Multi-empirical Discriminant Multi-Agent Reinforcement Learning Algorithm Based on Intra-group Evolution

ZHANG Zhong-lei\textsuperscript{a}, ZHENG Jiao-ling\textsuperscript{b}, ZOU Chang-jie\textsuperscript{c}

Department of Software engineering, Chengdu University of Information and Technology
\textsuperscript{a}yi.ni.a@163.com, +86-18408250743
\textsuperscript{b}zjl9191@163.com
\textsuperscript{c}xiaozouzou777@163.com

ABSTRACT. In order to find the optimal target in the unknown environment more quickly, a multi-empirical discriminant multi-agent reinforcement learning algorithm based on intra-group evolution is proposed based on the deep deterministic policy gradient algorithm (DDPG). In the unknown environment, the agent can find the target faster by performing Information exchange, group genetics and other mechanisms. The comparison with the traditional reinforcement learning algorithm shows that the proposed algorithm is superior to the traditional reinforcement learning algorithm in solving time and solving accuracy, and can quickly and effectively find the optimal target in the environment.

1. INTRODUCTION
Reinforcement learning is a reward-guided behavior in which the agent learns in a "trial and error" manner, and the goal is to maximize the reward of the agent through the interaction with the environment. Reinforcement learning is a sub-area of machine learning. Recently, intensive learning has been applied to solve many challenging problems, including: intelligent traffic information lights in urban road networks in smart cities. The control system of a driverless car. And in industrial applications, reinforcement learning is an effective part of large systems for the tuning of machines and equipment.

The above-mentioned application of reinforcement learning is a single-agent reinforcement learning algorithm in a simple scenario. However, there are also many important application scenarios involving interactions between multiple agents. In this common interaction evolution process, new states will appear and the problems will become more complicated. Therefore, a new topic proposed on the basis of the single agent reinforcement learning algorithm - "multi-agent reinforcement learning algorithm model research" is a very difficult and challenging topic. When designing an algorithm model, you will encounter the following difficulties:

First, when the environment in which the agent is located becomes larger, the time required for the agent to explore the environment increases exponentially, which brings challenges to the efficiency of training. When the environment is too large, the agent is effective in exploring the environment. The proportion of experience will be reduced, which greatly hinders the direct use of previous experience replay, which is an important reason for the rapid convergence of the algorithm.
Secondly, if you simply put multiple agents into the environment, you will find that many agents will repeatedly explore the environment that other agents have explored.

Furthermore, when multiple agents explore the same environment, there is usually more than one method of convergence to the optimal solution, and in the latter stage of training, a large number of agents repeatedly explore training in the same direction, which often wastes computing resources.

Finally, how the agent chooses to interact with other agents during the training process, and how to learn the excellent experience of other agents.

Aiming at the above difficulties, we propose a general multi-agent learning algorithm based on DDPG algorithm--multi-empirical discriminant multi-agent reinforcement learning algorithm based on intra-group evolution. The main contributions of this algorithm include:

1. In the environment, the agent has multiple different types of experience buffer pools. At different stages of the training, the agent extracts the experience proportionally from various experience buffer pools.
2. Experience interaction between agents can share high quality experience.
3. Group the agents according to the performance of each agent in each round, ensuring that more ways can be found to achieve algorithm convergence.
4. Treat each group's agent as a small population, and use the genetic algorithm to make the agent cross-mutation operation to form a new population.

Based on DDPG, this paper conducts research and development of multi-agent reinforcement learning algorithm, which mainly studies how multi-agents interact with each other under complex environmental conditions, so that intelligent bodies can use high-quality experience training to ease training pressure. The framework of distributed execution group training was used for modeling, and the stability of the algorithm was verified in several simulation experiments.

2. RELATED WORK
The algorithm framework of deep reinforcement learning was proposed in the 1990s. It is a combination of reinforcement learning and deep learning. It is an important branch of machine learning. In this learning paradigm, the agent constantly adjusts the strategy through interaction with the environment, in order to achieve the goal of maximizing the cumulative reward value. Classified by the number of agents, it is divided into single agent reinforcement learning algorithm and multi-agent reinforcement learning algorithm.

2.1 Single Agent Reinforcement Learning and Its Development
In 2015, Mnih V et al. combined the convolutional neural network and reinforcement learning to propose the DQN algorithm [1][2], which broke the predicament of the intensive training of deep reinforcement learning algorithms. In order to eliminate the correlation between the intensive learning transfer samples, DQN uses the experience return visit technique and the experience of evenly sampling early interactions and then trains the neural network. However, the uniform sampling method ignores the importance of experience. Schaul[3] proposed priority experience playback, using TD_error to measure the importance of the experience, and replaying the experience of the most important ones, thereby improving the learning efficiency. In addition, there is a huge improvement in the model structure of DQN. Hausknecht et [4] introduced the cyclic neural network into DQN and proposed that the DRQN model surpassed the standard DQN in partially observable reinforcement learning tasks.

The DQN algorithm is mainly applied to the spatial tasks of discrete motions. In the face of continuous action space tasks, the DRL algorithm based on the strategy gradient can obtain better decision-making effects. The literature[5][6] combines DRL and deep learning to propose a depth-determining strategy gradient algorithm DDPG, which assumes that the action generated by the strategy is determined, and the solution of the strategy gradient does not need to sample the integral in the action space, Mnih V et al. Based on this, an A3C algorithm[7][8] using the asynchronous actor-critic framework is proposed. This algorithm uses the multi-threaded function of the CPU to asynchronously execute multiple simulation processes. This parallel training breaks the correlation between training
samples. Compared with the traditional AC algorithm, the A3C algorithm based on multi-thread parallel training combines the dominant function to train the neural network, which greatly improves the learning efficiency of the algorithm.

2.2 Multi-agent Reinforcement Learning and Its Development

The complexity of the multi-agent task makes it difficult for the preset agent strategy to adapt to the changing environment. The agent must rely on its own learning to find a solution and gradually improve the performance of the agent or the entire multi-agent system. Usunier [9] use the full communication autonomous decision-making architecture to train a single network to control multiple similar agents through parameter sharing, and use the gradient-free estimation to update the policy network. Compared with other algorithms, this method can control up to 15 Units. Peng [10] used a two-way LSTM network to build a centralized decision-making architecture for all communications. The central network decision-making outputs the actions of each agent, which has good effects on different scales. The learning part has better performance. Explanatory. Kong [11] combined the advantages of centralized decision-making and autonomous decision-making, using the master-slave architecture of the full communication centralized decision-making architecture, surpassing all previous algorithms in the confrontation of more than ten units. Foerster [12]. Reused experience through fingerprinting and importance weighting, and achieved a good result in smaller combat scenarios using an autonomous decision-making architecture with under-communication. In a later study, Foerster [13] used a centralized critic and decentralized actor-critic architecture actor-critic algorithm to obtain an action advantage function using a counterfactual baseline to solve the credit assignment in multi-agent problems. Good results have been achieved in the under-communication decision-making architecture, and more than ten units can be controlled [14].

2.3 The Application of Group Intelligence in Reinforcement Learning

In the development of reinforcement learning, many group intelligence algorithms are applied to reinforcement learning such as genetic algorithms and particle swarm algorithms. Kenneth O. Stanley [15] proposed a method combining genetic algorithm with reinforcement learning to enhance topological neural evolution (NEAT), which is superior to optimal fixed topology in challenging benchmark reinforcement learning tasks. The method has the following improvements:

1. Different topologies are used
2. Specifications for protection structure innovation
3. Gradually grow from the smallest structure

These studies show that each component is required for the entire system and each other. NEAT makes an important contribution to genetic algorithms because it demonstrates the possibility of simultaneous optimization and composite solutions, providing the possibility of developing increasingly complex solutions among generations and enhancing biological evolution. Analogy.

2.4 DDPG Algorithm Description

DDPG is a model-independent classical reinforcement learning algorithm that finds the best action selection strategy for any given Markov decision process. One of the advantages of DDPG is that it does not require a complete environment model. It is unique in that it can choose between instant rewards and delayed rewards. Each time a decision is made, the robot observes the current state, selects an action and transfers it. Go to the next state. The ultimate goal is to find a sequence of actions that maximizes future cumulative returns. Based on this paper, a new algorithm framework is built based on the DDPG algorithm.

This article refers to robots moving in the environment as agents. The agent has the ability to observe its own environment. The environment in which the agent is located is called the current state st, and the agent changes the current state according to the state selection behavior at. When the agent selects a state through behavior, the environment rewards the agent according to the new state. The robot path planning algorithm framework based on DDPG algorithm is shown in the figure.
As shown in Figure 1, the agent described in this paper is a DDPG neural network composed of an Actor network and a Critic network. The Actor network selects the behavior \( a_t = \mu(s_t) \) by determining the sexual behavior policy \( \mu \). The Critic network uses the behavior-value function \( Q \) to calculate the expected value of the return \( r_t \) obtained after taking the action at and if the policy \( \mu \) is continuously executed, the specific formula is as follows:

\[
Q^\mu(s_t, a_t) = E[r_t(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]
\]

The agent stores the state, behavior, and reward value involved in this process as experience \((s_t, a_t, r_t, s_{t+1})\) in the experience buffer pool. After performing a certain number of steps, a certain amount of experience is randomly selected from the experience buffer pool and recorded as a minibatch into the DDPG network of the agent for training. In the training process, the actual reward is compared with the expected reward, and the loss value \( L \) is obtained.

\[
L = \frac{1}{N} \sum_t \left( r_t - Q(s_t, a_t|\theta^\phi) \right)^2
\]

Use function \( J \) to measure the quality of a strategy \( \mu \):

\[
\nabla_\mu J(\mu) = \frac{1}{N} \sum_{i=1}^N \nabla Q(s_t, a_t|\theta^\phi)_{s_t, a_t=\phi(s_t)} \cdot \nabla_\theta^\phi \mu(s_t|\theta^\phi)_{s_t}
\]

The ultimate training goal of DDPG is to minimize the loss value \( L \) while maximizing \( J(\mu) \).

3. **MGI_DDPG ALGORITHM**

In order to improve training efficiency, on the basis of DDPG algorithm, we proposed the following four-point improvement, proposed a generic multi-agent learning algorithm - based on much experience in the evolution of group discrimination multi-agent reinforcement learning algorithm.
As shown in Figure 2, compared to the DDPG algorithm, only one agent is trained. The algorithm proposed in this paper puts N agents in an unknown environment. Each agent has an independent DDPG network and three different experiences. The cache pool is the same as the DDPG algorithm. Each agent can modify the current state $s$ by the behavior $a$, and the environment returns the reward value $r$ according to the state $s$ of the agent. The agent puts the experience into different experience buffer pools according to the difference in return value. In order to improve the accuracy of the selection behavior, the agent extracts the experience group from the experience buffer pool of the same group of agents as a minibatch, and puts it into the DDPG network of the agent for training. After the end of the round, clustering is performed according to the trajectory similarity $D_s$ of the agent, and the agents are grouped. Treat each group as a population for genetic manipulation to generate new populations and invest in the next round of training.

The specific algorithm pseudo code is as follows:

**Algorithm 1 Algorithm Framework**

Step 1: Initialize N Agents  
Step 2: Initialize the Rpri, Rsuc and Rfai of Agents  
Step 3: For each episode, Loop the following steps:  
(1) Initial state $s_1$  
(2) For each step of the round, loop the following steps:  
1. Save $s_t$ to Rroad, Select the action $a_t$,  
   $a_t = \mu(s_t|\theta^\mu)$  
2. Execute the action $a_t$, get the reward $r_t$ and the new state $s_{t+1}$  
3. Save experience to the corresponding experience cache pool according to [3.1]  
4. According to [3.2], extract m experience from the experience cache pool as minibatch  
5. Minimize the loss $L$ to update the Critic network parameters:
$$L = \frac{1}{N} \sum \left( y_i - Q(s_i, a_i | \theta^0)^2 \right)$$

among:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta'^\mu) | \theta'^\mu)$$

6. Update Actor network parameters with the following gradients:

$$\nabla_{\mu} J(\mu) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mu} Q(s_i, a_i | \theta^0) \big|_{s_i, a_i, \mu(s_{i+1} | \theta'^\mu)} \nabla_{\mu} \mu(s_i, \theta'^\mu) \big|_{s_i, \mu}$$

7. Update parameters of Target-Critic network $Q'$ and Target-Actor network $\mu'$

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

$$\mu'^\mu \leftarrow \tau \mu'^\mu + (1 - \tau) \mu^\mu$$

(3) End step loop

Step 4: Calculate the path similarity $D_s$ between agents according to the [3.3], and cluster them by cluster

Step 5: Perform a [3.4] operation for each group to obtain a new set of agent groups.

Step 6: End episode loop

3.1 More Preferred Experience Buffer Pool

The experience playback mechanism and the experience explored by the agent are stored in the buffer pool, and the neural network is trained by random sampling to eliminate the correlation of the data and improve the stability of the neural network. However, the random extraction of training data does not take into account the quality of the data, resulting in low training efficiency and slow convergence. In order to solve this problem, T Schaul et al. proposed a priority experience caching mechanism. Chen Xiliang[16] proposed a deep reinforcement learning method based on resampling and preferred cached experience playback mechanism. Both of the above methods are high quality with large returns. Data is sent to the neural network for training, which has a certain effect on improving the average return and accelerating convergence. However, the above two algorithms only determine the importance of experience by calculating the TD error, and the convergence speed of the algorithm is still slow.

In the process of learning, human beings will review past experiences, analyze errors and reasons for success, and strengthen memory so as to avoid similar mistakes and deepen the memory of success. Inspired by the human learning process, this paper proposes a multi-preferential experience buffer pool method, which has the following three improvements.

First, the qualification trace factor is added, and the path that has passed is recorded with decay. In the traditional DDPG algorithm, although the agent will update the reward value every step, but there is no positive return before going to the target, the return value of each step of the agent will not be updated, only when the target is reached. The agent will update the return value of the previous step to the target. The agent knows that this step is beneficial to reach the target, and that the return value of all the steps taken before is not updated is considered to be irrelevant. After the introduction of the qualification trace factor, the agent updates the weight of the qualification trace factor for each step taken before reaching the target, and the agent knows that each step taken before is related to the target. Therefore, the chances of being taken after the update are higher in the next round. The Qualification Trace factor is a decay value that lets you know that the closer you are to the target, the more important it is. The qualification trace factor is defined as:

$$z_t = \gamma \lambda z_{t-1} + Q^\mu(s_t, a_t)$$

$$\gamma$$ is the discount coefficient; $\lambda$ is the Trace-Decay parameter, which satisfies the condition $0 < \gamma < 1$, and $Q(s_t, a_t)$ is the return value of the behavior $a_t$ in the $s_t$ state.
Secondly, the multi-experience cache pool mechanism is introduced. While retaining the original preferred cache pool \( R_{\text{priority}} \), two new new experience cache pools are created, namely the revenue cache pool \( R_{\text{success}} \) and the loss cache pool \( R_{\text{failure}} \). The revenue cache pool \( R_{\text{success}} \) is used to store the experience of gaining (revenue experience), the loss cache pool \( R_{\text{failure}} \) is used to store the experience of punishment (punishment experience). The experience calculation priority is placed in the preferred cache pool \( R_{\text{priority}} \), and the experience is judged if it is a revenue experience or a penalty experience. This will be stored in the corresponding experience cache pool.

Therefore, the priority of redefining experience in combination with TD error is defined as follows:

\[
P = A z_t + B \delta_t
\]

\( A, B \) is to make the qualification trace and TD error in the same magnitude, the specific value depends on the environment.

The TD error is:

\[
\delta_t = R_t + \gamma Q'(s_t, \arg \max Q(s_t, a_t) - Q(s_{t-1}, a_{t-1}))
\]

Finally, in the learning of the agent, a certain amount of experience will be extracted from the three experience pools in a certain proportion, which can eliminate the correlation between the data, improve the stability of the neural network training, and reduce the training shock. From the preferred cache pool \( R_{\text{priority}} \), the revenue cache pool \( R_{\text{success}} \) and the loss cache pool \( R_{\text{failure}} \) extraction ratio is as follows

\[
(1 - 2\alpha) : \alpha : \alpha = 0.3 \left( 1 - \frac{\text{episode}_n}{\text{episode}_{\text{max}}} \right)
\]

\( \text{episode}_n \) is the current number of rounds, \( \text{episode}_{\text{max}} \) is the total number of rounds.

The specific algorithm pseudo code is as follows

---

### Algorithm 2 Multi-experience Pool Optimization Algorithm

**Input:** experience \((s_t, a_t, r_t, s_{t+1})\)

**Output:** experience \((s_t, a_t, r_t, s_{t+1})\)

**Step 1:** Initialize the \( R_{\text{priority}}, R_{\text{success}} \) and \( R_{\text{failure}} \) of Agents

**Step 2:** Execute the DDPG algorithm until the step cycle 3 in the DDPG step 4

1. Store the experience \((s_t, a_t, r_t, s_{t+1})\) in the priority cache pool \( R_{\text{priority}} \)
2. Calculate the priority \( P \) of the experience and store it in the priority buffer pool \( R_{\text{priority}} \)
3. Calculate the qualification trace factor:

\[
z_t = \gamma \Delta z_{t-1} + Q^\theta(s_t, a_t)
\]

4. Calculate the td_error:

\[
\delta_t = R_t + \gamma Q'(s_t, \arg \max Q(s_t, a_t) - Q(s_{t-1}, a_{t-1}))
\]

5. Calculate priority: \( P = A z_t + B \delta_t \)

6. Determine whether the reward \( r_t \) is positive. If it is positive, the experience \((s_t, a_t, r_t, s_{t+1})\) is stored in the revenue buffer pool \( R_{\text{success}} \) according to the priority \( P \), and if it is the negative priority \( P \), it is stored in the loss buffer pool \( R_{\text{failure}} \).

**Step 3:** Modify the step process 4: Extract the m experience from the priority cache pool \( R_{\text{priority}} \), the revenue cache pool \( R_{\text{success}} \) and the loss cache pool \( R_{\text{failure}} \) as minibatch, the corresponding ratio is: \((1 - 2\alpha) : \alpha : \alpha\)

---
3.2 Multi-agent Empirical Interaction Mechanism

When the state of the environment is large, if you only invest one agent into the environment, it is a waste of resources. If you can invest in multiple agents in the environment, and the experience between agents can be used together, it will definitely improve training efficiency. To this end, this paper proposes a multi-agent empirical interaction mechanism. The main points of this mechanism are as follows:

First, investing in multiple agents in the environment, each agent has a set of independent DDPG neural networks and corresponding three experience buffer pools.

Secondly, during training, each agent is selected once in turn and executed as a turn step. Each time a round step is completed, the agent saves the experience to the corresponding experience buffer pool according to Algorithm 1. After each number of rounds of completion, the agent learns from its experience pool and the experience buffer pool of other agents to learn and update the DDPG neural network parameters.

Finally, when the agent extraction experience constitutes minibatch, not only the experience is extracted proportionally from the own cache pool, but also a certain amount of experience is extracted from the other agent's revenue cache pool $R_{success}$ and the loss cache pool $R_{failure}$. The ratios extracted from their preferred cache pool $R_{priority}$, revenue cache pool $R_{success}$, loss cache pool $R_{failure}$, and other agents' experience cache pools are as follows.

$$(1 - 3\alpha) : \alpha : \alpha : \alpha = 0.2 \left(1 - \frac{episode}{episode_{max}} \right)$$

$\alpha$ will become smaller as the number of rounds increases. This is to make the agent pay more attention to the preferred cache pool $R_{priority}$ in the early stage of training. The experience in the revenue buffer pool $R_{success}$ reduces some indifferent exploration actions in the experience buffer pool from other agents. When extracting experience, the Agent extracts the state (the Euclidean distance) closest to the state $s$ in the experience pool according to its own state $s$. This makes full use of the experience of other agents in the vicinity of the $s$ state, effectively avoiding the agent repeatedly exploring the same environment.

3.3 Multi-agent Grouping Mechanism

When the environment in training becomes larger, there are often more than one path that can reach the target. If there is no restriction on the experience sharing of the agent, the routes of all the agents in the latter part of the training are roughly the same. Training multiple agents in the later stages of training is similar to training an agent. To this end, this paper proposes a multi-agent grouping mechanism, which divides the trajectory-like agents into a group by clustering algorithm, and experiences sharing between the same group of agents. In this paper, the trajectory similarity $D_s$ is used to describe the similarity between the trajectories as the clustering basis.

$N$ agents have $N$ tracks after each round of training $\{t_0, t_1, t_2, ..., t_n\}$. The trajectory similarity $D_s$ between the trajectories is calculated by the dynamic time warping algorithm (DTW). For example, for tracks $t_0$ and $t_1$ their lengths are $m$ and $n$, respectively. First, we need to align the two tracks. We need to construct a matrix network with $m$ by $n$. The position of $(i, j)$ in the matrix represents the similarity of the $i$ state $s_i$ in the track $t_0$ to the $j$ state $s_j$ in the track $t_1$. $d(i,j)$, here is the Euclidean distance.
Figure 3 DTW Mapping Relationship

As shown in Figure 3, \( W = \{w_1, w_2, \ldots, w_k\} \) in the figure is a regular trajectory corresponding to time, which is actually expressed as a mapping relationship between the trajectories \( t_0 \) and \( t_1 \).

The regular trajectory satisfies the following constraints:

1. Boundary conditions: \( w_1 = (1, 1) \) and \( w_k = (m, n) \). Both tracks have the same start and end points.
2. Continuity: If \( w_{k-1} = (a', b') \), then the next point \( w_k = (a, b) \) for the path needs to satisfy \( (a - a') \leq 1 \) and \( (b - b') \leq 1 \).
3. Monotonicity: If \( w_{k-1} = (a', b') \), then the next point \( w_k = (a, b) \) for the path needs to satisfy \( 0 \leq (a - a') \) and \( 0 \leq (b - b') \).

Matching the two tracks \( t_0 \) and \( t_1 \) from the \((0, 0)\) point, the distance calculated by all the previous points is accumulated every time a point is reached. After reaching the end point \((n, m)\), this cumulative distance is the last total distance we said above. Using the idea of dynamic programming to find the best regular trajectory with the smallest accumulated distance, its cumulative distance is the path similarity \( D_s \) of the trajectories \( t_0 \) and \( t_1 \). The specific formula has

\[
D_s = \sum d(i, j) + \min \begin{cases} 
  d(i-1, j) \\
  d(i-1, j-1) \\
  d(i, j-1) 
\end{cases}
\]

Next, the paper clusters the trajectories \( \{t_0, t_1, \ldots, t_n\} \) generated by each round by hierarchical clustering. The specific algorithm steps are as follows

**Algorithm 3 Multi-agent grouping**

**Input**: Trajectory of the agent \( \{t_0, t_1, \ldots, t_n\} \), Target class number \( C_n \) (0 ≤ \( C_n \) ≤ \( N \))

**Output**: Agent class \( C_n \) sequence set \( \{0, 3\}, \{4, i\} \ldots \{i, n\} \)

**Step 1**: Initialize Agents

**Step 2**: Following steps are required to complete each episode.

1. Obtain the track of the round of the agent \( \{t_0, t_1, t_2, \ldots, t_n\} \), and treat each track as a class
2. Calculate the path similarity \( D_r \) of each two trajectories as the inter-class distance
3. Merge: Find the two closest classes between classes, classify the two classes into one class.
4. Calculate the distance between classes: use average-linkage as the distance criterion between classes, recalculate the class spacing.
5. Repeat (3) and (4) until the termination condition is met
3.4 Population Evolution Mechanism

When investing in multiple agents in the environment for training, the quality of the DDPG models trained in the environment is not good at the end of each round. At this time, we use the elite retention strategy, retain the good performance model, and use the genetic evolution strategy to make the quality poor. The model is cross-mutated and then placed in the environment as a new population for the next round of training. The process of evolutionary algorithm is as follows:

![Figure 4 Genetic Algorithm Process Diagram](image)

First, the neural network weight of each agent is real-coded. The encoding method is as follows:

\[ \text{Agent} = [\omega_{ij}, \omega_{jk}, \ldots, \omega_{im}, \omega_{mn}, \ldots, \omega_{nk}, \ldots] \]

Where: \( \omega_{ij} \) represents the weight of the input layer neuron i to the hidden layer neuron j in the neural network; \( \omega_{jk} \) represents the weight of the hidden layer neuron j to the output layer neuron k in the neural network.

In the process of optimizing the DDPG neural network, the genetic algorithm is based on the fitness function \( f(i) \) to assess the merits of the individual, so as to achieve the goal of survival of the fittest. Fitness is:

\[ f(i) = R_i + q \sum \delta_i \]

\( R_i \) is the total reward for each round of Agent\(_i\), \( \sum \delta_i \) is the total loss value of each round of agent\(_i\).

In order to prevent the optimal solution generated in the evolution process from being destroyed by crossover and mutation, this paper adopts the elite retention strategy to copy the most adaptable individuals in each group to the next generation. Next, the article selects individuals by roulette selection method according to probability \( P \). The probability \( P \) is:

\[ P(i) = \frac{f(i)}{\sum_{x} f(x)} \]

Because the setting of the crossover probability \( P_c \) and the mutation probability \( P_m \) will affect the convergence of the genetic algorithm and the error of the optimal solution and the real solution to a large extent, this paper selects the crossover probability and mutation probability that can be adaptively adjusted to ensure the population diversity:

\[ P_c = 1 - 0.5 / (1 + e^N) \]
\[ P_m = 1 - 0.05 / (1 + e^N) \]

The meaning of \( \Delta f \) is the difference between the maximum fitness value and the average fitness value in the population.

Since the coding scheme of this paper adopts the real coding scheme, the intersection and mutation operations respectively use the arithmetic cross operation and the uniform mutation operation.

The arithmetic crossover operation means that two agents generate two new agents by linear combination. Suppose the agent Agent\(_a\) and Agent\(_b\) cross each other, then the two new individuals generated after the crossover operation as follows:

\[ \text{Agent}_{a'} = P_c \cdot \text{Agent}_{a} + (1 - P_c) \cdot \text{Agent}_{b} \]
\[ \text{Agent}_{b'} = P_c \cdot \text{Agent}_{b} + (1 - P_c) \cdot \text{Agent}_{a} \]

Uniform mutation operation refers to replacing the original gene values at each locus in an individual coding string with a small probability by using random numbers that are uniformly distributed within a certain range. The specific operation process of uniform variation is: 1) sequentially designate each
locus in the individual coding string as a mutation point. 2) For each mutation point, the random probability is taken from the range of the corresponding gene to replace the original gene with the mutation probability Pm. Suppose there is an agent for

Agent = \[
\begin{bmatrix}
\omega_{i1} & \omega_{i2} & \ldots & \omega_{ip} \\
\omega_{j1} & \omega_{j2} & \ldots & \omega_{jk} \\
\omega_{m1} & \omega_{m2} & \ldots & \omega_{mk}
\end{bmatrix}
\]

If \( \omega_{ij} \) is a mutated point, \( \omega_{ij} \in [\omega_{\text{min}}, \omega_{\text{max}}] \) a random number is generated for \( \omega_{ij} \), and if the random number is smaller than \( P_m \), \( \omega_{ij} \) is replaced with a random number \( \omega_{ij}' \) in the range.

4. SIMULATION

4.1 Experimental Environment Selection and Parameter Setting

In this paper, three different multi-agent simulation environments are designed to verify the effectiveness of the MGI-DDPG algorithm. These three environments are composed of N agents and L obstacles. They exist in a two-dimensional space with continuous space and discrete time. In the world, the agent can take physical actions in the environment and can query the experience pool of other agents, and each agent can independently select behaviors. The environment will feedback the corresponding reward value according to the action and status of the agent. The specific rules of each environment are as follows:

Maze environment: As shown in Figure 5, there are L obstacles, one target, need to bypass the obstacle to find the target, the agent does not know the target position, which requires the agent to explore on its own. When the agent hits an obstacle or goes out of the map, it does not move and gets a -0.01 penalty. 1 points will be awarded when the agent moves to the target location.

Spaceship landing: As shown in Figure 6, the environment is modified from the LunarLander-v2 environment in the gym. The agent moves from the top of the environment to the landing point. If it does not collapse, it is considered to be 100 rewards for completing the mission. The spacecraft can pass Ignite the engine to modify its speed and angle. If the spacecraft is far from the landing point or the speed of the descent is too fast, the crash will get zero rewards.

Angry Birds: As shown in Figure 7, there are L obstacles in the environment. The birds need to avoid obstacles. The birds will get 1 reward for each frame in the environment, and will encounter obstacles every 48 frames. If you hit an obstacle, you will die and get a reward of -1 points. The death of the bird or the passage of all obstacles is considered the end of the round.
4.2 Experimental Results and Analysis

We implemented the algorithm described in this paper and evaluated it in the environment described above. In order to evaluate the quality of the strategy learned in the competitive environment, we use DDPG, DQN and A3C algorithms to compare with the MGI_DDPG algorithm, DDPG, DQN and A3C. The algorithm has an experience caching mechanism. Here, the algorithm cache pool size is set to 5000, the minibatch size is 64, and the total buffer pool capacity of each agent in the MGI_DDPG is 5000. The preferred size of the cache pool Rpriority is 2000. The size of the revenue cache pool Rsuccess is 1500 and the size of the loss cache pool Rfailure is 1500, and the size of the minibatch is 64.

We put the above algorithms into three different environments for training until convergence, and take the number of steps and rewards for each round to evaluate. In order to ensure the accuracy of the experiment, the data in this chapter are averaged after training multiple times. The result of Figure 8, Figure 9, Figure 10 shows the relationship between the average return and training rounds of the MGI_DDPG algorithm, the A3C algorithm, the DDPG algorithm and the DQN algorithm in the above three environments. It can be seen that the DDPG algorithm and the DQN algorithm take a long time to explore and try. It can be seen from the figure that the MGI_DDPG algorithm proposed in this paper takes the shortest time to explore and try, and the convergence speed is the fastest.

![Figure 6: Spaceship Landing and Angry Bird](image_url)

![Figure 7: Maze Environment](image_url)
5. CONCLUSION
In this paper, the depth determination strategy gradient algorithm is applied to the multi-agent reinforcement learning field, so that multiple agents can be grouped and experience interaction and group evolution within the group. The algorithm takes into account the use of high quality experience to make the training process more reasonable and efficient. The repeated training results of the algorithm in different simulation environments show that the algorithm converges faster and is more stable than A3C, DDPG and DQN algorithms.

REFERENCES
[1] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.
[2] Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with Deep Reinforcement Learning[J]. Computer
[3] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.
[4] Hausknecht M, Stone P. Deep recurrent q-learning for partially observable MDPs. arXiv preprint arXiv:1507.06527, 2015
[5] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//ICML. 2014.
[6] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning: U.S. Patent Application 15/217,758[P]. 2017-1-26.
[7] Babaeizadeh M, Frosio I, Tyree S, et al. Reinforcement learning through asynchronous advantage actor-critic on a gpu[J]. 2016.
[8] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International Conference on Machine Learning. 2016: 1928-1937.
[9] Usunier N, Synnaeve G, Lin Z, et al. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks[J]. arXiv preprint arXiv:1609.02993, 2016.
[10] Peng P, Yuan Q, Wen Y, et al. Multiagent BidirectionallyCoordinated nets for learning to play StarCraft combat games[J]. arXiv preprint arXiv:1703.10069, 2017
[11] Kong X, Xin B, Liu F, et al. Revisiting the Master-Slave Architecture in Multi-Agent Deep Reinforcement Learning[J]. arXiv preprint arXiv:1712.07305, 2017.
[12] Foerster J, Nardelli N, Farquhar G, et al. Stabilising experience replay for deep multi-agent reinforcement learning[J]. arXiv preprint arXiv:1702.08887, 2017.
[13] Foerster J, Farquhar G, Afouras T, et al. Counterfactual Multi-Agent Policy Gradients[J]. 2017.
[14] Liang Xingxing, Feng Yihe, Ma Yang, Cheng Guangquan, Jin Cai, Wang Qi, Zhou Yuzhen, Liu Zhong. A review of multi-agent deep reinforcement learning [J/OL]. Acta Automatica Sinica: 1-21[2019-09-16]. https ://doi.org/10.16383/j.aas.c180372.
[15] Stanley Kenneth O, Miikkulainen Risto. Evolving neural networks through augmenting topologies. [J]. Evolutionary Computation,2002,10(2).
[16] Chen Xiliang, Cao Lei, Li Chenxi, Xu Zhixiong, He Ming. A Deep Reinforcement Learning Method Based on Resampling Optimization Cache Experience Playback Mechanism[J]. Control and Decision, 2018, 33(04): 600-606.