Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PERFORMANCE COMPARISON OF REINFORCEMENT LEARNING ALGORITHMS IN THE CARTPOLE GAME USING UNITY ML-AGENTS

[1]EUN-HYEONG JO, [2]YOUNGSIK KIM
[12]Dept. of Game and Multimedia Engineering, Tech University of Korea, Republic of Korea
E-mail: [1]whdmsgud8@tukorea.ac.kr, [2]kys@tukorea.ac.kr (corresponding author)

ABSTRACT

Reinforcement learning is a field of machine learning where agents learn optimal actions through trial and error interactions with their environment. Games provide an effective benchmark to evaluate and compare the performance of reinforcement learning algorithms. This study utilized Unity's ML-Agents to implement the 'CartPole' game and applied various algorithms, including the Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO), to compare their performance. The primary research contribution of this work is the systematic comparison of these algorithms within a consistent environment, providing insights into their respective strengths and weaknesses. The study presents detailed analyses of the learning processes and outcomes of each algorithm, highlighting the DQN's superior performance in terms of stability and efficiency. Additionally, this work contributes new knowledge by demonstrating the practical applications and potential of reinforcement learning algorithms in simple game environments, thereby informing future developments in more complex domains.

Keywords: *Reinforcement Learning Algorithm, Deep Q-Network (DQN) Algorithm, Performance Comparison, CartPole Game*

1. INTRODUCTION

In contemporary society, artificial intelligence (AI) technologies are rapidly advancing and being utilized across various fields. Among these technologies, games have emerged as a compelling platform for evaluating reinforcement learning (RL) algorithms and developing new techniques. Games offer complex environments that mimic real-world scenarios, enabling agents to learn the skills necessary to solve actual problems through diverse situations and rules. Additionally, games provide a benchmark for effectively assessing and comparing the performance of RL algorithms [1].

Reinforcement learning, a subset of machine learning, enables an agent to interact with its environment through trial and error to learn optimal actions. In game environments, RL algorithms play several critical roles. RL algorithms can learn the rules and dynamics of a game environment and develop optimal strategies for gameplay. This capability can be applied to create AI-based game-playing systems that surpass human performance, contributing to the development of agents capable of outperforming human players in various games. RL algorithms can be employed in the design and development of new games. They can adjust game difficulty levels and introduce new elements that enhance player experience. By doing so, RL can create more engaging and entertaining game environments. Research and development of RL algorithms in game environments significantly contribute to the advancement of AI technologies. Through RL, agents can improve their ability to learn and solve problems in complex environments, which is crucial for applying AI technologies across diverse domains.

Reinforcement learning algorithms face a critical challenge in balancing exploration and exploitation during the learning process. Exploration involves the agent taking random actions to gain new experiences, while exploitation involves choosing the best-known actions based on learned knowledge. Both are essential for optimal learning performance, but finding the right balance is challenging.

If an agent focuses too much on exploration, it may fail to settle on stable actions, resulting in poor performance. While gaining new experiences is important, purely random actions can reduce learning efficiency and waste time.

Conversely, if an agent overemphasizes exploitation, it may miss out on discovering better strategies by not exploring new possibilities. Acting solely on learned knowledge without exploring new experiences can prevent the agent from developing optimal strategies.

The exploration-exploitation dilemma is a major factor that can hinder the performance of RL algorithms. Therefore, designing RL algorithms requires effective strategies to maintain a balance between exploration and exploitation.

Various strategies have been developed to address the exploration-exploitation dilemma in RL algorithms. Prominent methods include Epsilon-Greedy Policy, Boltzmann Exploration and Upper Confidence Bound (UCB) [1].

Epsilon-Greedy Policy selects a random action with a probability of epsilon (ε) and the best-known action with a probability of 1-ε. It effectively balances exploration and exploitation, although the performance depends on the choice of ε. For Boltzmann Exploration, actions are chosen based on a probability distribution that favors higher-valued actions but allows for some exploration of lesser-known actions. This method helps in gradually shifting towards exploitation while still exploring new actions. UCB algorithms choose actions based on a confidence interval that balances the estimated value of actions with the uncertainty in those estimates. This approach encourages exploration of actions with high uncertainty, helping to discover potentially better strategies.

The paper [2] provides a comprehensive review of the recent advances in reinforcement learning, particularly deep reinforcement learning (DRL), and its applications in the gaming industry. It discusses how RL has been used in classic games like Atari, Chess, and Go, highlighting the development of algorithms capable of outperforming human players.

The paper [3] delves into various deep reinforcement learning techniques applied to video games. It covers a range of DRL algorithms such as DQN, DQfD, and IQN, explaining their evolution and improvements. The paper [3] also explores the application of these algorithms in different game genres and the challenges faced in integrating RL with complex game environments.

The article [4] discusses the practical application of reinforcement learning methods like TD(0) and TD(λ) in game scenarios. It explains the theoretical foundations and implementation details of these algorithms, providing insights into their effectiveness in learning and decision-making within games.

The paper [5] introduces a novel approach that combines deep reinforcement learning (DRL) with supervised learning from human demonstrations to improve the efficiency and performance of DRL agents. The proposed method leverages human demonstrations to enhance the initial learning phase of the agent. By combining Q-learning with supervised learning, the agent can learn valuable behaviors from the demonstrations, which accelerates its ability to perform well in the environment.

Game environments are not only attractive platforms for evaluating RL algorithms and developing new technologies but also play an essential role in applying RL techniques across various fields. This paper aims to contribute to the advancement of this field through a comparative analysis of RL algorithms, highlighting their importance and impact on AI technology development.

2. BACKGROUNDS

Reinforcement learning (RL) algorithms can be broadly categorized based on their approach to handling state-action spaces and their complexity. Here, we discuss three major categories: table-based methods, temporal-difference (TD) learning, and deep learning-based RL [1] in Table 1.

2.1 Table-Based Methods

Table-based methods involve storing state-action pairs in a table and updating these values based on experiences. These methods are suitable for problems with relatively small state spaces. Two prominent algorithms in this category are Q-Learning and SARSA (State-Action-Reward-State-Action).

Q-Learning is an off-policy algorithm where the agent learns the value of the optimal policy independently of the agent's actions.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1. Deep Learning Algorithms.

Characteristics	Table-Based Methods	Temporal-Difference (TD) Learning	Deep Learning-Based Reinforcement Learning
Basic Idea	Store state-action pairs in a table	Learn using the difference between predicted and actual rewards	Approximating policy and value functions using neural networks
Representative Algorithms	Q-learning, SARSA	TD(0), TD(λ), Sarsa(λ)	DQN, A2C, PPO
State Space	little	middle	great
Learning Style	Offline/Off-Policy	Online/Onpolicy	Online/Off Policy
Policy Expression	table	table	neural network
Memory Requirements	low	low	high
Calculation Requirements	low	middle	high
Sample Efficiency	low	middle	high
Generalization Ability	doesn't exist	low	high
Memory Mechanism	not needed	not needed	Replay buffer, experience reuse
Exploration Strategy	ε(epsilon)-Greedy	ε(epsilon)-Greedy	ε(epsilon)-Greedy, add noise, move octaves
Function Approximation	doesn't exist	doesn't exist	use
Advantages	Simple, easy to implement	Relatively simple, fast convergence	Ability to solve complex problems and generalize
Disadvantage	Inefficient as state space grows	Inefficient as state space grows	High computational cost, complex

The Q-values (state-action pairs) are updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma a' \max Q(s',a')-Q(s,a)]$$

It is widely used due to its simplicity and effectiveness in various applications.

SARSA is an on-policy algorithm where the agent updates its Q-values based on the action actually taken by the policy being followed. The update rule is:

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$$

This method ensures that the learning process is more stable as it updates the Q-values based on the current policy's actions.

2.2 Temporal-Difference (TD) Learning

TD learning combines the ideas of Monte Carlo methods and dynamic programming. It uses the difference between predicted rewards and actual rewards to update the value functions. This approach is suitable for problems with medium-sized state spaces. Key algorithms include TD(0), TD(λ), and Sarsa(λ).

TD(0) algorithm updates the value function based on the immediate reward and the estimated value of the next state:

$$V(s) \leftarrow V(s)+\alpha[r+\gamma V(s')-V(s)]$$

It is simple and can be used for online learning as it does not require the entire episode to update values.

TD(λ) introduces eligibility traces, which allow it to take into account the entire history of states and actions. The update rule integrates the traces.

$$V(s) \leftarrow V(s) + \alpha\delta_t e(s)$$

where δt is the TD error and e(s) is the eligibility trace for state s.

Sarsa(λ), an extension of SARSA, it also uses eligibility traces to update the Q-values. This algorithm balances between immediate and long-term rewards using a decay parameter λ:

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]e(s,a)$$

where e(s,a) is the eligibility trace for the state-action pair.

2.3 Deep Learning-Based Reinforcement Learning

Deep learning-based RL methods utilize neural networks to approximate policy and value functions. These approaches are particularly effective for handling large and complex state spaces. Prominent algorithms include Deep Q-Network (DQN) [6][7][8][9], Advantage Actor-Critic (A2C) [9][10][11], and Proximal Policy Optimization (PPO) [9][12][13].

DQN uses a neural network to approximate the Q-value function. It incorporates techniques such as experience replay and fixed Q-targets to stabilize training:

$$Q(s,a;\theta) \leftarrow Q(s,a;\theta)+\alpha[r+\gamma a' \max Q(s',a';\theta-)-Q(s,a;\theta)]$$

where θ represents the network parameters and θ^{-} represents the parameters of the target network.

The paper [6] is the seminal paper on DQN, presenting the algorithm and its application to Atari 2600 games. It covers the core concepts of using deep learning with Q-learning and demonstrates the algorithm's effectiveness. The paper [7] provides an in-depth analysis of DQN's performance and its ability to achieve human-level control in Atari games. It elaborates on the techniques used to stabilize training, such as experience replay and target networks. The paper [8] introduces Double DQN, an improvement on the original DQN that addresses the overestimation bias in Q-learning. It discusses the theoretical background and presents empirical results showing enhanced performance. The book [9], while not specific to DQN, provides foundational knowledge in reinforcement learning, including Q-learning, which is crucial for understanding the principles behind DQN.

A2C uses two networks: an actor network to select actions and a critic network to evaluate them. The advantage function helps reduce variance in policy updates:

$$\text{Advantage} = R - V(s)$$

where R is the total reward and V(s) is the value function estimated by the critic.

The book [9] provides foundational concepts in reinforcement learning, including actor-critic methods. It is a comprehensive resource for understanding the theoretical background behind A2C. The paper [10] introduces the Asynchronous Advantage Actor-Critic (A3C) algorithm, from which A2C is derived. It presents the core ideas and the performance of A3C in various environments. The paper [11] discusses methods that improve the stability and performance of policy gradient methods, including A2C. It is useful for understanding enhancements to the basic A2C algorithm.

PPO improves upon traditional policy gradient methods by ensuring that policy updates are not too drastic. It uses a clipped objective to maintain the stability of learning:

$$L^{CLIP}(\theta) = E[\min(r(\theta)Adv, \text{clip}(r(\theta),1-\epsilon,1+\epsilon)Adv)]$$

where r(θ) is the probability ratio between the new and old policies.

The book [9] covers a wide range of reinforcement learning methods, including policy optimization techniques. It provides the theoretical background necessary to understand the principles behind PPO. The paper [12] is the seminal work on PPO, presenting the algorithm and its advantages over previous policy optimization methods such as Trust Region Policy Optimization (TRPO). It includes theoretical foundations, implementation details, and experimental results demonstrating PPO's effectiveness. The paper [13] provides important background on policy optimization methods, introducing concepts and techniques that influenced the development of PPO. Understanding TRPO helps in comprehending the improvements made by PPO.

The Additional Learning Nearest Neighbor (ALNN) algorithm [17][18], which might not be widely recognized in standard machine learning literature, seems to be a variant or a specific implementation involving nearest neighbor techniques with additional learning components. Given the name, it likely combines principles of nearest neighbor algorithms with some form of iterative learning or refinement.

The ALNN algorithm extends the traditional k-nearest neighbor (k-NN) algorithm by incorporating additional learning mechanisms. While the standard k-NN algorithm relies purely on distance metrics to classify data points based on their nearest neighbors, ALNN could involve updating weights or refining the decision boundaries through iterative learning from the data. Unlike traditional k-NN which uses a fixed set of training examples, ALNN likely adjusts its decision criteria based on additional training iterations, possibly improving its classification accuracy over time. It may involve adjusting the weights assigned to different neighbors based on their relevance or accuracy in classification tasks. The algorithm could iteratively refine its model by incorporating feedback from misclassifications or using cross-validation techniques to optimize performance.

The DQN algorithm effectively resolves the exploration-exploitation dilemma by employing experience replay memory and an epsilon-greedy policy. DQN leverages experience replay memory to store and utilize past experiences for learning. This approach allows the agent to learn from historical experiences, leading to more efficient Q-value updates. Additionally, experience replay memory mitigates data correlation and enhances the stability of the learning process.

DQN employs experience replay memory in the following manner. The agent interacts with the environment to gather new experiences. These experiences are stored in the replay memory as tuples of the form (state, action, reward, next state). During training, the agent samples random batches of tuples from the replay memory. The sampled tuples are used to update the Q-values.

Utilizing experience replay memory offers several benefits. The agent can learn from past experiences, improving the efficiency of Q-value updates. It reduces the correlation between data points, enhancing the stability of the learning process. Experience replay memory ensures efficient use of data, leading to improved learning performance.

The DQN algorithm maintains a balance between exploration and exploitation using an epsilon-greedy policy. This policy selects actions based on a probability threshold, where the agent either explores by choosing random actions or exploits by selecting actions based on learned Q-values.

DQN utilizes the epsilon-greedy policy as follows. Before selecting an action, the agent evaluates the epsilon value to decide whether to explore or exploit. If a random number is less than epsilon, the agent chooses a random action (exploration). If the random number is greater than or equal to epsilon, the agent selects the action with the highest Q-value (exploitation).

The epsilon-greedy policy offers several advantages. Balanced Exploration and Exploitation: It allows the agent to explore new experiences while leveraging learned knowledge to achieve better results. Faster Learning: The policy ensures that the agent learns quickly by balancing exploration and exploitation.

In different gaming environments, DQN outperforms other reinforcement learning algorithms. For instance, in the 'CartPole' game [14][15][16], DQN achieves higher average episode lengths and faster learning compared to other algorithms.

DQN's stability is attributed to experience replay memory, which allows the agent to learn from past experiences and efficiently update Q-values. This process reduces data correlation and minimizes errors during Q-value updates. The epsilon-greedy policy in DQN facilitates adaptation to new environments by encouraging exploration of new experiences and learning optimal actions.

DQN can be applied in various gaming environments for different purposes. For instance, it can be used to. Learn Optimal Game-Playing Strategies: DQN can develop superior game-playing strategies based on the rules and environment of the game. Design and Develop New Games: DQN can aid in adjusting game difficulty levels and introducing new elements to enhance player experience. Advance AI Technology: Research and development of DQN in gaming environments contribute to the broader field of AI.

3. IMPLEMENTATION

3.1 CartPole Game using Unity ML Agent

The CartPole game [14][15][16] is a reinforcement learning environment provided by OpenAI Gym [16]. The primary goal is to keep a pole balanced vertically on a moving cart by adjusting the cart's position. Players can control the cart's movement using the left and right arrow keys.

In the CartPole game, the action space is discrete with two possible actions: moving the cart left (0) or right (1). The observation space consists of a 4-dimensional vector that includes cart position, cart velocity, pole angle, and pole angular velocity.

Table 2. Comparison of reinforcement learning-related games.

Characteristics	CartPole	MountainCar	Pong	Atari Breakout
Simplicity	relatively simple	medium complexity	medium complexity	complicacy
State Space	little	little	middle	great
Action Space	little	little	little	middle
Learning Rate	speedy	slow	middle	slow
Clarity of Reward Signals	low	low	middle	high
Visual Complexity	low	low	high	high
Ease of Comparison	high	middle	middle	low
Need for Navigation Strategy	low	high	middle	middle
Generalization Ability	high	low	middle	middle

The cart's position can vary between ±4.8 units, while the pole's angle ranges between ±0.42 radians. Players receive a reward of 1 point for each time step the pole remains balanced. The game terminates if the pole falls beyond a certain angle or if the cart moves out of the designated boundaries. These features make the CartPole game an excellent platform for benchmarking the performance of various reinforcement learning algorithms in a standardized and controlled setting.

Figure 1 is a depiction of the CartPole environment. The blue rectangle represents the cart, and the red rectangle represents the pole. The objective is to balance the pole vertically by moving the cart left or right.

This paper implements the CartPole Game using Unity ML-Agents [19], applying ALNN [17][18], PPO [9][12][13], A2C [9][10][11], and DQN (E-Greedy) [6][7][8][9].

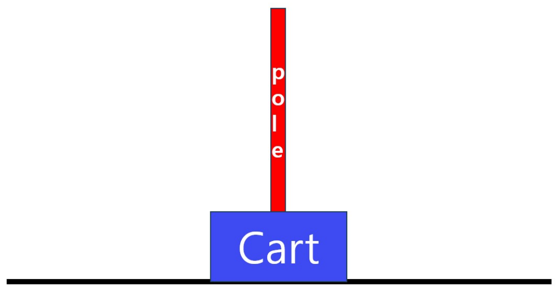

Figure 1. CartPole Game.

Unity ML-Agents [19] is a toolkit developed by Unity Technologies to enable the training of intelligent agents within the Unity game engine using reinforcement learning and other machine learning techniques. This toolkit provides a convenient interface for creating complex, interactive environments where agents can learn and improve their behaviors through trial and error.

Table 2 shows that the CartPole game is an ideal environment for rapidly evaluating the performance of reinforcement learning algorithms due to its simplicity and small state space. The game involves balancing a pole on a moving cart by taking left or right actions to keep the pole upright. Rewards are given based on the duration the pole remains balanced.

The straightforward environment and clear reward signal enable algorithms to converge quickly, facilitating rapid experimentation. Additionally, the low computational resource requirements make it an excellent platform for developing and testing reinforcement learning algorithms. The high reproducibility of experimental results and simple initial setup are valuable for comparing and evaluating various algorithms.

3.2 Algorithms used in the Experiment

Table 3. Pseudo code of ALNN-based AI Model

```
Algorithm: ALNN

Input:
 - Input data: X (input vector)
 - Actual output data: Y (actual output vector)
 - Learning rate: η (learning rate)
 - Activation function: ϕ (activation function)
 - Number of hidden layer neurons: n_h
 - Number of repetitions: epochs

Output:
 - Learned weight: W
 - Learned bias: b

Initialize:
 - Initialize weight W and bias b with small random
   numbers

Learning steps:
 for epoch from 1 to epochs do:
   for each (x, y) in (X, Y) do:
     # Forward propagation
     # From input layer to hidden layer
     for j from 1 to n_h do:
       $h_j = \phi(\Sigma w_{ij} * x_i + b_j)$
     # From hidden layer to output layer
     for k from 1 to m do:
       $y_k = \phi(\Sigma w_{jk} * h_j + b_k)$
     # Calculate loss
     $L = (1/N) \Sigma (y_i - \hat{y}_i)^2$
     # Backpropagation step
     # Update weights and biases from output layer
       to hidden layer
     for k from 1 to m do:
       for j from 1 to n_h do:
         $w_{jk} = w_{jk} - \eta * \partial L/\partial w_{jk}$
       $b_k = b_k - \eta * \partial L/\partial b_k$
     # Update weights and bias from hidden layer to
       input layer
     for j from 1 to n_h do:
       for i from 1 to n do:
         $w_{ij} = w_{ij} - \eta * \partial L/\partial w_{ij}$
       $b_j = b_j - \eta * \partial L/\partial b_j$

Output:
 - Learned weights and bias: W, b
```

The basic learning method of the ALNN algorithm [17][18] requires assigning and calculating weights for all combinations as shown in Table 3.

The flowchart of the pseudocode in Table 3 is as follows. In the initialization step, the weights W_ij and the threshold T_i are initialized randomly or using a specific initialization method.

In the learning stage, forward propagation calculates the activation of each neuron in the network. An equation is presented to calculate the activation of each neuron using the input data X and the current weights. Here the activation function determines the activation of the neuron for given inputs and weights.

Backward propagation calculates the error between the predicted output and the actual output, and updates weights and thresholds to minimize this error. Here we use optimization techniques such as gradient descent to update the weights and thresholds. Inference uses a trained network, given new input data, performs forward propagation through the trained network to obtain a predicted output. The process of building various resources necessary for ray tracing by traversing all objects in the entire scene, which is performed identically for both hybrid ray tracing and ray tracing only, is called the construction process. This process involves two main steps: building the hit group shader table and building the acceleration structure.

PPO [9][12][13] is an algorithm used in the field of reinforcement learning and uses a policy-based approach. This is a way to optimize the policy that determines what action to take in what state as the agent interacts with the environment. The main purpose of PPO is to achieve high performance through an efficient and stable learning process.

The flowchart of the pseudocode in Table 4 is as follows. The purpose of a PPO is to maximize expected returns. This is expressed as the objective function. Here, θ is a parameter that parameterizes the policy. The objective function is expressed as the sum of the expected discounted rewards, which is the sum of the rewards r_t at each time step discounted by gamma (γ). PPO updates the policy using a clipped surrogate objective function. This formula updates the policy in a way that minimizes it within the confidence region for r_t, which is the ratio of the new policy probability to the old policy probability. A_t represents the estimated advantage function, and ε is the clipping parameter. PPO improves policy learning through updates of the value function. Here we update the value function by minimizing the mean squared error. V_ϕ (s_t) represents the predicted value of state s_t, and V_target represents the target value.

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 4. Pseudo code of PPO-based AI Model

Algorithm: Proximal Policy Optimization (PPO)

```
Algorithm: A2C (Advantage Actor-Critic)

Input:
- Initial policy parameter: $\theta$
 - Initial value function parameters: $\phi$
 - Policy network: $\pi\_\theta(a|s)$
 - Value function network: $V\_\phi(s)$
 - Discount coefficient: $\gamma$
 - Learning rate: $\eta\_\theta$, $\eta\_\varphi$
 - Batch size: N
 - Number of epochs: K

Output:
 - Optimal policy parameter: $\theta$
 - Optimal value function parameters: $\phi$

Initialize:
 - Policy parameter $\theta = \theta\_0$
 - Value function parameter $\phi = \phi\_0$

Repeat:
 for iteration = 1, 2, ..., M do:
   Initialize collected samples: D = []
   # Sample collection step
   for actor = 1 to N do:
     Initial state: s_0
     for t = 0 to T do:
       Action selection: a_t ~ $\pi\_\theta(a\_t|s\_t)$
       Execute action in environment: s_{t+1},
           r_t = env.step(a_t)
       Add sample: D.append((s_t, a_t, r_t, s_{t+1}))
       if s_{t+1} is terminal:
           break
   # Extract samples from batch
   for each (s, a, r, s') in D do:
     # Calculate value
     R_t = r + $\gamma$ * $V\_\varphi$(s_{t+1}) if s' is not terminal
          else r
     A_t = R_t - $V\_\phi$(s_t)

     # Update policy network
     policy_loss = -log($\pi\_\theta$(a|s)) * A_t
     $\theta \leftarrow \theta - \eta\_\theta * \nabla\_\theta$ policy_loss

     # Update value function network
    value_loss = (R_t - $V\_\phi$(s_t))^2
    $\phi \leftarrow \phi - \eta\_\phi * \nabla\_\phi$ value_loss

Output:
 - Optimal policy parameter: $\theta$
 - Optimal value function parameters: $\phi$
 - Optimal policy parameter: $\theta$
 - Optimal value function parameters: $\phi$
```

Table 5. Pseudo code of A2C-based AI Model

A2C [9][10][11 is one of the algorithms used in reinforcement learning and is based on the Actor-Critic methodology. Here, "Actor" is responsible for deciding what action to take in a given state, and "Critic" is responsible for evaluating how good the action taken is. A2C uses these two components to optimize the learning process.

The flowchart of the pseudocode in Table 5 is as follows. The A2C algorithm uses an Actor-Critic architecture. An Actor is a policy network parameterized by θ, which is used to select actions based on observations. Critic is a value network parameterized by ϕ that evaluates the state value function. The goal of the policy gradient is to maximize the expected return. This objective function is defined as the expected return for policy ϕ _θ parameterized by policy parameter θ. Estimate the advantage function. The benefit function is defined as the difference between the state-action value function and the state value function. To update the policy and value networks, we use gradient descent to maximize θ and minimize the loss of each value function. Typically, the loss function for a value network is the mean squared error between the predicted and target values.

4. PERFORMANCE EVALUATION

4.1 Experimental Environment

Table 6 describes the CartPole game settings. As details, the episode length is a cartpole game where the agent must keep the bar vertical for 30 seconds. For this, we set an episode length of 1800 timesteps (30 seconds * 60 FPS). The agent must learn the behavior of keeping the bar as vertical as possible during this time. Learning Rate is an important hyperparameter used in the Q-value update process. Setting it to 0.001 will result in incremental improvements during the learning process. A learning rate that is too high can result in unstable learning, while a learning rate that is too low can result in slow learning. Discount Factor indicates how important the value of future rewards is compared to current rewards. Setting it to 0.99 makes future rewards of similar importance to current rewards. This allows the agent to learn optimal behavior over the long term. Exploration Probability refers to the probability that an agent will randomly choose an action. Setting it to 0.1 will cause the agent to choose the optimal action most of the time, but may occasionally explore new situations. This allows the agent to find better action strategies.

Table 6. CartPole Game Settings.

Items	Setting
Episode Length	30 seconds (1800 timesteps)
Learning Rate	0.001
Discount Factor	0.99
Exploration Probability	0.1

4.2 Experimental Results

Table 7. Experimental Results of CartPole Game.

Algorithms	Number of Simulation Iterations	Average Horizontal Angle	Minimum Horizontal Angle	Maximum Horizontal Angle	Standard Deviation
DQN	100	20.345	1.154	20.789	1.278
ALNN	100	19.567	2.345	23.456	1.987
PPO	100	22.678	1.978	26.789	2.134
A2C	100	20.987	1.567	24.567	2.345

As shown in Table 7, based on the results of 100 simulations, the DQN algorithm showed the lowest standard deviation, and the ALNN algorithm showed the lowest average horizontal angle. However, the DQN algorithm showed better results at the minimum horizontal angle. Therefore, the DQN algorithm is judged to be the best. The DQN algorithm showed higher performance than other reinforcement learning algorithms in the 'CartPole' game.

The DQN algorithm showed a faster learning rate compared to other reinforcement learning algorithms. This is because DQN utilizes experience replay memory to maximize efficiency in the learning process. Setting the discount rate of the DQN algorithm allowed us to effectively consider long-term rewards and achieve excellent results. The DQN algorithm was able to quickly learn the optimal action policy by setting an appropriate exploration probability. This contributed to maintaining a higher horizontal angle compared to other algorithms.

Based on the above, the DQN algorithm showed excellent performance in learning rate, discount rate, and exploration probability, achieving better results than other reinforcement learning algorithms.

According to experimental results, the DQN algorithm shows much better performance than other reinforcement learning algorithms in the 'CartPole' game. This is because the DQN algorithm effectively solves the dilemma of exploration and exploitation and increases the stability of learning by using experience replay memory. DQN algorithms can be effectively applied in a variety of game environments and can play an important role in game AI development.

5. CONCLUSION

In this paper, we examined the exploration-exploitation dilemma within a game environment and highlighted the superiority of the Deep Q-Network (DQN) algorithm. The DQN algorithm is a robust reinforcement learning approach that effectively addresses the exploration-exploitation trade-off while enhancing learning stability through the use of experience replay memory. Our research contributes by providing a comprehensive performance comparison of various reinforcement learning algorithms—namely DQN, A2C, and PPO—in the 'CartPole' game environment. This systematic comparison offers valuable insights into the respective strengths and weaknesses of these algorithms, contributing new knowledge to the field of reinforcement learning.

Experimental results demonstrated that the DQN algorithm significantly outperforms other reinforcement learning algorithms in the 'CartPole' game. The findings suggest that DQN algorithms can be efficiently applied to a variety of game environments, playing a crucial role in the development of game AI.

Furthermore, reinforcement learning holds substantial potential for application across various domains. Although it is currently predominantly applied in areas such as games, robot control, and autonomous vehicles, its principles can be extended to diverse fields including medicine, finance, manufacturing, and energy management. This underscores the necessity to identify new application areas and conduct focused research to explore these possibilities. By expanding the scope of reinforcement learning applications, we can leverage its capabilities to innovate and improve outcomes across different industries and sectors.

REFERENCES:

[1] Li, Yuxi. "Deep reinforcement learning: An overview." arXiv preprint arXiv:1701.07274 (2017).
[2] Souchleris, Konstantinos, George K. Sidiropoulos, and George A. Papakostas. "Reinforcement learning in game industry—Review, prospects and challenges." Applied Sciences 13.4 (2023): 2443.
[3] Shao, Kun, et al. "A survey of deep reinforcement learning in video games." arXiv preprint arXiv:1912.10944 (2019).
[4] Crespo, João, and Andreas Wichert. "Reinforcement learning applied to games." SN Applied Sciences 2.5 (2020): 824.
[5] Hester, Todd, et al. "Deep q-learning from demonstrations." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.
[6] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[7] Mnih, V., Kavukcuoglu, K., Silver, D. et al. "Human-level control through deep reinforcement learning", Nature 518, pp. 529–533, 2015.
[8] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.
[9] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
[10] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. PMLR, 2016.
[11] Wu, Yuhuai, et al. "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation." Advances in neural information processing systems 30 (2017).
[12] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
[13] Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.
[14] Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." IEEE Transactions on Systems, Man, and Cybernetics SMC-13.5 (1983): 834-846.
[15] Sutton, Richard S. "Generalization in reinforcement learning: Successful examples using sparse coarse coding." Advances in neural information processing systems (1996): 1038-1044.
[16] Brockman, Greg, et al. "OpenAI Gym." arXiv preprint arXiv:1606.01540 (2016).
[17] Bon-woo Gu, Jun-kyum Kim, and Eun-joo Rhee, "GLSL based Additional Learning Nearest Neighbor Algorithm suitable for Locating Unpaved Road", The Journal of Korea Institute of Information, Electronics, and Communication Technology, 12.1, pp. 29-36, 2019.
[18] Bon-Woo Gu, Kiu-Duck Hwang, and Taewoo Han, "ALNN-based Reinforcement Learning Framework for AI Model Development of 32Bit Client Games: Based on Gomoku games", Journal of Korea Game Society 23.3 (2023): pp. 63-70.
[19] Juliani, Arthur, et al. "Unity: A general platform for intelligent agents." arXiv preprint arXiv:1809.02627 (2018).