Project#2 report

111550116 王紹安

Video link:

https://youtu.be/Fl0Xq6ZQ1fk

https://youtu.be/n6ebNVSIwZg

1 Introduction

In this project, I implemented Q-learning and Deep Q-Network (DQN) algorithm using PyTorch. DQN combines the core idea of Q-learning with the function approximation capability of deep neural networks, enabling it to handle high-dimensional state spaces. To stabilize training, I implemented two classic techniques: Experience Replay and a Target Network.

For the learning environment, I used Gym (OpenAI Gym), which provides a standardized interface for reinforcement learning tasks. It allowed me to easily test and benchmark the DQN agent on classic control problems such as Blackjack-v1 and ALE/Assault-v5, while ensuring reproducibility and consistency across experiments.

2 Method

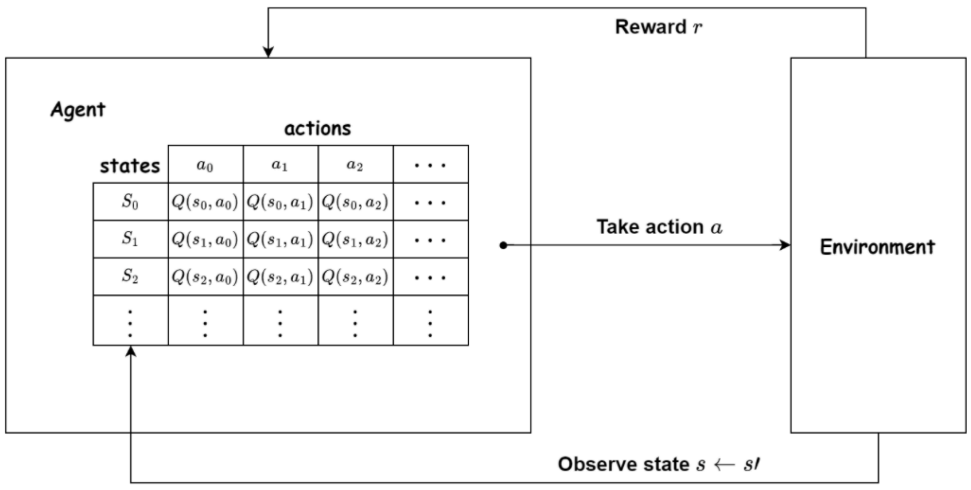
2-1 Q-learning [1][2]

Q-Learning is a model-free, off-policy reinforcement learning algorithm used to learn the optimal action-value function, known as the Q-function. It aims to estimate the maximum expected future reward for each state-action pair without requiring a model of the environment's dynamics. The Q-function is iteratively updated using the Bellman equation:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

where s is the current state, a is the selected action, r is the immediate reward, s' is the next state, α is the learning rate, and γ is the discount factor. The term max Q(s', a') represents the greedy estimate of the future reward, which is why Q-learning is considered off-policy — it learns the optimal policy independently of the agent's behavior policy.

The primary data structure in Q-Learning is the Q-Table, a table that stores the Q-values for all state-action pairs. During training, the agent queries the Q-Table for a given state s and selects the action with the highest Q-value. Additionally, querying the Q-Table often involves challenges such as value instability. For instance, if Q-values are initialized too high, the agent may develop an incorrect preference for certain actions early in training. If the Q-values are not properly normalized or constrained within reasonable bounds, the training process can become unstable, with the Q-values diverging or oscillating. Thus, while querying the Q-Table seems simple, it reflects the algorithm's heavy reliance on state design, initialization strategies, and exploration mechanisms.

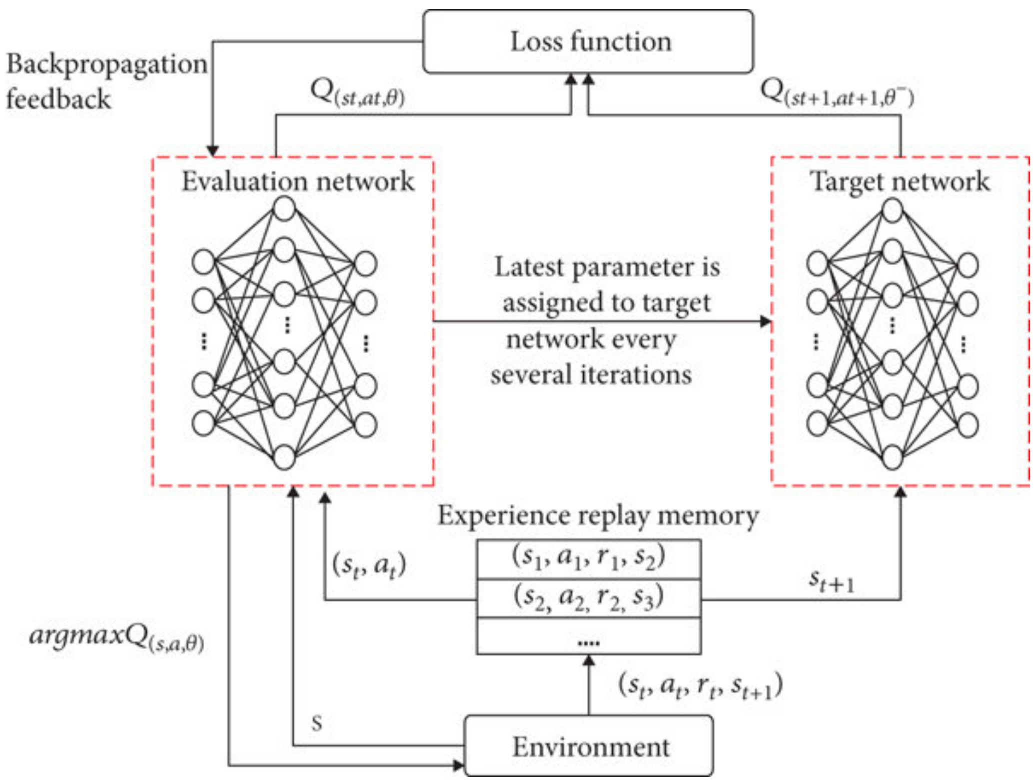


2-2 Deep Q-learning [3]

Deep Q-Network (DQN) is a reinforcement learning algorithm that combines Q-learning with deep neural networks to handle high-dimensional and continuous state spaces where traditional tabular Q-learning fails. The core idea of DQN is to approximate the Q-function — which maps state-action pairs to expected future rewards — using a deep neural network, typically a feedforward multi-layer perceptron. This allows the agent to generalize across similar states, instead of storing a separate Q-value for every possible state-action pair.

To stabilize the inherently unstable combination of function approximation and Q-learning, DQN introduces two critical techniques: Experience Replay and the use of a Target Network. Experience Replay stores past transitions (state, action, reward, next state) in a replay buffer and samples mini-batches randomly during training. This breaks temporal correlations and improves data efficiency. The Target Network is a separate copy of the Q-network whose parameters are updated less frequently. During training, the target Q-value is computed using the Target Network instead of the current (online) network, helping to reduce oscillations and divergence.

DQN was a foundational advancement in deep reinforcement learning, but it’s not a silver bullet. While powerful, it comes with multiple design trade-offs and potential failure points.



3 Background

3-1 Blackjack [4]

3-1-1 Description

The game starts with the dealer having one face up and one face down card, while the player has two face up cards. All cards are drawn from an infinite deck (i.e. with replacement). The card values are:

(1) Face cards (Jack, Queen, King) have a point value of 10.

(2) Aces can either count as 11 (called a 'usable ace') or 1.

(3) Numerical cards (2-10) have a value equal to their number.

The player has the sum of cards held. The player can request additional cards (hit) until they decide to stop (stick) or exceed 21 (bust, immediate loss).

After the player sticks, the dealer reveals their facedown card and draws cards until their sum is 17 or greater. If the dealer goes bust, the player wins. If neither the player nor the dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21.

3-1-2 Action Space

The action shape is (1,) in the range {0, 1} indicating whether to stick or hit.

- 0: Stick
- 1: Hit

3-1-3 Observation Space

The observation consists of a 3-tuple containing: the player's current sum, the value of the dealer's one showing card (1-10 where 1 is ace), and whether the player holds a usable ace (0 or 1). The observation is returned as (int(), int(), int()).

3-1-4 Starting state

The starting state is initialized with the following values.

Observation	**Values**
Player current sum	4, 5, …, 21
Dealer showing card value	1, 2, …, 10
Usable Ace	0, 1

3-1-5 Rewards

- win game: +1
- lose game: -1
- draw game: 0
- win game with natural blackjack: +1.5 (if natural is True) +1 (if natural is False)

3-1-6 Episode End

The episode ends if the following happens:

- Termination:

(1) The player hits and the sum of hand exceeds 21.

(2) The player sticks.

An ace will always be counted as usable (11) unless it busts the player.

3-2 Assault [5]

3-2-1 Description

You control a vehicle that can move sideways. A big mother ship circles overhead and continually deploys smaller drones. You must destroy these enemies and dodge their attacks.

3-2-2 Actions

Assault has the action space of Discrete (7) with the table below listing the meaning of each action's meanings.

Value	**Meaning**	**Value**	**Meaning**	**Value**	**Meaning**
0	NOOP	1	FIRE	2	UP
3	RIGHT	4	LEFT	5	RIGHTFIRE
6	LEFTFIRE				

3-2-3 Observations

Atari environments have three possible observation types:

- obs_type="rgb" -> observation_space=Box(0, 255, (210, 160, 3), np.uint8)

- obs_type="ram" -> observation_space=Box(0, 255, (128,), np.uint8)
- obs_type="grayscale" -> Box(0, 255, (210, 160), np.uint8), a grayscale version of the q”rgb” type

4 Experiments

4-1 Q-learning [6]

4-1-1 Hyperparameters

- Learning rate = 1e-3
- Number of episodes = 100,000
- Start epsilon = 1.0
- Final epsilon = 0.1
- Epsilon decay = start epsilon / (number of episodes / 2)

4-1-2 Discount factor

I experimented with adjusting the discount factor to observe its effect on the agent’s learning behavior. The discount factor, commonly denoted as γ, determines how much future rewards are valued compared to immediate rewards. A higher discount factor places more emphasis on long-term rewards, encouraging the agent to consider the long-term consequences of its actions. Conversely, a lower discount factor makes the agent focus more on immediate gains. By changing the discount factor, I was able to analyze how different values influenced the stability, convergence speed, and overall performance of the agent during training.

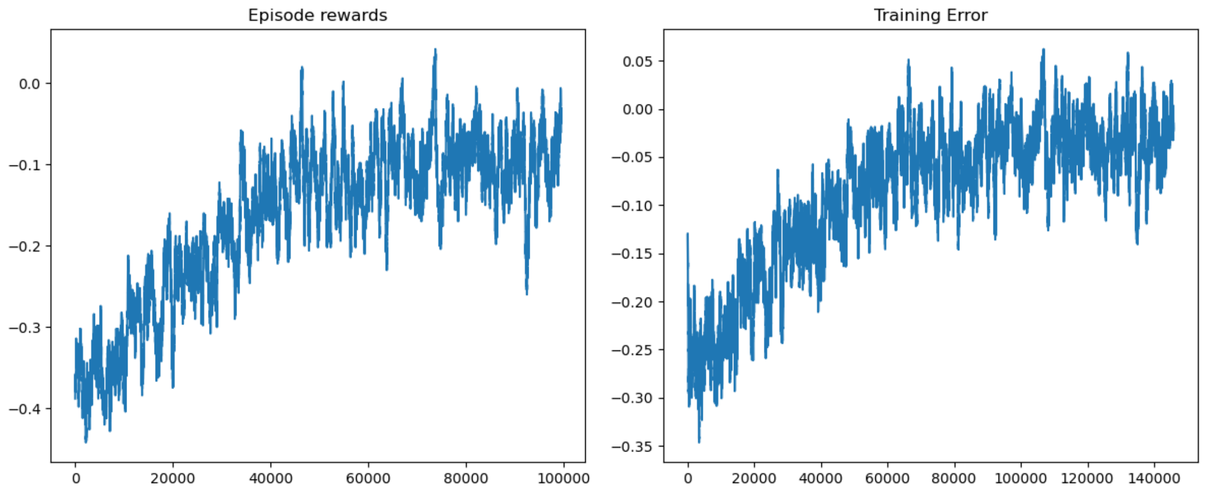


Figure 1: Episode rewards and Training loss under γ = 0.95

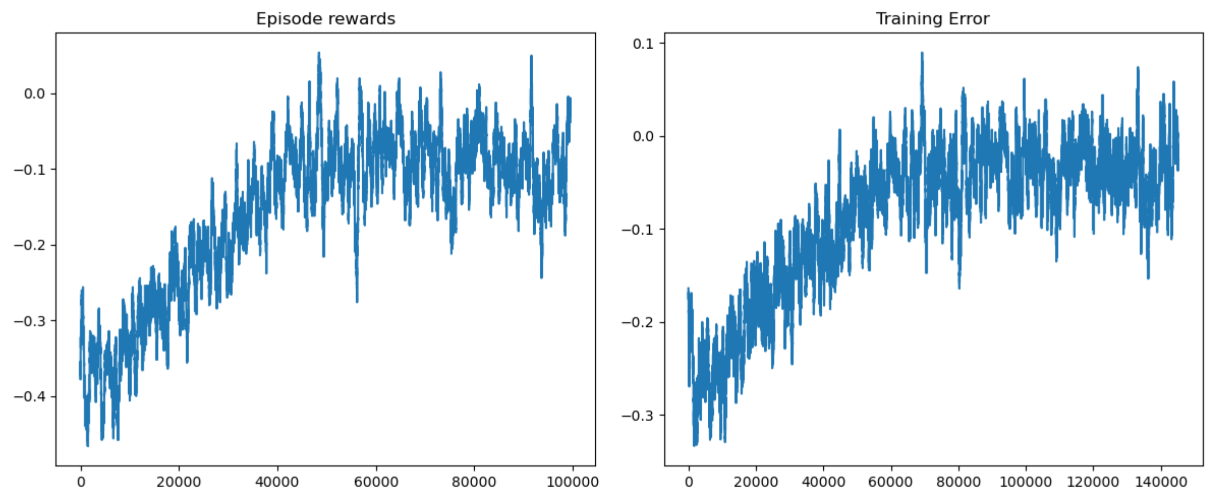


Figure 2: Episode rewards and Training loss under γ = 0.85

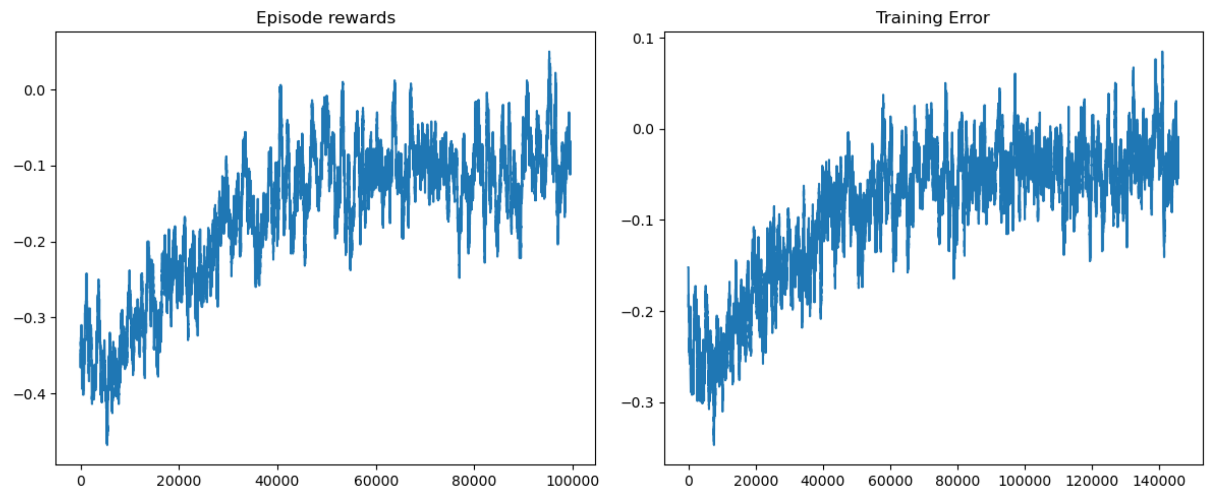


Figure 3: Episode rewards and Training loss under γ = 0.75

After running several experiments (γ=0.95/0.85/0.75), I observed that the episode rewards consistently converged to around -0.1 across different discount factor settings, indicating that the overall policy learned by the agent did not vary significantly in terms of cumulative return. However, there was a notable difference in training error: as the discount factor increased, the training error decreased.. A higher discount factor likely enables the agent to better incorporate long-term feedback, resulting in more consistent learning signals. Nevertheless, despite the reduction in error, the actual performance in terms of episode rewards did not improve significantly.

4-2 Deep Q-learning [7]

4-2-1 Hyperparameter

- Number of episodes = 200
- Steps target net per update = 1,000
- Epsilon decay(linear) = 6e-3

- Epsilon decay(exponential) = 50,000
- Gamma = 0.99
- Learning rate = 1e-3
- Start epsilon = 1.0
- Final epsilon = 0.1
- Memory sample batch = 8

4-2-2 Epsilon decay method

I implemented two epsilon decay methods to adjust the agent's behavior over time: linear decay and exponential decay. Linear decay decreases the epsilon value at a fixed rate until it reaches a predefined minimum, making it simple and predictable for tasks with a fixed number of steps or simpler environments. On the other hand, exponential decay follows the formula:

epsilon = final epsilon + (start epsilon – final epsilon) * exp(-steps / decay rate)

which causes a rapid drop in epsilon at the beginning and gradually approaches the minimum value. This ensures the agent maintains a degree of exploration even in later stages of training, reducing the risk of getting stuck in local optima.

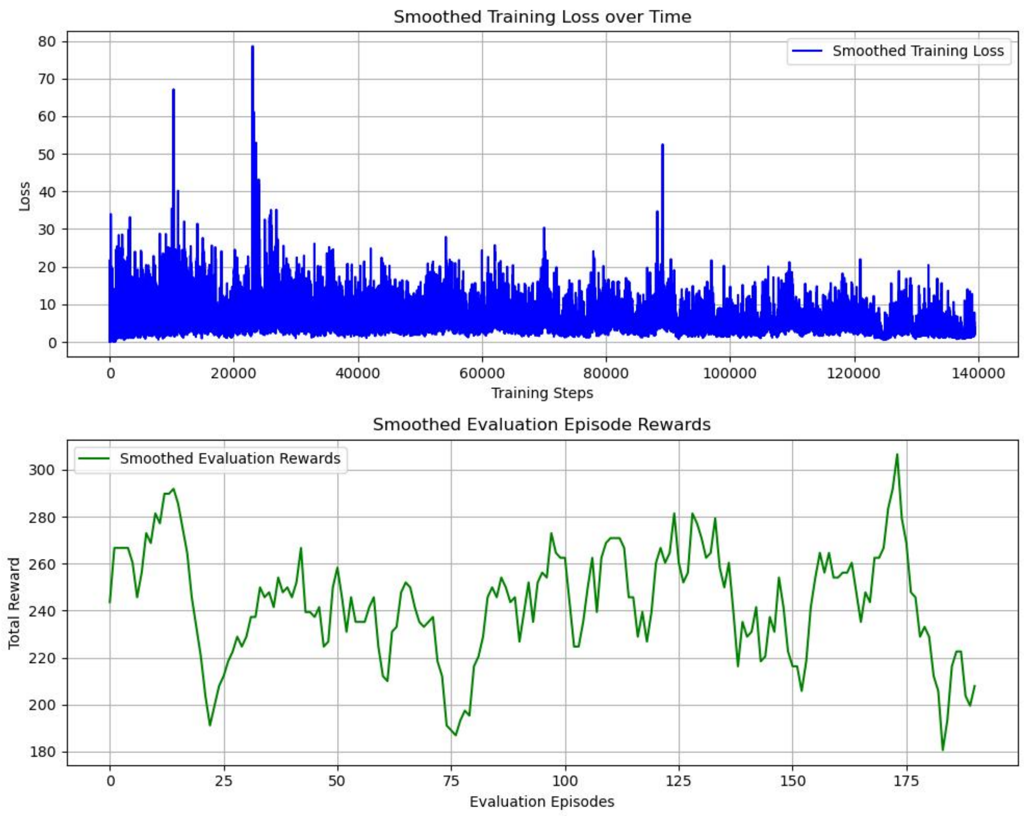


Figure 4: Training loss and Episode rewards using linear epsilon decay

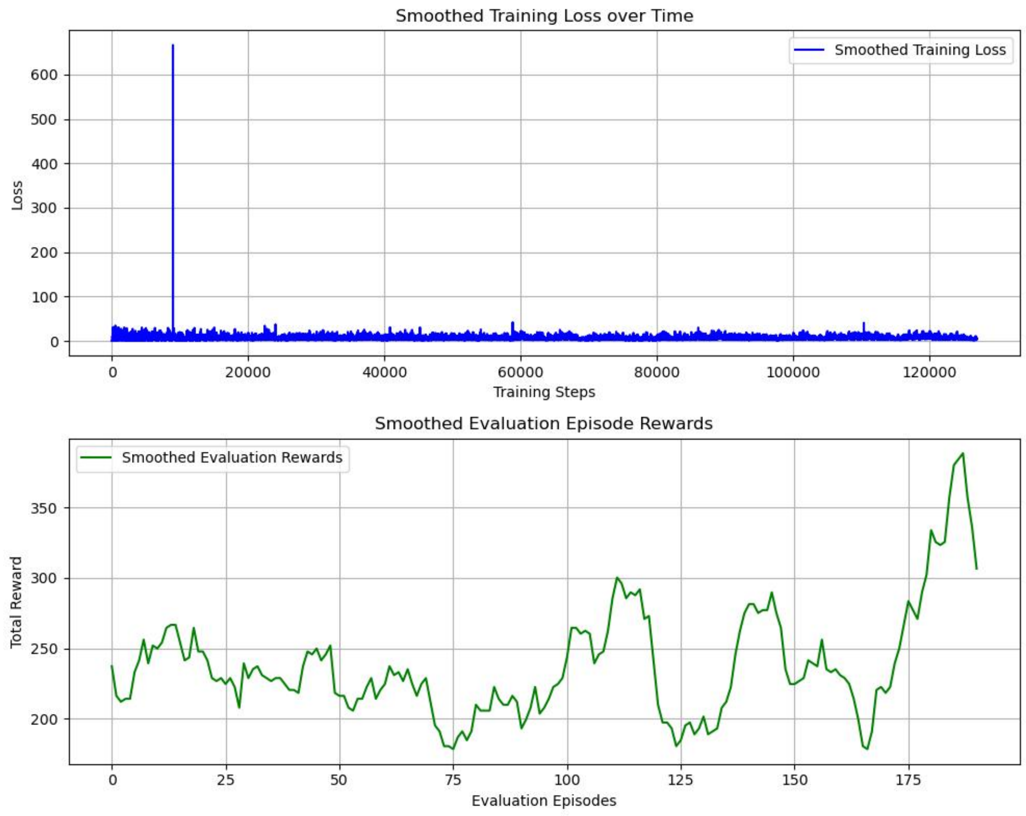


Figure 5: Training loss and Episode rewards using exponential epsilon decay

After conducting a series of experiments, I observed that both the linear decay and exponential decay strategies produced comparable performance in terms of training loss and episode rewards during the early and middle stages of training. This indicates that, in the initial phases, the rate and pattern of epsilon reduction did not significantly impact the agent's ability to learn a baseline policy. However, as training progressed into the later episodes, the exponential decay strategy began to outperform linear decay in terms of cumulative rewards. Specifically, the agent using exponential decay was able to achieve higher and more stable rewards over time. This improvement can be attributed to the nature of exponential decay, which allows the agent to retain a small but consistent level of exploration throughout the training process. Unlike linear decay, which reduces exploration at a constant rate and may reach the minimum epsilon too early, exponential decay maintains a gradually diminishing exploration rate, helping the agent continue discovering better policies and avoid premature convergence to suboptimal actions. These results suggest that exponential decay may offer a more effective balance between exploration and exploitation, particularly in environments where optimal strategies are discovered later in training.

5 Discussion

5-1 Experiment results

Experimental results revealed distinct effects of hyperparameter choices in different reinforcement learning algorithms. In Q-learning, I found that increasing the discount factor led to a noticeable reduction in training loss. This suggests that placing greater emphasis on future rewards results in more stable and consistent Q-value updates.

On the other hand, in the DQN setting, using exponential epsilon decay produced higher episode rewards in the later stages of training compared to linear decay. This is likely because exponential decay maintains a small but persistent degree of exploration throughout training, enabling the agent to continue improving its policy over time and avoid settling on suboptimal behaviors too early.

5-2 Further experiments if available

If I had more time to work on this project, I would extend the training to include more episodes to observe whether the trends I identified—such as the impact of the discount factor in Q-learning and the advantages of exponential epsilon decay in DQN—become more pronounced over a longer training period. For Q-learning, I would further investigate how different discount factors affect the stability of Q-value updates and the learning of optimal strategies as training progresses. In the case of DQN, I would focus on comparing the long-term performance of linear decay versus exponential decay to see whether exponential decay continues to yield higher rewards and more stable policies in later episodes. Extending the training period would not only allow for a clearer view of the learning trends but also help validate the effectiveness of the chosen strategies in more complex environments and with different hyperparameter settings.

5-3 Valuable insight from experiments

In this project, I gained a deeper understanding of how different hyperparameters influence learning behavior in reinforcement learning through both implementation and experimentation. In Q-learning, I observed the significant impact of the discount factor on the agent's decision-making strategy. A higher discount factor encourages the agent to prioritize long-term rewards, promoting farsighted actions, while a lower discount factor leads the agent to focus more on immediate gains. Although the final rewards across different settings were similar, I found that a higher discount factor resulted in lower training error, suggesting more stable Q-value updates during learning.

In the implementation of DQN, I investigated the effects of two commonly used epsilon decay strategies: linear decay and exponential decay. While both strategies produced similar training performance in the early and middle phases, the agent using exponential decay achieved higher cumulative rewards in the later episodes. This is likely due to the nature of exponential decay, which maintains a small but consistent level of exploration even in later stages, allowing the agent to continue refining its policy and avoid premature convergence to suboptimal solutions. Through these experiments, I not only deepened my understanding of Q-learning and DQN algorithms but also experienced firsthand the profound influence of hyperparameter tuning on learning outcomes.

6 Reference

[1]https://www.kdnuggets.com/2022/06/beginner-guide-q-learning.html

[2]https://wikidocs.net/174536

[3]https://www.researchgate.net/figure/Flow-chart-of-the-deep-Q-learning-network_fig2_345893622

[4]https://gymnasium.farama.org/environments/toy_text/blackjack/

[5]https://ale.farama.org/environments/assault/

[6] https://gymnasium.farama.org/tutorials/training_agents/blackjack_tutorial/

[7] https://github.com/MattC04/Atari-RL-project