HYPERPARAMETER TUNING FOR FROZENLAKE-V1 ENVIRONMENT

Group 11:
YangYueqi 12211056
OuWeijuan 12212252
LiSiqing 12212964

ABSTRACT

This report investigates the impact of hyperparameter tuning on the performance of the Deep Q-Learning (DQN) algorithm in the FrozenLake-v1 environment. Key hyperparameters such as learning rate, discount factor, epsilon-greedy parameters, and memory capacity were systematically analyzed using univariate and randomized search methods. The study also compares the performance of DQN with Double DQN and Dueling DQN across different parameter settings. Results highlight that Dueling DQN outperforms DQN in terms of stability, efficiency, and generalization, making it a more robust choice for complex reinforcement learning tasks. Additionally, factors contributing to reward fluctuations despite decreasing loss, such as exploration-exploitation tradeoffs and replay buffer diversity, are discussed.

1 INTRODUCTION

Reinforcement Learning (RL) is a machine learning paradigm where agents learn by interacting with an environment to maximize rewards. Deep Q-Network (DQN)(1) is an important algorithm in deep reinforcement learning that combines Q-learning with deep learning techniques. In Q-learning, the agent learns the state-action value function $Q(s, a)$, which represents the expected return of taking action a in state s. DQN approximates the Q-value function $Q(s, a)$ using a deep neural network, enabling it to handle high-dimensional state spaces (e.g., images). The input to the network is the state, and the output is the Q-values for all possible actions.

The performance of DQN is highly sensitive to hyperparameters. This report investigates the impact of these hyperparameters on DQN's performance in the FrozenLake-v1 environment. The exploration begins with a baseline configuration and extends to both univariate and randomized hyperparameter tuning experiments.

2 HYPERPARAMETER TUNING IN DQN

Deep Q-Network (DQN) is an algorithm that combines Q-learning with deep learning techniques to address the challenges of high-dimensional state spaces in reinforcement learning. DQN utilizes a deep neural network to approximate the Q-value function, which estimates the expected future rewards for a given state-action pair.

2.1 BASELINE CONFIGURATION

A baseline configuration for DQN was established with the following parameters:

The training process involved 10000 episodes with this configuration. The results were visualized through cumulative reward curves and loss curves, providing a baseline for further analysis.

The rewards(left panel) show an upward trend, stabilizing at 0.6–0.8 after 4000 episodes, indicating effective policy learning. Significant fluctuations persist even after convergence, suggesting room for improvement in exploration-exploitation balance.

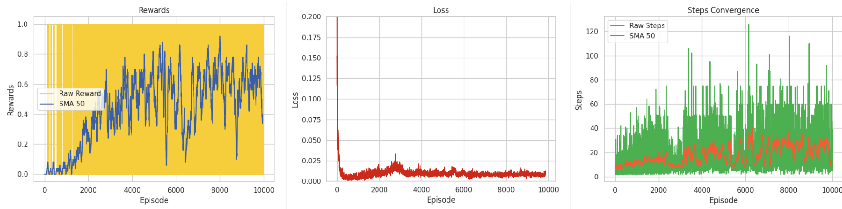


Figure 1: DQN Baseline Visualization

Loss(middle panel) decreases sharply in the first 500 episodes and stabilizes at a low value (0.01) after 1000 episodes, reflecting successful Q-value approximation. Occasional spikes in loss indicate the agent encountering rare transitions or new states. Steps Convergence(right panel) reveals the number of steps decreases and converges to 20–40 steps per episode after 4000 episodes, indicating efficient task completion. Rare spikes in steps suggest episodes with suboptimal action sequences due to exploration. The current parameter configuration enables the agent to learn effectively but shows limitations in achieving optimal performance. Adjustments, such as slower epsilon decay or reward shaping, could further enhance policy stability and convergence.

2.2 Univariate Hyperparameter Analysis

2.2.1 Hyperparameter Explaination

Univariate hyperparameter tuning was conducted to investigate the impact of varying individual hyperparameters while keeping others constant. Among the 16 parameters in the DQN algorithm, we selected five parameters that are most influential on performance for further exploration. The parameters analyzed were: **Learning Rate** $\left[1 \times 10^{-4}, 6 \times 10^{-4}, 1 \times 10^{-3}\right]$, which determines how much the model updates in response to an error during training; **Discount Factor** $[0.9, 0.93, 0.95]$, which controls the importance of future rewards in the agent's decision-making; **Epsilon Decay** $[0.995, 0.999, 0.9995]$, which gradually reduces the exploration rate (ϵ) as the agent learns; **Epsilon Min** $[0.01, 0.005, 0.0015]$ which sets the minimum exploration rate (ϵ) the agent can reach. It Ensures the agent always explores to some extent, preventing it from becoming overly exploitative; **Memory Capacity** $[2000, 4000, 8000]$, which defines the size of the replay buffer used to store past experiences for training.

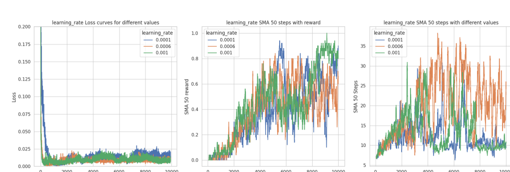
For each parameter, the training process was repeated for 1000 and 10000 episodes, and the results were recorded, including rewards, loss curves, and training steps. A sliding window of size 50 was used to compute the Simple Moving Average (SMA) for smoother visualizations of rewards and steps.

2.2.2 Visualization

Custom plots were generated for each parameter to visualize loss curves, the simple moving average (SMA) of rewards, and the SMA of steps per episode over the training episodes. This methodology facilitated a detailed comparison of how each hyperparameter influences the performance of DQN.

1. Univariate Search for Learning Rate

(a) Results for 1000 Episodes

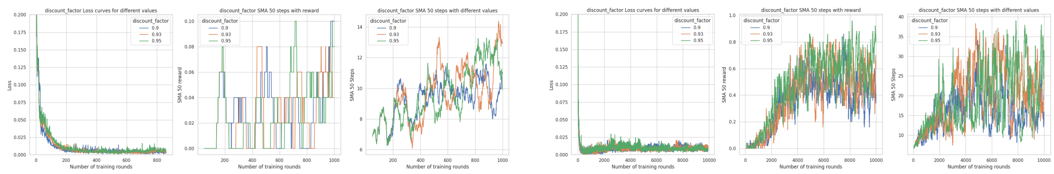


(b) Results for 10000 Episodes

Figure 2: Learning Rate Searching Results

The results indicate that a moderate learning rate of 0.0006 achieves the best balance between convergence speed, stability, and performance. In contrast, the higher learning rate of 0.001 initially converged faster but displayed greater volatility in later episodes, suggesting overcorrection or instability in the Q-value updates. The lowest learning rate of 0.0001 remained the slowest to converge, stabilizing at a higher loss value of approximately 0.01. Therefore, 0.0006 is the optimal choice for further experiments and refinement.

2. Univariate Search for Discount Factor

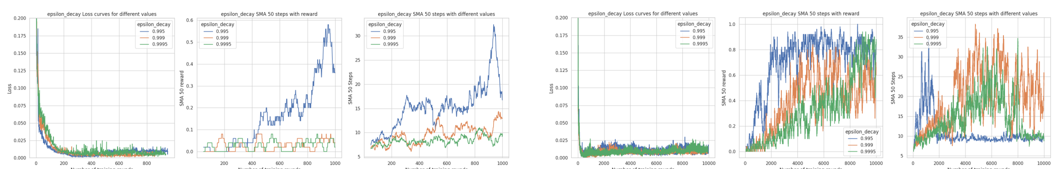


(a) Results for 1000 Episodes (b) Results for 10000 Episodes

Figure 3: Discount Factor Searching Results

The results demonstrate that a discount factor of 0.95 yields the best overall performance, maintaining a stable and low loss. In comparison, a discount factor of 0.93, slightly reduces the emphasis on future rewards, which has a less effective exploration of long-term strategies. The lowest discount factor, 0.9, converges more quickly but prioritizes immediate rewards. This short-sightedness can lead to suboptimal policies, as the agent may fail to recognize the value of delayed rewards.

3. Univariate Search for Epsilon Decay

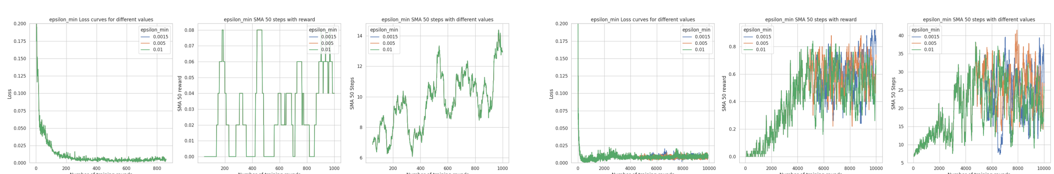


(a) Results for 1000 Episodes (b) Results for 10000 Episodes

Figure 4: Epsilon Decay Searching Results

The results indicate that an epsilon decay of 0.995 achieves the best balance between exploration and exploitation, leading to higher rewards and fewer steps per episode while maintaining stable and low loss over 5000 episodes. In contrast, smaller epsilon decay may shows faster initial convergence but results in erratic rewards and higher steps due to insufficient exploration. Meanwhile, 0.9995 promotes excessive exploration, leading to slower convergence and lower overall performance.

4. Univariate Search for Epsilon Min



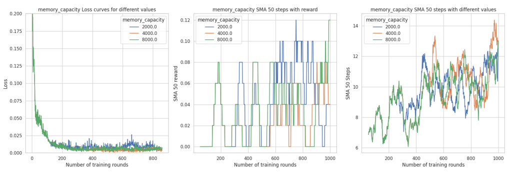
(a) Results for 1000 Episodes (b) Results for 10000 Episodes

Figure 5: Epsilon Min Searching Results

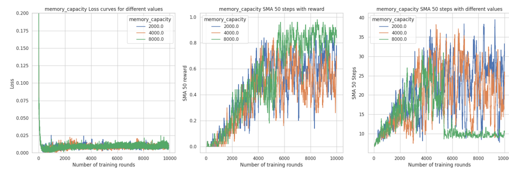
The overlap in curves for different epsilon minimum values (epsilon min) happens because epsilon just sets the minimum exploration level. During training, epsilon gradually decreases, making epsilon min important only later. This delay means the agent's performance is similar early on, with differences showing only later when exploration-exploitation balance matters.

Results show that epsilon min of 0.005 gives the best, yielding higher rewards and fewer steps over 5000 episodes with low, stable loss. Lower (0.0015) or higher (0.01) values lead to slower convergence or inefficient policies, respectively. Thus, 0.005 is the best choice for future experiments.

5. Univariate Search for Memory Capacity



(a) Results for 1000 Episodes



(b) Results for 10000 Episodes

Figure 6: Memory Capacity Searching Results

The results demonstrate that a memory capacity of 8000 achieves the best balance between performance, stability, and efficiency. A smaller memory capacity (2000) learns faster but has larger fluctuations and unstable long-term performance; a larger memory capacity (8000) learns slower but has higher final rewards and more stable performance, making it suitable for scenarios that require long-term results.

2.3 Randomized Hyperparameter Search

It is evident that exploring a single variable is insufficient for comprehensive hyperparameter tuning, as optimizing an individual hyperparameter does not guarantee the identification of the optimal combination for the model. To address this, a randomized search strategy was employed to optimize the hyperparameters of the DQN agent. This section presents two distinct approaches: *random grid search* and *Optuna-based optimization*, accompanied by a thorough analysis of their results and corresponding visualizations.

2.3.1 Random Grid Search

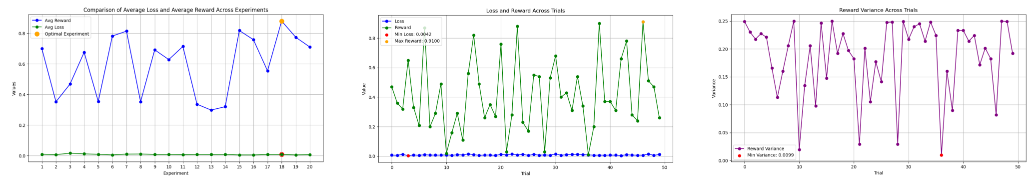
The initial stage of hyperparameter tuning involved a random grid search across a limited parameter space. Grid search can help us determine a parameter optimization interval. The key hyperparameters and their ranges included the learning rate (η) values of $[6 \times 10^{-4}, 7 \times 10^{-4}, 8 \times 10^{-4}]$, the discount factor ($\gamma$) values of $[0.95, 0.96, 0.98]$, the epsilon decay rate (ϵ_{decay}) options of $[0.998, 0.999, 0.9995]$, the minimum epsilon (ϵ_{min}) values of $[0.005, 0.0055]$, and the memory capacity ranging from $[4000, 8000]$.

A total of 30 configurations were evaluated, each trained for 5000 episodes.The performance metrics included the average rewards and average losses calculated over the last 100 episodes, along with their respective variances. Figure 7a visualizes the average loss and reward across the experiments. The best-performing configuration achieved an average reward of 0.880 with an average loss of 0.0072 (highlighted in orange).

As shown in Figure 7a, the reward distribution varied significantly across experiments, reflecting the sensitivity of the agent's performance to hyperparameters. While most configurations yielded moderate rewards, a few configurations outperformed others, demonstrating the importance of fine-tuning. The comparison of loss and reward metrics further reveals a trade-off between optimizing reward and minimizing loss. Additionally, reward variance across experiments indicates variability, with the most stable configuration achieving a variance of 0.0099.

2.3.2 Optuna-based optimization

Optuna is an automatic hyperparameter optimization framework that explores the parameter space through efficient sampling algorithms, terminates inefficient experiments in advance using pruning algorithms, and supports parallel optimization and multi-objective optimization to accelerate the parameter tuning process of machine learning models. It also provides visualization tools to help analyze and understand the impact of hyperparameters.



(a) Randomized Grid Search Result

(b) Optuna Random Search Average Loss and Reward

(c) Optuna Random Search Reward Variance

Figure 7: Results for Random Search

To further refine hyperparameter tuning, Optuna-based multi-objective optimization was employed, expanding the parameter search space as follows: the learning rate (η) ranges from 1×10^{-4} to 1×10^{-3} (log scale), the discount factor (γ) spans $[0.85, 1.0]$, the epsilon decay rate (ϵ_{decay}) is set between $[0.85, 1.0]$, the minimum epsilon (ϵ_{min}) varies from $[0.001, 0.2]$, and the memory capacity ranges from $[6000, 9000]$.

The objective was to minimize the average loss while maximizing the average reward (modeled as minimizing the negative reward). A total of 50 trials were conducted, with results summarized in Figure 7b. The best trial achieved an **average reward of** 0.910 and an **average loss of** 0.0073 using the following hyperparameters: a learning rate of 3.21×10^{-4}, a discount factor of 0.874, an epsilon decay rate of 0.908, a minimum epsilon of 0.0051, and a memory capacity of 7238.

Figure 7c highlights the reward variance across trials, with the most stable configuration achieving a variance of 0.0819. These results demonstrate the efficacy of Optuna in exploring a more granular parameter space, leading to improved performance and stability compared to the random grid search.

3 Double DQN Analysis

3.1 Traditional DQN Methods and Their Limitations

Deep Q-Network (DQN) has been widely used for reinforcement learning tasks but faces notable limitations:

Redundant Computation: DQN estimates Q-values $Q(s, a)$ for all actions independently, even when the differences between actions are insignificant, leading to unnecessary computation.

Generalization Issues: Traditional DQN lacks the ability to leverage shared information between actions, resulting in poor generalization across states.

Slow Convergence: Redundant Q-value estimation slows down the learning process and affects convergence speed.

Limited Interpretability: DQN does not differentiate between the intrinsic value of a state and the relative advantage of actions, reducing the interpretability of the model.

3.2 Double DQN

Double Deep Q-Network(Double DQN)(2) is an enhancement of the DQN algorithm designed to address the "maximization bias" issue in action value prediction. In DDQN, the target Q-value Y is computed using the following equation:

$$Y = r + \gamma Q(s', \arg\max_{a'} Q(s, a; \theta); \theta^{-}) \quad (1)$$

where Y is the target Q-value, r is the immediate reward, s' is the next state after taking action a, θ represents the parameters of the main Q-network, and θ^{-} represents the parameters of the target Q-network. The key aspect of this formula is the separation of action selection and value estimation. Here's a concise overview:

- **Principle:**Double DQN uses two separate Q-functions to eliminate overestimation bias, with one for action selection and the other for value evaluation.
- **Separation of Action Selection and Value Evaluation:**It selects actions based on the online Q-network and evaluates their Q-values using the target Q-network, which helps to reduce overestimation.
- **Addressing Overestimation:**By separating action selection from value evaluation, Double DQN mitigates the overestimation problem, even if there's bias in the target network's predictions.
- **Implementation:**It typically involves two identical Q-networks, experience collection, and updating the online Q-network parameters using sampled experiences and calculated target Q-values.

3.2.1 Optuna-based optimization

Optuna can be effectively utilized for hyperparameter tuning in Double DQN models.

Through 50 independent experiments aimed at minimizing losses and maximizing rewards, Optuna determined the optimal parameter combination based on the average loss and reward over the last 100 generations. The best result obtained was a reward of 0.79, with the corresponding parameter values as follows: a learning rate (η) of 0.0002158, a discount factor (γ) of 0.8786021, an epsilon decay rate (ϵ_{decay}) of 0.9250076, a maximum epsilon (ϵ_{max}) of 0.9972755, a minimum epsilon (ϵ_{min}) of 0.0162136, and a memory capacity of 6530.

Although Optuna successfully selected a parameter combination that maximizes reward and minimizes loss, these parameters were not satisfactory. In addition, during the model training process using these parameters, the average loss and reward values of the last 100 generations showed significant volatility, which may suggest issues with the stability and generalization ability of the model.

The unsatisfactory optimization results of Optuna may be due to the large parameter space or insufficient number of experiments. When the parameter space is too wide, Optuna needs to conduct more experiments to fully explore, and if the number of experiments is not enough to cover the vast parameter space, it will be difficult to find the optimal solution. Due to limitations in computing power, we are unable to obtain the truly optimal combination of parameters.

4 Dueling DQN Analysis

4.1 Limitations of Double DQN

Double DQN, despite its advancements over traditional Q-learning, has certain limitations that restrict its effectiveness in some scenarios:

- **Limited state value estimation:** Double DQN cannot effectively separate state value and action advantage, which reduces efficiency for redundant actions.
- **Poor adaptability to high-dimensional action spaces:** Double DQN lacks the capability to model relative action values accurately, limiting its performance in high-dimensional settings.
- **Weaker robustness to noise:** Double DQN is less stable in noisy environments compared to methods like Dueling DQN.
- **Slower convergence:** Double DQN exhibits a less efficient learning process, characterized by slower convergence and reduced stability in contrast to Dueling DQN.

4.2 Dueling DQN: Theory and Innovations

Dueling DQN(3) introduces a new architecture to address these limitations by decomposing the Q-value function $Q(s, a)$ into two components:

- **Value Function $V(s)$:** Represents the intrinsic value of the state s.

- **Advantage Function** $A(s, a)$**:** Captures the relative advantage of action a compared to other actions in the same state.

The Q-value is computed as:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right)$$

Here, $\frac{1}{|A|} \sum_{a'} A(s,a')$ normalizes the advantage function by subtracting the mean advantage, ensuring that the advantage function has a mean of zero.

4.3 Comparison of DQN and Double DQN Sensitivity to a Shared Hyperparameter

The following analysis compares the sensitivity of DQN and Double DQN to the same hyperparameter, highlighting their respective performance differences.

Learning Rate: Dueling DQN consistently outperforms DQN across various learning rates. While DQN performs stably only at low learning rates (e.g., 0.0001) but struggles with fluctuations at higher rates (e.g., 0.0006 and 0.001), Dueling DQN achieves smoother reward curves, faster convergence, and higher stability due to its architectural separation of Value and Advantage streams (see Figure 8a).

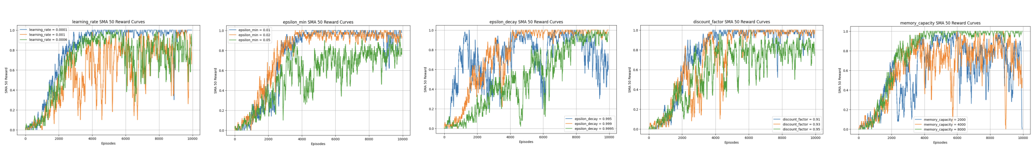
Epsilon Min: Across different minimum exploration probabilities, Dueling DQN shows superior performance. At lower values (e.g., 0.0015), both methods are stable, but Dueling DQN achieves higher final rewards. At moderate values (e.g., 0.005), it strikes a balance between exploration and exploitation, offering faster convergence and stability. At higher values (e.g., 0.01), Dueling DQN still outperforms DQN despite increased reward fluctuations (see Figure 8b).

Epsilon Decay: Dueling DQN demonstrates greater adaptability and stability across exploration decay rates. At slower decay rates (e.g., 0.995), it avoids the significant fluctuations seen in DQN. At moderate decay rates (e.g., 0.999), both methods converge quickly, but Dueling DQN achieves higher rewards. Even at faster decay rates (e.g., 0.9995), Dueling DQN mitigates early exploration stopping effectively (see Figure 8c).

Discount Factor: Dueling DQN outperforms DQN across discount factors by balancing short-term and long-term rewards more effectively. At lower discount factors (e.g., 0.91), Dueling DQN maintains higher rewards. At higher values (e.g., 0.95), it remains stable, unlike DQN, which experiences instability. Moderate values (e.g., 0.93) offer the fastest convergence and optimal reward levels (see Figure 8d).

Memory Capacity: Dueling DQN exhibits better stability and efficiency than DQN across memory capacities. While DQN struggles at smaller capacities (e.g., 2000) and stabilizes only at larger ones (e.g., 8000), Dueling DQN maintains stability across all capacities, achieving faster convergence and higher rewards, particularly at 8000 (see Figure 8e).

Each hyperparameter impacts performance differently; careful tuning based on task requirements is essential (see Figure 8 for an overview).



(a) Learning Rate (b) Epsilon Min (c) Epsilon Decay (d) Discount Factor (e) Memory Capa

Figure 8: Hyperparameter Impact on Performance of Dueling DQN

4.4 Performance Analysis

- **Optimal Hyperparameters:** Learning rate (η): 1×10^{-4}, Discount factor (γ): 0.93, Epsilon decay rate (ϵ_{decay}): 0.999, Maximum epsilon ($\epsilon_{\max}$): 0.999, Minimum epsilon ($\epsilon_{\min}$): 0.001, Memory capacity: 10,000.
- **Convergence Speed:** Dueling DQN converged within approximately 4000 steps during training.
- **Average Reward:** The average reward achieved by Dueling DQN stabilized close to 1.
- **Loss Trend:** Dueling DQN demonstrated a more stable and smoother loss reduction throughout the training process.

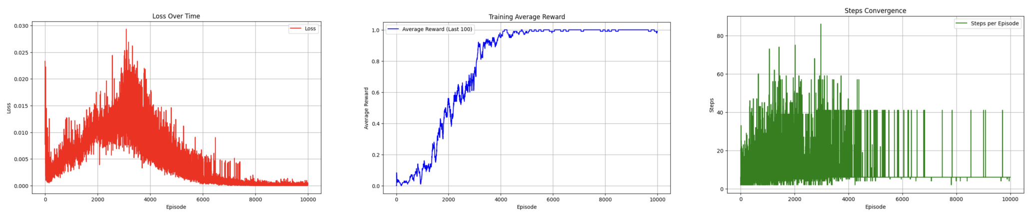


Figure 9: Best Performance Results of Dueling DQN

4.5 Advantages of Dueling DQN

The key advantages of Dueling DQN include: **Efficiency**, as it reduces redundant computation of Q-values for similar actions; **Stability**, achieved by separating state and action characteristics to improve learning robustness; **Generalization**, which enhances the model's ability to adapt across different states and actions; and **Interpretability**, offering better insights into the value of states and the relative importance of actions.

5 Discussion and Conclusion

5.1 Why Does the Loss Decrease While Rewards Fluctuate?

This section explains the phenomenon where the loss decreases while rewards exhibit fluctuations. The key factors include:

- **Exploration vs. Exploitation Tradeoff:** As epsilon decreases, exploration is reduced, and exploitation increases. However, environmental randomness can still lead to reward instability.
- **Local Optima Jumps:** The model transitions from one local optimum to another, causing changes in strategy performance and resulting in reward fluctuations.
- **Replay Buffer Diversity:** Updated experience samples shift the data distribution, leading to short-term performance variations and reward instability.

5.2 Conclusion

This study highlights the significant impact of hyperparameter tuning on the performance of the DQN algorithm and its variants in the FrozenLake-v1 environment. The results show that Dueling DQN provides superior stability, faster convergence, and better adaptability across different hyperparameter configurations compared to DQN and Double DQN. Specifically, it excels in high-dimensional action spaces and under varying exploration and exploitation settings. The analysis also sheds light on the underlying reasons for reward fluctuations despite decreasing loss, emphasizing the importance of balancing exploration and exploitation, managing replay buffer diversity, and addressing local optima jumps. Future work could further refine these models by incorporating advanced optimization methods and exploring their application in more complex environments.

References

[1] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.