Application and Optimization of Reinforcement Learning Based on Deep Q-Network (DQN) in Complex Environments

Xiaochi Zhang *

Dalian University of Technology-Ritsumeikan University, International School of Information Science & Engineering, Dalian University of Technology, Dalian, China

* Corresponding Author Email: candletime7@ldy.edu.rs

Abstract. Reinforcement Learning, as an important branch of machine learning, is dedicated to learning how to make optimal decisions to maximize the cumulative reward in an uncertain environment. Deep Q-Network (DQN) combines the advantages of deep learning and reinforcement learning and approximates the Q-value function through deep neural network. Despite the remarkable success of DQN in complex tasks, there are still challenges in continuous action space processing, high-dimensional state space exploration efficiency, adaptability to environmental changes, and delayed reward issues. To overcome these limitations, this paper proposes an optimization of the DQN algorithm, Double-Deep Q-Network, and the concept of DDQN. This paper first analyzes the challenges faced by DQN in complex environments and outline the introduction of DDQN and its comparative analysis with DQN. Then, a series of optimization measures are introduced, including network structure optimization, target network update strategy improvement, dynamic adjustment of exploration strategy and ensemble learning, to improve the performance and stability of DDQN. In addition, the application cases of DDQN in different fields such as autonomous driving, and game AI are explored, and the corresponding improvement strategies are analyzed. Finally, the optimization strategies of DDQN in practical applications are summarized, and its potential applications in future complex decision-making problems are prospected. Through continuous technological innovation and algorithm optimization, DDQN is expected to play a greater role in a wider range of fields and promote the development of intelligent decision systems.

Keywords: Reinforcement learning; double deep Q-Network; application and optimization.

1. Introduction

Reinforcement learning has been widely applied in many fields and achieved remarkable results: Reinforcement learning has achieved great success in the field of game AI. For example, AlphaGo defeated the world champion in the game of Go, demonstrating the ability of reinforcement learning in complex decision-making and strategic planning. In addition, the OpenAI's Dota 2 robot defeated several top professional players, further proving the advantages of reinforcement learning in multi-agent environments. Reinforcement learning has also made significant progress in the field of robot control.

Today, reinforcement learning is more fully developed, but it still faces some challenges, and there is also room for optimization and expansion. First, reinforcement learning algorithms require a lot of data to train, and the data that is actually available often takes a lot of effort. Meanwhile, the quality and diversity of data directly affect the performance of the algorithms. Although the model generalization ability has been improved, reinforcement learning algorithms may still perform poorly when facing unknown environments or tasks. The high demand for computing resources is also a point worthy of attention. The training process of reinforcement learning algorithms requires a large amount of computing resources, which increases the cost and threshold for the application of the algorithms.

Currently, DQN faces many challenges and deficiencies in complex environments. First, it lacks sufficient processing capability for continuous action spaces: DQN is mainly suitable for discrete action spaces and difficult to be directly applied to scenarios requiring fine-grained control. Second, it has inefficient exploration efficiency in high-dimensional state spaces: in complex environments, DQN may fall into local optimal solutions and fail to find global optimal policies.

Moreover, DQN has poor adaptability to environmental changes, and when the environment changes, it is difficult to learn online and adapt quickly, and it needs to be trained again. At the same time, DQN also has a delay problem, and when dealing with tasks with delayed rewards, DQN may have difficulty learning long-term dependence effectively.

The research content of this paper includes the optimization of DQN algorithm, DDQN introduction and comparative analysis. It attempts to introduce a continuous action space processing mechanism: by combining policy gradient methods, DQN can process continuous action spaces. At the same time, a more efficient exploration strategy is designed to accelerate the learning process and reduce the risk of falling into local optima. To enhance adaptability to environmental changes, online learning mechanisms or meta-learning techniques are introduced to improve DQN's rapid adaptability to environmental changes.

2. Reinforcement Learning

A crucial component of machine learning is reinforcement learning, which aims to teach agents how to make the best decisions by allowing them to interact dynamically with their surroundings. Maximizing the cumulative reward—a measure of the total amount of money earned by an agent in the environment—is the ultimate objective of reinforcement learning. Reinforcement learning has five basic elements: agents, which are entities that actively participate in the environment, perform actions, and receive subsequent feedback; The environment, the external domain in which the agent operates, accepts the agent's behavior and responds with updated status and rewards; State, which represents the current condition of the environment, based on which the agent makes decisions about subsequent actions; Action, the behavior choice made by the agent to change the state of the environment; Finally, there is the reward, the immediate gratification or punishment of the environment for the agent's behavior, which is the key signal that guides the agent's learning process.

2.1. Markov Decision Process

Markov Decision Process (MDP) is a basic framework in reinforcement learning, which is used to describe and solve decision-making problems. MDP models the decision-making process through states, actions, transition probabilities, and reward functions.

The stochastic Markov game is neither fully cooperative nor fully competitive so that the agents employ a utilitarian selection mechanism that maximizes the sum of all agents' rewards [1].

According to the Markov property, previous information is irrelevant for predicting the future because all relevant information required for making predictions is already available. Strictly speaking, a state is thought to possess Markov property if its historical records encompass all pertinent information, and once the present state is understood, all previous records become superfluous and the present state is capable of predicting the future.

The core of Markov processes lies in their "memoryless", that is, the current state contains all the useful information needed for future predictions, and that past state information is no longer important for future state transitions.

2.1.1. Markov Property

The Markov property states that a system's future state is only dependent on what it is at present and not on its previous state. In other words, all the data required to characterize the probability distribution of the future state is present in the current state. The fundamental formula for the MDP is as follows.

S is a finite set of states, P is a state transition probability matrix where S_t represents the state of the next moment and S_{t+1}represents the current state.

$$P = (S_{i+1}|S_i) = P(S_{i+1}|S_1, S_2, \ldots, S_t) \quad (1)$$

2.1.2. Introduction and Analysis of Markov Property

The goal of MDP is to find a policy that maximizes the long-term accumulated rewards given the initial state. MDP problems can be solved to find the optimal policy through dynamic programming such as value iteration and policy iteration or reinforcement learning methods such as Q-learning and policy gradients.

The definition of the MDP model's state space, action space, transition probability, and reward function are crucial. The other crucial issue is the MDP optimization and solution. While value iteration and policy iteration, two classic dynamic programming methods, work well in small-scale situations, their computational complexity grows exponentially in vast state and action spaces. Consequently, it has become essential to introduce approximation techniques and distributed computing. Especially, the sample-based reinforcement learning method shows strong adaptability in high dimensional continuous space. At the application level, the actual performance of MDP depends on the modeling accuracy of specific problems and the effectiveness of solving algorithms.

For example, in the now popular autonomous driving, MDP can provide decision support for the system, but it must face the challenges of dynamic environments and uncertainties. Therefore, it is necessary to combine deep learning technology, and deep reinforcement learning has become an effective way to solve complex MDP problems [2].

As the basic framework of reinforcement learning, MDP provides a systematic solution for decision-making problems in theory, but it also needs to be integrated with other algorithms in practical application (Fig.1). Its effectiveness is highly dependent on the modeling of state and action space, the accuracy of transition probability, the design of reward function and the optimization of solving algorithm.

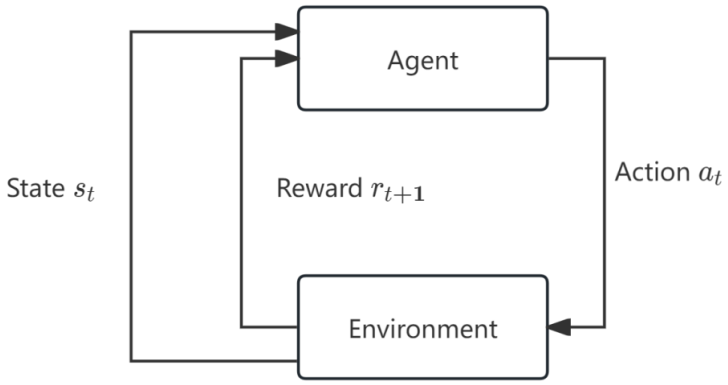

Fig 1. Markov Decision Process [2].

2.2. MDP and Deep Q Network

In DQN, MDP are essential building blocks. Deep Q learning and deep learning are combined in DQN, a reinforcement learning algorithm designed to address complicated decision problems.

MDP has many roles in DQN, starting with environment modeling. MDP provides a framework for DQN, in which the state transitions and reward mechanisms of the environment follow the definition of MDP to describe the agent's interaction with the environment. In addition, there are value function approximation and strategy optimization. In DQN, a deep neural network is used to approximate the action value function (Q function) to evaluate the value of the action performed in a given state. The goal of DQN is to find the optimal strategy, even with action selection rules that expect the maximum cumulative reward. By constantly updating the parameters in the neural network, DQN can gradually approach the optimal strategy, thus improving the decision-making ability of the agent. The joint optimization problem was described as a MDP [3]. Finally, exploration and balance. In DQN, the agent needs to strike a balance between exploring new actions and using known information.

The framework of MDP enables DQN to achieve this balance through, for example, ε-greedy strategies, where random actions are selected with a certain probability to explore the environment, otherwise actions that are currently considered optimal are selected to leverage known information.

2.3. Deep Q-Network

The DQN approach addresses the drawbacks of conventional reinforcement learning techniques when working with high-dimensional state Spaces by fusing deep learning and reinforcement learning. The DeepMind team first presented DQN in 2013, and it soon gained traction as a significant development in the field of deep reinforcement learning [3].

2.3.1. Core Idea and Training Process

The core idea of DQN is to approximate the action value function (Q function) in reinforcement learning using deep neural networks. In the traditional Q-learning algorithm, it uses a Q-table to store the value of each action in each state, but in the case of very large state space and action space, this method becomes unfeasible [4]. DQN solves the "dimensional disaster" problem by training a deep neural network to take the state as input and output the corresponding Q value for each action.

In the training process of DQN, an experience playback pool, an online network and a target network are first initialized. After then, the agent engages with the surroundings, taking actions, seeing the rewards and states that emerge, and recording these encounters in the playback pool. With the progress of training, a batch of experience samples are randomly selected from the playback pool, the target value of each sample is calculated using the target network, and the weight of the online network is updated using the mean square error loss function. During training, the weights of the online network are also regularly synchronized with the weights of the target network to ensure stability and prevent errors when the online network is updated.

2.3.2. The Key Technology of DQN

Experience playback and the target network are used by DQN during the training phase. The experience of the agent's contact with the environment is stored in the experience playback pool via the experience playback mechanism, and randomly selects a batch of experience for training during training, breaking the correlation between samples and improving the sample utilization. DQN combines the advantages of deep neural network and reinforcement learning [4]. The target network is the target value used to calculate the Q value, a network with the same structure but different weights as the online network. The online network is responsible for generating the predicted Q value, and the weights of the target network are periodically copied from the online network to maintain some stability, which is used to calculate the target value and reduce fluctuations during training [5].

2.3.3. Shortcomings and Challenges

Although DQN has achieved remarkable success in many complex tasks, As a reinforcement learning algorithm combining deep learning and Q-learning, DQN performs well in dealing with complex tasks, but it also has some shortcomings. It cannot be applied to continuous action control, and can only deal with short-term memory problems, network convergence problems and overestimation problems [6]. Overestimation is a common problem, which can reduce the performance and stability of the algorithm.

3. Double Deep Q Network

3.1. DDQN Principal Analysis

Because the target network remains unchanged for a period of time, its estimation of the future Q value is too optimistic, campared with DQN, the core advantage of DDQN is that it separates the action selection process from the Q value evaluation process, which effectively solves the common overestimation of Q value in DQN. As mentioned in 2.3, in DQN, the same network is responsible for selecting the optimal action and evaluating the Q value of the action at the same time, so the

estimation of the Q value is often too optimistic, which affects the optimality and stability of the strategy [7]. Therefore, DDQN introduces two independent neural networks-an online network for selecting the optimal action and a target network for evaluating the Q value of the selected action, which improves the accuracy of Q value estimation.

DDQN also inherits the strategy of delaying the update of the target network in DQN. The slow change characteristic of the target network in DDQN can provide a relatively stable value estimate, reduce the fluctuation and noise interference in the training process, and help the agent to converge to the effective policy faster [8].

DDQN also shows stronger generalization ability and adaptability in practical applications. In the face of complex game environments and challenging robot control tasks, DDQN can find effective solutions through its powerful learning and optimization capabilities (Fig.2).

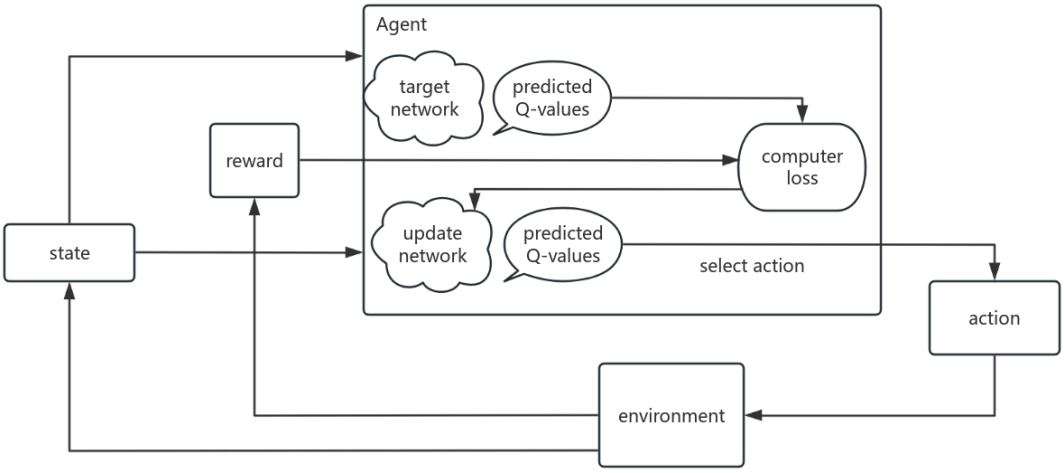

Fig 2. Double-Deep Q-Network Process [8].

3.2. DDQN-based Reinforcement Learning Optimization

Initialization of the Estimation Network and Target Network, two neural networks, is required first. Although the target network's parameters are updated rather slowly, the two networks appear to have the same structure. Experience Gathering Afterwards The sequence of state (S), action (A), reward (R), and future state (S') data that the agent generates during training is kept in the Experience Replay Buffer.

Next, sample sampling is carried out. A batch of data samples is chosen at random for network training from the experience replay pool, and the network is then predicted. The optimal action A* is identified by using the main network to forecast the Q value of the present state S and all feasible actions A. Subsequently, the Q-value of the subsequent state S' is anticipated using the target network, and the target Q-value is computed using this predicted value.

Lastly, the weight update and loss computation are carried out. The main network's weight is updated using the back propagation technique, and the loss function is computed based on the goal Q value and the main network's anticipated Q value. To achieve delayed update, the target network's weights are copied from the parent network within a specific time frame [9].

Iterations are also required, repeating steps until a preset number of training rounds is reached or other stopping criteria are met.

DDQN, by introducing a target network to decouple action selection and evaluation, mitigates overestimation but heightens computational complexity, requiring simultaneous updates and maintenance of two networks, thereby consuming more computing resources. Its target network's delayed update stabilizes training but also introduces sensitivity to the update frequency and mode, potentially impacting training stability and performance. Balancing exploration and exploitation via ε-greed strategies is challenging due to the fixedε's inability to adapt to dynamic environmental needs. Moreover, DDQN, despite correcting overestimation, remains susceptible to noise and outliers in training data, which can destabilize training and compromise the final strategy's effectiveness.

3.3. Improved Method based on DDQN

In order to improve the performance and stability of DDQN, researchers can take a series of improvement measures. Firstly, the computational complexity was reduced by optimizing the network structure, using parallel computing technology and optimizing the experience replay mechanism. Secondly, the adaptive update frequency and soft update strategy are used to optimize the update mode of the target network to reduce the fluctuation in the training process [10].

At the same time, the ε value in the ε-greedy strategy is dynamically adjusted or a more complex exploration strategy is adopted to balance the relationship between exploration and exploitation. In addition, data preprocessing, regularization techniques, and outlier detection and filtering are used to improve the robustness of the model to noise and outliers. Finally, multiple DDQN models can be considered for ensemble learning, and the stability and generalization ability of the model can be further improved through model fusion.

3.4. Application and Example Analysis

In view of the shortcomings of DDQN, different improvement strategies can be adopted in different application fields, such as optimizing the network structure and reducing the computational complexity, adaptive update frequency and soft update strategy, dynamic adjustment of ε value and complex exploration strategy, which can effectively improve the performance and stability of DDQN in practical application. Other technologies such as data preprocessing, regularization and outlier detection, and ensemble learning and model fusion can be used in real life technology processing. The following are application examples in several fields or aspects and analysis of improvement strategies.

At present, autonomous driving technology is becoming more and more popular. In addition to networking to obtain map information, the driving system is more important to perceive the surrounding environment and road conditions and make correct route planning and real-time decisions. In the automatic driving perception system, DDQN is used to predict the behavior of obstacles in front of the vehicle and make obstacle avoidance decisions. At this time, CNN is used to process the real-time video stream from the vehicle camera and extract key information such as road signs, pedestrians and other vehicles, which can provide accurate input data for DDQN to optimize driving decisions. In DDQN, multiple time steps or gradient updates of multiple samples can be calculated in parallel, thus speeding up the training process [11]. By using parallel computing devices such as Gpus or Tpus, the computing tasks of multiple neural networks can be processed simultaneously. Finally, optimizing experiential replay can also reduce consumption by adopting a prioritized experiential replay mechanism to select which samples to play based on their importance, such as TD error. However, the computational complexity and sensitivity of DDQN to target network updates can be limiting factors, it is proper to take compute resource optimization: Use hardware acceleration to process the computation of two networks in parallel, reducing overall training time.

At the same time, adaptive target network updating is adopted to dynamically adjust the update frequency of target network according to the complexity and stability of vehicle driving environment. For example, when the environment on the highway is relatively stable, the frequency of updates can be reduced; when the urban traffic environment is complex and changeable, the updating frequency should be increased [12].

The development of e-commerce is very rapid, at present, the e-commerce platform has basically realized the personalized recommendation function according to the user's purchasing habits.In the current recommendation system of many e-commerce platforms, a comprehensive method of adaptive update frequency and soft update strategy has been implemented. According to the performance of the model on the verification set and the stability index (such as the fluctuation of the loss function) during training, the update frequency of the target network is dynamically adjusted. When the model is stable, the update frequency is reduced to save resources and prevent overfitting; when the model exhibits fluctuations, the update frequency is increased to adjust the parameters quickly. At the same time, the soft update strategy is adopted, which does not copy the parameters of

the main network directly to the target network but sets a small update coefficient to realize the smooth transition of the parameters of the target network. Not only e-commerce, but also DDQN can be used to learn trading strategies. However, high noise and outliers in the data and rapid changes in the environment have high requirements for the adaptability and robustness of DDQN [13]. Therefore, data pre-processing can be improved, and data cleaning and denoising can be carried out before training to reduce the impact of noise and outliers on the model. Multiple DDQN models are combined for decision analysis.

Not only that, at present, people's requirements for game experience and difficulty are gradually rising, and more and more manufacturers are adopting AI technology. In game AI, where Double Deep Q-Network is used to control the behavior of game characters in pursuit of victory, researchers an use dynamic adjustment of εvalues and implement complex exploration strategies, balancing exploration and extraction is two key points. Exploration is about trying new, unproven behaviors to discover better strategies, while mining is about using currently known best behaviors for maximum immediate returns.

The ε-GREEDY strategy is a commonly used method to balance exploration and production, and ε represents the probability of conducting exploration (i.e. the probability of randomly selecting the action rather than the currently estimated optimal action). Dynamically adjusting the ε value at different stages of the game or facing different difficulties can significantly improve the performance of the AI. In the early or low difficulty phase of the game: at the beginning of the game or at low difficulty, setting a large ε value can encourage AI to explore more unknown behavioral space, help AI to discover potential better strategies, and avoid falling into the problem of local optimal solutions; As the game progresses or becomes more difficult, gradually decreasing the ε value encourages the AI to use more of what it has learned to perform the actions that are optimal for the current estimate. Make the most of what you know to win while maintaining stability.

In addition to the basic ε-greedy strategy, the game AI can further optimize its behavior selection by employing a more complex exploration strategy, where an indeaction-based exploration guides the exploration according to the model's uncertainty about the effect of the action.

4. Conclusion

This review has encompassed the optimization strategies employed in the DDQN, aimed at enhancing the algorithm's performance in intricate environments. By refining the network architecture of the original DQN, improving the strategy for updating the target network, dynamically adjusting exploration tactics, and introducing ensemble learning, DDQN has demonstrated significant advancements in both efficacy and robustness. Notably, this paper has conducted a detailed comparison between DDQN and DQN, elucidating their differences, and has explored the concrete applications of DDQN across diverse domains, such as autonomous driving, and AI in gaming, showcasing its superiority in addressing real-world challenges.

Given the outstanding theoretical and practical achievements of DDQN, this paper envisages a broad spectrum of future applications in complex decision-making scenarios. Continuous technological innovation and algorithmic refinement will likely empower DDQN to assume pivotal roles in a wider array of sectors, catalyzing the evolution and sophistication of intelligent decision-making systems. Future research endeavors should focus on further optimizing DDQN's capabilities, particularly in navigating larger-scale and more convoluted state-action spaces, by investigating more efficient neural network architectures and learning methodologies. Additionally, enhancing DDQN's adaptability in real-time, dynamic settings and exploring its applications within multi-agent systems will be instrumental in propelling this field forward. Through interdisciplinary collaboration and sustained empirical validation, DDQN holds the promise to unveil new horizons in domains such as smart transportation, financial market forecasting, and virtual reality experiences, thereby expanding its utility and impact significantly.

References

[1] Zhen He, Kien Phuoc Tran, Simon Thomassey, et al. Multi-Objective Optimization of the Textile Manufacturing Process Using Deep-Q-Network Based Multi-Agent Reinforcement Learning. Journal, 2020, 1-11.

[2] Wei, Z., Ma, Y., Yang, N., Ruan, S., & Xiang, C.. Reinforcement learning based power management integrating economic rotational speed of turboshaft engine and safety constraints of battery for hybrid electric power system. Energy, 2023, 263, Article 125752.

[3] Zhiyuan Liu, Xuefei Chen, Yan Chen, et al. Deep Reinforcement Learning Based Dynamic Resource Allocation in 5G Ultra-Dense Networks. In: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), 2019, 1-12.

[4] Bo Liu, Xiaoye Ye, Cheng Zhou, et al. The Improved Algorithm of Deep Q-learning Network Based on Eligibility Trace. In: 2020 6th International Conference on Control, Automation and Robotics (ICCAR), 2020.

[5] Zhiyu Yu, Xiaojun Yang, Feng GAO, et al. A Knowledge-based reinforcement learning control approach using deep Q network for cooling tower in HVAC systems. In: 2020 Chinese Automation Congress (CAC), 2020.

[6] Kiyoshi Kaneko, Takashi Komatsu, Zhou. Autonomous optimization of cutting conditions in end milling operation based on deep reinforcement learning (Offline training in simulation environment for feed rate optimization). Journal of Advanced Mechanical Design Systems and Manufacturing, 2023, 17(5).

[7] Jing Zhang, Fang Qian, Jian Yang. Online routing and spectrum allocation in elastic optical networks based on dueling Deep Q-network. Computers & Industrial Engineering, 2022, 173:108663.

[8] Xiaomin Li, Shaohu Yang, Jia Shi, et al. Deep reinforcement learning based resource allocation algorithm in cellular networks. Journal on Communications, 2019, 332-344.

[9] Zhuang Zhu, Jian Wang, Qi Qi, et al. Adaptive and Robust Network Routing Based on Deep Reinforcement Learning with Lyapunov Optimization. ACM, 2020.

[10] Xing Tang, Jinfeng Chen, Tian Liu, et al. Distributed Deep Reinforcement Learning-Based Energy and Emission Management Strategy for Hybrid Electric Vehicles. IEEE Transactions on Vehicular Technology, 2021, 70, 10.

[11] Yong Qian, Rui Wang, Jian Wu, et al. Reinforcement Learning Based Optimal Computing and Caching in Mobile Edge Network. IEEE Journal on Selected Areas in Communications, 2020, 1-11.

[12] Fan Wu, Lian Zhang, Yong He. Energy Efficiency Optimization in Downlink NOMA-Enabled Fog Radio Access Network Based on Deep Reinforcement Learning. In: 2021 Computing, Communications and IoT Applications (ComComAp), 2021.

[13] Yi Ke, Kai Huang, Xin Qiu, et al. Distributed Routing Optimization Algorithm for FANET Based on Multi-agent Reinforcement Learning. IEEE Sensors Journal, 2024, 24.