Intelligent Autonomous System Project4 Report

Gilberto E. Ruiz (ger83)

May 9, 2024

Introduction:

For the final project, I was given the task of applying reinforcement learning algorithms to solve decision making tasks within various Markov Decision Process (MDP) environments. Specifically, implementing Q-learning and REINFORCE algorithms, as well as a choice between Value Iteration and Policy Iteration algorithms. These methodologies are applied to the discrete state space of a Maze environment and the continuous state spaces of the Acrobot-v1 and MountainCar-v0 environments from OpenAI Gym. The project aims not only to find optimal policies but also to understand the dynamics of learning and the effectiveness of different algorithmic approaches in reinforcement learning.

Problem Formulation:

The project itself was divided into three main tasks: Maze Environment, Q-learning on Maze, and Continuous State Space Problems. For the Maze Environment, I had to choose to implement either Policy Iteration or Value Iteration to determine optimal policies and Q-values for a maze where a robot must collect flags and reach a goal.

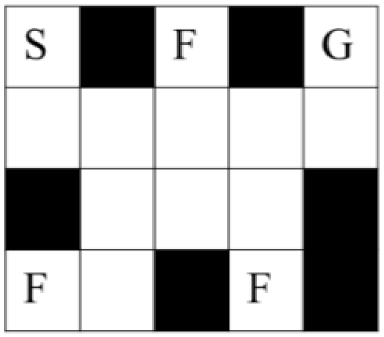
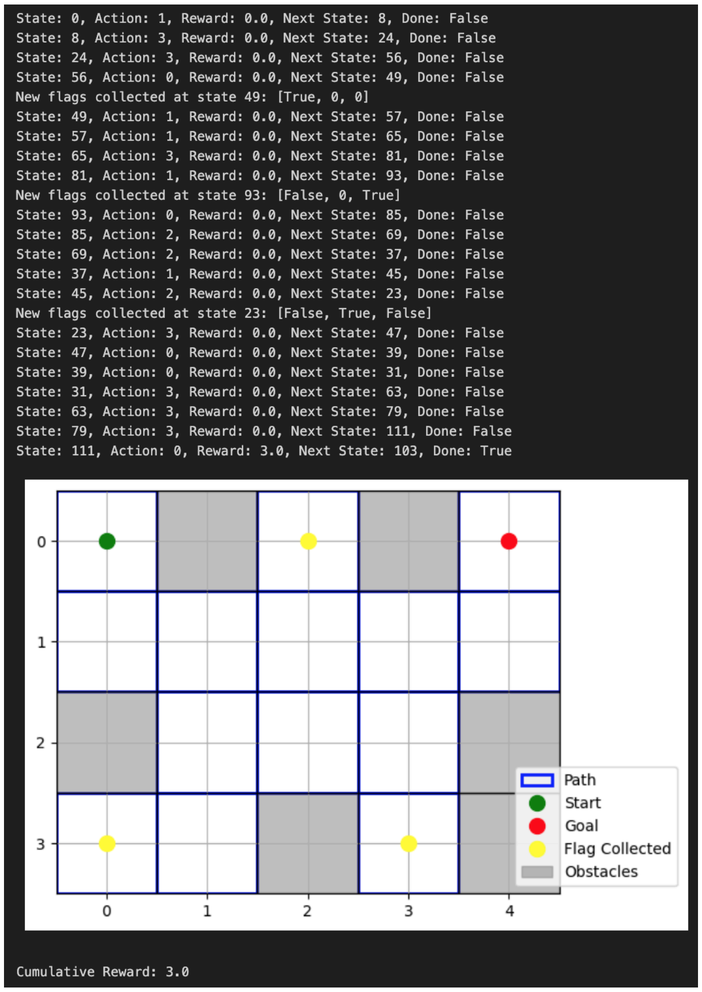


Figure 1: Maze to travel for part 1

For the second part of the assignment, I had to apply Q-learning to the Maze for a fixed number of steps, experimenting with different learning rates and action policies to understand their impact on the learning process and convergence to optimal Q-values. And for the final part of the assignment, I had to implement the REINFORCE algorithm for the Acrobot-v1 and apply Q-learning to the MountianCar-v0, the main focus in addition to the algorithm itself was the hyperparameter tuning and performance evaluation through episodic rewards and steps. For all three parts of the project, I completed them in my python notebook titled "RLearning.ipynb" and it is split up into 7 cells.

Technical Approach and Results:

For the Maze Environment, I implemented the value iteration method to compute the Q-table which describes the expected utility of taking a specific action in a specific state and following the optimal policy thereafter. The 'value_iteration' function initializes values for all states to zero and iteratively updates these values based on the expected rewards for possible actions in each state, adjusted by a discount factor to account for the value of future rewards. Adjustments for the maze environment included accounting for the probability of sliping and conditions for collecting flags, which provided additional rewards. The environment, customized through the 'retrievedMaze' class, simulates a typical maze scenario with slip conditions set to zero to simplify the path planning. This setting allowed the algorithm to focus on strategic decisions like flag collection and obstacle avoidance without the added complexity of stochastic movements. The iterative process continued until changes in the value function were below a small threshold, indicating convergence to an optimal policy. The resulting Q-table and policy were then saved to a specified directory for further use, the file is named "Optimal_Q_table.npy". Once the Q_table was calculated, I tested the effectiveness of the learned policy by simulating an episode within the maze. The function 'simulate_episode' utilizes the derived Q-table to navigate the maze from the start to the goal. The agent follows the path that maximizes expected returns, collecting flags and avoiding obstacles as dictated by the policy. Each action's outcome, including rewards and transitions, is recorded, showing how the agent would perform in a real scenario based on the learned policy. The function 'plot_episode' then visualizes the path, clearly marking important locations such as the start, goal, flags, and obstacles.



```
State: 0, Action: 1, Reward: 0.0, Next State: 8, Done: False
State: 8, Action: 3, Reward: 0.0, Next State: 24, Done: False
State: 24, Action: 3, Reward: 0.0, Next State: 56, Done: False
State: 56, Action: 0, Reward: 0.0, Next State: 49, Done: False
New flags collected at state 49: [True, 0, 0]
State: 49, Action: 1, Reward: 0.0, Next State: 57, Done: False
State: 57, Action: 1, Reward: 0.0, Next State: 65, Done: False
State: 65, Action: 3, Reward: 0.0, Next State: 81, Done: False
State: 81, Action: 1, Reward: 0.0, Next State: 93, Done: False
New flags collected at state 93: [False, 0, True]
State: 93, Action: 0, Reward: 0.0, Next State: 85, Done: False
State: 85, Action: 2, Reward: 0.0, Next State: 69, Done: False
State: 69, Action: 2, Reward: 0.0, Next State: 37, Done: False
State: 37, Action: 1, Reward: 0.0, Next State: 45, Done: False
State: 45, Action: 2, Reward: 0.0, Next State: 23, Done: False
New flags collected at state 23: [False, True, False]
State: 23, Action: 3, Reward: 0.0, Next State: 47, Done: False
State: 47, Action: 0, Reward: 0.0, Next State: 39, Done: False
State: 39, Action: 0, Reward: 0.0, Next State: 31, Done: False
State: 31, Action: 3, Reward: 0.0, Next State: 63, Done: False
State: 63, Action: 3, Reward: 0.0, Next State: 79, Done: False
State: 79, Action: 3, Reward: 0.0, Next State: 111, Done: False
State: 111, Action: 0, Reward: 3.0, Next State: 103, Done: True
```

```
Cumulative Reward: 3.0
```

Figure 2: Final Results of the 'plot_episode' function

For the Q-learning part of the project, I implemented the algorithm within the maze environment to optimize navigation strategies by learning from interactions. The code setup began by initializing the maze environment and defining various parameters critical for learning, including the exploration rate (epsilon), the learning rate, and the discount factor. The exploration rate started at 1.0, promoting initial exploration, and decayed over time to allow for increased exploitation of learned policies. Similarly, the learning rate was set to start high at .5 and decay, ensuring that updates to the Q-table reduced over time as the agent's experience increased. During the execution of 5000 steps, the agent interacted with the environment: exploring randomly according to the epsilon value of exploiting known strategies by selecting actions with the highest Q-values. After each action, the Q-table was updated based on the reward received and the maximum expected future rewards, adjusted by the learning rate and discount factor. To measure the performance and convergence of the algorithm, I periodically evaluated it every 50 steps by comparing the learned Q-values against a precomputed set of optimal Q-values using the Root Mean Square Error (RMSE). Additionally, I monitored the average number of steps taken and the average rewards received during these evaluations. The results showed a rapid initial decrease in RMSE, indicating swift learning of effective strategies, and the evaluations highlighted consistent improvements in the efficiency and effectiveness of navigational decisions made by the agent. These metrics were plotted to visually represent the learning progression and stability of the algorithm over time, demonstrating the successful application of Q-learning in a complex maze navigation task.

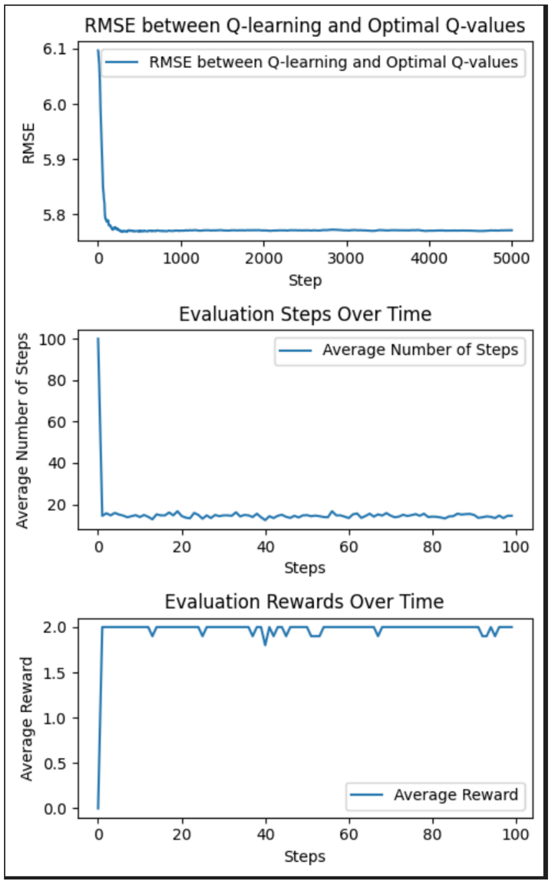


Figure 3: Plots of RMSE, Evaluation Steps and Evaluation Rewards

For the final part of the project, I implemented the REINFORCE algorithm to solve the Acrobot-v1 task. A policy network (PolicyNet) is employed, constructed using a simple neural network architecture with two fully connected layers. The first layer maps the state input to a hidden layer of 16 units using ReLU activation, and the second layer maps these to action probabilities using a softmax function. The training process involves running multiple episodes where actions are sampled from the policy's output distribution. For each action taken, a log probability is recorded, and rewards collected. Rewards are discounted using a backward approach, emphasizing the importance of immediate rewards over future ones. The accumulated log probabilities and the discounted rewards are then used to compute the policy gradient, which is applied to optimize the network. This optimization seeks to maximize the expected sum of discounted rewards, effectively refining the policy towards optimal actions over the episodes. Training diagnostics, such as episode lengths and rewards, are plotted to evaluate performance improvements.

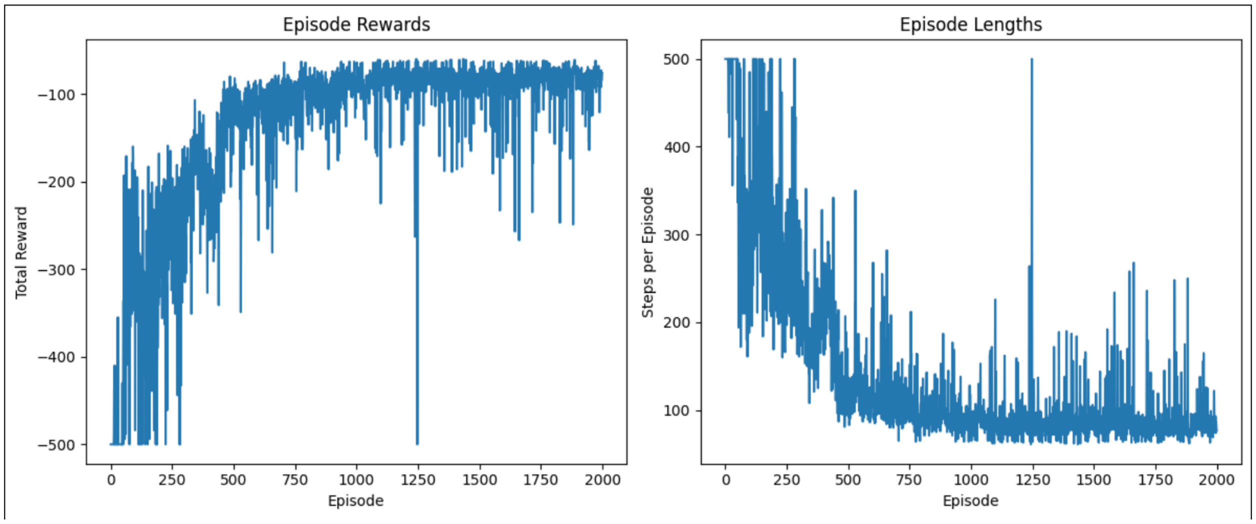


Figure 4: Training diagnostics for Acrobot-v1 plotted

In addition to plotting, I also rendered the REINFORCE environment to visually demonstrate the behavior of the trained policy. The environment is initialized in a mode that supports rendering ('render_mode="human"'). This visual output helps in qualitatively assessing how well the policy controls the Acrobot to reach the target position. However, this setup needs a manual intervention requirement to close the render window.

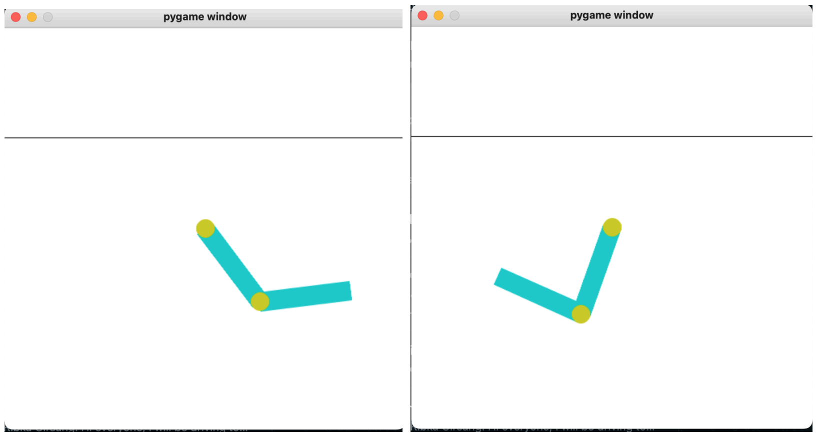


Figure 5: Acrobot-v1 Rendered Animation

For MountainCar-v0 on the other hand, I utilized Q-learning to handle continuous state spaces by discretizing the position and velocity states of the environment into bins. This discretization transforms the problem into a more tractable tabular Q-learning scenario, where a Q-table can be uploaded using traditional methods. An epsilon-greedy strategy governs the balance between exploration and exploitation, dynamically adjusting as the agent learns about the environment. The Q-values are updated following the Bellman equation, with adjustments in the learning rate to encourage faster convergence when the car moves closer to the goal. The reward function is also made to promote reaching the target by giving incremental rewards for positive velocities and positions moving towards the goal. The learning process is visualized by plotting the total rewards accumulated in each episode, showing the agent's progress in solving the task.

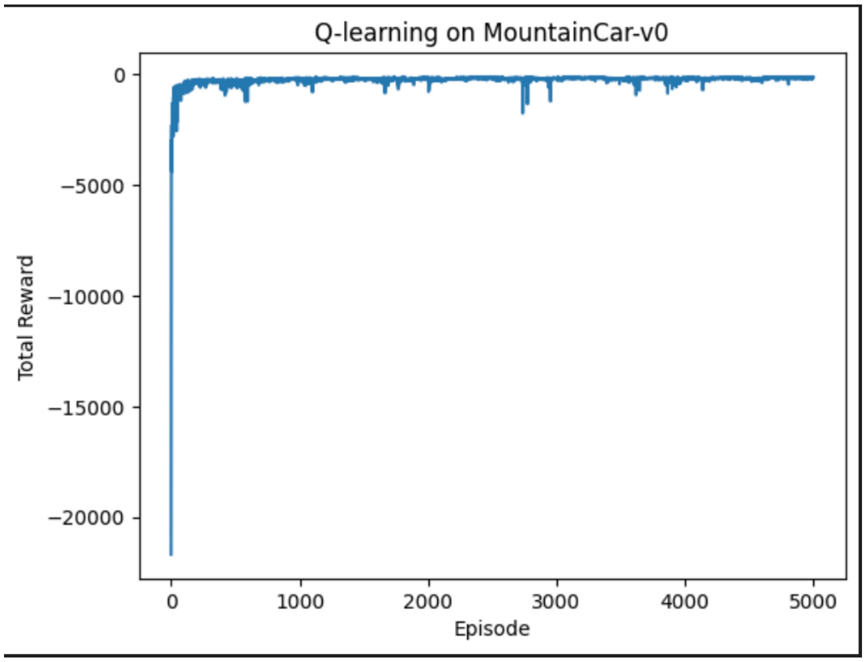


Figure 6: The learning process on MountainCar visualized

Similar to the Acrobot setup, the MountainCar-v0 environment is visualized using an animated display. The same Q-learning algorithm runs, but each state transition and the chosen actions are rendered on screen. This setup is especially useful for debugging and directly observing the strategic movements learned by the Q-agent, but with the need for manual termination of the session due to the continuous rendering.

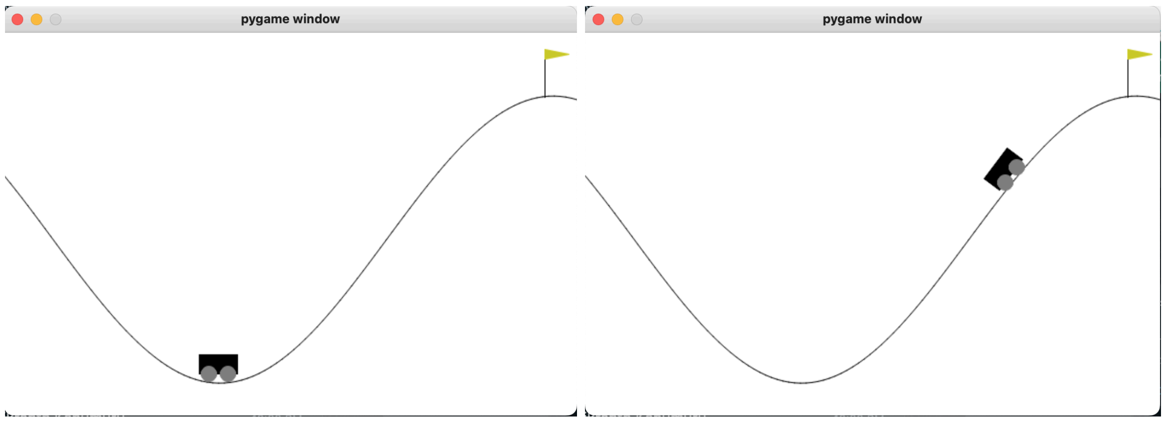


Figure 7: MountainCar-v0 Rendered Animation

Results and Discussion:

The results obtained from applying reinforcement learning algorithms to different environments provide valuable insights into the dynamics of various decision-making strategies in complex scenarios. For the maze Environment, the value iteration method effectively computed a robust Q-table, which allowed the autonomous agent to successfully navigate through the maze, demonstrating high efficiency in both flag collection and obstacle avoidance. The simulation of the maze traversal using the optimized Q-table showcased a significant alignment between the theoretical policy and practical execution, reinforcing the value of precise modeling in reinforcement learning tasks.

In the Acrobot-v1 challenge, the REINFORCE algorithm's implementation demonstrated how well policy gradient methods are in continuous state spaces. The neural network, despite how simple it is, was adept at learning a policy that significantly improved the performance over episodes. The plots of episode rewards and lengths exhibited a trend towards optimization, with increasing total rewards and decreasing steps per episode, indicating the policy's increasing efficiency. Rendering the Acrobot's movement provided a direct visual affirmation of the theoretical improvements, showing the agent's enhanced ability to control and stabilize the Acrobot swiftly.

On the other hand, the Q-learning approach applied to MountainCar-v0 highlighted some of the limitations encountered in sparse reward environments. Despite the innovative approach of discretizing the continuous state space and employing an epsilon-greedy strategy for balanced exploration and exploitation, the MountainCar consistently failed to reach the goal within the given episodes. This outcome underscores the challenge of applying Q-learning in environments where suitable rewards are not frequently encountered, needing further adjustments in the strategy or alternative methods that might better handle the task.

Conclusion:

In conclusion, this project has significantly enhanced my understanding and application of reinforcement learning algorithms within various environments. Through the practical implementation of Q-learning, REINFORCE, and value iteration methods, I have gained deeper insights of policy development, state space management, and the balancing act between exploration and exploitation. This experience has also improved my technical skills in adapting these algorithms to tackle complex, real-world problems in intelligent autonomous systems. The challenges encountered and the solutions developed have been crucial in improving my comprehension of the dynamic field of reinforcement learning.