Advances in Value-based, Policy-based, and Deep Learning-based Reinforcement Learning

Haewon Byeon
Department of Medical Big Data-College of AI Convergence,
Inje University, Gimhae 50834, Gyeonsangnamdo, South Korea

***Abstract*—Machine learning is a branch of artificial intelligence in which computers use data to teach themselves and improve their problem-solving abilities. In this case, learning is the process by which computers use data and algorithms to build models that improve performance, and it can be divided into supervised learning, unsupervised learning, and reinforcement learning. Among them, reinforcement learning is a learning method in which AI interacts with the environment and finds the optimal strategy through actions, and it means that AI takes certain actions and learns based on the feedback it receives from the environment. In other words, reinforcement learning is a learning algorithm that allows AI to learn by itself and determine the optimal action for the situation by learning to find patterns hidden in a large amount of data collected through trial and error. In this study, we introduce the main reinforcement learning algorithms: value-based algorithms, policy gradient-based reinforcement learning, reinforcement learning with intrinsic rewards, and deep learning-based reinforcement learning. Reinforcement learning is a technology that enables AI to develop its own problem-solving capabilities, and it has recently gained attention among AI learning methods as the usefulness of the algorithms in various industries has become more widely known. In recent years, reinforcement learning has made rapid progress and achieved remarkable results in a variety of fields. Based on these achievements, reinforcement learning has the potential to positively transform human lives. In the future, more advanced forms of reinforcement learning with enhanced interaction with the environment need to be developed.**

Keywords*—*Reinforcement learning; value-based algorithms; policy gradient-based reinforcement learning; reinforcement learning with intrinsic rewards; deep learning-based reinforcement learning

I. Introduction

Advances in artificial intelligence and machine learning technologies have led to the development and use of AI-based services in many industries. At the same time, models using reinforcement learning, a branch of machine learning, are growing rapidly.

Machine learning is a branch of artificial intelligence in which computers use data to teach themselves and improve their problem-solving abilities. In this case, learning is the process by which computers use data and algorithms to build models that improve performance, and it can be divided into supervised learning, unsupervised learning, and reinforcement learning [1, 2]. Among them, reinforcement learning [3] is a learning method in which AI interacts with the environment and finds the optimal strategy through actions, and it means that AI takes certain actions and learns based on the feedback it receives from the environment [4]. In other words, reinforcement learning is a learning algorithm that allows AI to learn by itself and determine the optimal action for the situation by learning to find patterns hidden in a large amount of data collected through trial and error. Reinforcement learning is a technology that enables AI to develop its own problem-solving capabilities, and it has recently gained attention among AI learning methods as the usefulness of the algorithms in various industries has become more widely known [5, 6, 7].

This study is structured as follows. Section II presents the history and components of reinforcement learning, and Section III describes the main reinforcement learning algorithms: Value Based Algorithms, Policy Gradient Based Reinforcement Learning, Reinforcement Learning with Intrinsic Reward, and Reinforcement Learning based on Deep Learning. Section IV describes applications of reinforcement learning. Finally, Section V presents trends in the application of Reinforcement Learning in networking and future research directions. Section VI outlines the limitations and Section VII presents the conclusion to the study.

II. History and Components of Reinforcement Learning

Reinforcement learning can be traced back to an optimisation method for solving sequential decision problems, mathematically modelled by the Markov Decision Process, developed in the 1950s [8]. A Markov decision process is defined as a tuple (S, A, P, R, γ), where S and A are the agent's state space and action space, respectively. P and R are transition probability and reward functions, respectively, where P is the probability distribution of the next state and R is the reward the agent will receive in the next state if it performs an action $a \in A$ in state $S \in S$. The reward is a metric for judging the goodness or badness of the agent's actions. γ is the discount rate used to write off future rewards when calculating cumulative rewards. This helps the agent reach the goal quickly and prevents the cumulative reward from drifting, so that learning is stable [9]. The optimal policy for the sequential decision problem defined by the Markov Decision Process presented above can be found by reinforcement learning.

A policy is a function that takes a state value as input and determines what action the agent should take. A reinforcement learning agent observes the state of itself and its environment based on its sensors and information from other agents, decides what to do based on the observed state values and policies, and is sometimes rewarded for its actions. At this point, the

reinforcement learning agent learns its policy to maximise its cumulative reward expectation, and the policy that maximises the cumulative reward expectation is the optimal policy. This is based on the Reward Hypothesis. The reward hypothesis states that any goal (e.g., solving a problem) can be described as maximising the agent's cumulative reward.

This implies the importance of designing a reward function. Since reinforcement learning is how an agent learns its policy by interacting with the environment through trial and error, relying on reward cues, a poor reward function will not only make it difficult for the agent to learn, but will also lead to unexpected side effects even if it does learn [10].

In summary, reinforcement learning generally consists of three main elements.

- Agent: An object that performs actions in the environment according to the current policy.
- Environment: The external system with which the agent interacts. The environment provides feedback to the agent in the form of rewards or punishments based on actions taken.
- Reward: Feedback is the positive or negative feedback an agent receives from the environment based on its actions.

The agent's goal is to maximise the total amount of rewards over time, which means that the goal of reinforcement learning is to find a strategy that maximises the cumulative reward in a given environment [11]. In most cases, this means emphasising long-term rewards over short-term rewards. This process of reinforcement learning can be thought of as a trial-and-error process, where the AI learns by taking actions and observing the rewards or punishments that result from those actions. The agent uses this information to update its policies so that it can make better decisions in the future.

III. Main Reinforcement Learning Algorithms

Reinforcement learning algorithms can be classified into value-based, policy-based, and model-based algorithms.

A. Value-based Algorithms

Value-based algorithms estimate the value of each state or state-action pair and select the optimal action to improve the agent's performance [12]. Typical examples are Q-learning and Deep Q-Network (DQN)(Fig. 1 and Fig. 2).

Q-learning based reinforcement learning approximates the Q-value for a state-action pair each time and then decides which action to take in which state [13]. For exploration, we often use ε-greedy policies. An ε-greedy policy is a method that chooses a random action in a given state with probability ε, and the action with the highest Q-value with probability (1 - ε). Representative examples are DQN and Rainbow [14], which combines DQN with six DQN improvement algorithms.

The technical features of DQN can be summarised as follows: first, the use of convolutional neural networks for image recognition; second, the introduction of empirical replay to eliminate the correlation between samples and increase the efficiency of sampling; and third, the separation of the online Q-network, which determines the agent's behaviour, and the target Q-network, which is used to calculate the target Q-value, for learning stability [15].

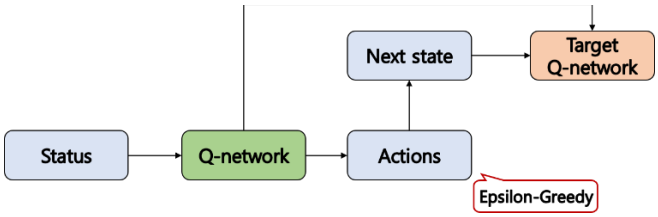

Fig. 1. The concept of deep Q network.

First, Rainbow is a technique that combines DQN with the following six DQN enhancement algorithms. For example, double Q-learning DQN takes the maximum value of the target Q-network in the current state when calculating the target Q-value, resulting in overestimation of the target Q-value and poor learning performance. Double Q-learning prevents the overestimation of the target Q-value by calculating the target Q-value using the behavioural value that maximises the online Q-net as the input of the target Q-net [16].

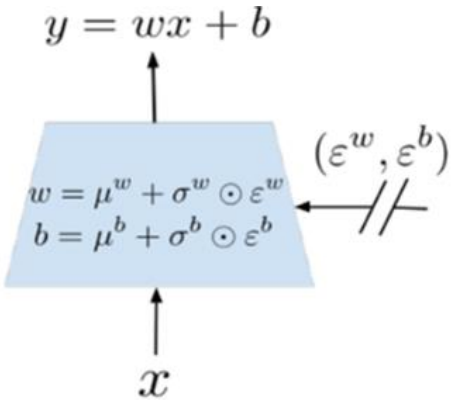

Fig. 2. Concept of applying gaussian noise to the weights of a neural network.

Second, prioritised experience replay - DQN learns by extracting experiences uniformly from experience replay, i.e. prioritised experience replay is a method of extracting samples that are more likely to be conducive to learning [17].

Third, Dueling Networking-DQN calculates Q-values on the fly. Since the Q-value takes into account both state and behaviour, it can be strongly influenced by behaviour when evaluating the value in a particular state. By decomposing the Q-value into the value of a particular state and the benefit of different actions that can be taken in that state, Dueling Networks can compute the value of a particular state more robustly while taking into account the value of actions [18].

Fourth, multi-step learning DQN uses the reward after the 1-step bootstrap, which is the very next state, to compute the target Q-value. By extending it to learn with the reward information after n-step bootstrapping, it evolves into multi-step learning (ex. with improved learning stability and speed) [19].

Fifth, Distributional RL-DQN, uses the expectation of the Q-value. In this case, the limitation is that it is difficult to

exploit the randomness inherent in the Markov decision process if only the expectation of the Q-value is used. In this case, Distributional RL is a method that uses a distribution of rewards instead of a single average. By using Distributional RL, you can not only improve learning performance, but also design safer agents by allowing agents to avoid risky behaviours [20].

Sixth, Noisy Nets-DQN uses the ε-greedy policy. However, the ε-greedy policy often leads to inefficient exploration because it outputs random behaviour regardless of the agent's current situation, and there is also the problem of setting the ε value. This is where Noisy Nets can be used (Fig. 3). By training a neural network (Noisy Nets) that adds noise to the weights and biases of the policy neural network when training the policy neural network, Noisy Nets has the advantage that the randomness of the agent's behaviour automatically adapts to the state the agent is in and over time (reducing randomness and promoting greedy choices as training progresses) [21].

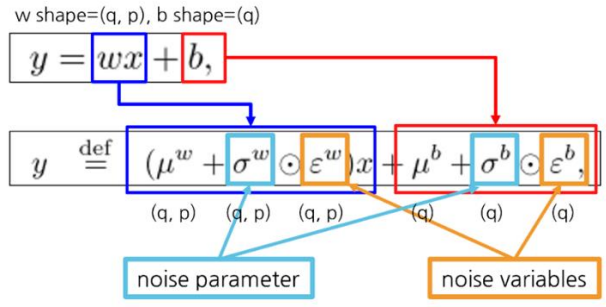

Fig. 3. The concept of noisy networks.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{bmatrix}$$

Fig. 4. The concept of element-wise multiplication.

B. Policy Gradient-based Reinforcement Learning

A policy-based algorithm directly optimises the policy that determines the agent's behaviour. In other words, policy gradient reinforcement learning directly yields a policy that determines which action to take in which state. A policy can be defined as a parameter vector θ, which is a function that takes observations as input and outputs an action value as output. The use of probabilistic policies in reinforcement learning is efficient for balancing exploration and exploitation. In deep reinforcement learning, this is approximated using a functional rate deep neural network, where the parameter vector θ is the weight and bias of the neural network. Policy gradient based reinforcement learning uses a policy gradient technique to compute this parameter vector, θ. The policy gradient technique is a method that uses gradient multiplication to find θ (Fig. 4). In other words, after finding the gradient of the objective function for a given θ, updating θ by a certain distance in the direction of the increasing gradient is repeated until the gradient converges or for a maximum time step.

The objective function is the expectation of the cumulative reward of acting according to the policy, expressed as in Eq. (1).

$$J(\theta) = E_{\pi_\theta}[r(s,a)] \tag{1}$$

The gradient of the objective function is defined by the policy gradient theorem [22], as shown in Eq. (2).

$$\nabla J(\theta) = E_{\pi_\theta}[Q_\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \tag{2}$$

The policy gradient updates the policy's parameters in the direction of maximising the objective function, as shown in Eq. (3).

$$\theta \leftarrow \theta + a\nabla j(\theta) \tag{3}$$

A prime example of policy-based reinforcement learning is Proximal Policy Optimisation (PPO) [22]. One of the drawbacks of policy-based reinforcement learning is that the policy of the parameters can change rapidly. This leads to learning instability, which results in slow learning speed and poor performance. To prevent such learning instability, TRPO (Trust region policy optimisation) [23] adds a condition that constrains the Kullback-Leibler (KL) divergence before and after a policy update to be below a certain level. TRPO attracted the attention of researchers due to its success in solving robot control problems with continuous action spaces, which were not solved by DQN. However, TRPO has the disadvantage that it requires a lot of computation to solve the constraints and is incompatible with various neural network structures (e.g., dropout, parameter sharing). To compensate for these disadvantages while maintaining the performance of TRPO, PPO removes the computationally intensive KL diveigenoe constraint and indirectly limits the number of policy renewals by simply clipping the ratio of pre- and post-policy renewals in the TRPO objective function. Although PPO was published in 2017, it is still the state-of-the-art algorithm and has been reported in many studies to be very good in terms of performance, computational efficiency and ease of implementation [24].

C. Reinforcement Learning with Intrinsic Reward

Reinforcement learning is a method of learning by exploring the environment through trial and error. In this case, the agent evaluates its behaviour and updates its policy based on the rewards it receives as a result of its actions. This can work well in environments where every action is rewarded, but policy learning is less successful in environments where rewards are infrequent and darkness is delayed. For example, suppose a game has a single reward for avoiding all obstacles, skeletons, etc. to get the key. In this case, it is difficult to determine whether an action taken in a particular state to get the key was beneficial, harmful or pointless. Even DQNs that outperform humans in many games are likely to fail in the game described above. One way to deal with this problem is to use intrinsic rewards.

Intrinsic rewards are rewards that are generated by the agent rather than given by the environment. It mimics the way humans learn through intrinsic motivation. A typical intrinsic reward function is prediction error. Prediction error is defined as the difference between the agent's predicted next state and

the actual next state. To predict the next state, an agent typically defines one or more prediction models (e.g., artificial neural networks) and trains them along with a policy. As the agent explores the environment and learns the prediction models, this prediction error will be lower for states that are familiar to the agent and higher for states that are unfamiliar to the agent, which has the effect of encouraging the agent to explore and, in games, discouraging the agent from dying and returning to the initial state. This is because the initial state is familiar to the agent, as it is the state to which the agent returns when it dies. The intrinsic reward function is designed so that the agent receives a reward for every action it performs, so it can learn well even in environments where the environment is sparse and black is a delayed reward. This also helps to broaden the application of reinforcement learning, as it reduces the need for a human to design a precise reward function for each task in the environment. However, there are a number of issues that need to be considered, such as the match between the directionality of the task the agent needs to perform and the directionality of the internal reward, the non-stationarity of the reward for performing the same behaviour in the same state as learning progresses, and the scaling of the reward across different environments.

To illustrate this, we refer the reader to Random Network Distillation [25]. Random Network Distillation consists of three neural networks: goal, prediction, and policy. The policy neural network is the one that determines the agent's behaviour, while the goal and prediction neural networks take the next state value as input and output some feature value. The goal neural network is fixed with randomly set weights, and the prediction neural network is a neural network with the same structure as the goal neural network, and is trained together with the policy neural network to produce the same output as the goal neural network. In other words, it is called Random Network Distillation because it has the effect of distilling a random network into a predictive neural network. In Random Network Distillation, the internal reward value function and the external reward value function are obtained separately and then combined, and PPO is used to optimise the policy neural network.

D. Reinforcement Learning (RL) Based on Deep Learning

Reinforcement learning refers to a group of methods for solving stochastic decision problems. Reinforcement learning can be classified as a model belonging to supervised learning or as an independent field of reinforcement learning. The reason it is classified as supervised learning is that it receives feedback or guidance from the environment, including humans, as it learns. On the other hand, it is classified as an independent model because the optimal decision process of reinforcement learning is a learning model that differs from the label-based discriminative approach typical of supervised learning. Reinforcement learning is very similar to the way humans learn, as it uses a trial and error process. For this reason, the core algorithm of AlphaGo, developed by Google DeepMind, is based on reinforcement learning. This makes reinforcement learning the closest model to artificial intelligence. Unlike supervised learning, reinforcement learning is not given training data. Instead, the reinforcement learning problem is given a reward function. The definition of solving reinforcement learning is to find a policy function that maximises the average of future reward values. To solve reinforcement learning, researchers have borrowed a mathematical model: the Markov decision process (MDP). Intuitively, the Markov property means that, given the present, the past and the future are independent. For example, the score I get on a test tomorrow depends only on my current state and how much I study today. An MDP has four main parts.

- A set of states
- A set of actions
- A transition function
- A reward function

In this study, we will illustrate the above four points with the example in Fig. 5. The most common example used to describe MDPs is the situation of a robot in a lattice space as shown below.

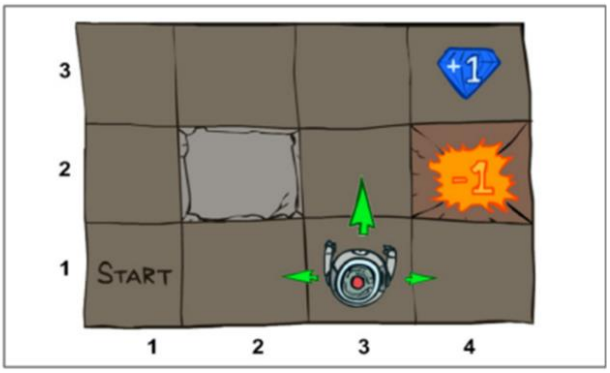

Fig. 5. A robot in a lattice space.

As shown in Fig. 5, the robot can be in one of twelve grids, which is its state space. In each grid the robot can move up, down, left, right or stay in place, and these five actions correspond to the action space. If the robot's movement through the grid is determined, it will move in the desired direction, but if it is stochastic, even if the robot tries to move up, it will move right or left with some probability. In this way, the transition function describes the probability of reaching the next state when a particular action is performed in a particular state. Finally, and most importantly for reinforcement learning, a reward function is defined for each state: in the above lattice space, reaching a gem is rewarded with a +1 reward, and reaching a fire is rewarded with a -1 reward. Given an MDP, solving reinforcement learning means finding a policy function that maximises the sum of expected future rewards. This has several implications, but the most important is that it involves future rewards. While rewards from current behaviour are important, ultimately we need to consider both current and future rewards. The next thing to remember is that we're dealing with a stochastic system. It is possible that our current behaviour will not lead us to the desired state, which means that if we can get the same reward, it is better to get it 'sooner'. The discount factor takes this into account. It is set to a value less than 1 and the reward earned over time is multiplied by this value.

So, given an MDP, how do we find the optimal policy function? The two most basic ways of solving reinforcement learning are value iteration and policy iteration. To explain this, we first need to define value. If we know not only the immediate reward we can get now, but also the consensus expectation of the rewards we can get when we start from that state, we can choose the action that maximises that function each time, and thus find the optimal policy function. This consensus expectation of future rewards is called the value function, V(s). Similarly, given a current state the expected future reward for taking an action in the current state is called the action value function or Q-function, Q(s,a). Value iteration refers to the method of finding this value function. The value function is difficult to define intuitively because it is not only about the current state, but also about future states and the rewards that can be obtained in those states. In general, reinforcement learning uses the Bellman equation to find this value function, which is defined as follows [26].

$$V(s) = \max_a \sum_{s'} T(s,a,s'')\left[R(s,a,s'') + \gamma V(s'')\right] \quad (4)$$

We can see that the above formula is a recursive equation with the value function V(s) we want to find on the left and right sides, and the rest is the same as MDP except for V(s). We can initialise V(s) to any value and run the above recursive equation for all states s until it converges, always finding the optimal V(s).In the previous section we looked at value iteration, but now we want to look at policy iteration, which is different from value iteration. Policy iteration consists of two phases: policy evaluation, which evaluates the performance of the current policy function, and policy improvement, which improves the policy based on the evaluation. These two phases alternate until the policy function converges. It is generally accepted that policy iterations converge to the optimal policy function faster than value iterations.

In this section we will investigate how to find the optimal policy function in an MDP when no model is given. This problem is commonly referred to as model-free reinforcement learning. The main difference from the model-based reinforcement learning described earlier is that we no longer know how the environment behaves. In other words, you do something in one state and get a reward for the next state, which is "passively" informed by the environment. Model-free reinforcement learning has several differences from model-based reinforcement learning, the most important of which is exploration. Since we don't know how the environment will behave, we have to experiment and use the results to gradually learn the policy function. Let's see how we can solve model-free reinforcement learning defined in this way. We can't use the Bellman equation directly because we don't know T(s, a, s'), which is the part of the Bellman equation used in model-based reinforcement learning.

Policy evaluation is a methodology for evaluating a given V(s), replacing a with pi(s) above.

$$V(s) = \max_a \sum_{s'} T\left(s,\pi(s),s''\right)\left[R\left(s,\pi(s),s''\right) + \gamma V\left(s''\right)\right] \quad (5)$$

The above formula is the policy evaluation formula used in model-free reinforcement learning. T(s, · (s), s') is an unknown value, but if the next state, s', comes from a model called T, we can replace the sum of T with the sample mean. One method that replaces the Bellman equation with sampling in this way is temporal difference (TD) learning. In an MDP, experience means that in a state s, I take an action via a given policy function (a= · (s)), and as a result I receive the next state s′ and a reward r. This reward r is a function of (s, a, s). As the experiences of (s, a, s', r) accumulate, we learn a value function V(s) and an action value function Q(s,a) based on these data. The expression for the Bellman equation for Q(s,a) is as follows.

$$Q(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_a, Q(s',a')] \quad (6)$$

The above formula can be used to update the behavioural value function Q(s,a). In this formula, we can replace T with the sample estimate, which gives us the following formula.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_a, Q(s',a')] \quad (7)$$

Solving reinforcement learning problems using the above formula is called Q-learning. All that is needed to update the Q-function using this formula is the experience of (s, a, s', r), i.e. the action taken in one state, the observation of the next state, and the reward received, and the Q-function can be obtained using the above formula. The advantage of Q-learning is that it can find the optimal policy function without knowing the model. However, Q-learning is not a one-size-fits-all method. The main disadvantage of model-less reinforcement learning is exploration. Due to the nature of reinforcement learning, rewards may be given only once or sporadically at the end. The difficulty of this exploration increases as the state space grows.

In deep reinforcement learning (DRL) [27], a method that combines this Q-learning with deep learning techniques, one of the most important things is how well it explores. One such method is called Deep Q Network (DQN), made famous by DeepMindtk. DQN is the algorithm that enabled DeepMind's AlphaGo to win the 2016 World Championship against Lee Sedol and Kasparov. From the algorithmic point of view of traditional reinforcement learning, DQN doesn't bring much new to the table. However, the main advantage of the QL formula is that it can update the Q function without any information about the environment. This property is more important than you might think, because the difficulty of RL compared to traditional SL is that it considers the sum of expected future rewards, but the above formula allows you to update the Q-function with just one experience. It also has the advantage that when choosing (s, a) from (s, a, s', r), Q(s, a) always converges to the optimal action value function after infinite time, even if a random action a is chosen each time. The right-hand side of the above formula is easy to find if we know the current Q(s, a) function. To find max a'Q(s', a'), the number of possible actions a must be finite. In other words, we can plug in all the possible actions and pick the one with the largest Q value. And if we think of Q(s, a) on the left side as the output of the Q function for some input (s, a) and the right side as the target for that input, the Bellman equation for the Q function above can be interpreted as giving us the input-output pairs for the regression function that we often use in supervised learning.

IV. Applications of Reinforcement Learning

Recently, there has been a lot of research and development on reinforcement learning, one of the artificial intelligence algorithms, to solve network system optimisation problems. Reinforcement learning is a system control method in which a reinforcement learning agent in a network management system uses information derived from the network environment to construct a reward function and achieve an optimal goal through iterative improvement. To do this, reinforcement learning agents go through an organic process of changing the state of the environment, controlling the behaviour of the agent, designing the value function, designing the reward function, improving the policy, and deriving the optimisation model. However, in order to learn a value function for decision making by predicting the expected value of output through predefined states and actions, a large amount of time must be invested in learning, and learning may not be performed well due to excessive environmental state information provided, or learning may be performed with the wrong goal. To overcome these problems, reinforcement learning models that improve learning efficiency and prediction accuracy performance by configuring the system's reward function as an artificial intelligence neural network have been studied. In addition, reinforcement learning models that can perform effective learning not only for discrete and limited number of behaviours, but also for very high degrees of freedom of the behaviours to be controlled are being studied.

V. Trends in the Application of Reinforcement Learning in Networking

As the network structure becomes more complex, various problems in the areas of routing, resource management, security and QoS/QoE arise, and to solve them, reinforcement learning application techniques are being studied, which include adaptive optimisation mechanisms for different environments. In the area of routing, research is being carried out to use reinforcement learning to optimise the routing process as network traffic grows exponentially. In resource management, researchers are applying reinforcement learning for efficient resource management and scheduling in rapidly changing network environments such as smart cities or edge clouds. This enables efficient network management by controlling network congestion or reducing overhead. In the area of network security, reinforcement learning is used to detect and respond to anomalies in the network, such as network congestion. In the area of QoS/QoE, researchers are using reinforcement learning to improve overall QoS/QoE by taking into account dynamically changing network characteristics.

VI. Limitations of Reinforcement Learning

One of the limitations of reinforcement learning is low data efficiency, especially in tasks where data selection is expensive, time consuming or dangerous. Therefore, one of the ways to deal with this is the off-policy technique, which can overcome the limitations of reinforcement learning to some extent if the behaviour policy and the target policy are different and the behaviour policy is carefully learned. Under this premise, imitation learning can also achieve good performance. However, most imitation learning algorithms have difficulties in achieving performance in suboptimal trajectory situations, and usually require interaction with the environment to overcome them. Therefore, in the future, it is necessary to develop imitation learning with enhanced interaction with the environment that can overcome suboptimal trajectory situations.

VII. Conclusion

In summary, reinforcement learning is a learning method at the heart of the AI revolution, enabling unimagined innovations in fields as diverse as autonomous driving, healthcare and gaming. As with any ML technology, it is important to consider the ethical implications of its use and ensure that it is applied in a responsible and beneficial way. In recent years, reinforcement learning has made rapid progress and achieved remarkable results in a variety of fields. Based on these achievements, reinforcement learning has the potential to positively transform human lives. In the future, more advanced forms of reinforcement learning with enhanced interaction with the environment need to be developed.

Acknowledgment

This research Supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF- RS-2023-00237287, NRF-2021S1A5A8062526) and local government-university cooperation-based regional innovation projects (2021RIS-003).

References

[1] M. I. Jordan, T. M. Mitchell, Machine learning: Trends, perspectives, and prospects. Science, vol. 349, no. 6245, pp. 255-260, 2015.

[2] B. Mahesh, Machine learning algorithms-a review. Int. j. sci. res, vol. 9, no. 1, pp. 381-386, 2020.

[3] Y. Li, Deep reinforcement learning: An overview. arXiv preprint, 2017.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey. IEEE Signal Process Mag, vol. 34, no. 6, pp. 26-38, 2017.

[5] R. Nian, J. Liu, B. Huang, A review on reinforcement learning: Introduction and applications in industrial process control. Computers & Chemical Engineering, vol. 139, pp. 106886, 2020.

[6] P. Dayan, Y. Niv, Reinforcement learning: the good, the bad and the ugly. Curr. Opin. Neurol, vol. 18, no. 2, pp. 185-196, 2008.

[7] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, D. Hassabis, Reinforcement learning, fast and slow. Trends Cogn. Sci, vol. 23, no. 5, pp. 408-422, 2019.

[8] M. L. Puterman, Markov decision processes. Handbooks in operations research and management science, vol. 2, pp. 331-434, 1990.

[9] W. S. Lovejoy, A survey of algorithmic methods for partially observed Markov decision processes. Ann. Oper. Res, vol. 28, no. 1, pp. 47-65, 1991.

[10] C. Guestrin, M. Lagoudakis, R. Parr, Coordinated reinforcement learning. In ICML, Vol. 2, pp. 227-234, 2002.

[11] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. P. van Hasselt, S. Singh, D. Silver, Discovering reinforcement learning algorithms. In Neural Information Processing Systems, vol. 33, pp. 1060-1070, 2020.

[12] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, Z. Li, Metalight: Value-based meta-reinforcement learning for traffic signal control. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, No. 01, pp. 1153-1160, 2020.

[13] A. Kumar, A. Zhou, G. Tucker, S. Levine, Conservative q-learning for offline reinforcement learning. In Neural Information Processing Systems, vol. 33, pp. 1179-1191, 2020.

[14] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, et al. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI conference on artificial intelligence. Vol. 32, No. 1, pp. 3215-3222, 2018.

[15] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, et al. Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning. IEEE access, vol. 7, pp. 39974-39982, 2019.

[16] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38, 2017.

[17] X. Wang, H. Xiang, Y. Cheng, Q. Yu, Prioritised experience replay based on sample optimisation. J. Eng, vol. 13, pp. 298-302, 2020.

[18] N. Van Huynh, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, Optimal and fast real-time resource slicing with deep dueling neural networks. IEEE J. Sel. Areas Commun, vol. 37, no. 6, pp. 1455-1470, 2019.

[19] J. F. Hernandez-Garcia, R. S. Sutton, Understanding multi-step deep reinforcement learning: A systematic study of the DQN target. arXiv preprint, 2019.

[20] Y. Tang, R. Munos, M. Rowland, B. Avila Pires, W. Dabney, M. Bellemare, The nature of temporal difference errors in multi-step distributional reinforcement learning. In Neural Information Processing Systems, vol. 35, pp. 30265-30276, 2022.

[21] S. Han, W. Zhou, J. Liu, S. Lü, NROWAN-DQN: A stable noisy network with noise reduction and online weight adjustment for exploration. arXiv preprint, 2020.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms. arXiv preprint, 2017.

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

[24] W. Yi, R. Qu, L. Jiao, Automated algorithm design using proximal policy optimisation with identified features. Expert Syst. Appl, vol. 216, pp. 119461, 2023.

[25] Y. Burda, H. Edwards, A. Storkey, O. Klimov, Exploration by random network distillation. arXiv preprint, 2018.

[26] B. O'Donoghue, I. Osband, R. Munos, V. Mnih, The uncertainty bellman equation and exploration. In International Conference on Machine Learning, pp. 3836-3845, 2018.

[27] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, S. Mannor, Learn what not to learn: Action elimination with deep reinforcement learning. In neural information processing systems, vol. 31, 2018