ECSE 555 Assignment 1

Alex Pi
261214211
McGill University
alex.pi@mail.mcgill.ca

Abstract

In this assignment, we implement and compare both traditional and deep reinforcement learning algorithms using two classic control tasks from the OpenAI Gym: `CartPole-v1` and `MountainCar-v0`. We implement Q-learning, SARSA, and a Deep Q-Network (DQN) to solve these tasks while analyzing the learning curves and the impact of various hyperparameters. In addition, we discuss key conceptual differences in reinforcement learning, including off-policy versus on-policy, model-based versus model-free, value-based versus policy-based, online versus offline learning, and learning versus planning. This report presents our experimental setups, detailed observations, and insights derived from these experiments.

1 Introduction

Reinforcement Learning (RL) has witnessed significant advancements in recent years, with applications ranging from game-playing agents to robotic control. A critical aspect of RL research is the availability of standardized benchmarks and environments. OpenAI Gym addresses this need by providing a diverse set of environments with a uniform API, which greatly facilitates algorithm comparison and reproducibility.

In this assignment, we focus on two classic control tasks: `CartPole-v1` and `MountainCar-v0`. Both tasks are formulated as Markov Decision Processes (MDPs) and serve as ideal platforms to implement and evaluate Q-learning, SARSA, and a deep RL method (DQN). We also explore the effects of various hyperparameters on algorithm performance, and discuss fundamental RL concepts with illustrative examples.

2 OpenAI Gym

OpenAI Gym is a widely adopted toolkit designed to facilitate the development, testing, and comparison of reinforcement learning algorithms. It offers a diverse collection of environments—ranging from simple, deterministic tasks to complex, stochastic simulations—that adhere to a standardized API, enabling researchers to easily benchmark and reproduce experiments. This consistency has made OpenAI Gym an essential resource in both academic and industrial settings, promoting collaborative research and accelerating the advancement of reinforcement learning methods. Moreover, its extensive library of environments allows users to experiment with a variety of challenges, from classic control problems to sophisticated game simulations, thus serving as a vital platform for both learning and innovation in the field.

3 CartPole-v1 Control Task

3.1 Task Overview

The `CartPole-v1` task involves balancing a pole on a moving cart by applying forces to the cart. The goal is to prevent the pole from falling while keeping the cart within a designated range.

3.2 MDP Settings

State Space: The state is represented by a 4-dimensional vector:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Angular Velocity

Index	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	$-\infty$	∞
2	Pole Angle	~ -0.418 rad ($-24°$)	~ 0.418 rad ($24°$)
3	Pole Angular Velocity	$-\infty$	∞

Table 1: State space of the `CartPole-v1` task.

Action Space: The action is discrete:

Action	Description
0	Push cart to the left
1	Push cart to the right

Table 2: Action space for the `CartPole-v1` task.

Episode Termination:

- **Termination:** The episode terminates if the pole angle exceeds $\pm 12°$ or if the cart position goes beyond ± 2.4 (i.e., the cart reaches the edge of the display).
- **Truncation:** The episode is truncated if the episode length exceeds 500 timesteps (200 for `v0`).

Rewards: A reward of +1 is awarded at every timestep to encourage keeping the pole upright. In `CartPole-v1`, the reward threshold is set at 475.

4 MountainCar-v0 Task

4.1 Task Overview

The `MountainCar-v0` task challenges the agent to drive a car up a steep hill. Since the car's engine is too weak to ascend the hill directly, the agent must learn to build momentum by oscillating between the hills.

4.2 MDP Settings

State Space: The state consists of two dimensions:

- Position (range: $[-1.2, 0.6]$)
- Velocity (range: $[-0.07, 0.07]$)

Index	Observation	Min	Max
0	Position	-1.2	0.6
1	Velocity	-0.07	0.07

Table 3: State space of the `MountainCar-v0` task.

Action Space: The action space is discrete:

Action	Description
0	Accelerate to the left
1	No acceleration
2	Accelerate to the right

Table 4: Action space for the `MountainCar-v0` task.

Episode Termination:

- **Termination:** The episode terminates when the car's position reaches or exceeds 0.5 (i.e., the goal is reached).
- **Truncation:** The episode is truncated after 200 timesteps.

Rewards: A penalty of -1 is incurred at every timestep to encourage reaching the goal as quickly as possible.

Transition Dynamics: The dynamics are given by:

$$v_{t+1} = v_t + (\text{action} - 1) \cdot \text{force} - \cos(3 \cdot p_t) \cdot \text{gravity}, \tag{1}$$

$$p_{t+1} = p_t + v_{t+1}, \tag{2}$$

where force $= 0.001$ and gravity $= 0.0025$. The position is clipped to $[-1.2, 0.6]$ and the velocity to $[-0.07, 0.07]$. Collisions with the boundary are inelastic, with the velocity set to zero upon collision.

5 Q-Learning for CartPole-v1

5.1 Algorithm Overview

Q-learning is an off-policy algorithm that updates the Q-value for a state-action pair using the Bellman equation:

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

where α is the learning rate, γ is the discount factor, r is the received reward, and s' is the next state.

5.2 Experimental Setup

Environment and Discretization: The continuous state space is discretized via binning:

- Cart Position: $[-2.4, 2.4]$
- Cart Velocity: $[-3.0, 3.0]$
- Pole Angle: $[-0.418, 0.418]$
- Pole Angular Velocity: $[-3.5, 3.5]$

A bin number of 6 is used for each state dimension.

Hyperparameter	Value
Learning Rate (α)	0.1
Discount Factor (γ)	0.99
Initial Exploration (ϵ)	1.0 (decays over time)
Minimum ϵ	0.01
ϵ Decay Rate	0.995
Number of Episodes	2000
Number of Test Episodes	200

Table 5: Hyperparameters for Q-learning on `CartPole-v1`.

5.3 Learning Curve Analysis

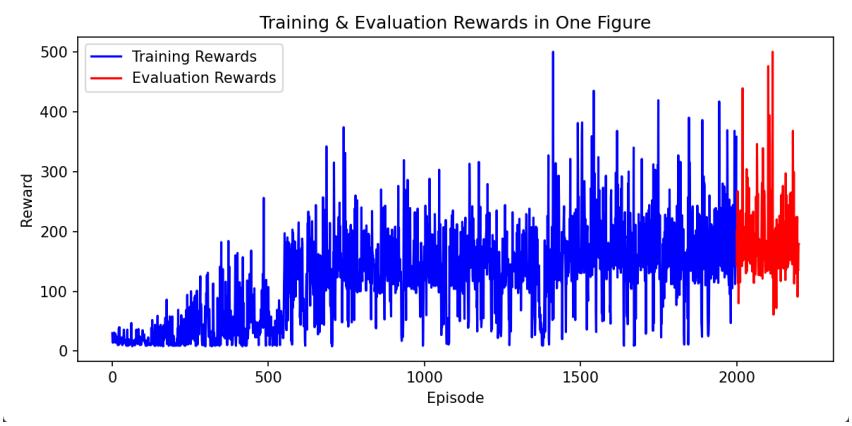

Figure 1: Learning curve of Q-learning on `CartPole-v1`. The x-axis represents episodes, and the y-axis shows the average reward per episode.

As shown in Figure 1, initially, the agent exhibits high variance due to random exploration. As ϵ decays, the learned Q-values guide the agent toward higher rewards, with the average reward stabilizing around 200 after approximately 600 episodes.

5.4 Conclusion

The Q-learning algorithm successfully learns a near-optimal policy for the `CartPole-v1` task. While early episodes are characterized by erratic performance due to exploration, a steady improvement is observed as the Q-table converges.

6 SARSA for CartPole-v1

6.1 Algorithm Overview

SARSA (State-Action-Reward-State-Action) is an on-policy method that updates Q-values based on the action actually taken:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]. \tag{4}$$

6.2 Experimental Setup

The same discretization and hyperparameter settings used for Q-learning are applied here (see Table 6).

Hyperparameter	Value
Learning Rate (α)	0.1
Discount Factor (γ)	0.99
Initial Exploration (ϵ)	1.0 (decays over time)
Minimum ϵ	0.01
ϵ Decay Rate	0.995
Number of Episodes	2000
Number of Test Episodes	200

Table 6: Hyperparameters for SARSA on `CartPole-v1`.

6.3 Learning Curve Analysis

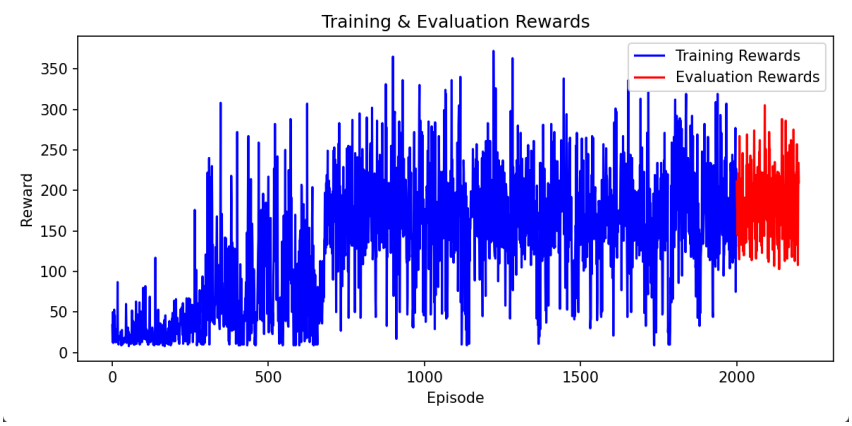

Figure 2: Learning curve of SARSA on `CartPole-v1`.

As shown in Figure 2, the SARSA algorithm shows a gradual increase in the average reward, eventually stabilizing around 200. The on-policy nature of SARSA results in slightly slower convergence compared to Q-learning.

6.4 Conclusion

SARSA effectively learns a stable policy for balancing the pole, though its convergence is more gradual compared to the off-policy Q-learning method.

7 Deep Q-Learning for CartPole-v1

7.1 Algorithm Overview

Deep Q-Networks (DQN) extend Q-learning by approximating the Q-value function with a neural network. In addition to the standard Q-learning update, DQN utilizes experience replay and a target network to stabilize training.

7.2 Experimental Setup

Network Architecture:

- **Input Layer:** 4 neurons (state dimensions).
- **Hidden Layer 1:** 128 neurons with ReLU activation.
- **Hidden Layer 2:** 128 neurons with ReLU activation.
- **Output Layer:** 2 neurons (Q-values for the two actions).

Hyperparameter	Value
Learning Rate (α)	10^{-3}
Discount Factor (γ)	0.99
Batch Size	64
Replay Memory Size	10,000
Target Network Update Frequency	10 episodes
Initial Exploration (ϵ)	1.0 (decays over time)
Minimum ϵ	0.01
Number of Episodes	2000
Number of Test Episodes	200

Table 7: Hyperparameters for DQN on `CartPole-v1`.

7.3 Learning Curve Analysis

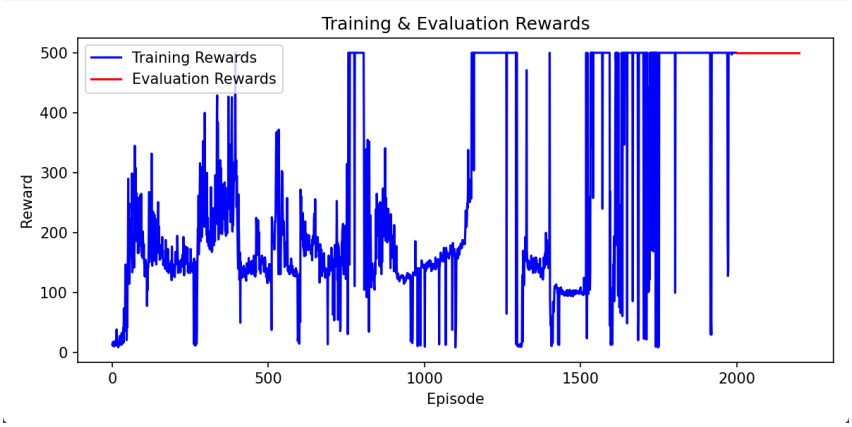

Figure 3: Learning curve of DQN on `CartPole-v1`.

As shown in Figure 3, DQN shows rapid initial improvement and a smoother learning progression compared to tabular methods. The target network and replay memory help stabilize the updates, enabling the agent to consistently achieve high rewards.

7.4 Conclusion

The DQN approach demonstrates the effectiveness of deep function approximation for handling continuous state spaces. It converges quickly to a near-optimal policy, highlighting the benefits of incorporating experience replay and target network updates.

8 Q-Learning for MountainCar-v0

8.1 Algorithm Overview

For the `MountainCar-v0` task, Q-learning is implemented using discretized state spaces. The agent learns to reduce the number of timesteps required to reach the goal by building momentum.

8.2 Experimental Setup

Environment and Discretization:

- Position: $[-1.2, 0.6]$
- Velocity: $[-0.07, 0.07]$

We use 20 bins per state dimension.

Hyperparameter	Value
Learning Rate (α)	0.1
Discount Factor (γ)	0.99
Initial Exploration (ϵ)	1.0 (decays over time)
Minimum ϵ	0.01
ϵ Decay Rate	0.995
Number of Episodes	6000
Number of Test Episodes	200

Table 8: Hyperparameters for Q-learning on `MountainCar-v0`.

8.3 Learning Curve Analysis

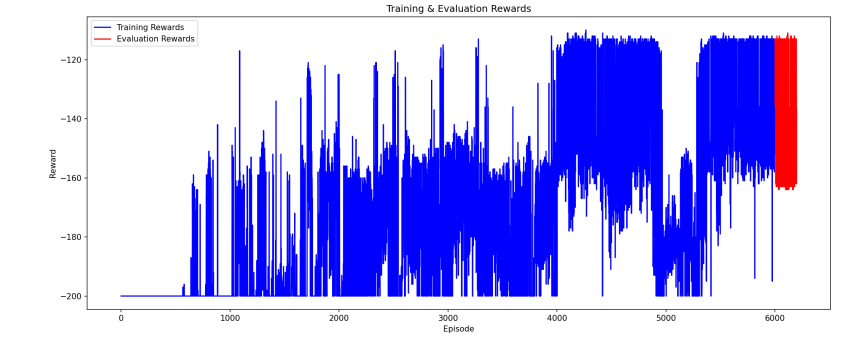

Figure 4: Learning curve of Q-learning on `MountainCar-v0` over 6000 episodes.

As shown in Figure 4 and 5, the learning curve reveals three phases:

- **Exploration (0–1000 episodes):** Rewards remain near -200.
- **Improvement (1000–4000 episodes):** A steady increase in average reward towards -170.
- **Stabilization (4000–6000 episodes):** The reward plateaus around -140.

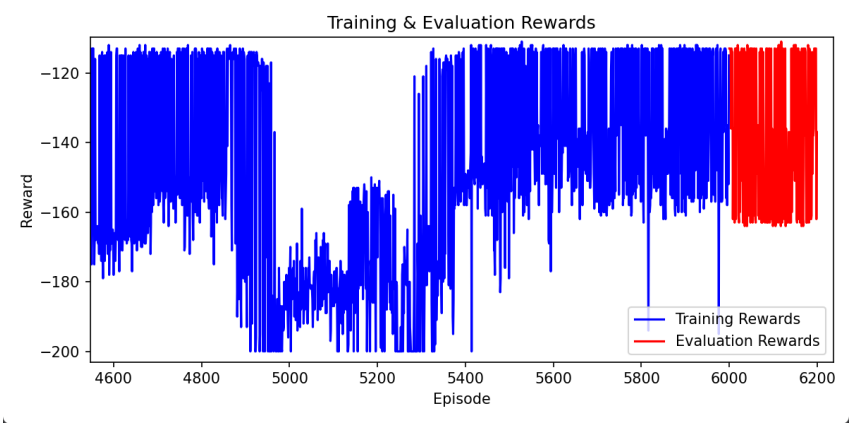

Figure 5: Learning curve of Q-learning on `MountainCar-v0` for the final 1000 episodes.

8.4 Conclusion

The Q-learning agent gradually learns to build momentum, as indicated by the improvement in average reward from -200 to -140. Finer state discretization or additional hyperparameter tuning could potentially yield even better performance.

9 SARSA for MountainCar-v0

9.1 Algorithm Overview

SARSA is applied to `MountainCar-v0` using the same discretization and similar hyperparameter settings as Q-learning.

9.2 Experimental Setup

Discretization uses 20 bins per state dimension (position and velocity), with hyperparameters as shown in Table 9.

Hyperparameter	Value
Learning Rate (α)	0.1
Discount Factor (γ)	0.99
Initial Exploration (ϵ)	1.0 (decays over time)
Minimum ϵ	0.01
ϵ Decay Rate	0.995
Number of Episodes	6000
Number of Test Episodes	200

Table 9: Hyperparameters for SARSA on `MountainCar-v0`.

9.3 Learning Curve Analysis

As shown in Figure 6, the SARSA agent shows similar trends to Q-learning, with rewards improving from -200 to approximately -130 over 6000 episodes, albeit with slightly higher variance.

9.4 Conclusion

SARSA is capable of learning an effective policy for the MountainCar-v0 task. However, its on-policy updates result in marginally slower convergence compared to Q-learning.

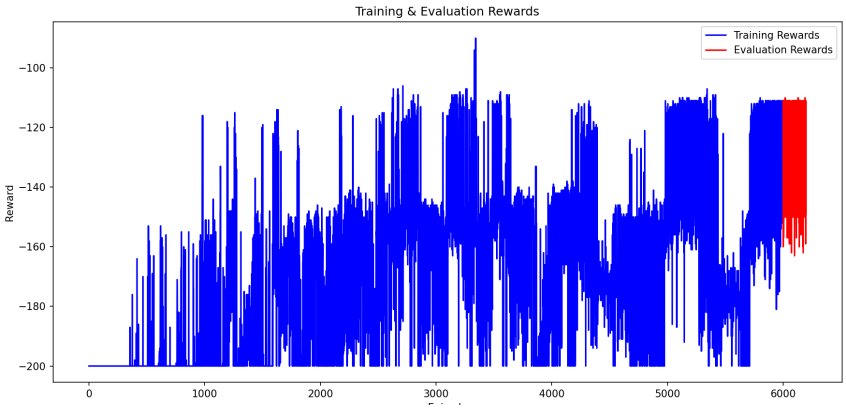

Figure 6: Learning curve of SARSA on `MountainCar-v0`.

10 Deep Q-Learning for MountainCar-v0

10.1 Algorithm Overview

Deep Q-Networks (DQN) approximate the Q-function using a neural network, making them well-suited for continuous state spaces. For MountainCar-v0, DQN is implemented with experience replay and a target network.

10.2 Experimental Setup

Network Architecture:

- **Input Layer:** 2 neurons (position and velocity).
- **Hidden Layer 1:** 64 neurons with ReLU activation.
- **Hidden Layer 2:** 64 neurons with ReLU activation.
- **Output Layer:** 3 neurons (Q-values for the three actions).

Hyperparameter	Value
Learning Rate (α)	10^{-3}
Discount Factor (γ)	0.99
Batch Size	64
Replay Memory Size	100,000
Minimum Replay Buffer Size	1000
Initial Exploration (ϵ)	1.0 (decays over time)
Number of Episodes	2000
Number of Test Episodes	200

Table 10: Hyperparameters for DQN on `MountainCar-v0`.

10.3 Learning Curve Analysis

As shown in Figure 7, the DQN agent initially achieves rewards near -200 and then shows steady improvement, reaching average rewards around -110 by the end of training. The smoother progression demonstrates the advantages of function approximation in complex environments.

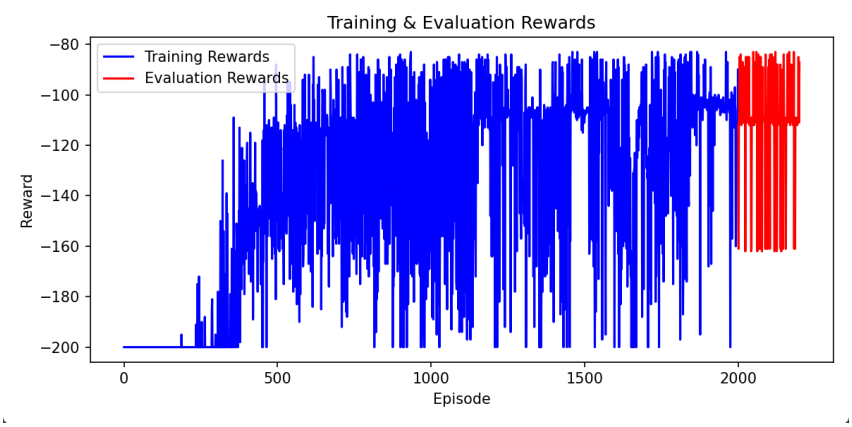

Figure 7: Learning curve of DQN on `MountainCar-v0`.

10.4 Conclusion

While DQN does not fully converge within 2000 episodes for MountainCar-v0, it shows promising improvement and a smoother learning curve compared to tabular methods. Future work may incorporate techniques such as prioritized experience replay to further enhance performance.

11 Hyperparameter Change Experiment Analysis

To assess the sensitivity of each algorithm to hyperparameter choices, we conducted experiments on `CartPole-v1` by varying learning rate, exploration decay, batch size (for DQN), replay memory size, and target network update frequency.

11.1 Q-Learning on CartPole-v1

Learning Rate:

- **Low ($\alpha = 0.05$):** Results in smooth but slow convergence (see Figure 8).
- **Default ($\alpha = 0.1$):** Provides a balanced convergence speed and stability (see Figure 1).
- **High ($\alpha = 0.2$):** Yields rapid initial improvement but unstable oscillations (see Figure 9).

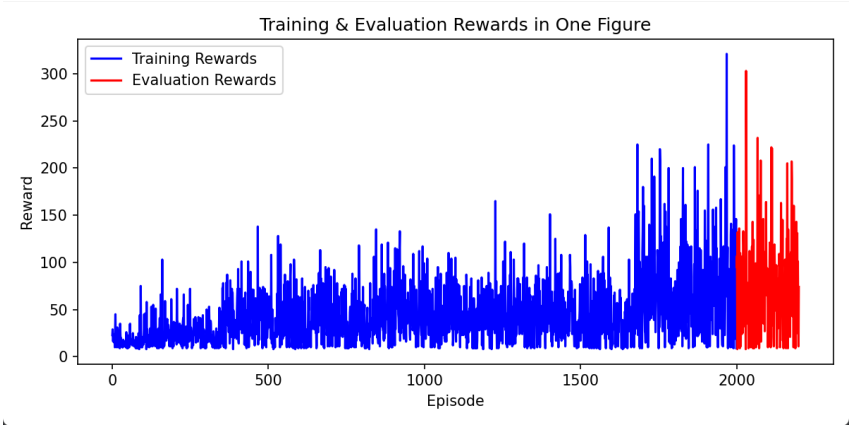

Figure 8: Learning curve of Q-learning on `CartPole-v1` with $\alpha = 0.05$.

Exploration Decay:

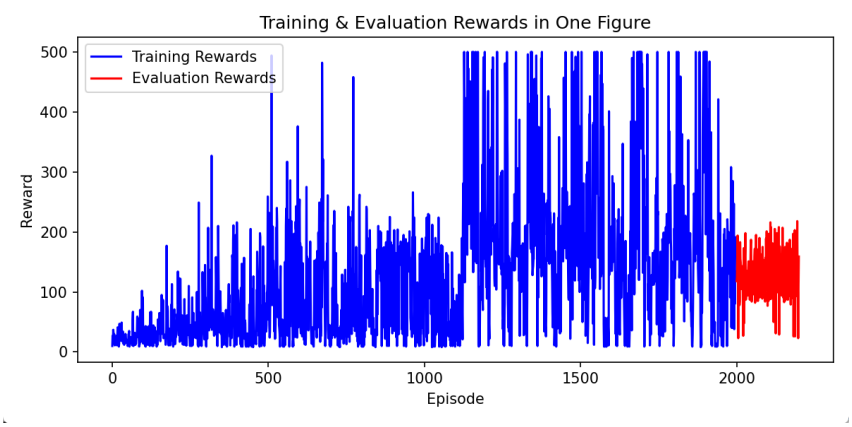

Figure 9: Learning curve of Q-learning on `CartPole-v1` with $\alpha = 0.2$.

- **Faster Decay (ϵ decay=0.99):** Leads to premature exploitation (see Figure 10).
- **Default (ϵ decay=0.995):** Provides a balanced exploration-exploitation trade-off((see Figure 1)).
- **Slower Decay (ϵ decay=0.999):** Maintains exploration longer and stabilizes performance (see Figure 11).

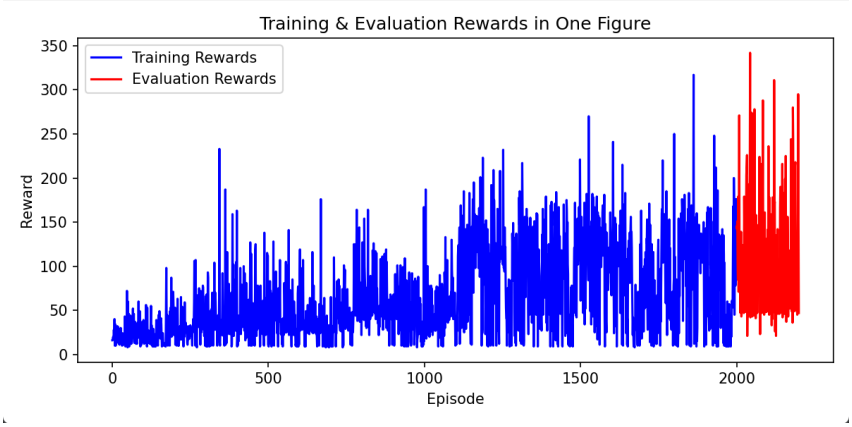

Figure 10: Learning curve of Q-learning on `CartPole-v1` with ϵ decay=0.99.

11.2 SARSA on CartPole-v1

Similar hyperparameter variations for SARSA reveal that:

- Lower learning rates lead to smoother yet slower convergence.
- Higher learning rates induce faster initial improvement but more oscillatory behavior.
- The exploration decay parameter similarly affects the balance between exploration and exploitation.

11.3 Deep Q-Network (DQN) on CartPole-v1

For DQN, we additionally investigated:

Batch Size:

- **Default (64):** Provides stable gradient estimates (see Figure 3).

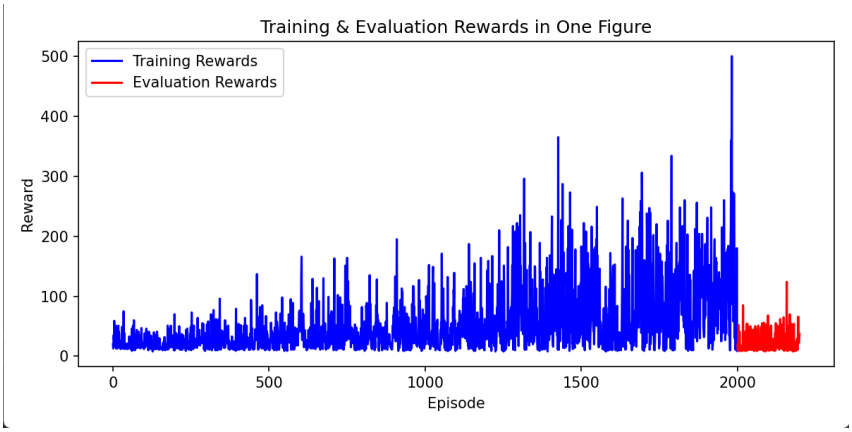

Figure 11: Learning curve of Q-learning on `CartPole-v1` with ϵ decay=0.999.

- **Increased (128):** (see Figure 12) yields smoother learning curves at a slight cost in per-episode convergence speed.

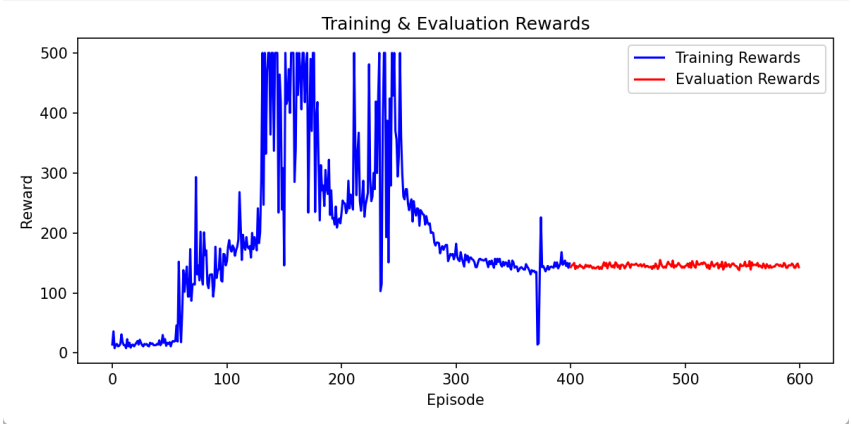

Figure 12: Learning curve of DQN on `CartPole-v1` with batch size=128.

Replay Memory Size:

- Smaller memory (5000) increases variance (see Figure 13).
- Default memory (10000) provides a good balance (see Figure 3).

Target Network Update Frequency:

- Updating every 5 episodes yields rapid but unstable learning (see Figure 14).
- Updating every 10 episodes (default) strikes a good balance (see Figure 3).
- Updating every 20 episodes slows down learning (see Figure 15).

11.4 Experiments on MountainCar-v0

For the more challenging `MountainCar-v0` task, extended training (6000 episodes) and finer state discretization (20 bins) are crucial. Trends observed in CartPole-v1 largely hold here, though slower convergence necessitates prolonged exploration.

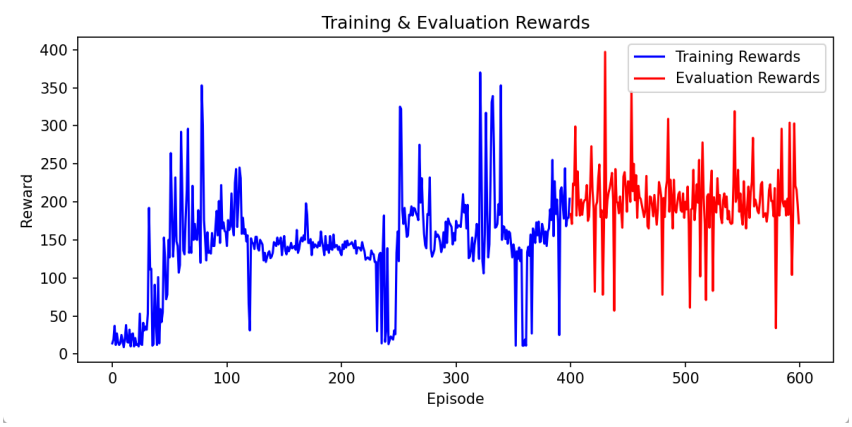

Figure 13: Learning curve of DQN on `CartPole-v1` with replay memory size=5000.

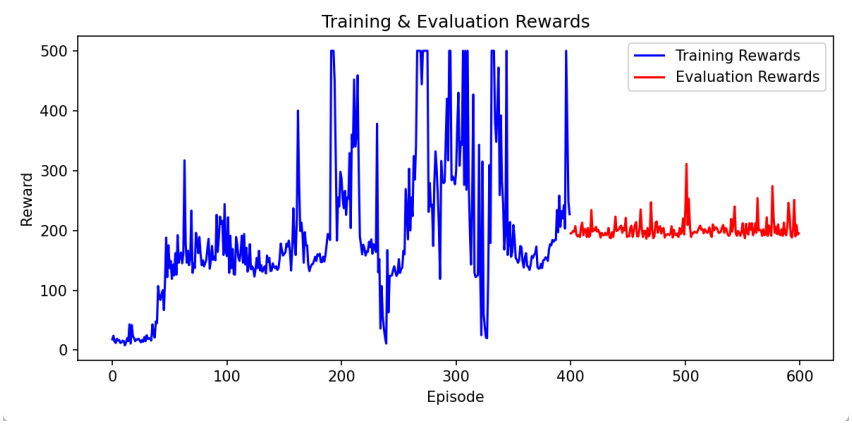

Figure 14: Learning curve of DQN on `CartPole-v1` with target network update frequency=5.

11.5 Summary of Findings

- A moderate learning rate (around 0.1) achieves a balance between speed and stability.
- The exploration decay rate critically impacts performance: too fast may trap the agent in a suboptimal policy, too slow may delay convergence.
- For DQN, larger batch sizes and replay memory sizes stabilize gradient updates, while the target network update frequency must be chosen carefully.

12 Discussion of Results

Our experiments across `CartPole-v1` and `MountainCar-v0` reveal:

- **Tabular Methods:** Q-learning (off-policy) converges faster than SARSA (on-policy), though both eventually learn effective policies.
- **Deep Q-Networks:** DQN demonstrates smoother learning curves due to its function approximation capabilities, but it is sensitive to hyperparameter tuning.
- **Hyperparameter Sensitivity:** A careful balance of learning rate, exploration decay, and (for DQN) batch size and replay memory is essential for optimal performance.

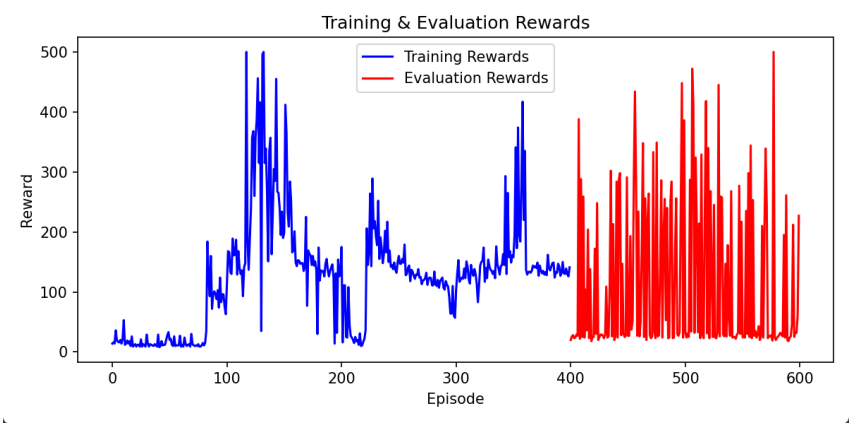

Figure 15: Learning curve of DQN on `CartPole-v1` with target network update frequency=20.

13 Conceptual Differences in Reinforcement Learning: Examples

Understanding fundamental distinctions is key to selecting the right algorithm for a given task. Here we illustrate some of these differences:

13.1 Off-policy vs. On-policy

- **Off-policy:** Q-learning updates using the maximum future Q-value, independent of the action taken (e.g., learning the optimal policy regardless of exploration).
- **On-policy:** SARSA updates based on the action actually taken, which can lead to more conservative policies.

13.2 Model-based vs. Model-free

- **Model-free:** Methods like Q-learning and SARSA learn directly from experience without modeling the environment.
- **Model-based:** Algorithms such as Dyna-Q incorporate a learned model to simulate future experiences, integrating planning with learning.

13.3 Value-based vs. Policy-based

- **Value-based:** Approaches such as Q-learning and DQN estimate state-action values and derive policies by selecting actions with the highest value.
- **Policy-based:** Methods like REINFORCE optimize the policy directly, which is advantageous in continuous action spaces.

13.4 Online vs. Offline Learning

- **Online Learning:** The agent continuously updates its policy based on new interactions with the environment.
- **Offline Learning:** The agent learns from a fixed dataset, which is useful when further interaction is costly or risky.

13.5 Learning vs. Planning

- **Learning:** Involves updating policies or value functions from direct experience (e.g., Q-learning updates).
- **Planning:** Uses a model of the environment to simulate future states and improve decision-making (e.g., in Dyna-Q).

14 Conclusion

This assignment provided hands-on experience with both traditional and deep reinforcement learning algorithms using two classic control tasks. By analyzing learning curves under various hyperparameter settings and discussing key RL concepts, we have developed a deeper understanding of the trade-offs involved in algorithm design and tuning. Our findings lay the groundwork for future explorations into more advanced deep RL techniques.

15 Running Environment

numpy: 1.24.3 matplotlib: 3.7.2 gym: 0.26.2 torch: 2.4.1

References

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. `https://arxiv.org/abs/1606.01540`.

[2] OpenAI Gym. *CartPole-v1*. Retrieved from `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`.

[3] OpenAI Gym. *MountainCar-v0*. Retrieved from `https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py`.