ends when a goal is scored, when the ball leaves the field, at 500 steps, or when the ball is intercepted by the goalkeeper. We use the state space relative to the agent itself. This includes the agent's position, position relative to the ball, goal, penalty area, opponents, teammates, and angle, and whether one can kick the ball or not. Constitute a vector of 58 + 9 * n dimensions.

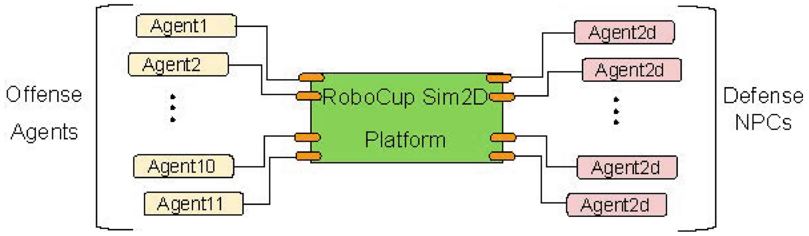

Fig. 2. RoboCup2D standard opponent platform

As opposed to HFO [19], a half-field offensive soccer environment, we added some mid-level action to make it more similar to gamepad controls. The game is a very successful and mature approach in RL. The action space used in the experiment is shown in Table 1. Action spaces.

Table 1. Action spaces

TORIGHT	TOPLEFT
BOTTOMRIGHT	TOP
BOTTOM	TOPRIGHT
BOTTOMLEFT	TOLEFT
AUTOPASS	AUTOTACKLE
MOVE	SHOOT
GO_TO_BALL	NOOP

The movement space is shown in Table 1. Action spaces, which first includes eight movement actions: left, top left, top, top right, right, bottom right, bottom, and bottom left. When holding the ball, the movement becomes a ball-carrying action. AUTOPASS is a passing movement that automatically selects the target and passes the ball based on the angle and distance between teammates and yourself, as well as whether an opponent player is on the passing route; MOVE is moving to the formation point (a preset catch point according to the ball's position); GO_TO_BALL is running to the ball; SHOOT is a shot action, which selects the optimal shot angle based on the goalkeeper and the enemy player. For the current reinforcement learning task, it is easier to use the above action space for effective control of the agent. It is also possible to obtain good results after experiments. The reward uses the HFO setting.

In this paper, we collect consecutive trajectories from the demonstration policy according to fixed rules. The goal sequence of the left demonstration policy is shown in Fig. 3, and the state of the current moment, including the relative position coordinates, velocity, body angle, and head facing angle of each agent; the position and velocity of the ball; and the position coordinates, velocity, body angle, current action, and next action of all teammates and opponents, sorted according to the distance from the current agent state at the next step, and combined with the rewards generated by our reward function. As demonstrated by the different agents listed in the dictionary. At the beginning of the sampling process, adding higher weight to the demonstration, the data gives some guidance to the policy of the agents and stabilizes the transition from demonstration learning to the policy of reinforcement learning. After the training reaches stability (no further improvement for a long time), the adaptive weights β are adjusted, and the demonstration data are stopped to prevent the negative impact of the demonstration data on reinforcement learning.

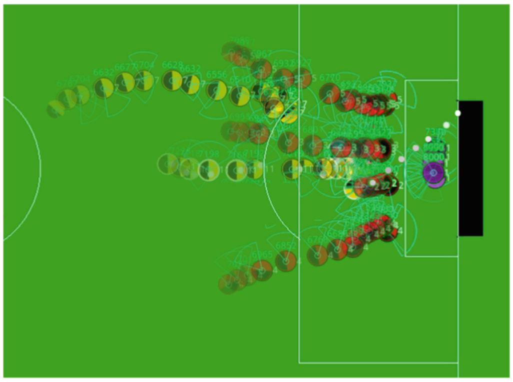

Fig. 3. Collection of demonstration policy trajectories.

To verify the performance of the method in this paper, the convergence speed is reflected in a 2 vs. 2 offensive environment where the adversary is the agent2d base [20], all World Cup teams currently use agent2d for their base, and the agent of agent2d is compared in a 4 vs. 5 scenario to further verify the ability of the algorithm to converge in a large-scale scenario, and the impact of different β is compared in a 2 vs. 5 scenario.

For comparison, the following algorithms are used: IMPALA [21], which uses distributed data collectors to explore in parallel in the environment, improving exploration capabilities also used to solve multiagent problems; Behavioral Cloning [9], which uses supervised learning to exactly imitate demonstration policies; and IPPO [15], which extends PPO to multiagent tasks where each agent runs a set of PPO [16] algorithms independently, with good results in the StarCraft environment. The deep reinforcement learning network framework in the experimental scenario of this paper is implemented by torch with a Core i7-10700 processor, 16 GB RAM, GeForce RTX 1060 GPU, and Adam as the optimizer with a learning rate of 0.0006.

4.2 Results and Analysis

4.2.1 2 vs. 2 Experiment

In the 2 vs. 2 experiment, the number of offensive players is set to 2, the number of defensive players to 1, and a goalkeeper is added. A 2 vs. 2 offensive scenario is set. At each time step, the agent observes their own and their teammates' positions in the environment, as well as the positions of the opposing defensive players. Each iteration is 3,000 steps, and the number of rounds in each iteration is different because the length of each training trajectory is different. The number of goals scored in the current iteration divided by all the rounds is the goal rate.

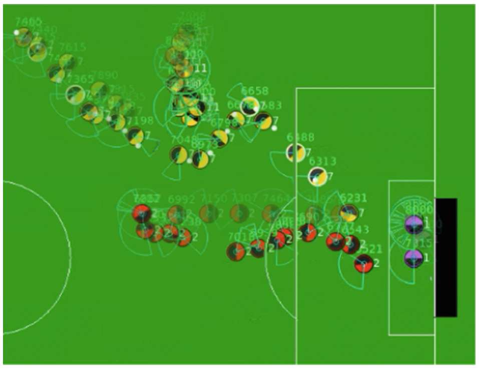

Fig. 4. 2 offensive agents vs. 2 defensive NPC scenarios with goal tracks. (Color figure online)

To illustrate the training effect of successful DCDAC, we take one of the goal trajectory graphs as an example. As shown in Fig. 4, yellow is the trained agent, red is the agent2d standard soccer agent, and white is the ball. Where the light color is the initial position and the brighter color is the later position, the yellow No. 7 player agent learns to move upward first after getting the ball, to lure the goalkeeper upward to make a defensive hole, and then shoot with the ball afterward.

The goal rate comparison of the algorithm is shown in Fig. 5. Comparison of goal rate under 2 offensive agents vs 2 defensive player scenarios. As the number of iterations increases, the behavioral clone remains stable after 300 iterations, and the goal rate of the agents in the other methods also increases, meaning that the agents have gradually learned the policies for successful goals. The addition of demonstration data will make the agents obtain the reward of scoring goals and the demonstration of behavior earlier, thus helping the agent learn the optimal policy faster and better. The IMPALA algorithm, on the other hand, shows better results relative to other algorithms because of its use of parallel workers, which can explore different states in parallel. When the number of iterations reaches 1500, the success rate of the IMPALA algorithm and DCDAC

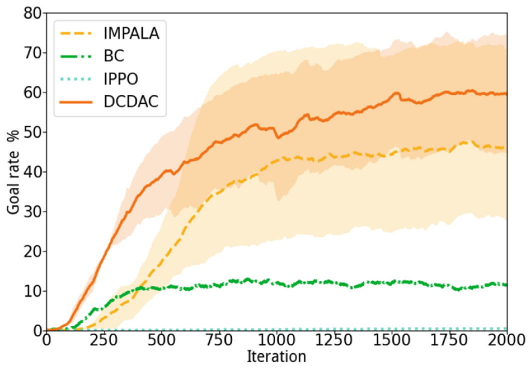

Fig. 5. Comparison of goal rate under 2 offensive agents vs 2 defensive player scenarios.

algorithm stabilizes. When the number of iterations reaches 2000, the goal rate of the DCDAC algorithm stabilizes at approximately 0.6, which can achieve better results than IMPALA but is still inferior to the algorithm proposed in this paper. It can be seen that in the complex multiagent environment, the method proposed in this paper makes a great contribution to the training of the agents.

4.2.2 4 vs. 5 Large-Scale Experiments

In this experiment, the difficulty is increased in terms of the number of agents. Set the number of offensive agents to 4 and the number of defensive players to 5, a scenario with 4 defensive players and one goalkeeper. Compared with the abovementioned 2 vs. 2 half-court offensive experiment, at this time, the agents need to cooperate to break through the defense of the opposing defender to score a goal. The increase in the number of agents also increases the amount of information that the neural network needs to process, which increases the burden on the deep neural network. The perfect NPC that the enemy has won the World Cup also makes it more difficult for the agent to score goals, and the agent's training becomes more difficult.

The algorithm success rate comparison is shown in Fig. 6. Due to the increased difficulty of the experiment, the success rate of all four groups of algorithms has been decreased due to the lack of effective rewards in the huge action space, and state-space IMPALA and PPO algorithms have failed with a goal rate of only 1% after 5000 iterations. The BC algorithm directly through the demonstration data on the imitation in 500 iterations after the goal success rate was maintained at approximately 2%, due to the demonstration data containing only part of the policy, does not contain all cases. In the environment, the agent will encounter states that never occur, which leads to the BC approach being far below the example policy. Since the starting β was set to 0.1 in this experiment, the DCDAC algorithm did not converge as fast as the behavioral clone in the early stage, but because it was able to obtain many goal trajectories in the early stage, it was able to use them to learn and have a good initialization of the policy, which effectively reduced the cost of global exploration. Then, getting rid of the demonstration data to explore on its own after 2000 iterations also had some degree of performance decrease in the early stage but eventually far outperformed the other algorithms in the

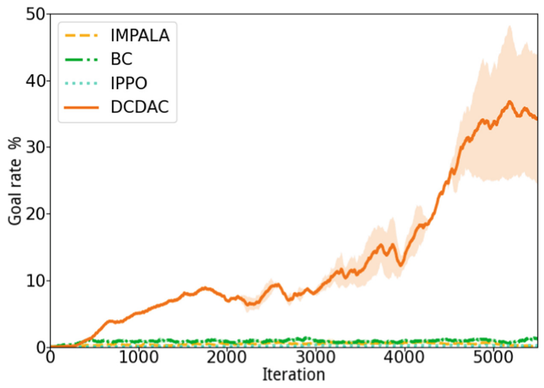

Fig. 6. Comparison of goal rate under 4 offensive agents vs 5 defensive NPC scenarios.

later stages of training. It can be seen that in the 4 vs. 5 attack environment, compared to 2 vs. 2, although the state space is larger, the task is more difficult and convergence is more difficult, the method proposed in this paper is still effective and adaptable.

The policy with a fixed rule that provides demonstration data versus the goal rate after the completion of DCDAC training is shown in Table 2.

Table 2. Comparison of goal rates between the demonstration policy and the DCDAC method

Scenes	Demonstration policy	DCDAC
2 vs. 2	52%	63%
4 vs. 5	20.5%	35%

The goal rate comparison between the DCDAC algorithm and the demonstration policy is shown in Table 2, where our method can produce better policies than the demonstration policy after training stabilization. Since policies generated based on complex rules (decision trees, evaluation of states under complex conditions, dynamic formation changes) are difficult to characterize directly with neural networks, we also do not obtain demonstration policies using BC. For methods such as IPPO, it is difficult to generate goal-scoring behavior in a huge space of states and actions, indicating that reinforcement learning without good demonstration has difficulty surpassing policies for which humans already have complex rules through pure exploration.

4.2.3 Impact of β Weighting on Performance

Finally, we study the effects of different β on the performance of the algorithm in an experimental scenario of 2 vs. 5, with two offensive agents and five defensive players. The goal-scoring rate is shown in Fig. 7. During the training process, when β is 0, there is no demonstration guideline, which leads to no good policy emerging from the agents. When β is 0.1, giving a little demonstration to the agents will make the agents' policies improve. When β is 0.2, it produces some improvement in the early training period when the performance of the agent is weak, but the negative effect brought by too many off-policy samples in the second half of the training period is more obvious, and the negative effect in the second half of the training period is more obvious when β is 0.5. Therefore, demonstration data that are too high will not be better.

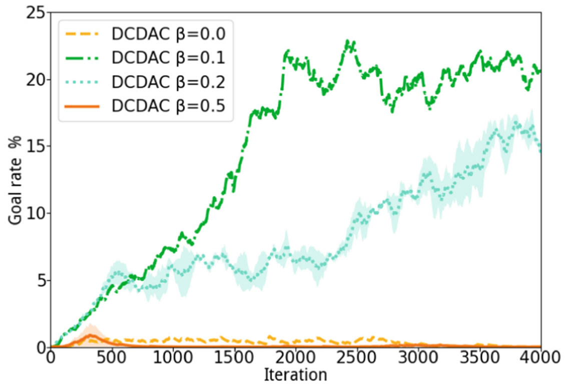

Fig. 7. Comparison of different parameters in 2 offensive agents vs 5 defensive NPC scenarios.

5 Discussion

In this paper, the role of demonstration data in multiagent systems is studied to solve the problem of difficult convergence in large-scale multiagent reinforcement learning due to the difficulty of emerging effective rewards from initial policies. A multiagent deep reinforcement learning algorithm with adaptive weight fusion demonstration data is proposed. The algorithm combines demonstration data with a distributed multiagent reinforcement learning algorithm and experiments in a RoboCup2D simulation environment for comparison, which can make full use of the respective advantages of demonstration data and reinforcement learning and improve the convergence speed and robustness of the algorithm. The experimental results also verify the effectiveness of the algorithm in this paper. The algorithm in this paper is a discrete action space, which leads to the inability to produce more accurate and reasonable actions, and further research on the hybrid action space will also be conducted. In addition, it is found that the cooperation level is still low in the case of multiagents in the experiments. Future research will also be conducted on these aspects of planning to improve them through course learning and self-play.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
2. Li, Y., Liu, T., Zhu, J., Wang, X.: IoT security situational awareness based on Q-learning and Bayesian game. In: Zeng, J., Qin, P., Jing, W., Song, X., Lu, Z. (eds.) ICPCSEE 2021. CCIS, vol. 1452, pp. 190–203. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-5943-0_16
3. Chu, T., Wang, J., Codeca, L., Li, Z.: Multi-agent deep reinforcement learning for large-scale traffic signal control. IEEE Trans. Intell. Transp. Syst. **21**(3), 1086–1095 (2019)
4. Hausknecht, M., Mupparaju, P., Subramanian, S., Kalyanakrishnan, S., Stone, P.: Half field offense: an environment for multiagent learning and ad hoc teamwork. In: AAMAS Adaptive Learning Agents (ALA) Workshop (2016)
5. Chang-Yin, S., Chao-Xu, M.: Important scientific problems of multi-agent deep reinforcement learning. Acta Automatica Sinica **46**(7), 71–79 (2020)
6. Nguyen, D.T., Kumar, A., Lau, H.C.: Policy gradient with value function approximation for collective multiagent planning. In: Advances in Neural Information Processing Systems: Proceedings of NIPS, pp. 4–9 (2017)
7. Peng, P., et al.: Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play starcraft combat games. arXiv preprint arXiv:1703.10069 (2017)
8. Fujimoto, S., Meger, D., Precup, D.: Off-policy deep reinforcement learning without exploration. In: International Conference on Machine Learning, pp. 2052–2062. PMLR (2019)
9. Levine, S., Kumar, A., Tucker, G., Fu, J.: Offline reinforcement learning: tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643 (2020)
10. Zhan, E., Zheng, S., Yue, Y., Sha, L., Lucey, P.: Generative multi-agent behavioral cloning. arXiv (2018)
11. Hernandez-Leal, P., Kartal, B., Taylor, M.E.: A very condensed survey and critique of multi-agent deep reinforcement learning. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, pp. 2146–2148 (2020)
12. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296 (2017)
13. Matignon, L., Laurent, G.J., Le Fort-Piat, N.: Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. Knowl. Eng. Rev. **27**(1), 1–31 (2012)
14. Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., Wu, Y.: The surprising effectiveness of MAPPO in cooperative, multi-agent games. arXiv preprint arXiv:2103.01955 (2021)
15. de Witt, C.S., et al.: Is independent learning all you need in the StarCraft multi-agent challenge? arXiv preprint arXiv:2011.09533 (2020)
16. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
17. Hester, T., et al.: Deep Q-learning from demonstrations. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
18. Wang, Q., Xiong, J., Han, L., Sun, P., Liu, H., Zhang, T.: Exponentially weighted imitation learning for batched historical data. In: NeurIPS, pp. 6291–6300 (2018)
19. Hausknecht, M., Mupparaju, P., Subramanian, S., et al.: Half field offense: an environment for multiagent learning and ad hoc teamwork. In: AAMAS Adaptive Learning Agents (ALA) Workshop (2016)

20. Akiyama, H., Nakashima, T.: Helios base: an open source package for the RoboCup soccer 2D simulation. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 528–535. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_46
21. Espeholt, L., et al.: IMPALA: scalable distributed Deep-RL with importance weighted actor-learner architectures. In International Conference on Machine Learning, pp. 1407–1416. PMLR (2018)