Overview of Deep Reinforcement Learning Improvements and Applications

Junjie Zhang[1], Cong Zhang[1], Wei-Che Chien[2]

[1] School of Mathematics and Computer science, Wuhan Polytechnic University, China
[2] Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan
1281259317@qq.com, hb_wh_zc@163.com, wcc@gms.ndhu.edu.tw

Abstract

The deep reinforcement learning value has received a lot of attention from researchers since it was proposed. It combines the data representation capability of deep learning and the self-learning capability of reinforcement learning to give agents the ability to make direct action decisions on raw data. Deep reinforcement learning continuously optimizes the control strategy by using value function approximation and strategy search methods, ultimately resulting in an agent with a higher level of understanding of the target task. This paper provides a systematic description and summary of the corresponding improvements of these two types of classical method machines. First, this paper briefly describes the basic algorithms of classical deep reinforcement learning, including the Monte Carlo algorithm, the Q-Learning algorithm, and the most primitive deep Q network. Then the machine improvement method of deep reinforcement learning method based on value function and strategy gradient is introduced. And then the applications of deep reinforcement learning in robot control, algorithm parameter optimization and other fields are outlined. Finally, the future of deep reinforcement learning is envisioned based on the current limitations of deep reinforcement learning.

Keywords: Deep reinforcement learning, Value function, Policy gradient, Sparse reward

1 Introduction

One of the main goals of research in the field of Artificial Intelligence (AI) is the implementation of a fully autonomous agent [1]. The agent not only learns the optimal strategy under the current task based on previously performed actions and feedback from the environment, but also continuously improves the action strategy through repeated experimentation. The emergence of deep reinforcement learning accelerates the process of achieving this goal. The deep learning part of deep reinforcement learning uses deep neural networks, which gives reinforcement learning a powerful ability to characterize strategies and states that can be used to simulate complex decision-making processes. In addition, reinforcement learning allows the agent to learn on its own, interact with its environment, and progress through trial and error [2]. Deep reinforcement learning, as an important branch of artificial intelligence research, is considered to be the key to achieve humanoid intelligence and has received wide attention from both academic and industrial communities.

Unlike classical machine learning, reinforcement learning is a self-supervised learning method in which agents in reinforcement learning on the one hand can autonomously interact with the environment, observe and obtain environmental feedback; on the other hand, it can train based on the reward value of the action and environmental feedback and optimizes the action strategy. Earlier approaches to reinforcement learning were based on optimal control theory, which described the sequential decision problem of reinforcement learning as an adaptive dynamic programming (ADP) problem [3]. Based on this approach, researchers extend the sequential decision problem to obtain a strategy-based reinforcement learning problem and propose various strategy search algorithms to solve the problem. Further, in order to understand the advantages and disadvantages of strategies in a simple and intuitive way, scholars have introduced the value function as a criterion for strategy evaluation and proposed a series of classical reinforcement learning models such as Q-learning [4]. At present, the development of reinforcement learning has entered a phase where it is integrated with deep learning. Traditional reinforcement learning methods are limited by the ability to represent strategies and can only deal with simple decision problems. The emergence of deep learning breaks this limitation, and the combination with deep learning injects new power into reinforcement learning theory and applications.

Google's artificial intelligence team DeepMind combines deep neural networks with perceptual

*Corresponding Author: Cong Zhang; E-mail: hb_wh_zc@163.com
DOI: 10.3966/160792642021032202002

capabilities in deep learning and reinforcement learning with decision-making capabilities to form Deep Reinforcement Learning. In 2017, the DeepMind team constructed AlphaGo using deep reinforcement learning algorithms and used AlphaGo to defeat the world's No. 1 human player with a score of 3:0. Not only that, but the DeepMind team has constructed and implemented human expert-level Agents in many challenge areas. These Agents construct and learn about their own knowledge directly from raw input signals, without any manual coding or a prior knowledge. Therefore, deep reinforcement learning is an end-to-end machine learning model that is highly versatile. At present, deep reinforcement learning has been widely used in simulation [5], optimization and scheduling [6-8], game gaming [9-10] and other fields.

Deep reinforcement learning models, while performing well for virtual decision problems, such as game decisions, do not perform well for many other decision problems, especially those in real environments. For example, in the field of robot control, a robot needs to make decisions about the next action to be performed based on information about its state and environment. In this seemingly simple process, the robot not only needs to do the work of executing the strategy, but also needs to do the work of evaluating and optimizing the strategy. For deep strategies or value function networks, a large number of samples are required for training. In the reinforcement learning model based on value function networks, the robot can get one sample in one motion-observation cycle. In contrast, in the reinforcement learning model based on deep strategy search, the robot can obtain one sample after many motion-observation cycles. Therefore, in order to train a better deep reinforcement learning model, more training samples are required, which requires a large amount of training time. An alternative is to train in a virtual environment and fine-tune the trained model in a real environment, however, this relies heavily on the ability of the environmental simulator to simulate the real environment, and the development of high-performance general-purpose simulators is also of interest to reinforcement learning researchers [11].

Deep reinforcement learning is still in the emerging stage and belongs to the emerging research field of artificial intelligence, which has a broad development space and bright application prospect. In the second part of this paper, we briefly introduce the basic theories of reinforcement learning, including the Monte Carlo algorithm, Q-learning algorithm and deep reinforcement learning network, etc. The third part introduces the current common deep reinforcement learning models and the improvement methods of the reinforcement learning models based on value function networks and policy search. The fourth section outlines the applications of deep reinforcement learning in the areas of game decision making, robot control, and optimal scheduling. The fifth part discusses the current limitations of deep reinforcement learning and discusses the theoretical dilemmas of reinforcement learning and the solutions proposed by current hot papers. The sixth part briefly discusses the prospects for future applications of deep reinforcement learning.

2 Basic Theory of Reinforcement Learning

2.1 Monte Carlo Algorithm

The Monte Carlo (MC) algorithm, also known as the computer stochastic simulation method, is a computational method based on "random numbers". It requires only the simulation of sample data series (states, actions, rewards) or actual data obtained through interaction with the environment to find the optimal (better) strategy. When the problem to be solved is the probability of a random event, or the expected value of a random variable, the probability of this random event is estimated by an "experimental" method with the frequency of this event, or some numerical characteristics of this random variable are obtained and used as a solution to the problem.

The MC algorithm has four advantages.

(a) they can learn optimal behaviour directly from environmental interactions, but this does not require a dynamic model of the environment.

(b) They can be used as simulation or sample models. For many applications, the MC algorithm can easily simulate sample plots even though it is difficult to construct exact models of migration probabilities in the DP method.

(c) It is easy and efficient to use the MC algorithm on a small subset of states.

(d) The MC algorithm is less harmful when it violates Markovian properties.

The MC algorithm solves the reinforcement learning problem based on averaging the sampling returns. Assume that there is a termination state and that any strategy can reach this termination state in a finite number of steps with probability 1. In order to estimate the value function, we need to execute the strategy multiple times. Based on this feature of the MC algorithm, the problem it solves must be able to be decomposed into fragments. The updated state - value function for the MC algorithm is as follows:

$$V(s_t) = V(s_{t+1}) + \alpha\left[R_t - V(s_t)\right]. \quad \textbf{(1)}$$

where α denotes the step parameter and R_t denotes the reward value at time t.

2.2 Q-learning Algorithm

The Q-Learning algorithm is a model-independent reinforcement learning algorithm, which was introduced by Watkins in 1989. It is an important milestone in reinforcement learning, the algorithm is a

powerful mutual combination of relevant theories of dynamic planning and the psychology of animal learning, with the goal of solving sequential optimal decision problems with delayed reporting. The iterative formula of Q-Learning algorithm is shown in followed equation.

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max Q(s_{t+1},a_t) - Q(s_t,a_t)\right]. \quad (2)$$

where α is the learning factor (or learning step), γ is the discount rate, and r_{t+1} is the reinforcement value returned by the environment to the learning system at moment $t+1$. $Q(s, a)$ is the value function.

The flow chart of the Q-learning algorithm is shown in Figure 1.

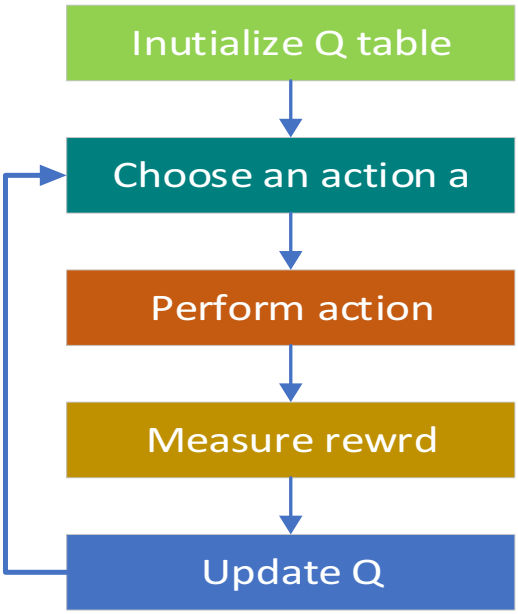


Figure 1. Flowchart of the Q-learning algorithm

2.3 Deep Reinforcement Learning Algorithms

2.3.1 Deep Learning

Deep learning is a general term for a class of pattern analysis methods that primarily involve convolutional neural networks (CNNs), deep self-coding neural networks, and deep confidence networks. The deep learning model consists mainly of multiple non-linear operational units. Data is input from the input layer of the deep neural network and after passing through several hidden layers, it is output from the output layer. In this way, the model can automatically learn the distributed features in large-scale data [12]. Compared with traditional machine learning methods, deep learning models have a more powerful data representation capability and can even extract high-latitude data features that cannot be directly understood by humans. At present, some of the more representative models for deep learning are stacked self-encoders [13-14], restricted Boltzmann machines [15-16], deep belief networks [17-18], recurrent neural networks [19-20] and so on.

2.3.2 Deep Reinforcement Learning

With the growth of training data and the improvement of computational power, more and more experts and researchers combine deep learning with classical reinforcement learning to form the initial deep reinforcement learning model. Riedmiller et al. proposed a neural fitting Q iterative algorithm (NFQ) using a multilayer perceptual machine instead of a Q-value function [21]. Lange [22] and others combined self-encoders with deep learning and proposed a deep autoencoder model (DAE). However, the real landmark was the Deep Q Network (DQN) proposed by Mnih et al.

Deep Q networks are an extension of the Q-learning algorithm. It possesses not only the representational capabilities of deep convolutional networks, but also the decision making capabilities of reinforcement learning. The deep convolutional network in DQN can perceive complex environmental information, and the reinforcement learning part makes an analysis of the perceived information and makes decisions about the next action to be performed. DQN uses deep convolutional networks to approximate the value function in Q-learning, but also destroys the unconditional convergence of the Q-Learning algorithm. To address this issue, DQN has made improvements in the following two aspects:

On the one hand, two deep neural networks are constructed in the DQN, which are identical in structure. However, the roles of the two neural networks are different. One of the neural networks is used to approximate the value function in Q-Learning, while the other neural network is only used to compute the target Q value. In addition, the two neural networks update their parameters in different ways and at different times. The first neural network updates the neural network parameters based on the loss function after each round of training. The other neural network is not involved in the training and is directly synchronized with the parameters of the first neural network after several rounds of training. The updated formula of loss function is as follows:

$$I = (r + \gamma \max Q_{a'}(s', a'; \omega^-)) - Q(s, a; \omega))^2. \quad (3)$$

Where s represents the current state, a represents the action performed, and r represents the reward value of the environment to the agent. $Q(s,a;\omega)$ is the output value of the current Q-Evaluate network to evaluate the value function of the current state-action pair when an action a is executed in the state of s. $Q(s',a';\omega^-)$ is the Q value of the target value function calculated using the Q-Target network.

The updated formula of Q-Evaluate network

parameters is as follows:

$$\omega_{t+1} = \omega_t + \alpha[\mathrm{r} + \gamma \mathrm{maxQ}_{a'}(\mathrm{s}', a'; \omega^-) - \mathrm{Q}(\mathrm{s}, a; \omega)]\nabla \mathrm{Q}(\mathrm{s}, a; \omega) \quad \textbf{(4)}$$

Where ω_t is the parameter values of the neural network at time t, s is the current state, a is the action performed, and r is the reward value of the environment to the agent. $Q(s,a;\omega^-)$ is the output value of the current Q-Evaluate network to evaluate the value function of the current state-action pair when an action a is executed in the state of s. $Q(s',a';\omega^-)$ is the Q value of the target value function calculated using the Q-Target network.

On the other hand, a memory bank is added to the DQN to store the training samples over a period of time. The state space can be approximated as a continuous space in the agent training process. Without processing, the correlation between two adjacent training samples is high, which is not conducive to the training and learning of deep neural networks. Instead, use a memory bank to store training samples over a period of time, and then randomly remove a portion of the samples during training. This can reduce the correlation between the training samples and improve the utilization of the training samples.

3 Improvement Methods of Deep Reinforcement Learning

3.1 Deep Reinforcement Learning Methods Based on Value Functions

In order to accelerate the training rate and improve the stability of the model, experts and scholars in the computer field have proposed various improved versions based on DQN. According to the different emphasis on DQN improvement, they can be divided into three major categories: improvement of algorithm logic, improvement of neural network structure and improvement of introducing new mechanisms. In addition, there are a number of other improvement methods. In the following, we will elaborate on these improvement methods, including their research background, improvement ideas, advantages and disadvantages.

3.1.1 Double Deep Q Network (DDQN)

When using deep Q-network to make decisions and evaluate actions, the Q-max value will be referred to. However, the Q-max value is often larger than the Q value of the next action performed by the agent. This can therefore lead to an overestimation of the Q action value function. To solve this problem, Hasselt et al. proposed the DDQN algorithm.

The deep neural network structure of DDQN is exactly the same as that of DQN, but its parameter update method is different from that of DQN. Equation (5) is the parameter update method for the deep neural network in DQN, and it can be seen that the DQN optimizes the neural network with the largest error Q-max value every time, which leads to the overestimation of the Q action value function by the neural network. However, the DDQN selects the action based on the current Q network for the target Q value and uses the target Q network to calculate the Q value corresponding to that action. By this method, the problem of overestimation of the Q action value function by the neural network is solved.

The updated formula of DQN's deep neural network parameters are as follows:

$$Y_t = \mathrm{r}_{t+1} + \gamma \max Q(s_{t+1}, a; \theta_t). \quad \textbf{(5)}$$

Where, r_{t+1} is the reward value of the environmental feedback at moment $t+1$. γ is the discount factor. $Q(s_{t+1}, a; \theta_t)$ represents the output value of the Q network with parameter θ_t after performing action a in state s_{t+1}.

The updated formula of DQN's deep neural network parameters are as follows:

$$Y_t = \mathrm{r}_{t+1} + \gamma Q(s_{t+1}, \arg\max Q(s_{t+1}, a; \theta_t), \theta_t). \quad \textbf{(6)}$$

Where, rt+1 is the reward value of the environmental feedback at moment t+1. γ is the discount factor. $Q(s_{t+1}, a; \theta_t)$ represents the output value of the Q network with parameter θ_t after performing action a in state s_{t+1}.

3.1.2 Deep Q-learning from Demonstration (DQfD)

Since classical deep learning models are trained with a large number of interactions with the environment in order to achieve better learning performance. It is acceptable when the tasks of the deep learning model can be accurately simulated by a computer. In practice, however, many problems cannot be simulated accurately by the computer, which severely limits the scope for using deep reinforcement learning in real-world tasks. Therefor it is essential that the agent has a good performance at the beginning of its training. While it is difficult to find a completely accurate simulator for real-world problems, most of these problems have data from systems that were run under previous controllers. In other words, the agent needs to learn as much as possible from the demonstration data during the pre-training phase. Hester et al. in Google's AI team have proposed the Demonstration Deep Q Learning algorithm, which uses demonstration data to greatly speed up the learning process.

Demonstration DQNs have an additional model pre-training phase compared to classical DQNs. First, the

demonstration DQN is pre-trained on exemplary data. The purpose of the pre-training phase is to allow the initialized value function to simulate the demonstrator so that the agent gets a deep neural network with better performance before interacting with the real environment. The update of the deep neural network combines four loss functions: the 1-step double Q-learning loss, an n-step double Q-learning loss, a supervised large margin classification loss, and an L2 regularization loss on the network weights and biases. supervised loss is used to classify the demonstrator's actions. The Q learning loss ensures that the network satisfies the Bellman equation and can be used as a starting point for TD learning. The overall loss function formula is:

$$L(Q) = L_{DQ}(Q) + \lambda_1 L_n(Q) + \lambda_2 L_E(Q) + \lambda_3 L_2(Q) \quad (7)$$

Where, λ is the weight of the different loss functions. the *n*-step returns corresponding to $L_n(Q)$ are：

$$L_n(Q) = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max \gamma^n Q(s_{t+n}, a) \quad (8)$$

The large interval classification loss function is defined as：

$$L_E(Q) = \max[Q(s,a) + l(a_E, a)] - Q(s, a_E) \quad (9)$$

Where, a_E is the action taken by the expert demonstrator in state s, $l(a_E, a)$ is an interval function, it is equal to 0 when $a_E=a$, and positive in the other cases.

In all 42 Atari games, the initial scores of the demonstrator DQNs were higher than those of the preferentially competing double DQN [23-25], with better initial performance.

3.1.3 Least Squares Deep Q Network (LS-DQN)

When a deep reinforcement learning model is used to learn high-dimensional data, the model has a low feature learning rate in some cases and the performance of the model is not satisfactory. When reinforcement learning uses a linear function as a reward-value function approximator, the learning process is more stable and requires only a small number of hyperparameters to be adjusted. Levine et al. [26] combined deep reinforcement learning with linear least squares and proposed the least-squares DQN.

The least-squares DQN is divided into two main stages.

(a) The least-squares DQN optimizes the weights of the deep neural network using the conventional approach. In this case, the optimization targets are the parameters in the entire deep neural network, including all parameters in the hidden and output layers.

(b) After n rounds of parameter updates in the neural network, the goal of supervised learning is constructed using a dataset D with sample number N and the approximated reward value function Qn-1 from the last update: $y_i = r_i + \gamma \max Q^{n-1}(s_{i+1}, a')$. Then consider yi as the true value to be approximated by $Q(s, a)$. To solve the approximation Q^N for the next iteration by supervised learning:

$$Q^N = \arg\min \sum_{i=1}^{N} (Q(s_i, a_i) - (r_i + \gamma \max Q^{N-1}(s_{i+1}, a')))^2 \quad (10)$$

The key to the LS-DQN learning process is the introduction of the Bayesian regularization term used for least-squares updating, which uses the parameters of the last hidden layer of the DQN as a Bayesian prior for fitting the Q-iteration algorithm, preventing overfitting of the data. Experiments show that the deep reinforcement learning algorithm incorporating the least-squares update method performs significantly better than DQN and DDQN in some Atari games.

3.1.4 Average DQNs

In order to improve the stability of deep reinforcement learning algorithm and reduce the negative impact of uncertainty on it, Anschel et al. [27] improved the classical DQN algorithm and proposed the average DQN algorithm. The algorithm saves the output value of each deep neural network and averages the output value of several previous neural networks. Using this method makes the training process of the model more stable. In addition, the model performance is improved by reducing the target approximation error.

The average DQN averages the previously learned Q values.

$$Q_{i-1}(s,a) = \frac{1}{\mathrm{M}} \sum_{\mathrm{m}=1}^{\mathrm{M}} Q(s,a;\theta_{i-\mathrm{m}}) . \quad (11)$$

The updated formula of depth neural network is.

$$y = r + \gamma \max Q_{i-1}(s,a) . \quad (12)$$

It is shown that an appropriate increase in the size of M not only allows the model to learn better strategies, but also reduces the overestimation of Q values, significantly improving the stability and game performance of the model. Moreover, since the average DQN does not modify the deep neural network structure of the DQN, the average DQN can be combined with other improved DQN models, such as DDQN, competitive network structure and preferred experience replay.

3.1.5 Dueling DQN

Wang et al. [28] proposed the Dueling DQN by dividing the neural network into two parts. One part is

the advantage function part and the other part is the value function part. The output value of the advantage function part is affected by both the state s and the action a of the input, while the output value of the value function part is only related to the current state s and not to the specific action a to be performed. Ultimately, the output value of DuDQN is given by the following formula:

$$Q(s,a;\theta,\alpha,\beta) = \mathrm{V}(\mathrm{s};\theta,\beta) + \mathrm{A}(\mathrm{s},\, a;\theta,\alpha) \quad \textbf{(13)}$$

Where, s is the input state, a is the action performed. θ is the weight of the input and hidden layers of the deep neural network. β is the neural network parameter of the value function part and α is the neural network parameter of the advantage function part. $\mathrm{V}(\mathrm{s};\theta,\beta)$ is the output value of the value function part. $\mathrm{A}(\mathrm{s},\, a;\theta,\alpha)$ is the output value of the advantage function part.

Without processing, it is impossible to determine the difference between the neural network of the advantage function part and the neural network of the value function part. Therefore, some appropriate adjustments were made to the previous Q-value formula for the final Q-value calculation. The adjusted formula is:

$$Q(s,a;\theta,\alpha,\beta) = \mathrm{V}(\mathrm{s};\theta,\beta) + \mathrm{A}(\mathrm{s},\, a;\theta,\alpha) - \max_{a'\in A} A(s,a';\theta,\alpha) \quad \textbf{(14)}$$

In practical use, the mean value of the dominant function is usually used instead of the maximum value of the dominant function to solve the problem, which improves the stability of optimization to some extent under the premise of ensuring the performance.

Since DuDQN improves DQN only in terms of neural network structure, DuDQN can be combined with other improved DQN models to form a more stable deep reinforcement learning model. Experiments have shown that the DuDQN model that combines the DDQN and preferred experience replay mechanisms performs better in some Atari games than the DQN and DDQN models that only use preferred experience replay. In addition, Raghu et al. [29] combined DDQN with DuDQN and proposed the Dueling Double-Deep Q Network. They used this model to address the problem of learning the best treatment strategy for sepsis and achieved better results.

3.1.6 Deep Recurrent Q Network (DRQN)

The limitations of DQNs are revealed when they are used to make game decisions. DQN has limited memory capacity. Also, good game performance of the DQN is strongly correlated with the integrity of the game screen. To solve this problem, Hausknecht et al. [30] proposed a Deep Recurrent Q Network (DRQN) by combining long- and short-term memory networks [31] with DQNs.

Previously, when a DQN is used to make game decisions, the last four frames of the game are extracted as input to the deep neural network, but then the DQN is only able to learn these four frames. In other words, the DQN can only remember 4 frames. In some games, when the agent needs to analyze and make decisions based on more than 4 frames, the model cannot satisfy Markovianity: the future state depends on the present state. The DRQN model replaces the first fully connected layer in the DQN to LSTM and uses LSTM to remember the previous training process.

The neural network structure of the DRQN is shown in Figure 2.

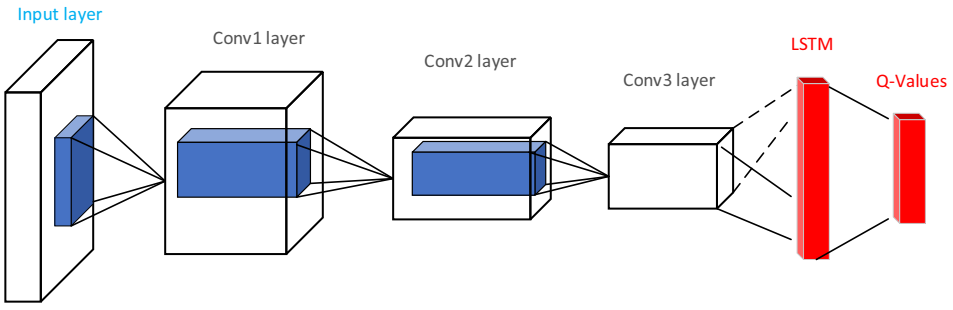


Figure 2. Network structure of the DRQN

3.1.7 Bootstrap DQN

How to learn to explore effectively remains a major challenge for reinforcement learning. The agent takes a series of actions in anticipation of maximizing the cumulative reward value. However, unlike the standard planning questions, agents in reinforcement learning learn through their own post-action experiences and do not have any knowledge of the environment prior to learning. Thus, the agent's search strategy for the environment is critical to the training and learning of the overall model. Common random jitter search strategies such as the ε-greedy search algorithm are used. This algorithm attempts to randomly select the action to be performed with a certain probability by not selecting the action that has the highest payoff in the current state. The probability of randomly selecting an

action can be adjusted by ε.

However, in complex situations, the ε-greedy algorithm is difficult to achieve the desired results. This is because it is difficult for a simple random strategy to reach a state far from the current state. The whole process is still searching around the Q value of the neural network. In other words, a simple random strategy does not affect the general direction of the neural network search. To solve this problem, Osband et al. [32] proposed the Booststrapped DQN (Booststrapped DQN) algorithm, which is the first practical reinforcement learning algorithm that combines reinforcement learning with a deep adventurous exploration process.

The bootstrapped DQN network structure includes a shared network and a K bootstrapped head (Head) network. The shared network is used to learn the joint feature representation of all data. Each bootstrap head is trained on its bootstrap sample dataset. The bootstrap sample data is obtained from a random sample of the overall sample using the bootstrap algorithm [33]. The usual form of the bootstrap algorithm is to randomly sample M times with return from the original sample set with sample number M. The resulting sample set is the bootstrap sample set. The bootstrap DQN collects the Q value of each bootstrap head before each round of training, and the agent explores based on the Q value of the bootstrap head until the end of the round. Then, in the next round, one of the bootstrapped heads is selected at random for exploration until the end of the training round.

The network structure of the bootstrap DQN is shown in Figure 3.

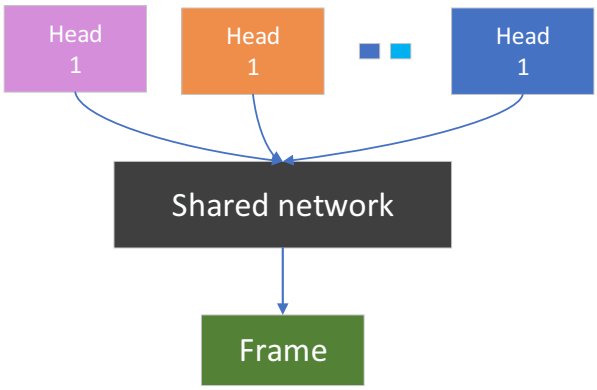


Figure 3. Network structure of the bootstrap DQN

3.1.8 Priority Experience Replay (PER)

By storing each training sample in a memory bank, the classical DQN model allows the model to remember previously learned experiences and reduces the correlation between adjacent training data. However, DQN only uses a random method to draw training samples from the memory bank with equal probability of each training sample being drawn, and does not take into account that the importance of different samples is different. Therefore, Schaul et al. proposed a preferential experience replay method that allows samples to be drawn based on the importance of the training samples. More important training samples have a higher probability to be drawn, thus the training of the Q network is more efficiency.

The importance of the samples in the memory bank is determined by the time-difference score (TD) error. In a Q network, the TD error is the difference between the target Q value computed by the target Q network and the current Q value computed by the current Q network. The larger the TD error, the greater the effect of its sample on the propagation of the neural network. Therefore, the larger the absolute value of TD error, the higher the probability that a sample will be extracted. The random priority sampling method is used when sampling samples of different priority levels. That is, an importance sampling mechanism is used to avoid the bias caused by the update process. Experiments have shown that in some Atari games, the DQN and DDQN models that incorporate the priority experience replay mechanism perform better than the games that use the DQN and DDQN models alone.

3.1.9 DQNs that Introduce Internal Fear Mechanisms

The agent can make catastrophic mistakes when using reinforcement learning for real tasks. And, due to the optimization of neural networks using function approximation, agents will always periodically enter more dangerous states. To enable agents without memory functions to avoid catastrophic mistakes, Lipton et al. [34] propose DQNs that introduce the Intrinsic Fear (IF) mechanism to alleviate these problems by avoiding dangerous states.

In classical DQNs, the neural network is optimized with the goal of maximizing the cumulative reward value. In contrast, in DQNs that introduce Intrinsic Fear, the neural network is optimized with the goal of minimizing the probability of a catastrophic state. Compared to classical DQNs, DQNs that introduce internal fears also have a hazard model. This hazard model is a biclass neural network with the same input and hidden layers as the DQN's Q network, and the output layer consists of two neurons. The role of the hazard model is to predict the likelihood of a catastrophe occurring within a short period of time for the agent in its current state.

Introducing internal fears, the formula for calculating the optimal target value of DQN is:

$$y^{IF} = r + \max Q(s', a'; \theta_Q) - \lambda \cdot d(s'; \theta_d). \quad \textbf{(15)}$$

In the formula, d(-) is the hazard model. λ is the fear factor that determines the size of the effect of the internal fear mechanism on the Q function update.

Experiments have shown that for some tasks, DQN that introduces internal fear can enjoy the benefits of

using function-approximation optimization neural networks while avoiding permanent catastrophe. But it does not have an accurate prediction of the dangerous state in the state space. In a problem, if a single action from any state leads to a catastrophe with high probability, this model will perform poorly.

3.1.10 Summary

This summary presents nine improved approaches to DQN based on value functions. The various versions of DQN-based improvements are classified according to the different approaches to DQN improvement. These models can be broadly categorized into three main groups: improvements to the algorithm logic, improvements to the neural network structure, and improvements that introduce new mechanisms. The proposed problem solving, improvement methods, and experimental results of the various improved algorithms are presented in the following table:

Table 1. Characteristics and experimental results of the DQN improvement method

Model	Issues to be addressed	Improved methodology	Experimental results
DDQN	Overestimation of action value functions in Q learning	Decoupling the selection of target Q-value actions and the computation of target Q-values using the target Q network	Outperforms DQN in Atari games
DQfD	Poor performance of the model after initialization	During the pre-training phase, the model is allowed to learn from the demonstration data.	Outperforms DQN at the start of most Atari games
LSDQN	The model does not perform well when learning high-dimensional data	Combining Least Squares and DQN to improve model stability	Outperforms DQN in Atari games
Average DQN	The instability and variability of the model during training can negatively affect it	Averaging the Q values of the model output from previous training sessions	Reduced over-estimation of Q values and better game performance in Atari and Gridworld games
Dueling DQN	Unsatisfactory convergence rate and stability of the model	Dividing the output layer of the Q network in the DQN into two parts to make the functionality of the network clearer	DuDQN combining the DDQN and preferred experience replay mechanisms outperformed DQN and DDQN using only preferred experience replay in the Atari game
DRQN	During training, model observations of the environment are erroneous and not always complete.	Combining a long- and short-term memory network with a DQN gives the DQN the ability to remember the game environment	DRQN outperformed DQN when trained with partial observations and evaluated with full observations
Bootstrap DQN	Classical DQN's exploration of the environment is shallow, leading to an unsatisfactory convergence rate of the model	Propose a deep exploration strategy and estimate the uncertainty of DQNs	Improved learning speed of models on many Atari games
PER	Training samples of different importance have equal probability of being learned by the neural network, which is not conducive to fast convergence of the model	Prioritize samples based on their TD errors to improve the probability of drawing quality samples	Improved performance of DQN and DDQN in the game
IF	The agents in the model periodically go into catastrophic states	Add a fear model. The model assesses the probability of an agent entering a catastrophic state in a short period of time	Outperformed DQN in the Cart-Pole Game

3.2 Deep Reinforcement Learning Approach Based on Strategy Search

Corresponding to the value function-based reinforcement learning method is the strategy search-based reinforcement learning method. Instead of evaluating the merits of the actions performed in each round, a strategy search-based reinforcement learning approach directly evaluates the strategies performed throughout the round based on sampling, and then uses maximizing the cumulative returns as the target optimization model.

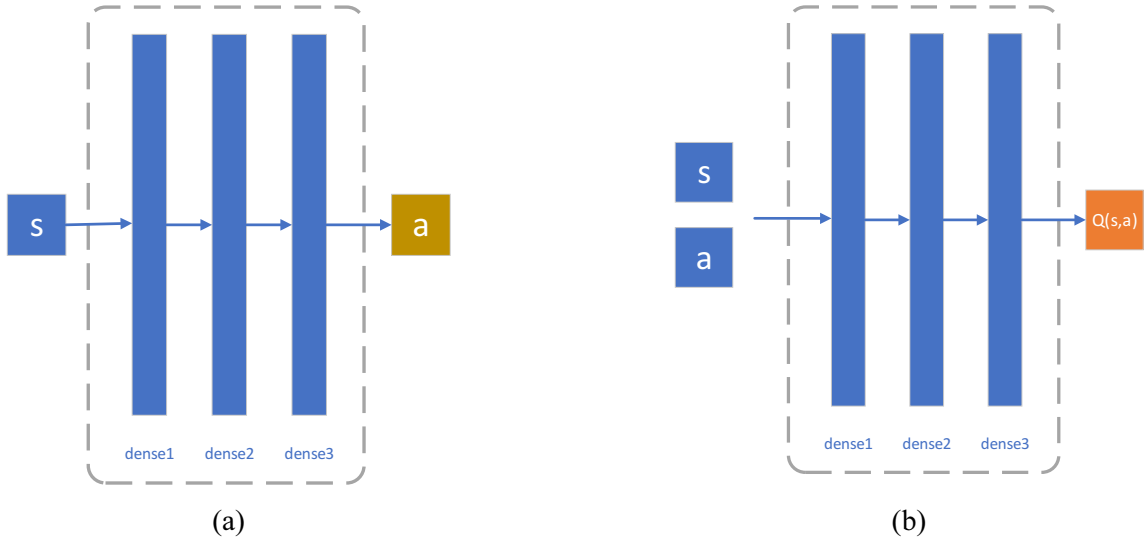
3.2.1 Actor-Critic (AC) Algorithm

For tasks where the action space is continuous or the spatial dimension is too high, reinforcement learning methods based on approximation of value functions are usually difficult to obtain a more optimal solution.

Reinforcement learning methods based on strategy search can directly parameterize the strategy and approximate the optimal strategy, which works better results. However, the strategy search algorithm is weak in evaluating the strategies, which is not conducive to the convergence of the whole model.

The Actor-Critic algorithm is a combination of value function-based and strategy gradient-based methods [35-37]. The algorithm consists of two neural networks, which are the Actor network and the Critic network. The role of the Actor network is to analyze the current state of the environment and make decisions about the actions to be performed, while the role of the Critic network is to evaluate the decisions made by the agent, and the Critic network will continuously update itself as training progresses.

The network structure of the AC algorithm is shown in Figure 4.



(a) (b)

Figure 4. Actor network (a) and critic network (b)

The updated formula of the Actor network is:

$$\theta_{a_i} = \theta_{a_i} + \alpha \sum_t \nabla_{\theta_{a_t}} \ln \pi_{\theta_{a_t}}(s_t, a_t) v_t . \tag{16}$$

Where, θ_{at} is the neural network parameter for the Actor network. θ_{vt} is the neural network parameter for the Critic network.

The updated formula for the Critic network is.

$$\theta_{v_i} = \theta_{v_i} + \alpha' \sum_t \nabla_{\theta_{v_t}} (r_t + \gamma V^{\pi_{\theta_{i-1}}}(s_{t+1}; \theta_{v_i}) - V^{\pi_{\theta_i}}(s_t; \theta_{v_i}))^2 . \tag{17}$$

Where, the input and hidden layers of the Actor network and Critic network have the same neural network structure.

3.2.2 Deep Deterministic Policy Gradient (DDPG)

Lillicrap et al. [38] combine two neural networks in the AC algorithm with the idea of DQN to propose a reinforcement learning model based on the Deterministic Policy Gradient: Deep Deterministic Policy Gradient.

Classical DQN performs well in discrete and low-dimensional action spaces. However, when DQN is trained in a continuous and high-dimensional action space, they usually discretize the continuous action space, which leads to the phenomenon of "dimensional catastrophe". In other words, the number of actions in the agent's action space increases exponentially with increasing dimensionality. This phenomenon will hinder the updating and optimization of neural networks. DDPG uses the empirical replay mechanism in DQN and a separate target network to reduce the correlation between neighboring training samples and increase the stability and robustness of the algorithm. Since classical DQN can not handle tasks with continuous action spaces, DDPG uses AC approach based on the DPG algorithm.

The empirical replay mechanism used in DDPG is identical to DQN, but the target network is updated in a different way. In DQN, the target network does not participate in the training of the model and its parameters are derived from the Q network assignment. In DDPG, however, the respective target networks of Actor and Critic participate in the training of the entire model and are updated in a slower manner. In this way, DDPG improves the stability of the learning process.

$$\theta^- = \tau\theta + (1-\tau)\theta^- . \tag{18}$$

$$w^- = \tau w + (1-\tau)w^- . \tag{19}$$

Where, θ^- is the parameter of the Actor target network. w^- is the parameter of the Critic target network. τ controls the rate at which the target network is updated.

The Schematic illustration of DDPG is shown in Figure 5.

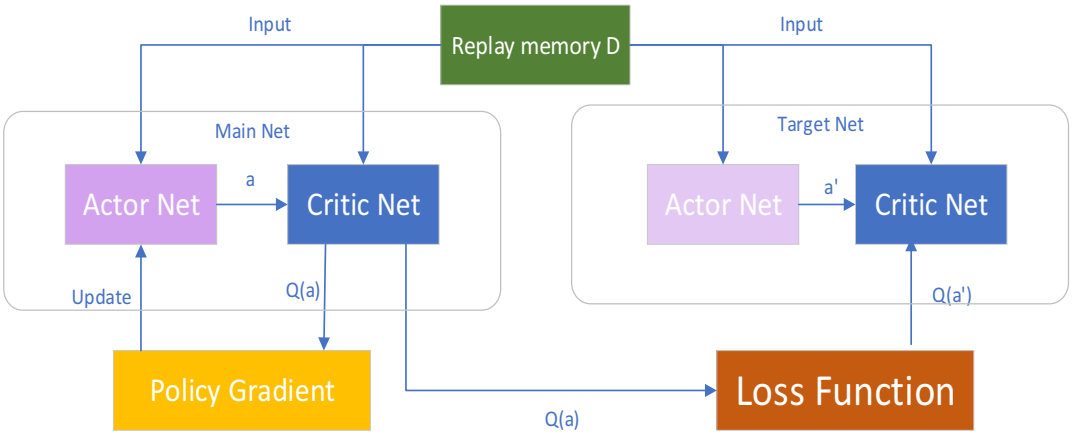


Figure 5. Schematic illustration of DDPG

Experiments show that DDPG can achieve good results in more than 20 physical simulation tasks using the same learning algorithm, network structure and network parameters. This shows that DDPG has good robustness. Not only that, DDPG can directly use raw pixels as input data into the model for end-to-end learning capabilities. However, DDPG, like most model-free reinforcement learning methods, requires a large number of training samples and training time to enable the model to have a more optimal performance when the model is trained.

3.2.3 Probability Surrogate Action Deterministic Policy Gradient (PSADPG)

Wang and Jing [39] proposed surrogate agent-environment interface (SAEI) in reinforcement learning. They developed the probability surrogate action deterministic policy gradient (PSADPG) algorithm based on SAEI. SADPG is an improved DDPG algorithm that allows continuous control of discrete actions.

The Schematic illustration of surrogate agent-environment interface is shown in Figure 6.

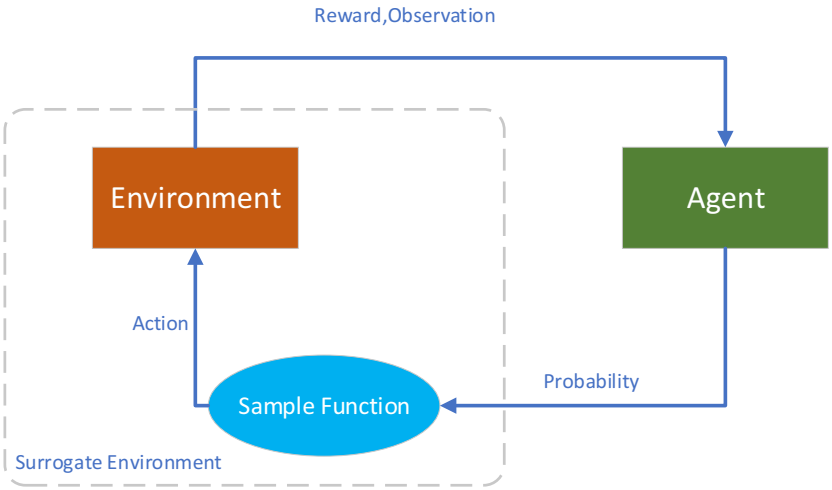


Figure 6. Schematic illustration of PSADPG

PSADPG does not directly adopt the actions obtained from decisions made by the agent, but rather by extracting the sampling steps from the decision task of the agent and integrating them into the environment. Experiments show that PSADPG performs comparable to DQN in Acrobot with discrete control tasks.

3.2.4 DDPG from Demonstrations (DDPGfD)

When learning in a high-dimensional complex space using a DDPG model, it is difficult to define a reward function with excellent performance. To solve this problem, Vecerik et al. [40] proposed the demonstration DDPG model. The model is a common model-free off-strategy model that approximates the demonstration data as a reward value function with good performance to train and optimize the model.

There are five differences between DDPGfD and DDPG:

(1) The data from the manual demonstration is first converted into the form of a sample of reinforcement learning data: (s, a, s0, r). The demonstration transfer

data are then placed in the memory bank of the reinforcement learning model before training begins. In addition, these demonstration transfer data will be permanently stored in the memory bank and the amount of these data will not be reduced as the training progresses.

(2) The sampling ratio between the model transfer data and the agent data in the memory bank is automatically adjusted by a priority replay mechanism.

(3) Both single-step and n-step reward values are used to update the deep neural network in the model.

(4) The deep neural network is updated multiple times in each round. The DDPG is updated only once in each round, which significantly reduces the efficiency of using the samples in the memory. Thus, taking a multiple update approach allows the agent to fully learn from the data from previous interactions with the environment.

(5) Add L2 regularization on both the Actor and Critic networks to stabilize the final learning performance.

The loss function for the Critic network is calculated as:

$$L_{Critic}(w) = L_1(w) + \lambda_1 L_n(w) + \lambda_2 L_{reg}^C(w) \quad (20)$$

The updated gradient for Actor network is:

$$\nabla_\theta L_{Actor}(\theta) = -\nabla_\theta J(\theta) + \lambda_2 \nabla_\theta L_{reg}^C(\theta). \quad (21)$$

3.2.5 A3C

Mnih et al. [41] proposed an asynchronous update reinforcement learning model (A3C) based on the AC algorithm, in which four asynchronous training reinforcement learning algorithms were implemented, namely one-step Q-learning, one-step Sarsa, n-step Q-learning, and advantage actor-critic, respectively.

Compared to Actor-Critic, A3C is optimized in the following three aspects:

In the first aspect, A3C creates multiple agents that perform actions and learning in multiple environments without interfering with each other. In other words, A3C creates multiple agents for training and learning in multiple threads, and also creates a shared public network. The agents in each thread independently interact with and obtain empirical data from the environments, which operate independently of each other. When the agents in each thread have interacted with the environment to a certain amount of data, they begin to compute the gradients of the neural network loss function in their own threads. This gradient information is then passed to the public neural network and the public network is instructed to make updates. Multiple threads independently interact with the environment and pass the gradient information to the shared network. At regular intervals, the neural networks in the threads are synchronized with the parameters of the public network.

In the second aspect, in Actor-Critic, two different Actor networks and Critic networks are used. Whereas in A3C, the two networks are put together. The Actor network and Critic network share an input layer, the Actor network outputs state values and the Critic network outputs the corresponding strategies. The network structure of A3C is shown in Figure 7.

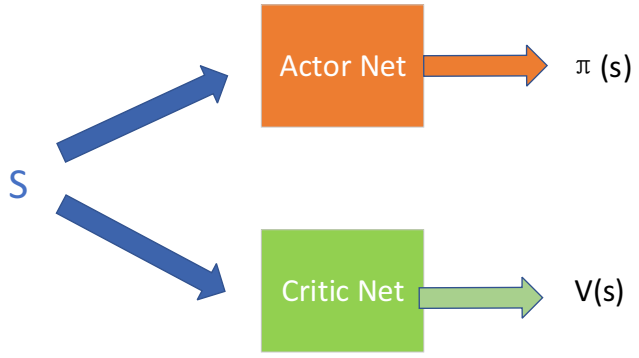


Figure 7. Network structure of A3C

In the third aspect, a natural gradient update method based on the advantage function is proposed, and regularization is used to reduce the variance of the strategy gradient, thus ensuring the stability of strategy learning. The expression for the advantage function is.

$$A(S,t) = R_t + \gamma R_{t+1} + \ldots + \gamma^{n-1} R_{t+n-1} + \gamma^n V(S') - V(S) \quad (22)$$

Experiments have shown that in many of the Atari 2600's gaming tests, A3C is used less than previous GPU-based reinforcement learning algorithms. This may be because the A3C can run on servers with multi-core, multi-threaded CPUs. In addition, A3C performs better than other classical deep reinforcement learning models on some continuous motion control problems. A3C is now the most versatile and successful deep reinforcement learning algorithm.

3.2.6 GPU-based A3C (GA3C)

Babaeizadeh et al. [42] propose an updated version of A3C: the GPU-based A3C algorithm (GA3C). GA3C takes advantage of the GPU's fast vector operations to speed up the training rate of the entire algorithm. The A3C algorithm, however, is based on the processor's multi-threaded capabilities to increase the training rate of the algorithm.

The GA3C has three main components [43].

(1) Agent. the GA3C agent is the same as the A3C agent. The agent selects actions and collects samples based on the learned strategy. However, the GA3C agents do not need to make a copy of the model respectively. They simply add the current state to the prediction queue as a request before selecting an action. After the agent performs a certain number of steps, it

will work backwards to calculate the total return for each step and finally add the total return and the experience data generated during training to the training queue.

(2) Predictor. the request samples from the prediction queue are taken out of the queue and entered as training data into the GPU's DNN model. When the prediction is complete, the Predictor returns the predicted results to the corresponding agent. To reduce latency, multiple predictors can be run simultaneously.

(3) Trainer. the request samples from the training queue are taken out of the queue and input as training data into the GPU's DNN model. To reduce latency, multiple trainers can be run simultaneously.

Experiments show that the training rate of GA3C is higher than that of the multi-core processor-based A3C, and GA3C reduces the memory consumption during training. However, GA3C has the following two problems.

(1) It is necessary to coordinate the number of Agents, Predictor and Trainer in GA3C.

(2) There may be a strategy delay. That is, the strategy that produces the current training sample is not the current strategy to be updated, resulting in instability of the algorithm.

3.2.7 Summary

This summary first introduces the basic principles of the reinforcement learning algorithm based on strategy search. The original AC algorithm is then introduced and improved reinforcement learning algorithms based on the AC algorithm are presented, including DDPG, which combines the AC algorithm and DNN; DDPGfD that can learn from a demonstration sample; PSADPG, which can continuously control discrete actions; A3C and the GPU version of A3C, which will enable asynchronous updates. The algorithms described above, along with their respective solutions and experimental results, are shown in the following table.

Table 2. Characteristics and experimental results of the DDPG and A3C improvement method

Model	Issues to be addressed	Improved methodology	Experimental results
AC	For tasks that are continuous in the action space, reinforcement learning methods based on value functions are often difficult to achieve a more optimal solution	Combine Actor and Critic networks to leverage the strengths of each	More stable algorithms and shorter training times
DDPG	The phenomenon of "dimensional catastrophe" occurs when reinforcement learning methods are trained in a continuous, high-dimensional action space.	Combining AC algorithms and DQN ideas to make them applicable to continuous action domains	Can solve more than 20 physical simulation tasks with performance comparable to traditional planning algorithms
PSADPG	Reinforcement learning cannot be applied in the task of continuous control of discrete actions by Yu Yao	The sampling step is extracted from the decision task of the intelligences and integrated into the environment.	PSADPG rivals DQN's performance in Acrobot with discrete control tasks
DDPGfD	It is difficult to assess rewards for each action in the actual task	The model is trained and optimized by approximating the demonstration data to a reward value function with good performance.	DDPGfD performs better than DDPG in a task in which four simulated robots insert objects at specified locations
A3C	The usual AC algorithm uses only one agent for learning, which is inefficient	Create multiple agents in multiple virtual environments, train in parallel, and achieve asynchronous updates	In the Atari game, A3C's training time is less than that of GPU-based algorithms.
GA3C	A3C uses processor multithreading to accelerate learning, but GPU vectoring is even faster	Increase the amount of training data generated and processed per second using GPUs	Faster training rate than CPU-based A3C algorithm

4 Application of Deep Reinforcement Learning

4.1 Robot Control

In the field of robot control, formalizing a specific task and determining a reward value function that performs well is difficult [44]. In addition, the reward value of the environment in a virtual scenario can be fed directly to the agent in the form of a signal. However, in a robot control task, the robot needs the spider to observe the environmental feedback and summarize the reward. This process places considerable demands on the robot's perceptual abilities.

Due to the difficulty of applying reinforcement learning to physical tasks, current robot control tasks

are very simple [45]. Levine [46] et al. combined trajectory optimization and supervised learning to train humanoid robots in a simulation environment to achieve walking motion gait control. Schulman et al. [47-48] proposed TRPO and PPO algorithms and implemented stable training algorithms using the actor-critic framework. Peng et al [49] used hierarchical deep reinforcement learning algorithms to train the motion of a 3D bionic robot in a simulation environment. Recently, more efficient and stable deep reinforcement learning algorithms such as MPO algorithm [50], SAC algorithm [51], and TD3 algorithm have been proposed. In addition to training the motion gait of the bionic robot in the simulation environment, Hwangbo et al [52] applied the training results in the simulation environment directly to the ANYmal quadrupedal robot to achieve stable and efficient quadrupedal motion gait control; Haarnoja et al [53] used an improved SAC algorithm to train directly on the Minitaur quadrupedal robot for multiple motion gait control.

There are a number of problems with current reinforcement learning algorithms in the field of robot control:

(1) Training samples are used inefficiently. Humans can quickly learn motor gait such as walking, running and jumping, whereas the existing reinforcement learning methods require a lot of trial and error. Therefore, it takes a lot of time to collect and train samples. Not only that, the large number of interactions will also lead to wear and tear on the bionic robot.

(2) Inability to effectively multi-task learning. Most of the existing reinforcement learning algorithms can only learn a single task. When learning other types of tasks, they need to be retrained.

(3) Poor migration from the simulation environment to the actual platform. Most of the existing reinforcement learning methods build a simulation model of the robot and train the robot's gait in the simulation environment, however, the actual model and the simulation model are often quite different, which leads to a large deviation when the strategy trained in the simulation environment is directly transferred to the actual robot.

(4) Poor robustness. In actual robot control tasks, the information obtained by the robot is incomplete due to errors in the sensors. However, the information observed by robots in the simulated environment is complete. Therefore, the model trained in this situation does not perform well in the real task.

4.2 Parameter Optimization

In traditional neural networks, the parameters of the network are generally optimized by methods such as gradient descent. However, it takes a lot of time and effort to repeatedly adjust the learning rate in the gradient descent method. Therefore, through some mechanism to automatically set the learning rate and initial parameters of the network according to different tasks before the training of the network will greatly improve the training rate of the model.

To address the above problem, Hansen et al. [54] proposed a gradient descent method based on Q-values. The method can automatically set different learning rates according to the specific task. Andrychoiwicz et al. [55] proposed an agent self-learning model. The model learns to optimize the parameters of other neural networks by training one neural network. Liu et al [56] combined the dual elite collaborative algorithm and deep learning to propose an improved deep reinforcement learning model. The model can optimize the parameters of semi-variant functions in the Kriging interpolation method and improve the interpolation accuracy of the algorithm. In addition, Google uses the reinforcement learning algorithm to optimize the parameter settings of data center servers and save 40% of power energy.

In summary, the application of deep reinforcement learning in the direction of optimizing parameters is still at a preliminary stage for the time being. However, deep reinforcement learning has been proven to have considerable potential in this direction. With more and more experts and researchers studying this direction in depth, the method of deep reinforcement learning model for automatic parameter optimization will certainly be widely used in various optimization tasks.

4.3 Other Directions

In addition to robot control, the field of algorithmic parameter optimization, reinforcement learning has applications in computer vision, game decision making, natural language processing, automated driving, and game theory [57-59].

In the field of machine vision, deep reinforcement learning models with visual perception capabilities can predict the actions that need to be performed afterwards based on the original image picture. Oh et al [60] performed a long-term prediction task for high-dimensional video images by controlling the input of actions with a deep reinforcement learning model. Caicedo et al. [61] combined CNN and DQN models with image perception capabilities to achieve precise localization of target regions in images.

In most complex strategy game tasks, the decision making ability of agents in deep reinforcement learning now comprehensively surpasses that of the top human gamers. Siliver et al. [62] added two deep convolutional neural networks to AlphaGo and combined the training methods of reinforcement learning with supervised learning. The end result is a model with a level of performance that surpasses that of human Go world champions.

5 Challenges and Perspectives for Reinforcement Learning

Although more and more researchers are exploring and extending deep reinforcement learning, there are still many challenges in applying it to practical tasks. Based on these challenges and the limitations of reinforcement learning itself, we attempt to prospect the development direction of deep reinforcement learning.

In the process of strategy implementation, the agent, on the one hand, needs to fully learn the strategies it has already learned and avoid blind experimentation. On the other hand, the agent needs to explore as many unknown strategies as possible in order to discover the best strategy to solve the problem. The currently used stochastic strategy allows the agent to traverse various states to avoid the agent falling into a local optimal solution. However, the random strategy execution makes the agent search back and forth between unnecessary states, which wastes more training time and computational resources. Therefore, how to balance exploring the environment and exploiting the experience may be a key research direction for the future.

(1) How to effectively evaluate the merits of a strategy is the key to improving agent learning efficiency. At present, strategy evaluation relies mainly on the reward-value function. However, in some complex decision-making tasks, it is difficult to design a reward-value function with good performance. Not only that, there is not always a suitable positive or negative feedback for every action in the actual task. When the state space dimension is high, the problem of sparse reward value will occur. Therefore, how to set a good reward value function according to different tasks and avoid the sparse reward problem will be the focus of future research.

(2) Since most of the current research in the field of deep reinforcement learning focuses on model-free methods, a large number of training samples are required to train the agent using this method. It is unrealistic to use agents and the environment for a large number of interactions in a real task. A model-based (model-based) approach can effectively solve this problem and improve the efficiency of sampling. Therefore, model-based methods for deep reinforcement learning will receive much attention in the future.

(3) Although deep neural network-based reinforcement learning can handle many high-dimensional input problems, it still requires a large number of samples. To accelerate the learning rate of deep reinforcement learning, the model can be pre-trained using the knowledge previously gained from the relevant task. At this point, the combination of migration learning and deep reinforcement learning can reduce the number of interactions between the agent and the environment. Therefore, using migration learning to allow deep reinforcement learning methods to solve multiple different tasks after a small number of debugging may be a key research focus in the future.

6 Conclusions

Deep reinforcement learning, as one of the most popular research directions in the field of artificial intelligence, has gained the attention of many professionals. This paper first introduces the current research status and development trend of deep reinforcement learning, and systematically introduces the basic algorithm of reinforcement learning. Then the deep reinforcement learning method based on value functions and its improvement methods are introduced in detail, including methods to improve the training logic of algorithms: DDQN, DQNfD, Least Squares DQN and Average DQN, methods to improve the network structure: Dueling DQN, DRQN and Bootstrap DQN, methods to improve by adding new mechanisms: DQN with preferential replay and DQN with internal fear mechanism, followed by a detailed description of the machine improvement method of deep reinforcement learning method based on strategy gradient. Then, the applications of deep reinforcement learning in robot control, algorithm parameter optimization and other fields are introduced. Finally, the future of deep reinforcement learning is envisioned based on the current limitations of deep reinforcement learning. Although there are still many challenges in the field of deep reinforcement learning, as research and the theory continues to develop, deep reinforcement learning will become an important part of building general artificial intelligence systems.

7 Conflicts of Interest

All authors declare that: (a) no support, financial or otherwise, has been received from any organization that may have an interest in the submitted work; and (b) there are no other relationships or activities that could appear to have influenced the submitted work.

Acknowledgments

This work was supported in part by the Hubei Provincial Major Science and Technology Special Projects under Grant 2018ABA099, in part by the Natural Science Foundation of Hubei Province under Grant 2018CFB408, in part by the National Natural Science Foundation of China under Grant 61272278, in part by the innovation and education promotion fund of science and technology development center of Ministry of education in 2019 under Grant 2018A01038, and in part by the Wuhan Polytechnic University Talent

Introduction (Training) Scientific Research Project under Grant 2019RZ02.

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, *A brief survey of deep reinforcement learning*, https://arxiv.org/abs/1708.05866, 2017.

[2] J. Hou, H. Li, J. Hu, C. Zhao, Y. Guo, S. Li, Q. Pan, A review of the applications and hotspots of reinforcement learning, *2017 IEEE International Conference on Unmanned Systems (ICUS)*, Beijing, China, 2017, pp. 506-511.

[3] A. Gosavi, Reinforcement learning: A tutorial survey and recent advances, *INFORMS Journal on Computing*, Vol. 21, No. 2, pp. 178-192, Spring, 2009.

[4] C. J. Watkins, P. Dayan, Q-learning, *Machine learning*, Vol. 8, No. 3-4, pp. 279-292, May, 1992.

[5] Q.-M. Fu, Q. Liu, H. Wang, F. Xiao, J. Yu, J. Li, A novel off policy Q (λ) algorithm based on linear function approximation, *Chinese Journal of Computers*, Vol. 37, No. 3, pp. 677-686, March, 2014.

[6] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, *The International Journal of Robotics Research*, Vol. 32, No. 11, pp. 1238-1274, September, 2013.

[7] Y. Wei, M. Zhao, A reinforcement learning-based approach to dynamic job-shop scheduling, *Acta Automatica Sinica*, Vol. 31, No. 5, pp. 765-771, September, 2005.

[8] E. Ipek, O. Mutlu, J. F. Martínez, R. Caruana, Self-optimizing memory controllers: A reinforcement learning approach, *ACM SIGARCH Computer Architecture News*, Vol. 36, No. 3, pp. 39-50, June, 2008.

[9] G. Tesauro, TD-Gammon, a self-teaching backgammon program, achieves master-level play, *Neural computation*, Vol. 6, No. 2, pp. 215-219, March, 1994.

[10] X. Wang, T. Sandholm, Reinforcement learning to play an optimal Nash equilibrium in team Markov games, *Advances in neural information processing systems*, Vancouver, British Columbia, Canada, 2002, pp. 1603-1610.

[11] Q. Liu, J. Zhai, Z. Zhang, S. Zhong, Q. Zhou, P. Zhang, J. Xu, A Survey on deep reinforcement learning, *Chinese Journal of Computers*, Vol. 41, No. 1, pp. 1-27, January, 2018.

[12] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature*, Vol. 521, No. 7553, pp. 436-444, May, 2015.

[13] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, *Proceedings of the 25th international conference on Machine learning*, Helsinki, Finland, 2008, pp. 1096-1103.

[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *Journal of machine learning research*, Vol. 11, pp. 3371-3408, March, 2010.

[15] I. Sutskever, G. E. Hinton, G. W. Taylor, The recurrent temporal restricted boltzmann machine, *Advances in neural information processing systems*, Vancouver, British Columbia, Canada, 2008, pp. 1601-1608.

[16] G. E. Hinton, A practical guide to training restricted Boltzmann machines, in: G. Montavon, G. B. Orr, K. R. Müller (Eds.), *Neural networks: Tricks of the trade*, Springer, 2012, pp. 599-619.

[17] H. Lee, R. Grosse, R. Ranganath, A. Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, *Proceedings of the 26th annual international conference on machine learning*, Montreal, Quebec, Canada, 2009, pp. 609-616.

[18] S. Kombrink, T. Mikolov, M. Karafiát, L. Burget, Recurrent neural network based language modeling in meeting recognition, *Twelfth annual conference of the international speech communication association*, Florence, Italy, 2011, pp. 2877-2880.

[19] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, D. Wierstra, *Draw: A recurrent neural network for image generation*, https://arxiv.org/abs/1502.04623, 2015.

[20] K. Simonyan; A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, https://arxiv.org/abs/1409.1556, 2014.

[21] M. Riedmiller, Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method, *European Conference on Machine Learning*, Porto, Portugal, 2005, pp. 317-328.

[22] S. Lange, M. Riedmiller, Deep auto-encoder neural networks in reinforcement learning, *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain, 2010, pp. 1-8.

[23] H. Van Hasselt, A. Guez, D. Silver, *Deep reinforcement learning with double q-learning*, https://arxiv.org/abs/1509.06461, 2015.

[24] T. Schaul, J. Quan, I. Antonoglou, D. Silver, *Prioritized experience replay*, https://arxiv.org/abs/1511.05952, 2015.

[25] L.-J. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Machine learning, Vol. 8, No. 3-4, pp. 293-321, May, 1992.

[26] N. Levine, T. Zahavy, D. J. Mankowitz, A. Tamar, S. Mannor, Shallow updates for deep reinforcement learning, *Advances in Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 3135-3145.

[27] O. Anschel, N. Baram, N. Shimkin, Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, *International Conference on Machine Learning*, Sydney, Australia, 2017, pp. 176-185.

[28] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, *International conference on machine learning*, New York, NY, USA, 2016, pp. 1995-2003.

[29] A. Raghu, M. Komorowski, L. A. Celi, P. Szolovits, M. Ghassemi, *Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach*, https://arxiv.org/abs/1705.08422, 2017.

[30] M. Hausknecht, P. Stone, *Deep recurrent q-learning for partially observable mdps*, https://arxiv.org/abs/1507.06527, 2015.

[31] H. Sak, A. Senior, F. Beaufays, *Long short-term memory*

based recurrent neural network architectures for large vocabulary speech recognition, https://arxiv.org/abs/1402.1128, 2014.

[32] I. Osband, C. Blundell, A. Pritzel, B. Van Roy, Deep exploration via bootstrapped DQN, *Advances in neural information processing systems*, Barcelona, Spain, 2016, pp. 4026-4034.

[33] J. Krolik, G. Niezgoda, D. Swingler, A bootstrap approach for evaluating source localization performance on real sensor array data, *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toronto, Ontario, Canada, 1991, pp. 1281-1284.

[34] Z. C. Lipton, K. Azizzadenesheli, A. Kumar, L. Li, J. Gao, L. Deng, *Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear*, https://arxiv.org/abs/1611.01211, 2016.

[35] J. Peters, S. Schaal, Natural actor-critic, *Neurocomputing*, Vol. 71, No. 7-9, pp. 1180-1190, March, 2008.

[36] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, M. Lee, Natural actor–critic algorithms, *Automatica*, Vol. 45, No. 11, pp. 2471-2482, November, 2009.

[37] T. Degris, P. M. Pilarski, R. S. Sutton, Model-free reinforcement learning with continuous action in practice, *2012 American Control Conference (ACC)*, Montreal, QC, Canada, 2012, pp. 2177-2182.

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, *Continuous control with deep reinforcement learning*, https://arxiv.org/abs/1509.02971, 2015.

[39] S. Wang, Y. Jing, Deep Reinforcement Learning with Surrogate Agent-Environment Interface, https://arxiv.org/abs/1709.03942, 2017.

[40] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, M. Riedmiller, *Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards*, https://arxiv.org/abs/1707.08817, 2017.

[41] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, *International conference on machine learning*, New York, NY, USA, 2016, pp. 1928-1937.

[42] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, J. Kautz, *Reinforcement learning through asynchronous advantage actor-critic on a gpu*, https://arxiv.org/abs/1611.06256, 2016.

[43] L. Wan, X. Lan, H. Zhang, N. Zheng, A Review of deep reinforcement learning theory and application, *Pattern Recognition and Artificial Intelligence*, Vol. 32, No. 1, pp. 67-81, January, 2019.

[44] J. Liu, F. Gao, X. Luo, Survey of deep reinforcement learning based on value function and policy gradient, *Chinese Journal of Computers*, Vol. 42, No. 6, pp. 1406-1438, June, 2019.

[45] F. Y. S. Lin, C. H. Hsiao, Y. F. Wen, S. T. Kuo, Markov Decision Process to Achieve Near-Optimal Admission Control Mechanism for 5G Cloud Radio Networks, *Journal of Internet Technology*, Vol. 20, No. 5, pp. 1561-1573, September, 2019.

[46] S. Levine, V. Koltun, Learning complex neural network policies with trajectory optimization, *International Conference on Machine Learning*, Beijing, China, 2014, pp. 829-837.

[47] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, *International conference on machine learning*, Lille, France, 2015, pp. 1889-1897.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, *Proximal policy optimization algorithms*, https://arxiv.org/abs/1707.06347, 2017.

[49] X. B. Peng, G. Berseth, K. Yin, M. Van De Panne, Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning, *ACM Transactions on Graphics (TOG)*, Vol. 36, No. 4, pp. 1-13, July, 2017.

[50] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, M. Riedmiller, *Maximum a posteriori policy optimization*, https://arxiv.org/abs/1806.06920, 2018.

[51] S. Fujimoto, H. Van Hoof, D. Meger, *Addressing function approximation error in actor-critic methods*, https://arxiv.org/abs/1802.09477, 2018.

[52] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, M. Hutter, Learning agile and dynamic motor skills for legged robots, *Science Robotics*, Vol. 4, No. 26, pp. 1-13, January, 2019.

[53] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, S. Levine, *Learning to walk via deep reinforcement learning*, https://arxiv.org/abs/1812.11103, 2018.

[54] S. Hansen, *Using deep q-learning to control optimization hyperparameters*, https://arxiv.org/abs/1602.04062, 2016.

[55] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, *Advances in neural information processing systems*, Barcelona, Spain, 2016, pp. 3981-3989.

[56] Y. Liu, C. Zhang, Application of Dueling DQN and DECGA for Parameter Estimation in Variogram Models, *IEEE Access*, Vol. 8, pp. 38112-38122, February, 2020.

[57] Q. Zhang, C. Zhou, N. Xiong, Y. Qin, X. Li, S. Huang, Multimodel-based incident prediction and risk assessment in dynamic cybersecurity protection for industrial control systems, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 46, No. 10, pp. 1429-1444, October, 2016.

[58] L. Ma, W. Xie, Y. Zhang, X. Feng, Extreme Learning Machine Based Defect Detection for Solder Joints, *Journal of Internet Technology*, Vol. 21, No. 5, pp. 1535-1543, September, 2020.

[59] R. Fei, Y. Zhu, Q. Yao, Q. Xu, B. Hu, A Deep Learning Method Based Self-Attention and Bi-directional LSTM in Emotion Classification, *Journal of Internet Technology*, Vol. 21, No. 5, pp. 1447-1461, September, 2020.

[60] J. Oh, X. Guo, H. Lee, R. L. Lewis, S. Singh, Action-conditional video prediction using deep networks in atari games, *Advances in neural information processing systems*, Montreal, Quebec, Canada, 2015, pp. 2863-2871.

[61] J. C. Caicedo, S. Lazebnik, Active object localization with deep reinforcement learning, *Proceedings of the IEEE international conference on computer visio*n, Santiago, Chile, 2015, pp. 2488-2496.

[62] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, *Nature*, Vol. 550, No. 7676, pp. 354-359, October, 2017.

Biographies

Junjie Zhang received the B.S. degree in computer science from Wuhan Polytechnic University in 2018. He is currently a master of Wuhan Polytechnic University, China. His research interests include social learning, machine learning, recommender system, and data mining.

Cong Zhang received the Ph.D. degree in computer application technology from Wuhan University, in 2010. He is currently a Professor with the School of Mathematics and Computer Science, Wuhan Polytechnic University, China. His main research interests include multimedia signal processing and artificial intelligence algorithm optimization.

Wei-Che Chien is currently an Assistant Professor with the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include wireless rechargeable sensor networks, 5G mobile networks, AIoT, fog computing and cloud computing.