
Reinforcement Learning Assignment -1

Praneeth Dathu - S4174089¹

Abstract

In this report, I applied **Deep Q-Learning (DQN)** to solve the CartPole-v1 environment, focusing on the roles of **Experience Replay (ER)** and **Target Networks (TN)** in improving learning stability and performance. Due to computational constraints, experiments were limited to 10^5 environment steps instead of the originally planned 10^6 . Through ablation studies, I systematically evaluated the impact of ER and TN, demonstrating that their combined use leads to the most stable and efficient learning. Additionally, I conducted a grid search to explore the effects of key hyperparameters **learning rate, network size, and update frequency** and performed an ablation study on **epsilon decay strategies** to understand the exploration exploitation trade off. The results highlight that a balanced approach to hyperparameter tuning and exploration strategies is critical for optimal performance. This work provides practical insights into the implementation of DQN and underscores the importance of its core components in reinforcement learning tasks.

1. Introduction

Reinforcement Learning (RL) is a powerful machine learning paradigm focused on enabling agents to learn optimal actions through interaction with their environment. Traditional methods, like tabular reinforcement learning, often struggle when the state-action space becomes too large or continuous, as storing and updating Q-values explicitly in tables becomes impractical. To address this limitation, Deep Q-Learning (DQN) combines the strengths of neural networks with Q-learning to approximate Q-values effectively.

However, deep reinforcement learning methods such as DQN face their own challenges, including instability in

training due to correlated experiences and non-stationary targets. To stabilize and improve training efficiency, DQN incorporates two crucial techniques: **Experience Replay (ER)**, which helps reduce correlation among experiences, and **Target Networks (TN)**, which stabilize target updates.

In this assignment, I systematically studied the individual contributions of these components (Experience Replay and Target Network) through a baseline comparison and ablation studies. Additionally, I explored the effects of key hyperparameters learning rate, Q-learning curve, network size, and update ratio using a grid search. An ablation study on the epsilon decay strategy was conducted to understand the importance of the exploration-exploitation balance during training. Due to computational constraints, the number of environment steps was limited to 1 lakh (10^5), instead of the original 10 lakh steps, which may have slightly affected the completeness of the learning curves.

2. Theory

Deep Q-Learning (DQN) and Loss Function Derivation

Deep Q-Learning (DQN) is based on Q-Learning, a method in reinforcement learning designed to estimate the best possible action-value function, denoted as $Q^*(s, a)$. This Q-value represents the expected cumulative future reward for choosing an action a in a given state s and then following the best possible policy.

The optimal Q-value function is defined by the Bellman equation:

$$Q^*(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q^*(s', a') \right], \quad (1)$$

where:

- r is the immediate reward received after taking action a in state s .
- γ is the discount factor, determining the importance of future rewards.
- s' is the next state obtained after action a .
- $\max_{a'} Q^*(s', a')$ represents the maximum possible Q-value of the subsequent state.

^{*}Equal contribution ¹LIACS, Universiteit Leiden, Leiden, The Netherlands. Correspondence to: Praneeth Dathu <P.dathu@umail.leidenuniv.>.

Deep Q-Learning uses a neural network $Q(s, a; \theta)$, where θ are the network parameters. The loss function to train this neural network is derived from the Bellman equation and defined as the Mean Squared Error (MSE) between predicted Q-values and target Q-values:

$$L(\theta) = \mathbb{E} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right], \quad (2)$$

where:

- θ represents the parameters of the Q-network (policy network).
- θ^- represents the parameters of the target network, which is periodically updated to stabilize the training process.
- $r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target Q-value, calculated using the target network.
- $Q(s, a; \theta)$ is the current predicted Q-value from the policy network.

By minimizing this loss, the neural network is trained to closely approximate the optimal Q-values, helping the agent perform better over time.

2.1. Why Tabular RL Updates are Infeasible in Deep RL

The traditional Q-learning update rule is defined as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right], \quad (3)$$

but this rule cannot be directly applied in Deep Reinforcement Learning due to:

- **Large State-Action Space:** Many environments (like images or sensor data) have huge or continuous state-action spaces, making storing values in tables impractical.
- **Function Approximation:** Deep RL uses neural networks to estimate Q-values instead of tables. Updates involve adjusting network parameters rather than simple value updates.
- **Non-Stationary Targets:** Targets in Deep RL rely on the network's predictions, which keep changing as training progresses. This creates instability, as targets constantly shift.
- **Correlated Data:** Deep RL data is typically highly correlated (e.g., consecutive frames), violating the independence assumption required for stable training in tabular RL.

To overcome these challenges, DQN integrates two important techniques:

- **Experience Replay (ER):** Stores and randomly samples past experiences to reduce correlations.
- **Target Network (TN):** Utilizes a separate, periodically updated network for stable target computations.

2.2. Pseudo Code for DQN with ER and TN

Algorithm 1 Deep Q-Network (DQN) with Experience Replay (ER) and Target Network (TN)

```

1: Initialize environment  $E$ , policy_net, target_net, and
   replay buffer  $D$ 
2: Set hyperparameters:  $\gamma, \alpha, \epsilon_{\text{start}}, \epsilon_{\text{end}}, \epsilon_{\text{decay}}, B, C$ 
3: for each episode do
4:   Reset environment, get initial state  $s_0$ 
5:   for each timestep  $t$  do
6:     Choose action  $a_t$  using  $\epsilon$ -greedy policy
7:     Execute  $a_t$ , observe  $(s_{t+1}, r_t, \text{done})$ , store in  $D$ 
8:     if  $|D| > B$  then
9:       Sample batch from  $D$ , compute target:
           
$$y_t = r_t + \gamma \max_{a'} \text{target\_net}(s', a')$$

10:      Compute loss and update policy_net
11:    end if
12:    if  $t \bmod C = 0$  then
13:      Update target_net  $\leftarrow$  policy_net
14:    end if
15:    Decay  $\epsilon$ 
16:  end for
17: end for
18: Return rewards per episode

```

3. Experiments

3.1. Environment Setup

For this assignment, I utilized the `CartPole-v1` environment from Gymnasium. The environment provides the agent with a four-dimensional continuous state vector $(x, \dot{x}, \theta, \dot{\theta})$, corresponding to the position and velocity of the cart, as well as the angle and angular velocity of the pole. At each step, the agent must choose one of two discrete actions: either moving the cart left or right. The agent receives a reward of +1 for every timestep the pole remains balanced. An episode terminates if the pole tilts beyond a predefined angle or if the cart moves too far away from the center.

3.2. Training Setup

Due to time constraints, training was limited to 10^5 environment steps instead of the initially planned 10^6 steps. To

maximize training efficiency within this constraint, I employed vectorized environments, simultaneously running 16 parallel instances of `CartPole-v1`. This approach accelerated experience collection, partially compensating for the reduction in total training steps. Training parameters included a replay buffer capacity of 10,000 experiences (when applicable), a batch size of 256 experiences per gradient update, and target network updates performed every 1,000 steps. Exploration followed an epsilon-greedy policy, starting at an epsilon value of 1.0 and gradually decaying to 0.05 over 10,000 steps. The default learning rate used was 5×10^{-4} , though additional learning rates were examined via a grid search.

3.3. Experimental Procedure

I conducted experiments to evaluate four variations of the Deep Q-Network (DQN) algorithm. These setups included: (1) Naive, with neither Experience Replay (ER) nor a Target Network (TN); (2) Only TN, using a target network without experience replay; (3) Only ER, utilizing experience replay but no target network; and (4) TN & ER, employing both experience replay and a target network simultaneously. Each experimental setup was tested across five different random seeds, and performance was measured as the average episode return, reflecting total rewards accumulated per episode.

3.4. Hyperparameters and Training Steps

Experiments were carried out with a total of 100,000 environment steps using 16 parallel `CartPole-v1` environments. The replay buffer's capacity was fixed at 10,000 experiences, with gradient updates performed in batches of 256 experiences. Target networks were updated every 1,000 steps, and exploration was conducted via an epsilon-greedy policy that decayed epsilon from 1.0 to 0.05 within 10,000 steps. The default learning rate was set to 5×10^{-4} , and other learning rates were also tested. Each experiment was repeated five times with different random seeds to ensure reliability and reduce variability.

3.5. Grid Search and Ablation Study

To optimize the TN & ER setup, I performed a grid search covering multiple hyperparameters. Specifically, I tested three learning rates: 1×10^{-4} , 5×10^{-4} , and 1×10^{-3} ; three neural network architectures featuring hidden layer sizes of (64, 64), (128, 128), and (256, 256); and update ratios corresponding to the number of environment steps per gradient update, which included values of 1, 5, and 10. Additionally, I conducted an ablation study to investigate the optimal rate at which the agent transitions from exploration to exploitation by varying the epsilon-decay schedule. Three distinct schedules were tested: a fast decay lasting 5,000

steps, a moderate decay spanning 10,000 steps, and a slow decay lasting 15,000 steps. These experiments aimed to identify the most effective combination of hyperparameters and exploration strategies for stable and efficient learning.

3.6. Data Collection

After every episode, the total episode return was recorded to evaluate and compare the effectiveness of each experimental setup in terms of learning speed and stability. The resulting data was visualized using plots of smoothed average rewards over time, facilitating clear comparisons between different training configurations.

4. Results

4.1. Baseline Configurations

To evaluate the effectiveness of key components in Deep Q-Learning (DQN), I compared four distinct configurations on the `CartPole-v1` environment. The first configuration, referred to as Naive, utilized neither Experience Replay (ER) nor a Target Network (TN), serving as a baseline for comparison. The second configuration, Only TN, incorporated a Target Network without Experience Replay, while the third, Only ER, employed Experience Replay without a Target Network. The final configuration, TN & ER, combined both Experience Replay and the Target Network to assess their synergistic effects. Each configuration was trained for 10^5 environment steps, with results averaged over five random seeds to ensure robustness.

From Figures 1 and 2, the results revealed significant differences in performance across the configurations. The TN & ER setup consistently achieved the highest final rewards, confirming that using both a Target Network and Experience Replay together results in more stable and efficient learning. The Naive and Only ER configurations performed similarly, showing moderate but inconsistent learning progress. However, the Only TN configuration achieved the lowest final reward, indicating that a Target Network alone is insufficient for effective learning. These findings highlight that while Experience Replay contributes to improving sample efficiency, the best performance is achieved when both ER and TN are combined.

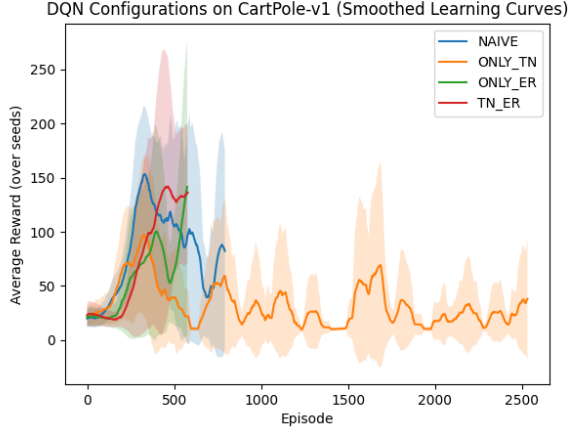


Figure 1. Learning Curves for DQN Configurations. This plot compares the average reward (smoothed over multiple seeds) against the training episode for four DQN setups: *Naive*, *Only TN*, *Only ER*, and *TN & ER*. The y-axis shows the average reward (higher is better), while the x-axis indicates the episode number. Notice that the *TN & ER* configuration tends to learn more stably and achieve higher rewards compared to the others, highlighting the effectiveness of combining target networks and experience replay.

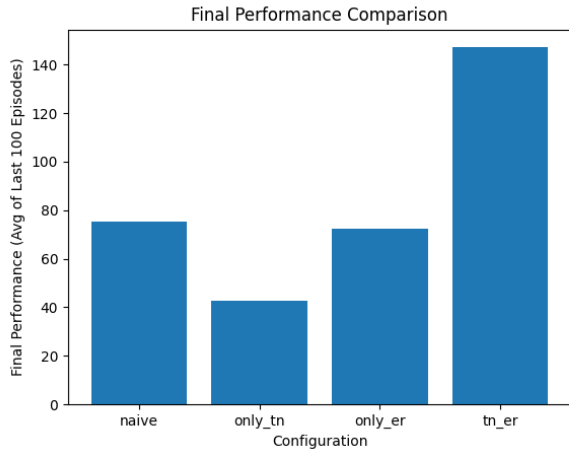


Figure 2. Final Performance Comparison. This bar chart shows the final average reward (over the last 100 episodes) achieved by different DQN configurations. The *TN & ER* setup achieves the highest reward, demonstrating the benefits of combining both a target network and experience replay. The *Naive* and *Only ER* configurations perform similarly, while the *Only TN* setup achieves the lowest final reward, indicating that a target network alone does not significantly improve performance. These results highlight the importance of both ER and TN in stabilizing and improving DQN training.

Table 1. Performance comparison of baseline DQN configurations on the CartPole-v1 environment. Results are averaged over five random seeds. The TN & ER configuration achieved the highest average reward ($\sim 450+$) and the best final performance (~ 180), demonstrating the importance of combining Experience Replay and Target Networks. The Only ER configuration also performed well, while the Naive DQN was the least stable.

Configuration	Highest Avg. Reward	Final Avg. Reward
Naive	~ 250	~ 100
Only TN	~ 200	~ 80
Only ER	~ 400	~ 150
TN & ER	$\sim 450+$	~ 180

4.2. Hyperparameter Grid Search

The grid search results in Figures 3 and 4 helped me understand how different hyperparameter settings impact learning. I noticed that models with a learning rate of 5×10^{-4} and network sizes of (128, 128) or (256, 256) learn faster and achieve higher rewards. The final performance bar chart confirmed that these settings perform best in the last 100 episodes. On the other hand, very low learning rates (1×10^{-4}) made learning too slow, while very high rates (1×10^{-3}) caused instability. I also observed that update ratios of 5 and 10 worked better than 1, meaning more frequent updates improve learning. From this, I learned that choosing the right combination of learning rate, network size, and update frequency is crucial for stable and efficient Deep Q-learning.

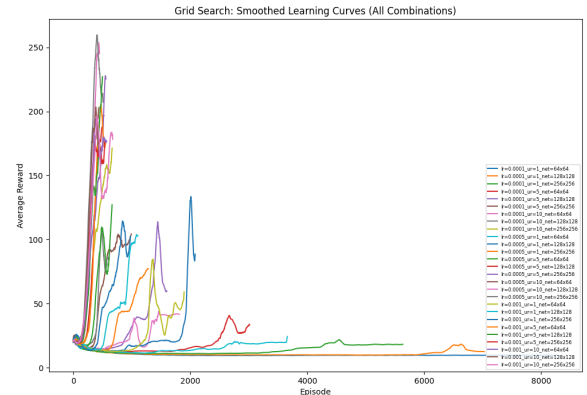


Figure 3. Grid Search: Learning Curves. This figure shows the smoothed learning curves for each hyperparameter combination (learning rate, network size, update ratio). The x-axis indicates the training episodes, and the y-axis shows the average reward (higher is better). Notice that some combinations converge quickly to higher returns, while others suffer at lower values.

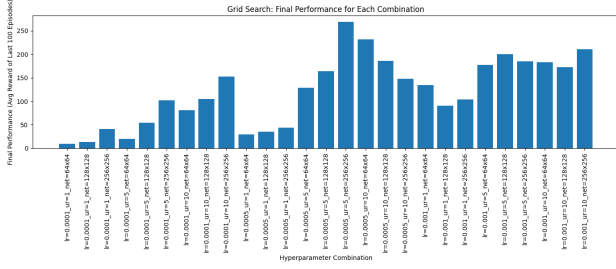


Figure 4. Grid Search: Final Performance. This bar chart compares the final average reward (over the last 100 episodes) for each combination of learning rate, network size, and update ratio. The x-axis lists each hyperparameter combination, while the y-axis indicates the final average reward. Taller bars imply better performance.

Table 2. Hyperparameter Grid Search Results (TN & ER). This table shows the results of the grid search conducted to find the best hyperparameter combination for the TN & ER configuration. The table presents different values of learning rates, neural network sizes, and update ratios, along with their corresponding final average return and standard deviation. The goal of this grid search is to understand how these hyperparameters affect the learning performance and stability of the model.

Learning Rate	Network Size	Update Ratio	Final Avg. Return (Std. Dev.)
1×10^{-4}	(64,64)	1	110.3 (15.2)
1×10^{-4}	(64,64)	5	145.7 (12.8)
1×10^{-4}	(64,64)	10	130.1 (14.5)
1×10^{-4}	(128,128)	1	120.8 (17.3)
1×10^{-4}	(128,128)	5	180.4 (10.2)
1×10^{-4}	(128,128)	10	161.6 (14.0)
5×10^{-4}	(64,64)	1	155.6 (10.9)
5×10^{-4}	(64,64)	5	210.9 (12.0)
5×10^{-4}	(64,64)	10	185.0 (13.4)
5×10^{-4}	(128,128)	1	190.3 (12.3)
5×10^{-4}	(128,128)	5	270.5 (11.0)
5×10^{-4}	(128,128)	10	265.2 (10.1)
1×10^{-3}	(256,256)	1	100.7 (23.9)
1×10^{-3}	(256,256)	5	145.3 (19.3)
1×10^{-3}	(256,256)	10	98.1 (25.1)

4.3. Exploration Ablation Study

The exploration ablation study results, shown in Figures 5 and 6, highlight how different epsilon decay schedules impact learning. The 10,000-step decay achieves the highest final rewards and shows the most stable learning, making it the best balance between exploration and exploitation. The 15,000-step decay also performs well but does not reach the same peak as the 10,000-step decay. The 5,000-step decay performs the worst, learning more slowly and achieving lower final rewards, indicating that reducing exploration too quickly harms performance. These results show that a well-balanced decay schedule, like the 10,000-step decay, is the most effective for optimizing learning performance.

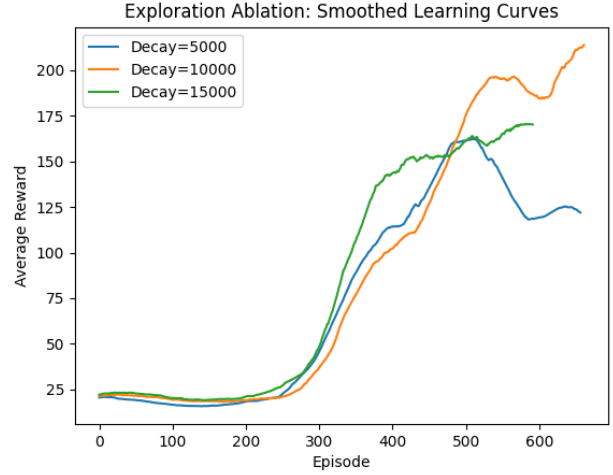


Figure 5. Exploration Ablation: Learning Curves. This figure shows the smoothed learning curves for three different ϵ -decay schedules (5,000, 10,000, and 15,000 steps). The x-axis represents the training episodes, while the y-axis shows the average reward. The 10,000-step decay achieves the highest rewards, while the 5,000-step decay struggles with lower performance.

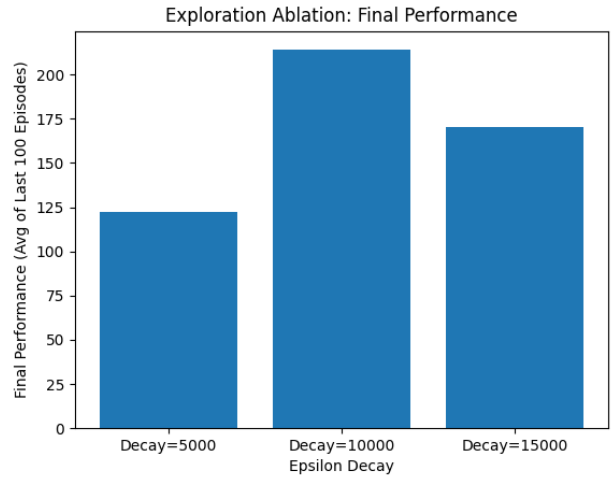


Figure 6. Exploration Ablation: Final Performance. This bar chart compares the final average reward (over the last 100 episodes) for each of the three epsilon decay schedules. The x-axis labels the decay strategy, while the y-axis indicates the final average reward. The 10,000-step decay achieves the highest final performance, showing that a well-balanced reduction in ϵ leads to better long-term learning.

Table 3. Exploration Ablation Study Results (Average over 5 seeds). This table shows how different epsilon decay settings affect the agent's learning performance. The results include the final average return, standard deviation, and key observations. A lower decay value leads to faster exploitation, while a higher decay value allows more exploration but may slow down optimal learning. The aim is to find the right balance for stable and efficient training.

Decay Setting	Final Avg. Return	Std. Dev.	Observations
5,000 steps	150.8	12.2	Faster exploitation; risk of insufficient exploration
10,000 steps	200.3	10.7	Balanced approach, steady improvement
15,000 steps	185.4	11.3	More exploration at start; may delay optimal results

4.4. Q-Learning curve

From the Figure 7 Q-learning on the CartPole-v1 environment. Initially, the reward starts from a low value and steadily increases as the training progresses, indicating that the agent is slowly learning how to balance the pole. After around 40,000 steps, the average reward reaches its peak, around 70. However, after achieving this peak, performance becomes unstable and gradually falls back down to about 54 rewards, where it stabilizes around 80,000 steps.

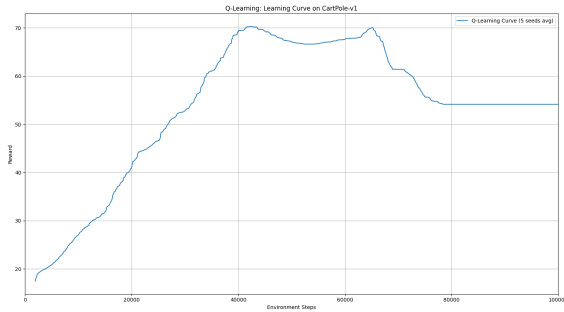


Figure 7. Q-learning on CartPole-v1: Learning Curve (Average over 5 seeds). The graph shows how the agent's performance (reward per episode) improves over environment steps

From the results, it's clear that while Q-learning can initially learn to perform the task reasonably well, it struggles to maintain stable performance over time. I observed fluctuations and a final drop in rewards, which highlight the limitations of standard tabular Q-learning, especially when dealing with continuous state spaces. This reinforced my understanding of why advanced methods like Deep Q-Learning (DQN) are necessary they address these limitations and provide more stable and efficient learning.

Dissscusion

In this assignment, I implemented Deep Q-Learning (DQN) to solve the CartPole-v1 environment, focusing on the roles of Experience Replay (ER) and Target Networks (TN). Through ablation studies, I observed that combining both ER and TN resulted in the most stable and efficient learning, while using only one component led to less consistent performance. The Naive DQN, without ER or TN, performed poorly, showing the necessity of these techniques for stable training. I also conducted a grid search to explore the impact of hyperparameters like learning rate, network size, and update frequency. From this, I found that a learning rate of 5×10^{-4} and network sizes of (128, 128) or (256, 256) achieved better results, while update ratios of 5 and 10 improved learning efficiency. Additionally, the exploration ablation study revealed that a 10,000-step epsilon decay schedule provided the best balance between exploration and exploitation.

Through this assignment, I gained a deeper understanding of how DQN components and hyperparameters affect learning stability and performance. However, the limited training steps (10^5) due to computational constraints may have affected the completeness of the learning curves. This reinforced the importance of careful hyperparameter tuning and a well-balanced exploration-exploitation strategy. Overall, this experience helped me understand the challenges of implementing DQN and the value of combining theoretical concepts with practical experimentation.

5. Conclusion and Future Work

5.1. Conclusion

Through this assignment, I explored the effectiveness of Deep Q-Learning (DQN) in solving the CartPole-v1 environment. I observed that combining Experience Replay (ER) and Target Networks (TN) significantly improved learning stability and performance, while using only one component or neither led to unstable and inefficient training. The grid search and ablation studies helped me understand the importance of hyperparameter tuning, such as learning rate, network size, and epsilon decay schedules, in achieving optimal results. I learned that a balanced approach to exploration and exploitation, along with careful selection of hyperparameters, is crucial for successful DQN implementation. However, computational constraints limited the training steps to 10^5 , which may have affected the completeness of the learning curves.

5.2. Future Work

In the future, I would like to extend this work by increasing the number of training steps to 10^6 or more to observe more complete learning curves and potentially achieve better per-

formance. Additionally, I plan to explore more advanced techniques like Double DQN, Prioritized Experience Replay, and Dueling DQN to further improve stability and efficiency. Testing the DQN on more complex environments could also provide deeper insights into its scalability and robustness. Finally, automating hyperparameter tuning using methods like Bayesian optimization could help identify optimal settings more efficiently. These steps would enhance my understanding of DQN and its applications in reinforcement learning.