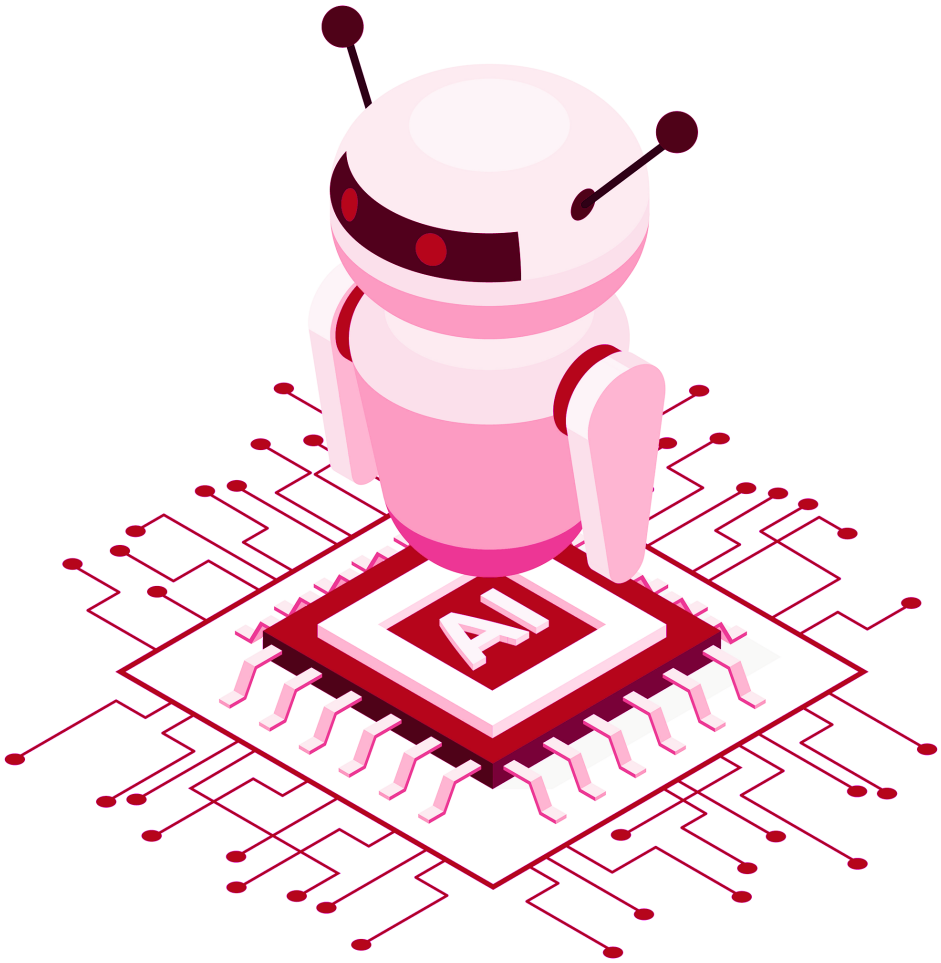


Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 1:

Introduction to RL

By:

Amirmahdi Meighani

400105274

RIML

Deep RL Course

Spring 2025

Contents

1 Task 1: Solving Predefined Environments [45-points] **1**

2 Task 2: Creating Custom Environments [45-points] **6**

2.1 Introduction 6
2.2 Environment Setup 6
2.3 Q-Learning Experiments 6
2.4 Deep Reinforcement Learning Algorithms and Hyperparameters 8
2.5 Results and Analysis 8
2.6 Conclusion 9

3 Task 3: Pygame for RL environment [20-points] **10**

3.1 Introduction 10
3.2 Environment 1: Simple Grid-Based Environment 10
3.2.1 Environment Description 10
3.2.2 Parameters 10
3.2.3 Step Function 10
3.2.4 Reward Function 10
3.2.5 Results 10
3.3 Environment 2: Complex Dynamic Environment 11
3.3.1 Environment Description 11
3.3.2 Parameters 11
3.3.3 Step Function 11
3.3.4 Reward Function 11
3.3.5 Results 11
3.4 Conclusion 12

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: Solving Predefined Environments	45
Task 2: Creating Custom Environments	45
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing a wrapper for a known env	10
Bonus 2: Implementing pygame env	20
Bonus 3: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Task 1: Solving Predefined Environments [45-points]

Overview

I used the Taxi and CartPole environments from the Gym library to implement and test different reinforcement learning (RL) algorithms. The algorithms were evaluated with various hyperparameters to understand their performance and convergence behavior.

CartPole Environment

Algorithms Tested:

- A2C (Advantage Actor-Critic)
- PPO (Proximal Policy Optimization)
- A2C with low gamma (a2c_gamma_low)
- PPO with low gamma (ppo_gamma_low)
- A2C with high learning rate (a2c_lr_high)
- PPO with high learning rate (ppo_lr_high)

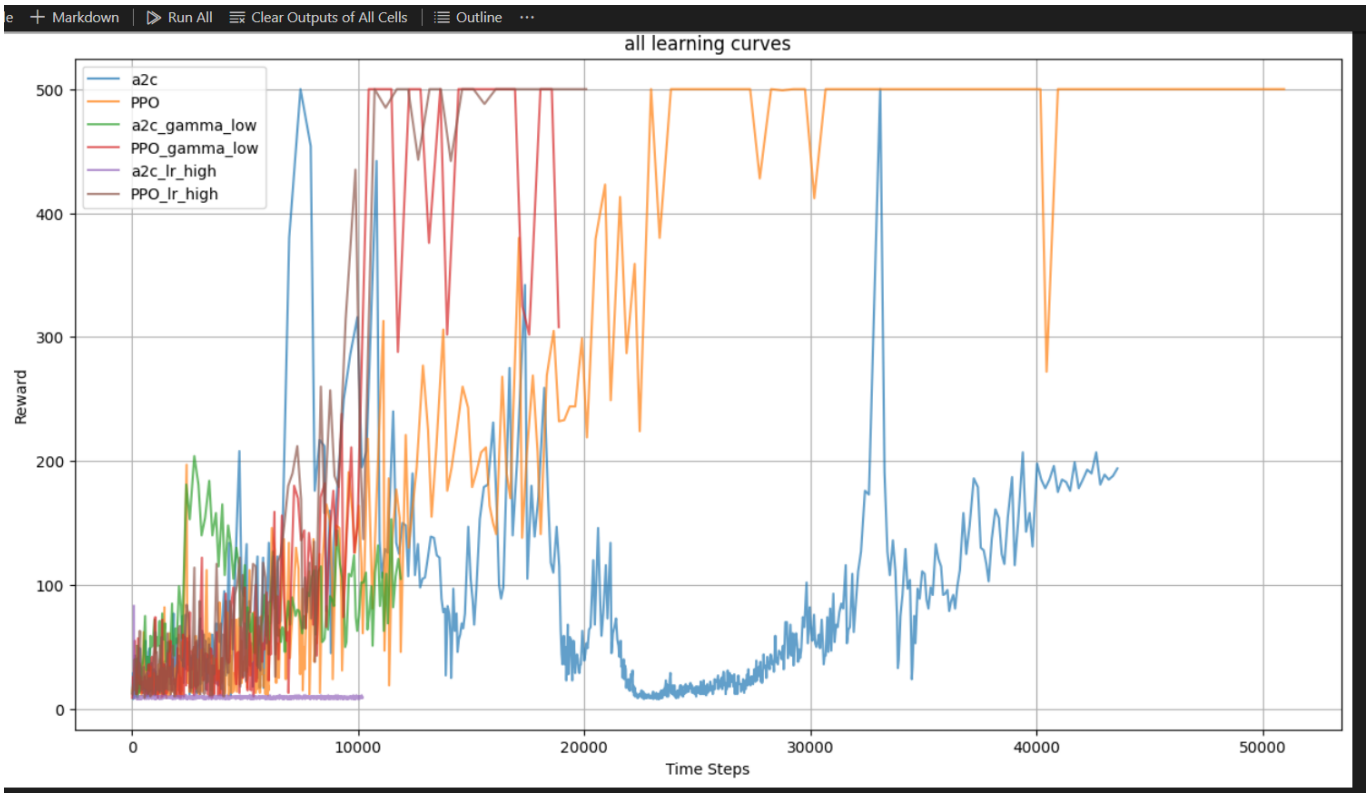
Results and Analysis:

- PPO consistently outperformed other algorithms, showing faster and more stable convergence.
- PPO with high learning rate achieved the fastest convergence.
- A2C struggled to converge and often failed.

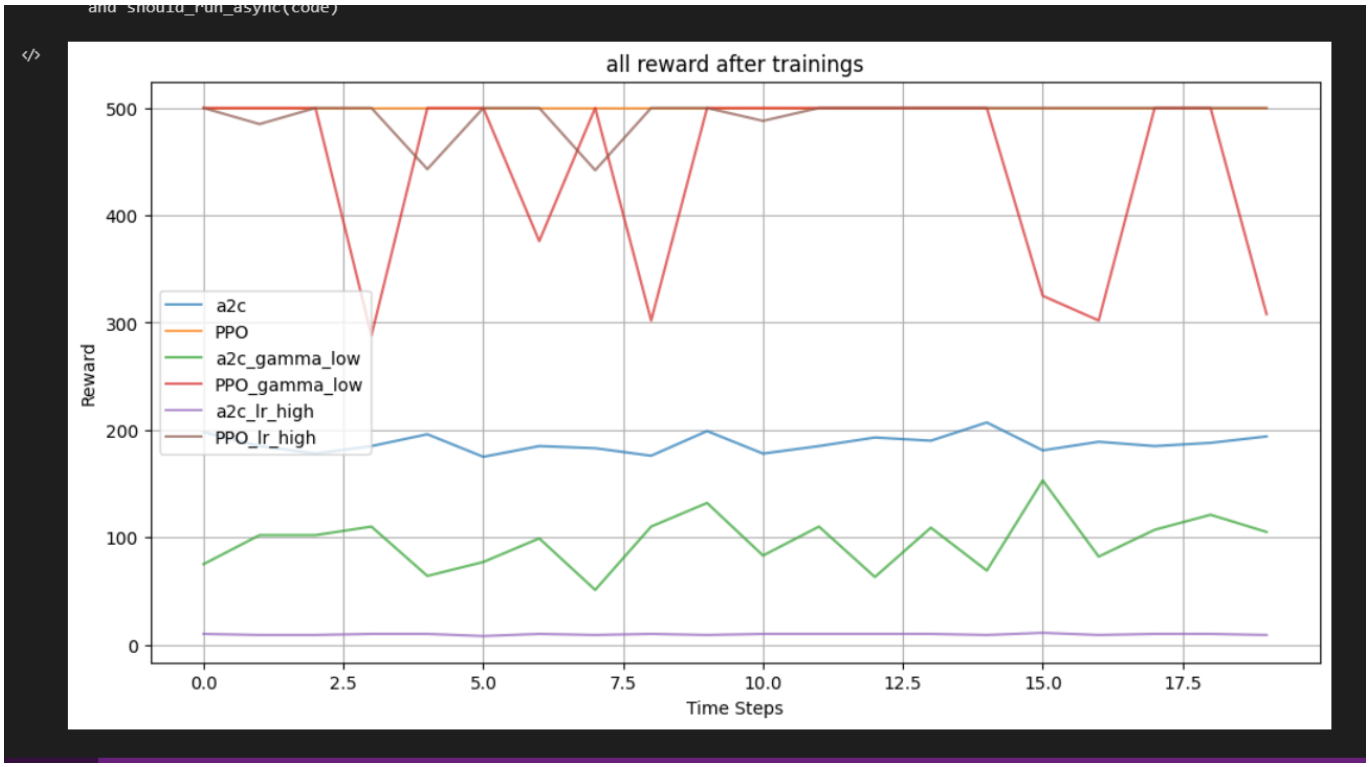
Reasons for Results:

- PPO's superior performance: PPO employs a clipped objective function, which helps maintain stable updates and prevents the policy from changing too drastically, making it more reliable for continuous control tasks like CartPole.
- High learning rate in PPO: PPO can effectively handle higher learning rates due to its policy clipping mechanism, enabling faster convergence.
- A2C's poor performance: A2C lacks the policy clipping mechanism, making it more sensitive to high variance in policy updates, which can hinder convergence.

You can see the learning curve of each algorithm here:



You can see the final result of each algorithm here:



Taxi Environment

Algorithms Tested:

- A2C (Advantage Actor-Critic)
- PPO (Proximal Policy Optimization)

- A2C with low gamma (a2c_gamma_low)
- PPO with low gamma (ppo_gamma_low)
- A2C with high learning rate (a2c_lr_high)
- PPO with high learning rate (ppo_lr_high)

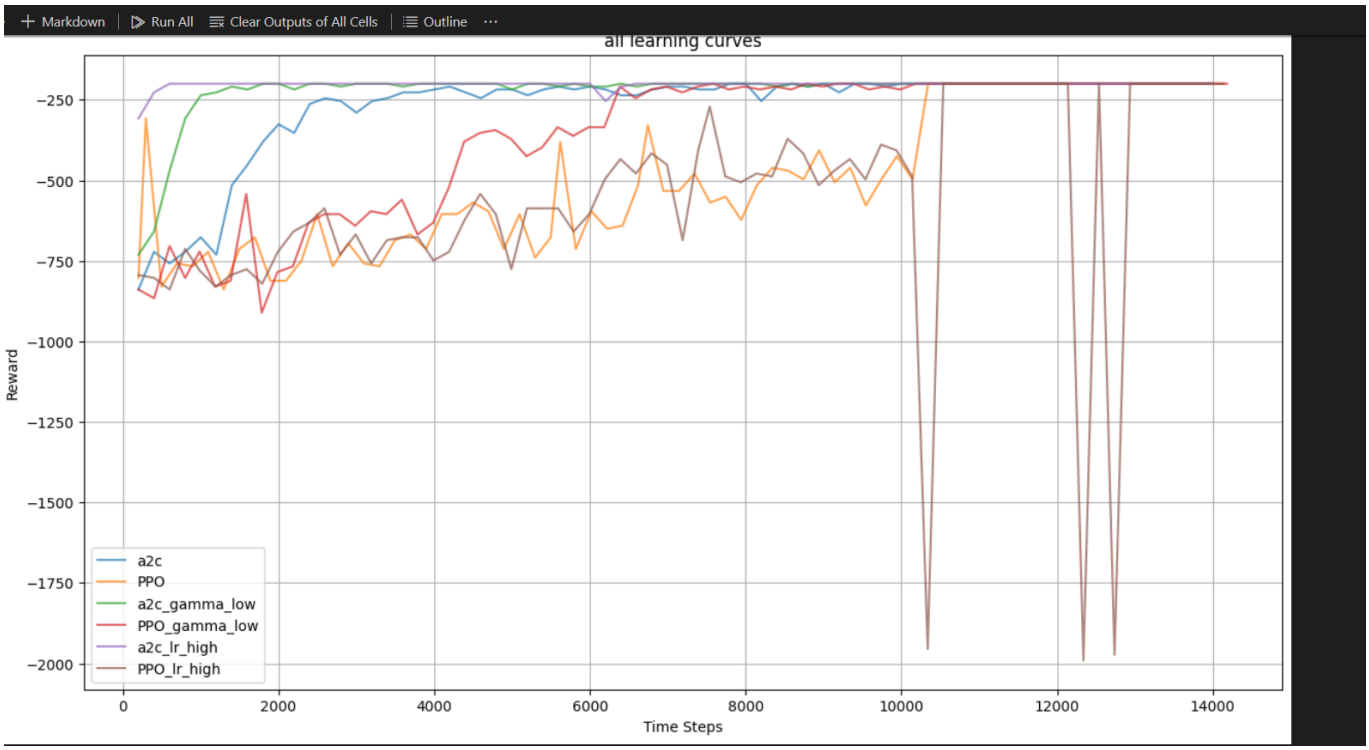
Results and Analysis:

- Almost all algorithms converged, indicating that the Taxi environment is relatively simpler than CartPole.
- A2C with high learning rate converged the fastest.
- PPO with different hyperparameters converged more slowly.
- PPO with high learning rate exhibited instability.

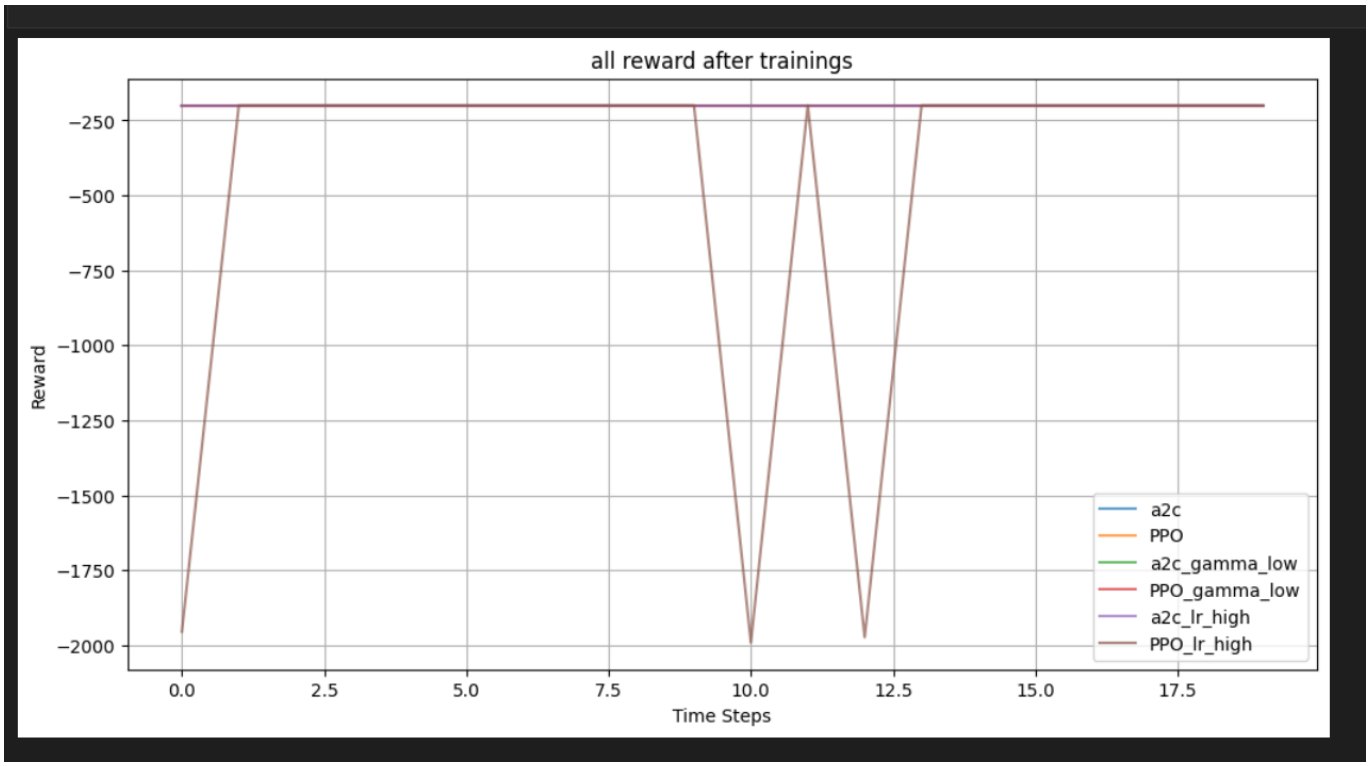
Reasons for Results:

- Simpler environment: Taxi is a discrete-state, turn-based problem, which makes convergence easier for most algorithms.
- A2C's advantage with high learning rate: In the Taxi environment, the discrete state space and deterministic transitions allow A2C to learn optimal policies quickly without the risk of divergence.
- PPO instability with high learning rate: Despite its clipping mechanism, PPO's performance in simpler environments can degrade with a high learning rate due to overfitting to early experiences.

You can see the learning curve of each algorithm here:



You can see the final result of each algorithm here:



Reward Wrapper for Taxi

I implemented a reward wrapper to scale rewards for the Taxi environment.

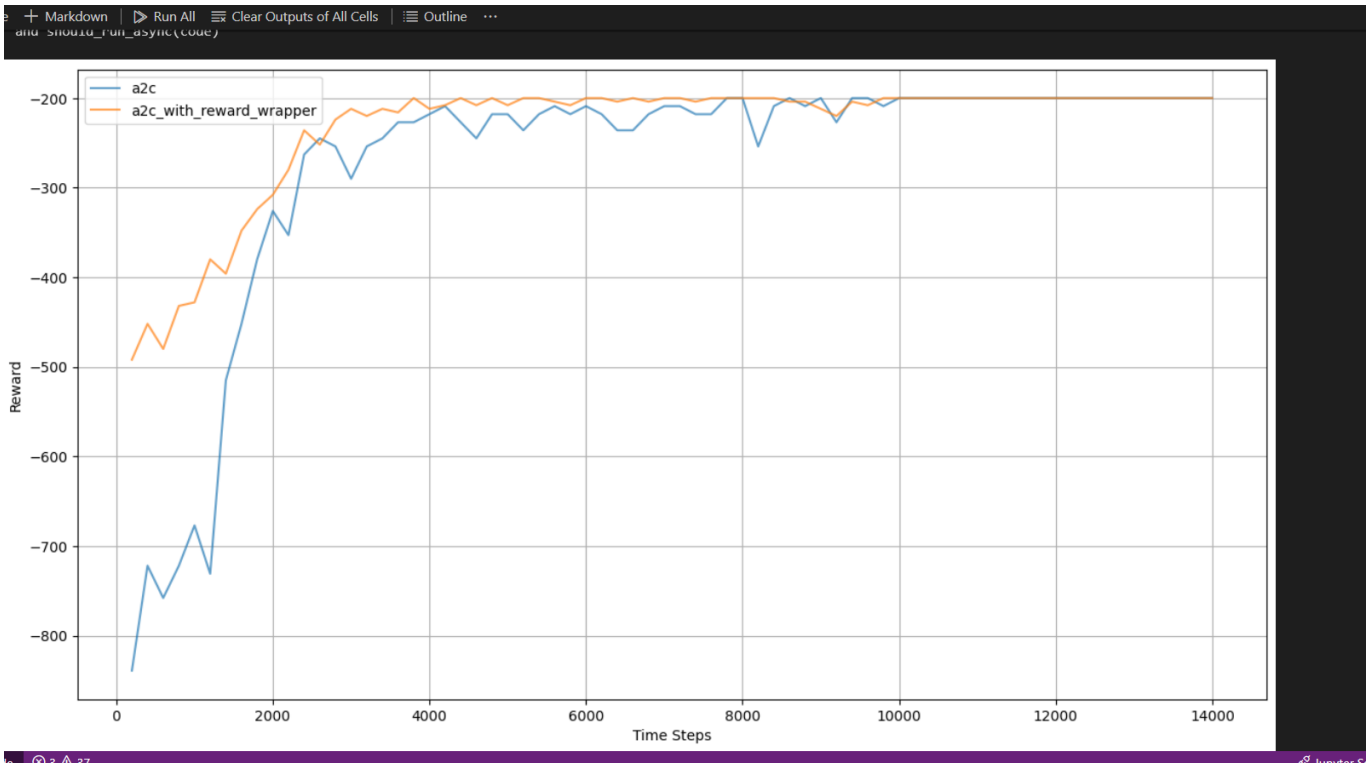
Impact on Convergence:

With this reward wrapper, convergence was faster.

Reason for Improvement:

- Stronger positive signals: Increasing the reward for successful drop-offs incentivized the agent to learn efficient routes more quickly.
- Weaker penalties: Reducing the penalty for mistakes encouraged the agent to explore more without severe setbacks, leading to faster learning.

You can see the results here:



Conclusion

- In the CartPole environment, PPO was superior due to its stable learning updates, while A2C struggled without policy clipping.
- In the Taxi environment, A2C performed better with high learning rates due to the simpler, discrete nature of the problem.
- Adding a reward wrapper in the Taxi environment accelerated learning by providing clearer feedback signals.

This experiment highlights the importance of choosing suitable algorithms and reward structures based on the environment and problem characteristics.

2 Task 2: Creating Custom Environments [45-points]

2.1 Introduction

In this project, I implemented a custom Gym environment using `gym.Env`, representing an $n \times n$ grid containing an agent, a goal, and obstacles. The agent navigates the grid to reach the goal while avoiding obstacles. I experimented with Q-learning and various deep reinforcement learning algorithms to analyze their performance.

2.2 Environment Setup

The environment was defined with the following specifications:

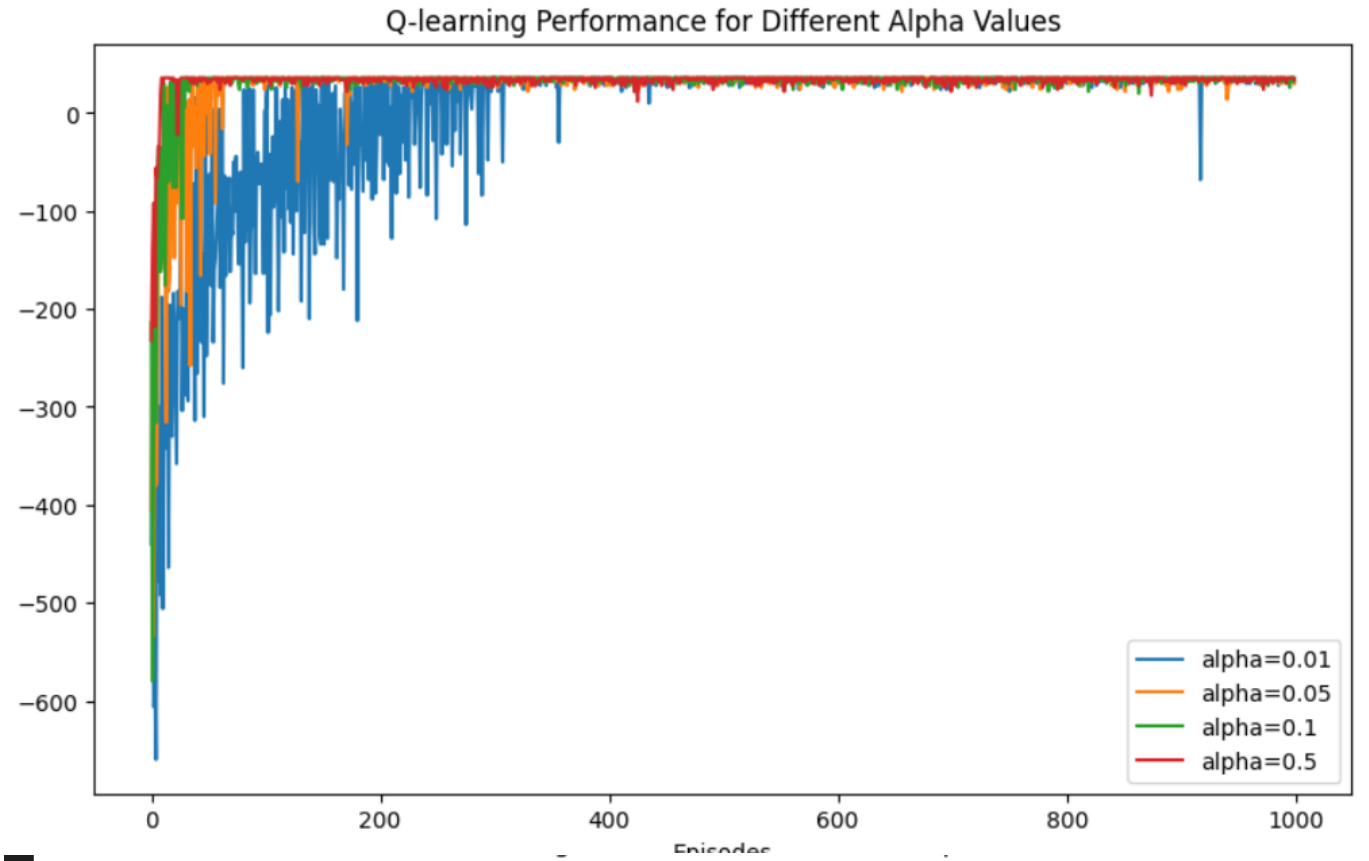
- **Action space:** `spaces.Discrete(4)`, corresponding to movements: 0: Up, 1: Down, 2: Left, 3: Right.
- **Observation space:** `spaces.Discrete(self.grid_area)`, where `grid_area` represents the total number of grid cells (n^2).
- **Reward function:** A reward of 50 was given upon reaching the goal (completion), and a penalty of -2 was applied for every step taken.

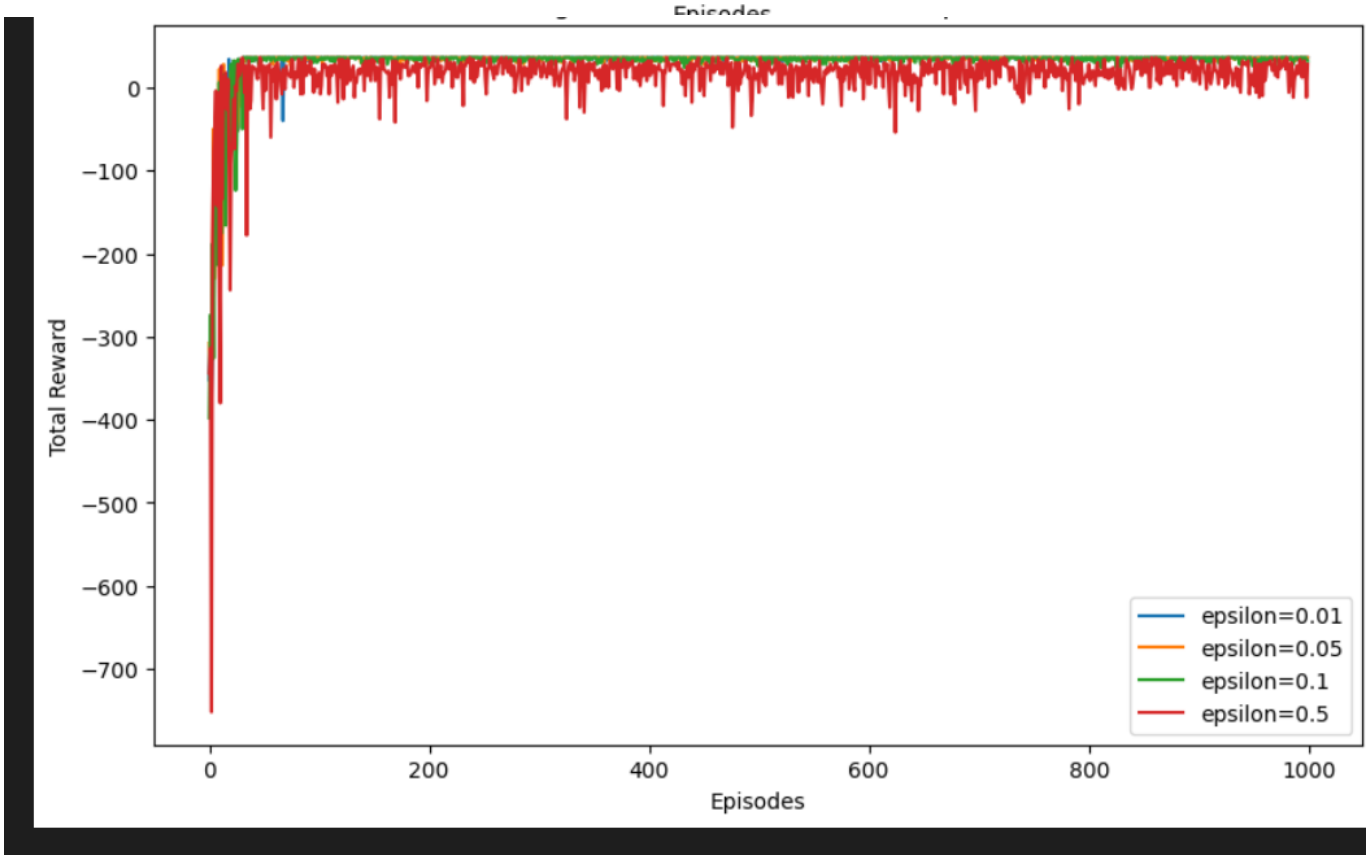
2.3 Q-Learning Experiments

Q-learning was applied with different values of learning rate (α) and exploration rate (ϵ) to observe their effect on learning performance.

- Higher α values led to faster learning but increased instability.
- Higher ϵ values resulted in more exploration, reducing the likelihood of early convergence to suboptimal policies.

You can see the result of Q-Learning with different parameters:


Q-learning Performance for Different Alpha Values
0
−100
−200
−300
−400
−500
−600
0
200
400
600
800
1000
alpha=0.01
alpha=0.05
alpha=0.1
alpha=0.5

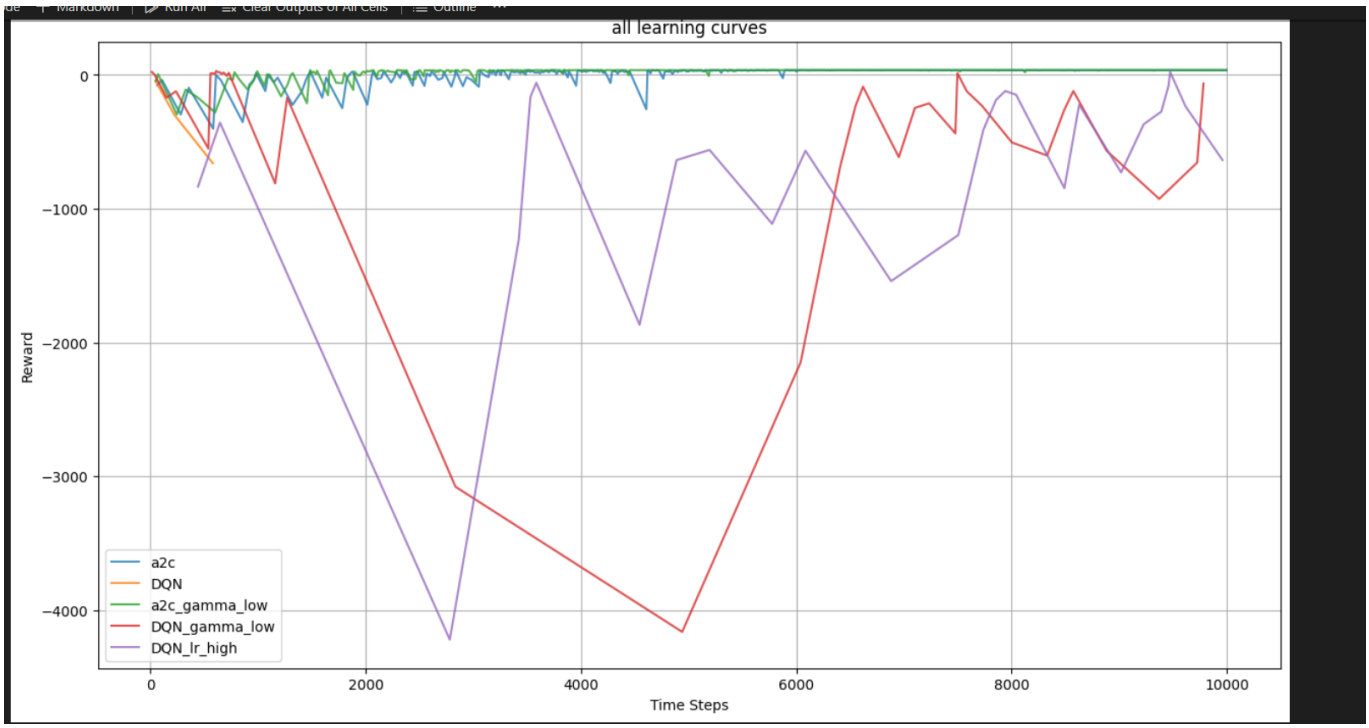

Total Reward
0
−100
−200
−300
−400
−500
−600
−700
0
200
400
600
800
1000
Episodes
epsilon=0.01
epsilon=0.05
epsilon=0.1
epsilon=0.5

2.4 Deep Reinforcement Learning Algorithms and Hyperparameters

The following algorithms and their respective hyperparameters were used for training:

- **A2C (Advantage Actor-Critic)**
- **DQN (Deep Q-Network)**
- **A2C with Low γ (discount factor)**
- **DQN with Low γ (discount factor)**
- **DQN with High Learning Rate (α)**

You can see the result of different algorithms and hyper parameters:



2.5 Results and Analysis

The results of the experiments revealed distinct differences in algorithm performance:

- **DQN Performance:** The DQN algorithm exhibited significant fluctuations and instability, especially with high learning rates and low discount factors. This behavior is due to the algorithm's sensitivity to hyperparameters and its reliance on experience replay, which can introduce variance in learning.
- **A2C Performance:** Both standard A2C and A2C with low discount factor converged quickly to a stable, high reward. A2C's advantage function and synchronous updates contributed to faster convergence and more stable learning.

2.6 Conclusion

In summary, the experiments demonstrated that while DQN can struggle with stability, A2C provides more consistent results due to its actor-critic architecture. The choice of hyperparameters, particularly learning rate and discount factor, plays a crucial role in the performance of reinforcement learning algorithms.

3 Task 3: Pygame for RL environment [20-points]

3.1 Introduction

This report describes the implementation of two custom reinforcement learning (RL) environments using Pygame, made compatible with OpenAI Gym for training RL agents. Both environments are solved using the PPO algorithm from Stable-Baselines3 (SB3).

3.2 Environment 1: Simple Grid-Based Environment

3.2.1 Environment Description

The first environment is a basic grid-based environment where the agent navigates to a goal while avoiding obstacles.

3.2.2 Parameters

- Grid Size: GRID_SIZE
- Obstacles: $\{(2, 2), (3, 3)\}$
- Goal Position: $(4, 4)$
- Action Space: Discrete (4) - Up, Right, Down, Left
- Observation Space: 2D coordinates (Agent Position)

3.2.3 Step Function

The step function updates the agent's position based on the action:

```
if action == 0 and y > 0: y -= 1
elif action == 1 and x < GRID_SIZE-1: x += 1
elif action == 2 and y < GRID_SIZE-1: y += 1
elif action == 3 and x > 0: x -= 1
```

3.2.4 Reward Function

- -0.1 per step (to encourage efficient navigation)
- $+10$ if the agent reaches the goal
- Termination if hitting an obstacle or reaching the goal

3.2.5 Results

Using PPO from SB3, the agent learned to reach the goal efficiently. The rendering function worked well on Google Colab, displaying the agent's navigation.

3.3 Environment 2: Complex Dynamic Environment

3.3.1 Environment Description

The second environment is more complex, simulating an agent moving with velocity and acceleration towards a goal while avoiding obstacles.

3.3.2 Parameters

- Agent Radius: 10
- Maximum Speed: 10
- Acceleration: 2
- Goal Position: $(450, 450)$
- Obstacles: $(200, 200), (300, 300)$
- Maximum Steps per Episode: 400
- Action Space: Discrete (4) - Accelerate Up, Right, Down, Left
- Observation Space: Position (x, y) and Velocity (vx, vy)

3.3.3 Step Function

The step function updates the agent's velocity and position based on the chosen action:

```
if action == 0:
    self.vel[1] = np.clip(self.vel[1] - ACCELERATION, -MAX_SPEED, MAX_SPEED)
elif action == 1:
    self.vel[0] = np.clip(self.vel[0] + ACCELERATION, -MAX_SPEED, MAX_SPEED)
elif action == 2:
    self.vel[1] = np.clip(self.vel[1] + ACCELERATION, -MAX_SPEED, MAX_SPEED)
elif action == 3:
    self.vel[0] = np.clip(self.vel[0] - ACCELERATION, -MAX_SPEED, MAX_SPEED)
self.pos = np.clip(self.pos + self.vel, 0, WINDOW_SIZE)
```

3.3.4 Reward Function

- -0.002 times distance to goal (to encourage moving closer)
- -200 if collision with an obstacle
- $+1000$ if reaching the goal
- Episode terminates on collision, reaching the goal, or exceeding 400 steps

3.3.5 Results

The PPO algorithm from SB3 successfully trained the agent to navigate towards the goal while avoiding obstacles. The rendering function also worked correctly in Colab.

3.4 Conclusion

Both custom RL environments were implemented and trained using PPO from SB3. The simple environment provided a basic testbed for the RL agent, while the complex environment tested advanced navigation capabilities.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, 2020. Available online: http://incompleteideas.net/book/the-book-2nd.html

[2] A. Raffin et al., "Stable Baselines3: Reliable Reinforcement Learning Implementations," GitHub Repository, 2020. Available: https://github.com/DLR-RM/stable-baselines3.

[3] Gymnasium Documentation. Available: https://gymnasium.farama.org/.

[4] Pygame Documentation. Available: https://www.pygame.org/docs/.

[5] CS 285: Deep Reinforcement Learning, UC Berkeley, Pieter Abbeel. Course material available: http://rail.eecs.berkeley.edu/deeprlcourse/.

[6] Cover image designed by freepik