

Design of Car Control Algorithm Based on Deep Reinforcement Learning

Yu Lei

College of Intelligence System Science and Engineering, Harbin Engineering University, Harbin, China.

shenyangshidoujiu@163.com

Abstract. Reinforcement learning algorithm is widely used in the field of agent control because of its high robustness and strong adaptability. The problem of Mountain-Car-v0 is a classic control problem, which attracts the attention of many researchers. Traditional control methods, for example, PID control, are widely used in industrial and robot control fields, but they show obvious limitations in the face of such highly nonlinear and unknown environments. Therefore, this paper proposes a car control algorithm based on deep reinforcement learning, aiming at overcoming the shortcomings of traditional methods and realizing efficient control of complex dynamic systems. By combining the automatic extraction of high-dimensional features of deep learning with the dynamic decision-making mechanism of reinforcement learning, an adaptive control algorithm is designed, which can continuously learn and optimize in the interaction with the environment without establishing an accurate system model in advance. The experimental results show that compared with traditional control methods and other intelligent control strategies, the proposed algorithm shows higher efficiency and better adaptability in the task of Mountain-car-V0, and obviously improves the success rate of the Mountain-Car-v0.

Keywords: DQN algorithm; deep learning; Mountain-Car-v0; neural network.

1. Introduction

In the past decade, the intelligent control field, an important branch of artificial intelligence technology, has gradually become an indispensable thrust to promote global scientific and technological innovation. In China, with the promotion of Chinese government's attention to the new generation of artificial intelligence and its strong support in this field, the field of intelligent control has encountered unprecedented development opportunities. According to the *Development Plan of New Generation of Artificial Intelligence*, the Chinese government not only provided financial support and policy guidance for the field of intelligent control, but also vigorously promoted personnel training and project support, which provided a solid foundation for the research and application of intelligent control technology. The rapid development of intelligent control technology benefits from its application in automation, robotics, intelligent manufacturing and other fields. The development of intelligent control technology has greatly improved the efficiency and accuracy of the above fields, thus promoting the smooth transformation and innovation and upgrading of related industries. At the same time, data-driven and algorithm innovation play a key role in the progress of intelligent control technology. They not only optimize the performance of the control system, but also expand the application scope.

Before the development of intelligent control, as the basis of control engineering, the theoretical system and methodology of traditional control supported the development of traditional industrial technology for a long time. For example, the traditional PID controller (Proportional-Integral-Differential controller) is a widely used example in industrial control. With its advantages of simple structure and easy realization and adjustment, PID control has been used in many common industrial production fields, such as pressure control, temperature regulation and liquid flow control. However, the performance of PID controller has common defects in the traditional control field, and its control ability depends on the detailed system model and accurately set parameters. Nonetheless, in the real world, the system is complex and dynamic, and the performance of PID controller is usually limited. Facing the increasing complexity and uncertainty, the traditional control strategy shows its inherent

limitations in some fields. Traditional control strategy usually depends on the premise of the accurate model of the system, but in practical application, such an accurate model is often difficult to obtain, especially in highly complex and dynamic systems.

In view of this defect, intelligent control has been developed. As an important pillar of intelligent control, Deep Reinforcement Learning (DRL) is attracting more and more scholars' and researchers' attention. DRL can find an effective control strategy in the absence of a clear system model by combining the characteristic learning ability of deep learning with the decision-making mechanism of reinforcement learning. Its core advantage lies in realizing self-optimization of control strategy through continuous interaction with the environment, thus effectively coping with highly complex and uncertain system challenges. Furthermore, deep reinforcement learning is outstanding in dealing with many challenges that traditional control methods cannot overcome, such as high-dimensional input processing, automatic feature extraction, and adaptability to dynamic environment. These advantages make deep reinforcement learning show a wide application prospect in many fields, such as automatic driving, intelligent robots, energy system management and so on. Many scholars in both home and abroad have made a lot of achievements in these aspects: in the micro-world, Professor Takaaki Abe's team in Japan made a particle robot that can operate accurately at the particle level by using DRL, taking the particles at each stage as the control object, inputting the target terminal position and time step for the system, and finally controlling the particle's walking path strategy, taking whether the task can be completed as the value of the output [1]. In the macro world, . . . the control system of multi-resource mixed utilization is established based on DRL to solve the problem of driving control of smart cars. The control object is smart cars; and the system inputs environmental information for smart cars, including traffic light information, whether there are pedestrians, etc., and outputs it as driving strategy: whether to brake or change lanes, whether the direction control is correct, whether there is line pressing problem, and whether to deviate from the lane as evaluation criteria [2].

Reinforcement learning plays a key role in unmanned articulated vehicle path tracking, unmanned surface vessel formation path planning, and intelligent obstacle avoidance control for fixed-wing UAV formations. By combining deep reinforcement learning, improved artificial potential field algorithms, and deep deterministic policy gradient (DDPG) algorithms, these studies demonstrate the significant benefits of reinforcement learning in increasing adaptivity, training speed, resolving planning conflicts, and improving obstacle avoidance in complex environments. These results not only contribute to the technological advancement of autopilot systems, autonomous maritime operations, and UAV formation flight safety, but also provide a strong algorithmic support and empirical basis for future control strategies of autonomous systems in complex environments.

2. Deep Reinforcement Learning Theory

2.1 Introduction to basic theory of reinforcement learning

In the field of reinforcement learning, we usually use markov decision processes (MDP) to simulate the decision-making process of agents. MDP is a mathematical framework, which is often used to describe decision-making problems in a specific environment. MDP has four basic elements: state space (S), action space (A), reward function (R) and state transition probability function (P), which together constitute a random decision-making process. S includes all possible environmental states, A defines all executable actions in a given state, R gives an immediate reward after each state transition, and is used to evaluate the advantages and disadvantages of decision-making. P describes the probability distribution of transitions to various possible states after performing a specific action in the current state.

The process of reinforcement learning is to evaluate the total rewards of various strategies through the interaction between agents and the environment, and learn a strategy (π), which can maximize the expected cumulative rewards. The whole process mainly depends on two functions: action value function (Q) and state value function (V). These two functions provide estimates of the expected cumulative rewards in a given state and action. The action value function ($Q(s, a)$) provides a

quantitative measure for the expected return of taking action (A) in a given state (S). The definition of the function (Q) is as follows:

$$Q(s, a) = E[R_{t+1} + \gamma \max_{a'} Q(s', a') | s, a]$$

Among them, R_{t+1} represents the instant reward from state s to states' after performing the action a , and γ is a discount factor, which is used to adjust the importance of future rewards relative to instant rewards. This expectation reflects the cumulative reward that can be obtained when taking specific actions in the current state and following the optimal strategy. Cumulative rewards include both current immediate rewards and expected rewards in the future.

At the same time, the state value function V evaluates the expected total return under the state S following strategy Π :

$$V(s) = E[R_{t+1} + \gamma V(s') | s]$$

Different from the function Q , the function V does not consider the specific action choice, but provides an estimate of the expected total return when following the optimal strategy in a specific state.

There is a close relationship between functions Q and V , and they gradually approach the real action value and state value through iterative updating. In this iterative process, the function V can be updated by the function Q :

$$V(s) = \max_a Q(s, a)$$

This shows that the value of a given state is equal to the value of the action with the greatest expected return among all actions that may be taken in this state. This process embodies the balance of "exploration and utilization" in reinforcement learning, aiming at constantly learning and adjusting strategies through the interaction between agents and the environment, so as to maximize long-term rewards.

2.2 Deep reinforcement learning theory

Traditional reinforcement learning methods mainly rely on tabular formulas such as Q-learning and Sarsa, and directly store a value (such as Q value) for each state-action pair in a table, indicating the expected return of taking the action in that state. This method is very effective when the state space and action space are small and discrete, and also in some early function approximation techniques. However, in the face of high-dimensional continuous state space or complex environment that needs to make decisions directly from the original perceptual input (such as images), their performance will be greatly reduced.

Deep reinforcement learning (DRL) uses neural network to fit Q function and V function. Compared with the traditional reinforcement learning method, it strengthens the ability to deal with high-dimensional state space and generalization, and can learn strategies directly from the original perceptual data. Although training deep neural network requires a lot of computing resources, once the training is completed, the neural network can quickly infer and provide real-time decision-making. In addition, deep reinforcement learning improves the utilization efficiency and learning stability of samples through technologies such as experience playback and target network.

2.2.1. Analytical solution of reinforcement learning

When we evaluate a strategy, our traditional calculation method is to use Bellman expectation equation to get the analytical solution of the state function. According to the four basic elements of markov decision processes, S , A , R and P , the state function $v^\pi(s) (\forall s \in S)$ is obtained for a given strategy π when the state transition matrix $P(s', r | s, a)$ is known. The discount factor γ , between 0 and 1, is introduced to calculate the current value of future awards. r_π represents the awards under the strategy π [5].

$$v^\pi = (1 - \gamma P^\pi)^{-1} r_\pi$$

The traditional analytical method is too complicated to solve the above matrix, so we introduce neural network.

2.2.2. Deep reinforcement learning theory

In large-scale intelligent control theory, the traditional calculation power is insufficient, so neural network is introduced and trained by gradient descent.

The diagram is as follows:

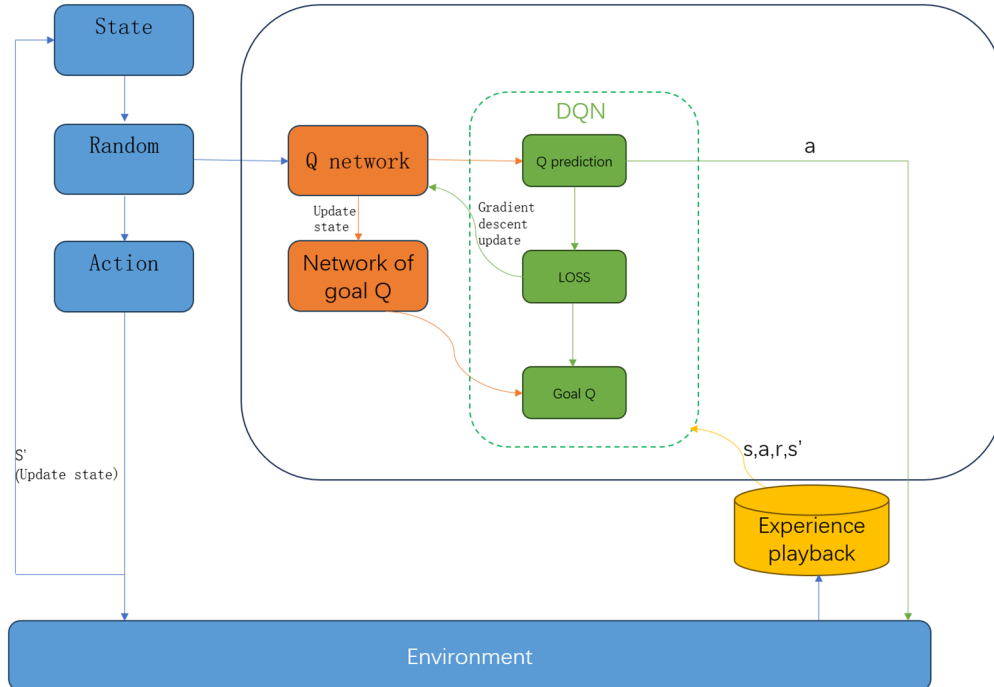


Figure 1. Flow chart of DQN algorithm work

When using DQN algorithm, we will first set the ϵ in the greedy algorithm as small as possible, so that the actions selected by agents are as random as possible, and the resulting action-reward pairs will be put into the experience pool. After the experience pool is full, DQN network will start random sampling from the experience pool and perform gradient descent calculation, thus updating the Q network, and starting the interaction between agents and the environment again, so as to generate new experience pairs, which will eventually make the Q network converge.

Neural network replaces Q function, replaces V function, and gradient descends-updating neural network.

In traditional Q-learning, the state-action pairs are usually stored in the Q table, but this method becomes infeasible when faced with a large number of data. In this plan, we use neural network to replace the Q-value function, estimate the current state for the samples extracted from the experience playback pool, calculate the loss for bellman equation, and update the network parameters by gradient descent method.

2.3 Mountain-Car-v0 problem

Mountain-Car-v0 is a typical reinforcement learning problem, which is used to test and demonstrate the effect of reinforcement learning algorithm. In this problem, the goal of an agent (representing a car) is to drive from a deep valley to the top of the mountain. The car is initially located at the bottom of the valley, and the power is not enough to directly drive it to the nearest hillside. Therefore, the car needs to swing back and forth, gradually increase its height by inertia, and finally reach the top of the mountain. The problem of Mountain-Car-v0 mainly solves how to learn a strategy in the limited state and action space, so that the agent can successfully reach the target position in the least time. Agents need to explore the balance between (trying different actions to understand the environment) and (using known information to make the best decision), and learn how to effectively use the dynamics of the environment (that is, swing) to achieve the goal. This problem

not only tests the ability of reinforcement learning algorithm to deal with continuous state space, but also tests the performance of the algorithm in the face of long-term planning and strategy development.

2.4 Control strategy based on DQN algorithm

2.4.1. Environment construction

In order to solve the problem of Mountain-Car-v0, we first created a simulation environment through OpenAI Gym. The car needs to drive from the bottom of the valley to the top of the mountain. At the start, the car does not have enough power to go straight up the mountain, and it needs to use the other side of the valley to get power. We have established a deep neural network model to achieve the goal of car learning. This model can formulate the current strategy (forward, backward or maintain) according to the car's environmental state, that is, the real-time position and speed, and then evaluate the action strategy. In the training process, we have adopted the experience playback mechanism, that is, we have stored every attempt and result of the car, and then randomly selected samples from these stored experiences to train the neural network. With the training, the car has gradually reduced the proportion of random exploration.

For Gym library construction and environment, we provide an agent, interaction to get some feedback.

2.4.2 Description of the environment of the Mountain-Car-v0

Table 1. Description of the environment of the Mountain-Car-v0

State space (s)	Action space (A)	Reward function (r)	State transition probability function (P)
Speed, position	Climb, reverse, keep	If the agent successfully moves the car between flagpoles on the top of the mountain, it will get a positive reward value. Otherwise, if the car fails to reach the flagpole, the agent will get a negative reward value.	Probability of three actions in specific state

State space (S): The state of the car consists of two continuous variables, namely the position and speed of the car. The position indicates the current position of the car in the valley, and the speed indicates the moving speed of the car (which can be positive or negative).

Action space (A): At each time step, the agent can choose one of three possible actions: accelerating forward (climbing), accelerating backward (reversing) or not accelerating (keeping the current speed and direction).

Reward (R): The agent usually receives a small negative reward at each time step until it reaches the top of the mountain. This reward mechanism encourages the agent to complete the task as soon as possible, because the longer it takes to complete the task, the greater the sum of negative rewards it receives.

Goal: The goal of the agent is to find a strategy that can reach the peak sign from the starting position with as few negative rewards as possible.

2.4.3 Problem solving

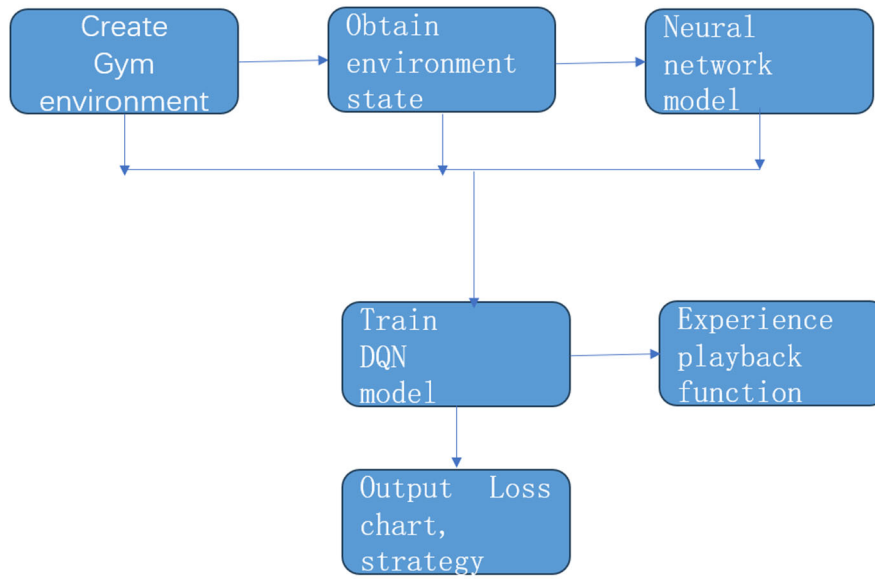


Figure 2. flow chart

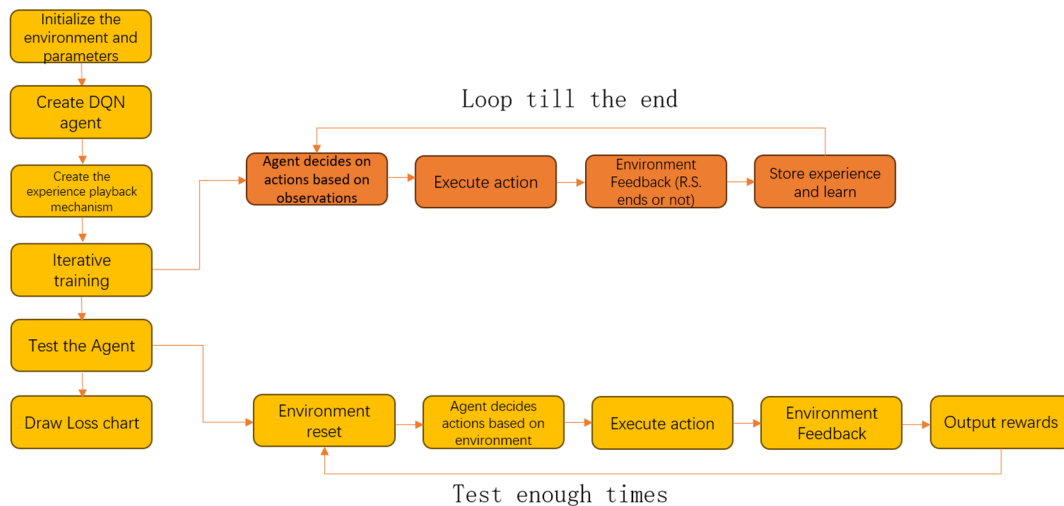


Figure 3. Flow chart of problem solving

In the problem solving code, we set the Batch size to 32, that is, the number of samples extracted from the memory is 32 per training, so as to train the DQN model faster. Adam's learning rate is 0.001, which controls the step size of model parameter updating.

Firstly, the Mountain Car environment is created by using the OpenAI Gym library. This environment simulates a car at the bottom between two mountains, which needs to swing to gain enough momentum and finally reach the goal of the top of the mountain. Then a series of parameters needed in the training process are set, including discount factor, exploration rate, learning rate, etc. Constructing neural network model: TensorFlow is used to construct a deep neural network model, which is used to approximate the Q function to estimate the value of performing every possible action in a given state. This model has two hidden layers, each containing 24 nodes, and uses ReLU activation function. Memory playback function: a memory playback function is defined, which is used to randomly select samples from stored experiences and update the neural network model. This function will also update the exploration rate. Training DQN model: In each turn, agents interact with the environment, take actions and observe the feedback from the environment. Intelligent experience will store these experiences in memory and learn and optimize according to the samples in memory. At the end of each round, the score, current exploration rate and average loss of the round will be

printed. Drawing loss change diagram: after all training rounds, the average loss change diagram is drawn by using matplotlib library, so as to observe the downward trend of loss during model training.

Through continuous training and optimization, the whole process enables agents to learn how to act effectively in the Mountain Car environment, so as to maximize the cumulative reward and achieve the goal.

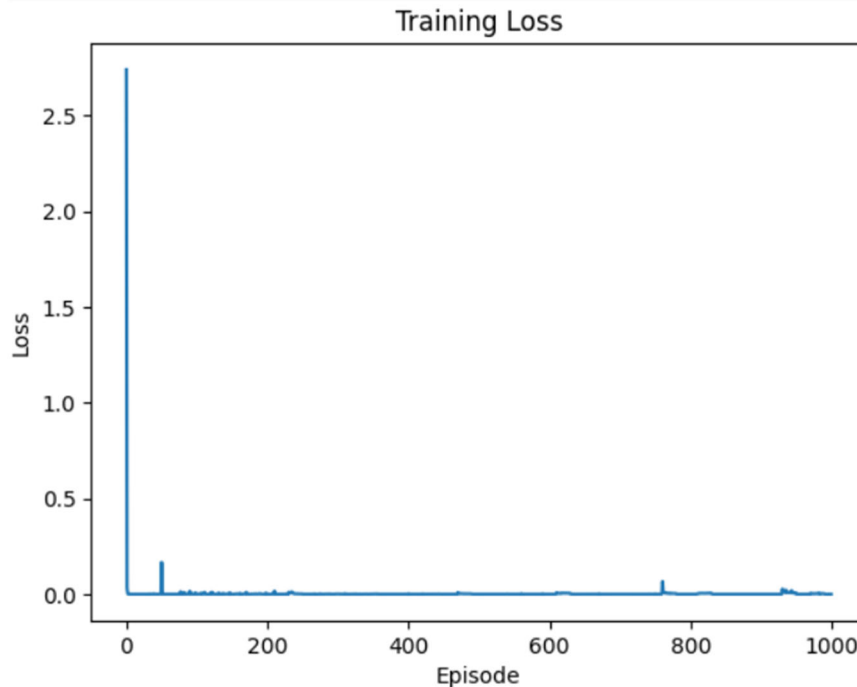


Figure 4. Training Model Diagram

With the increase of the number of rounds, the training loss gradually tends to become 0, and slightly changes around 0, indicating that the model gradually learns the patterns and laws in the data during the training process. In other words, the model is close to the real target in the training data, and its generalization ability to new data is also improved. It means that the model is learning and optimizing effectively.

During the training of this reinforcement learning model, we observed the trend of the loss value and the performance of the model. Overall, the loss value shows a trend of decreasing with the increase in the number of training rounds, which indicates that the model is gradually learning and adapting to the environment to optimize its decision-making strategy. However, in some training phases, we also noticed a sudden increase in the loss value, which may be due to the difficulties encountered by the model in exploring a new state space, or due to noise and uncertainty in the training data. In addition, the model was considered good and saved at different training stages, which suggests that the model is able to achieve a better level of performance under specific conditions.

In general, these observational results illustrate the learning dynamics and challenges of reinforcement learning models during training. In order to further improve the performance and stability of the model, future research can consider adopting more advanced exploration strategies, improving the design of the reward function, and employing more effective training techniques to reduce the effects of noise and uncertainty.

3. Conclusion and Prospect

This paper aims to solve the problem of Mountain-car-V0, and improve and optimize the cars' ability to reach the target position in a challenging environment through innovative deep reinforcement learning methods. This paper introduces how to use the improved algorithm to enhance learning efficiency and decision-making quality in detail, especially when dealing with tasks with high uncertainty and complex dynamics. Innovations of this paper mainly include the optimization

of algorithm design, such as introducing a more efficient exploration mechanism and a more robust reward structure to promote a fast and effective learning process. In addition, how to improve the adaptability and robustness of the model to environmental changes through algorithm adjustment is also discussed. Through the study of Mountain-car-V0 problem, the paper shows the effectiveness of its method in improving task completion rate and learning efficiency, and provides valuable insights for solving similar challenging problems.

There will be many problems when using DQN. Firstly, over-estimation: in DQN, the update of Q-Learning depends on the size of the maximum Q value, but the estimation of Q value is often too large, which will lead to the instability of the learning process and may eventually choose a sub-optimal scheme. Secondly, the sample efficiency of DQN is low. DQN needs a large number of samples to train neural network. In order to break the correlation among samples, the samples will be put into an experience playback pool after being generated, and the samples will be discarded after being used once, which leads to its low sample utilization rate. When dealing with high-dimensional action space, DQN is often used to deal with discrete or relatively small action space, so when dealing with a large number of continuous action tasks, DQN's performance will be significantly reduced.

Therefore, in order to optimize the defects of DQN, we can adopt more innovative methods, such as Double DQN, which separates the Q neural network for selecting actions from the Q neural network for evaluating actions, that is, the Q neural network for calculating cumulative rewards, which is the online network and the target network respectively. The online network value only selects one action to transfer the state, and does not evaluate the value of the action selected by the online network. The target network is used to evaluate the value of the action selected by the online network, and updates the online network in real time by using the difference between the target Q value and the current Q value, so as to reduce overestimation and reduce the risk of choosing a sub-optimal plan. In addition to this method, we can adopt DDPG. Compared with DQN, DDPG adopts Actor-Critic framework. Actor is responsible for outputting certain actions, and Critic is responsible for evaluating the value of actions. When evaluating the value of actions, DDPG updates the network by adopting the algorithm of strategic gradient and function approximation, and according to $\theta' = z\theta + [1 - z]\theta'$, θ is the parameter of the main network, and z is a very small number (for example, 0.001), which can ensure that the parameters of the target network change slowly, thus maintaining the stability of training. In this way, DDPG can effectively learn strategies in continuous action space, and improve the stability of training process through target network and experience playback.

References

- [1] Abe, T., Oh-hara, S., & Ukita, Y. (2023). Integration of deep reinforcement learning to simple microfluidic system toward intelligent control: Demonstration of simultaneous microbeads manipulation. *Sensors and Actuators B: Chemical*, 397, 134636.
- [2] (2017). Reinforcement learning path tracking control algorithm for unmanned articulated vehicles. *Journal of Agricultural Machinery*, 48(3), 7.
- [3] [3] Lee, H., Song, C., Kim, N., & Cha, S. W. (2020). Comparative analysis of energy management strategies for hev: dynamic programming and reinforcement learning. *IEEE Access*, 8, 67112-67123.
- [4] Toichi Ito. (2022). Introduction to Python Deep Reinforcement Learning: Search and Control of Reinforcement Learning and Deep Learning. Machinery Industry Press.