CISC 856 – Reinforcement Learning

Prof. Sidney Givigi

Final Project Report
Mountain Car

Group: Abdallah Ibrahim 20398554
Ali Aboelela 20398556
Amr Sayed 20398048

Contents

Introduction 3
Problem Formulation 3
Solution Overview 3
1- Algorithms: 4
Q-Learning: 4
B- DQN 5
C- SARSA 6
2- Performance Metrics: 7
Convergence Speed (Learning Curve – Cumulative Rewards): 7
Optimal Actions: 7
Trajectories: 7
Results: 8
Q-Learning 8
1st Trial: 8
2nd Trial: 9
3rd Trial: 11
DQN 13
1st Trial 13
2nd Trial 15
3rd and 4th trials 17
SARSA .... **Error! Bookmark not defined.**
1st Trial 17
2nd Trial 19
Performance Summary: 21
Conclusion: 22
References: 23

Introduction

The Mountain Car problem is an interesting and challenging problem in the field of reinforcement learning. The problem involves a car that is stuck in a valley and the goal is to reach the top of a hill by applying throttle. The car is underpowered to drive up the hill directly and has to build up enough momentum by first moving away from the goal. This problem is particularly challenging because it requires the agent to learn how to balance between exploring new actions and exploiting previously learned knowledge to reach the goal efficiently. Additionally, the continuous state space of the problem presents a challenge in finding an optimal policy.

The objective of this project is to apply various RL algorithms to solve the Mountain Car problem and analyze their performance in terms of convergence speed and optimal solutions. The project aims to explore the effectiveness of different RL algorithms and their hyperparameters in solving this problem.

Problem Formulation

The Mountain Car problem is a reinforcement learning problem where the goal is to reach the top of a mountain on a one-dimensional track using a car that has limited engine power. The state space consists of the car's position and velocity, while the action space consists of three possible actions: move left, move right, or do nothing. The reward scheme encourages the car to reach the goal as quickly as possible, with a penalty for each timestep taken. Reinforcement learning algorithms like Q-learning, DQN and SARSA can be used to train the agent to learn an optimal policy. Continuous adaptation is important due to the car's limited engine power and steep terrain. The OpenAI Gym environment is used for the Mountain Car problem, and various RL algorithms will be applied, including Q-learning, SARSA, and DQN, with hyperparameter tuning to find the optimal settings. Training performance will be evaluated by plotting the cumulative reward per episode and the number of timesteps required to solve the task per episode. The performance of each algorithm will be analyzed in terms of convergence speed and optimal solutions.

Solution Overview

The algorithms used are Q-Learning, DQN and SARSA. The metrics used are cumulative rewards, optimal actions and trajectories. Two Trials were executed for each algorithm. One trial with a few episodes and one with many episodes. Additionally, two extra trials were done for the DQN algorithm with random search for hyperparameter tuning and one trial with grid search for the Q-Learning hyperparameters tuning. Each trial is eventually evaluated using the cumulative rewards and trajectory. This approach was taken to check the best approach to solve the problem.

1- Algorithms:

Q-Learning:

Q-learning is a reinforcement learning algorithm that learns an optimal policy by estimating the action-value function Q(s,a) for each state-action pair. At each time step, the agent observes the current state of the environment, selects an action, receives a reward, and observes the next state of the environment. The agent then updates its estimate of the action-value function Q(s,a) using the Q-learning update rule, which takes into account the observed reward and next state. The algorithm iteratively updates the Q-values for all state-action pairs until it converges to the optimal Q-values. Once the Q-values have converged, the optimal policy can be derived by selecting the action with the highest Q-value for each state. Q-learning is a simple and effective algorithm that can handle large state and action spaces, but it requires complete knowledge of the MDP and can be slow to converge for large state and action spaces.

Architecture:

- The Q-Learning function is defined, which takes several parameters, including the environment (env), the learning rate (learning_rate), the discount factor (discount_factor), the exploration rate (exploration_rate), the minimum exploration rate (min_exploration_rate), the number of training episodes (train_episodes), and the number of test episodes (test_episodes).
- The size of the discretized state space is calculated based on the environment's observation space, and a Q-table is initialized with random values.
- The function tracks the rewards and timesteps for both training and test episodes.
- The episodic reduction in the exploration rate is calculated based on the number of training episodes.
- The Q-learning algorithm is run for a specified number of training and test episodes. In each episode, the agent interacts with the environment, chooses actions based on an epsilon-greedy exploration strategy, and updates the Q-table based on the observed rewards and next states. The exploration rate is decayed every training episode until it reaches the minimum exploration rate.
- After every amount of training episodes, the function runs a specified number of test episodes to evaluate the agent's performance. The mean reward and timesteps for these test episodes are reported.
- Finally, the function returns the lists of rewards and timesteps for both training and test episodes.

B- DQN

DQN is a reinforcement learning algorithm that combines Q-learning with deep neural networks to learn the Q-value function for a given environment. It uses a neural network to approximate the Q-value function for each possible action, and it uses experience replay to improve the stability and efficiency of learning. DQN also uses a target network to improve learning stability. DQN has been shown to be effective in various applications, but it has some limitations, such as the need for large amounts of training data and the tendency to overestimate Q-values in some cases.

Architecture:

- The DQN function is defined, which takes several parameters, including the environment (env), the learning rate (learning_rate), the discount factor (discount_factor), the exploration rate (exploration_rate), the minimum exploration rate (min_exploration_rate), and the number of episodes (episodes).
- The size of the discretized state space is calculated based on the environment's observation space, and a Q-network is initialized with random weights.
- The function tracks the rewards and average rewards for each 100 episodes.
- The episodic reduction in the exploration rate is calculated based on the number of episodes.
- The DQN algorithm is run for a specified number of episodes. In each episode, the agent interacts with the environment, chooses actions based on an epsilon-greedy exploration strategy, and updates the Q-network based on the observed rewards and next states using the Q-learning update rule. The exploration rate is decayed every episode until it reaches the minimum exploration rate.
- After every amount of episodes, the function calculates the average reward for the previous number of episodes and tracks it in the ave_reward_list.
- Finally, the function returns the ave_reward_list and the Q-network weights.

In the main part of the code, the parameters for the DQN algorithm are set, and the DQN function is called to run the algorithm. The average reward over episodes is plotted and saved as an image. Is.

C- SARSA

SARSA is a reinforcement learning algorithm that learns the optimal policy for an agent interacting with an environment. It updates the Q-values based on the tuple (state, action, reward, next state, next action) and considers the next action the agent will take. SARSA is like Q-learning but may converge more slowly and get stuck in local optima. The SARSA update rule considers the reward obtained by taking an action, the next state, and the next action obtained by applying the policy to the Q-values for the next state.

Architecture:

The code implements the SARSA (State-Action-Reward-State-Action) algorithm for solving a reinforcement learning problem. Here's a breakdown of the code's architecture:

- The SARSA function is defined, which takes several parameters, including the environment (env), the learning rate (learning_rate), the discount factor (discount_factor), the exploration rate (exploration_rate), the minimum exploration rate (min_exploration_rate), and the number of episodes (episodes).
- The size of the discretized state space is calculated based on the environment's observation space, and a Q-table is initialized with random values.
- The function tracks the rewards and average rewards for each amount of episodes.
- The episodic reduction in the exploration rate is calculated based on the number of episodes.
- The SARSA algorithm is run for a specified number of episodes. In each episode, the agent interacts with the environment, chooses actions based on an epsilon-greedy exploration strategy, and updates the Q-table based on the observed rewards and next states using the SARSA update rule. The exploration rate is decayed every episode until it reaches the minimum exploration rate.
- After every amount of episodes, the function calculates the average reward for the previous number of episodes and tracks it in the ave_reward_list.
- Finally, the function returns the ave_reward_list and the Q-table.

In the main part of the code, the parameters for the SARSA algorithm are set, and the SARSA function is called to run the algorithm. The average reward over episodes is plotted and saved as an image.

2- Performance Metrics:

Convergence Speed (Learning Curve – Cumulative Rewards): which is the rate at which an RL algorithm learns an optimal policy for a given task. In the context of the Mountain Car problem, convergence speed refers to how quickly the agent reaches the top of the hill with the optimal policy. Faster convergence speed means that the agent learns an optimal policy more quickly and requires fewer training episodes. <u>The learning curve for each RL algorithm shows the average reward obtained over a certain number of episodes of the MountainCar-v0 environment</u>,

Several factors can affect the convergence rate of RL algorithms, including learning rate, discount factor, exploration rate, state representation, environment complexity, and algorithm choice. A good state representation captures relevant features of the environment and makes it easier for the algorithm to learn a good policy. Environment complexity can also affect convergence rate, and different RL algorithms may perform better or worse on different problems.

Optimal Actions: The code visualizes the optimal actions for each state in the state space of the Mountain Car problem the resulting image shows the optimal actions for all states in the state space, and it can be used to gain insights into the agent's behavior and to evaluate the effectiveness of the learning algorithm. Visualization can help us understand the agent's learning process and identify areas for improvement. It can also be used to evaluate the robustness of the agent's policy and to compare the performance of different RL algorithms. The visualization uses a colormap to represent the optimal actions, where blue represents the action to move left, red represents the action to move right, and white represents the action to do nothing. The intensity of the color indicates the confidence in taking that action. Overall, visualization provides a clear and intuitive way to understand the optimal actions for each state in the state space of the Mountain Car problem and can be a powerful tool for analyzing and improving the performance of RL algorithms.

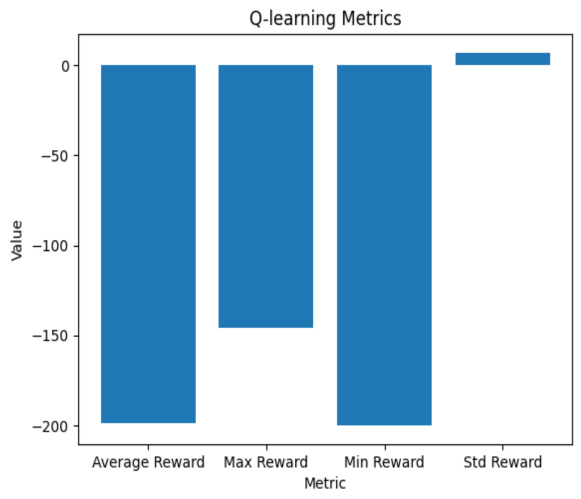
Trajectories: The concept of trajectories in the Mountain Car problem, which is a sequence of states that the agent reaches while attempting to climb the mountain. The trajectory can be visualized as a plot of the car's position and velocity over time, and analyzing the trajectory can provide insights into the agent's learning process and performance. By analyzing the trajectory, we can gain insights into whether the agent is getting stuck in a local minimum or exploring different regions of the state space. We can also use trajectory analysis to adjust hyperparameters, modify the reward function, change the algorithm, or add more features to the state representation to improve the agent's performance. To determine whether the agent has successfully climbed the mountain, we check whether its position has exceeded a certain threshold value, which represents the top of the mountain – 0.5 in our case.

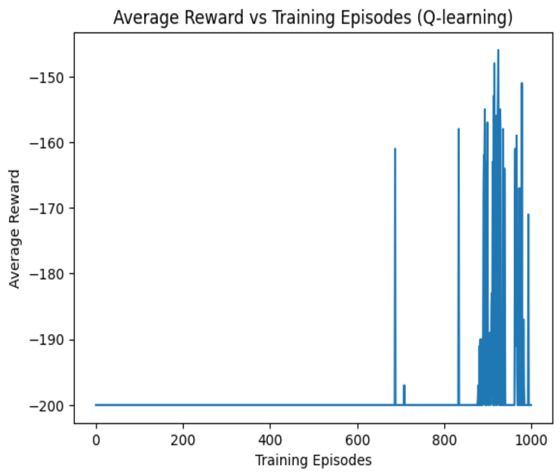
Results:

Q-Learning

1st Trial:

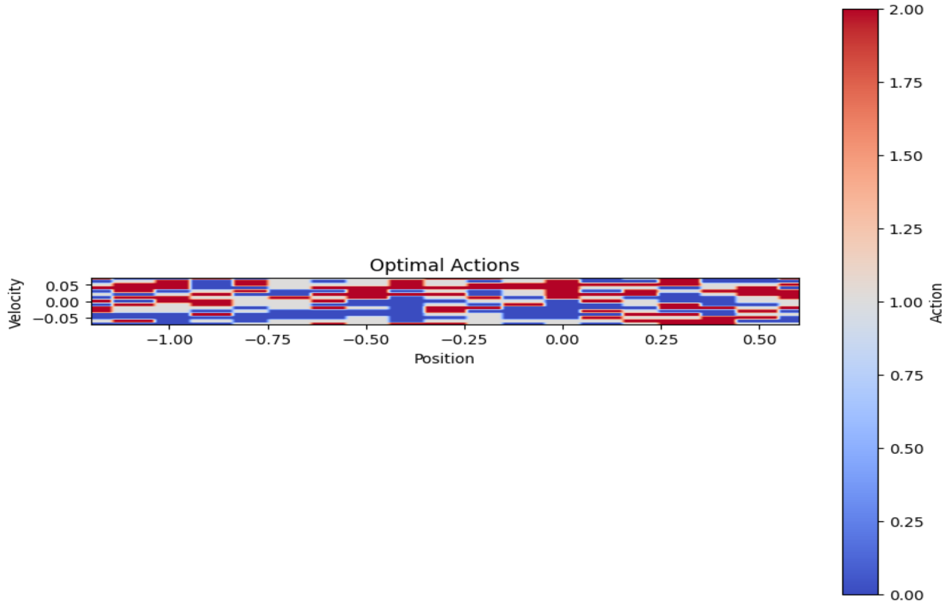
- Cumulative Rewards



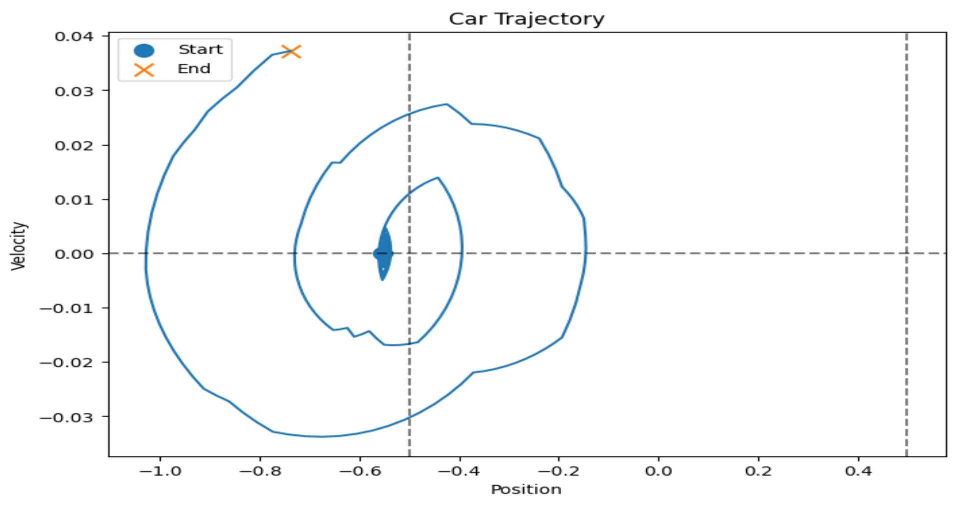


Q-learning Metrics:
Average Reward: -198.628
Max Reward: -146.0
Min Reward: -200.0
Std Reward: 6.9169079797262025

- Optimal Actions



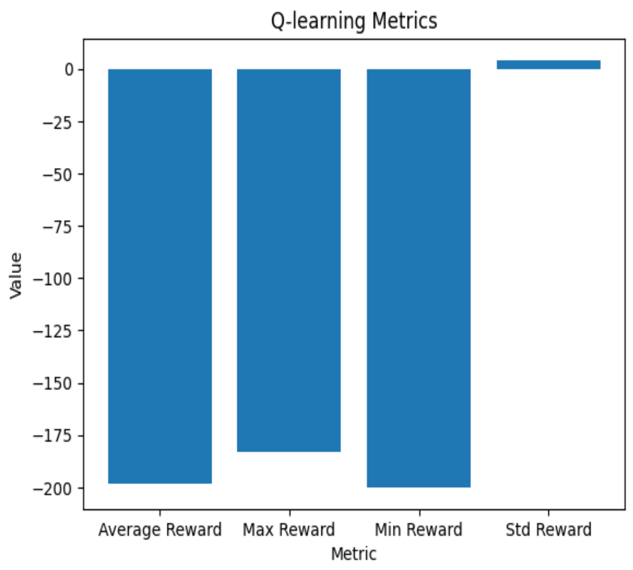
- Trajectory

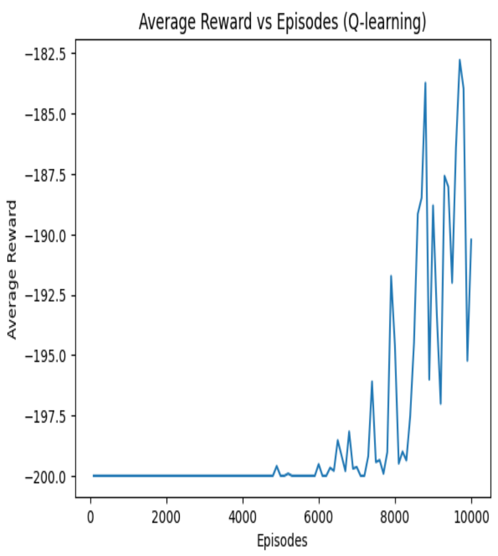


Observation: The agent failed to reach the top of the mountain. Rewards were acceptable; however, trajectory shows the failure to reach the top.

2nd Trial:

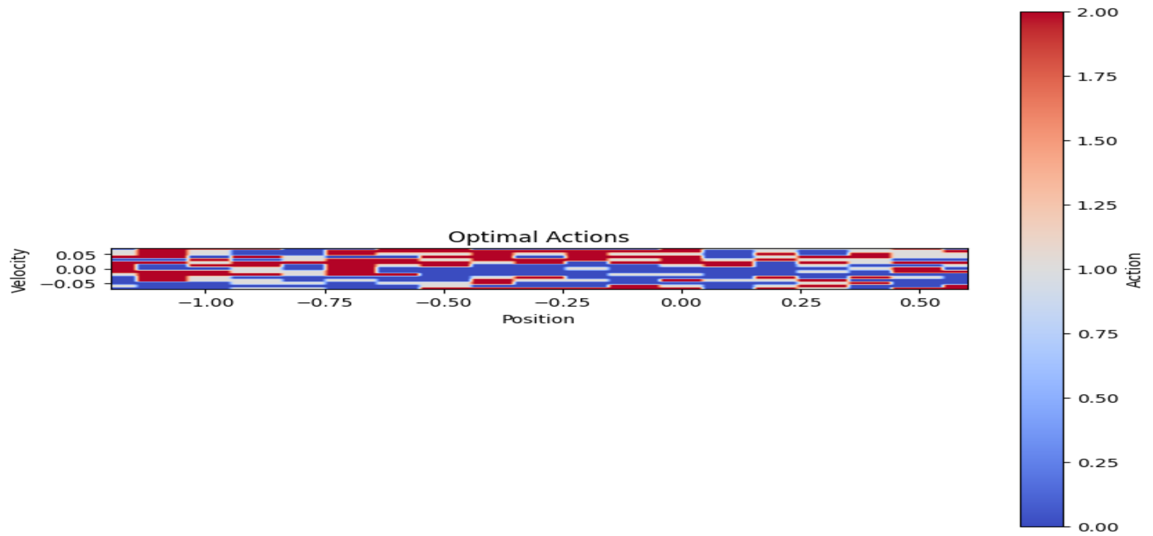
- Cumulative Rewards



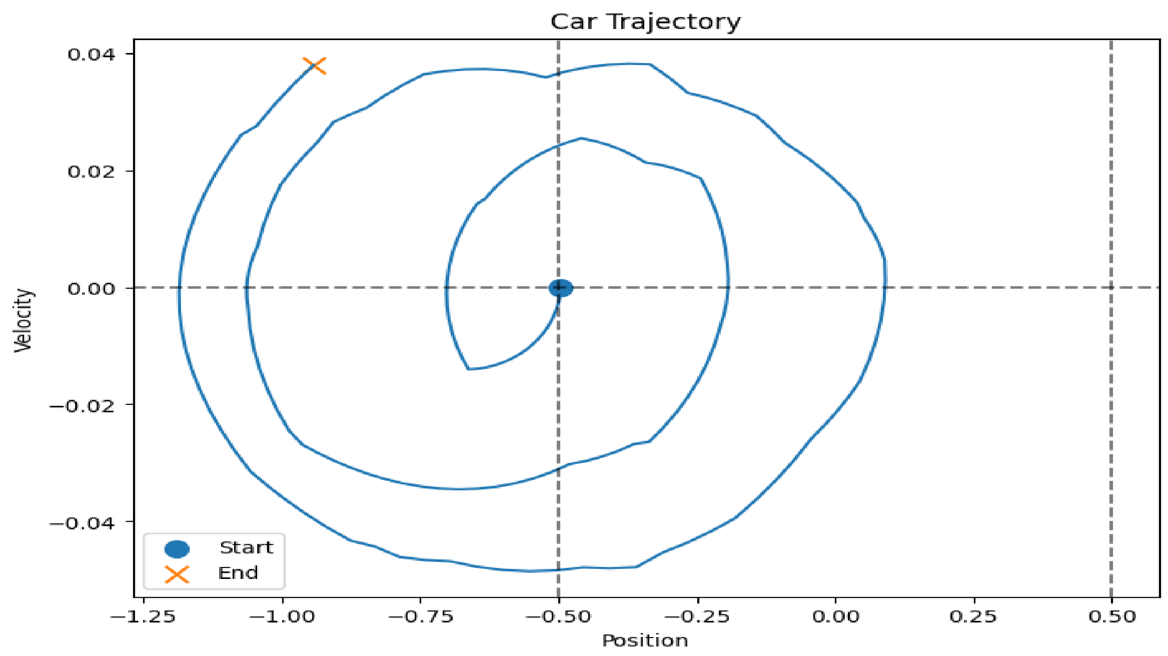


Q-learning Metrics:
Average Reward: -198.05539999999996
Max Reward: -182.78
Min Reward: -200.0
Std Reward: 4.133102084391335

-Optimal Actions



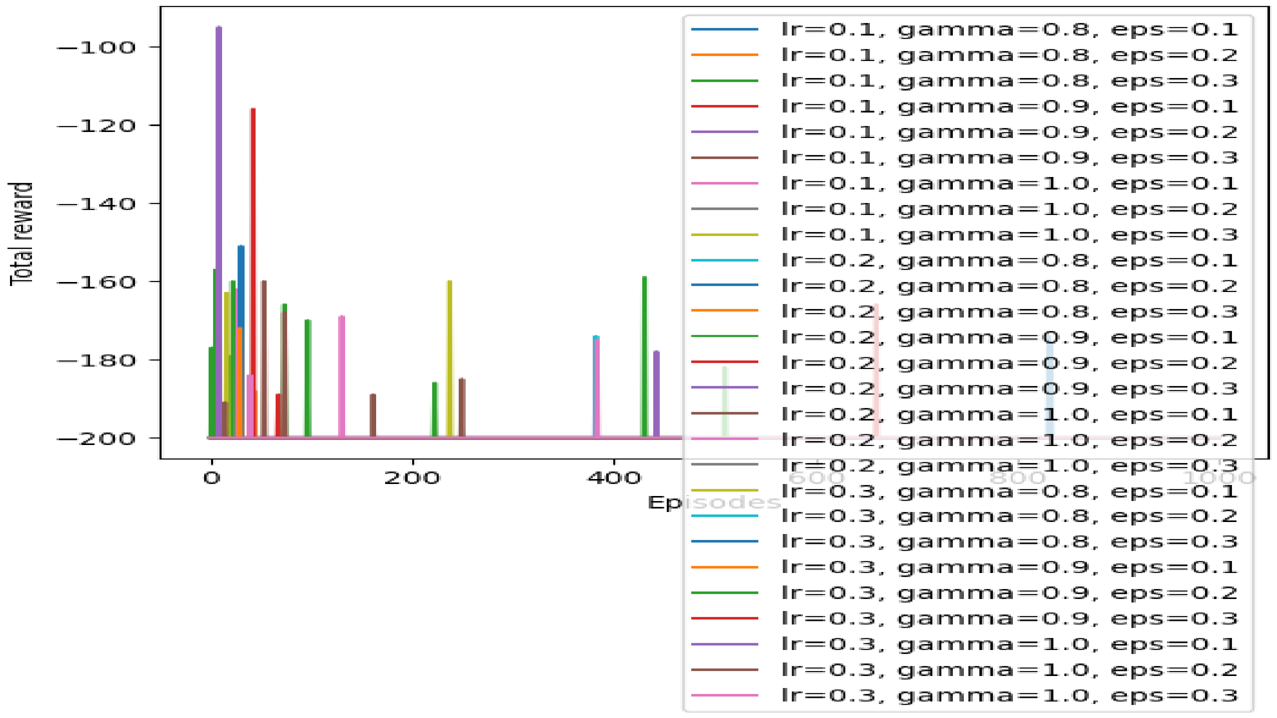
- Trajectory



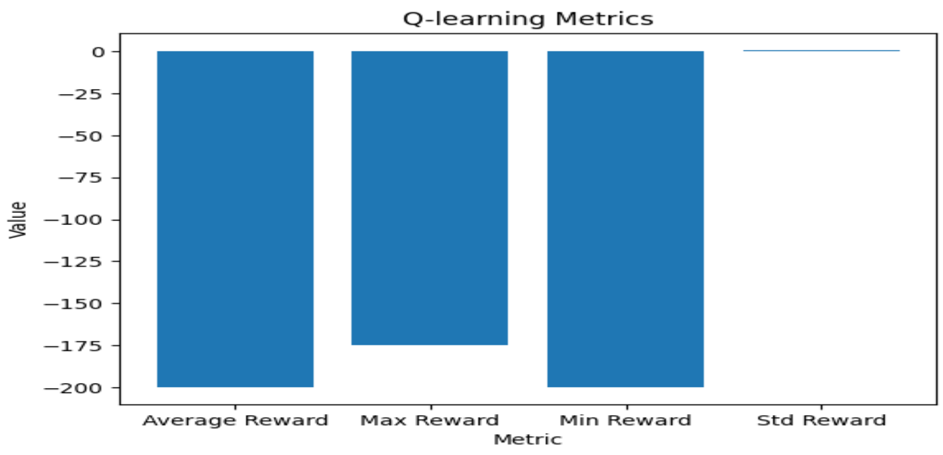
Observation: Rewards were good. This trial gave almost similar behavior to the 1st trial with respect to rewards but with better rewards std which might be due to larger no of episodes. Trajectory shows that the agent kept revolving without reaching the top as well.

3rd Trial:

- Cumulative Rewards and Best Hyperparameter

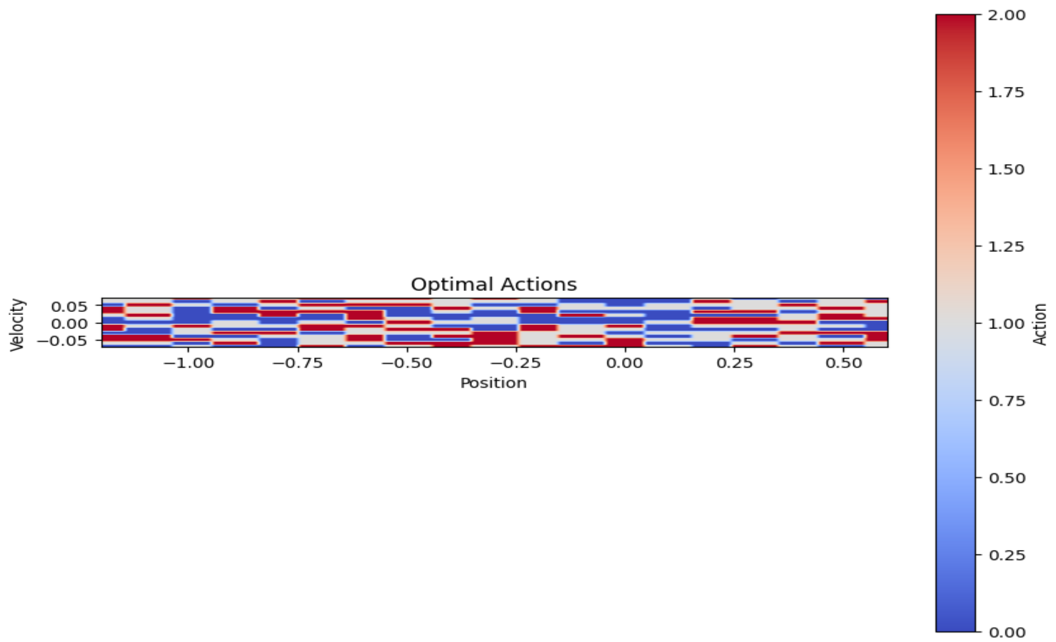


Best hyperparameters: {'lr': 0.2, 'gamma': 0.9, 'eps': 0.1} Best reward: -199.887

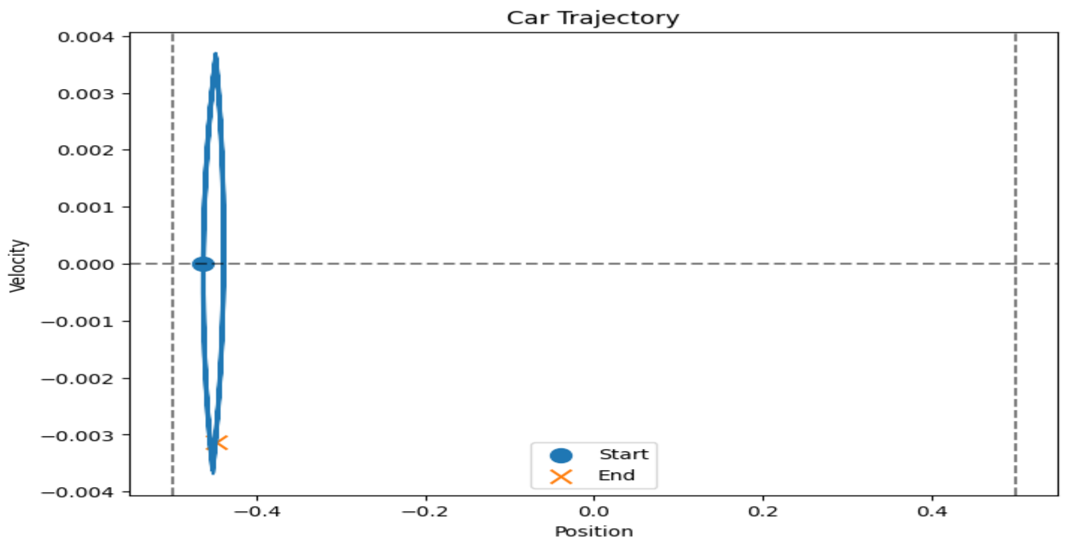


Q-learning Metrics:
Average Reward: -199.975
Max Reward: -175.0
Min Reward: -200.0
Std Reward: 0.7901740314639555

-Optimal Action



-Trajectory

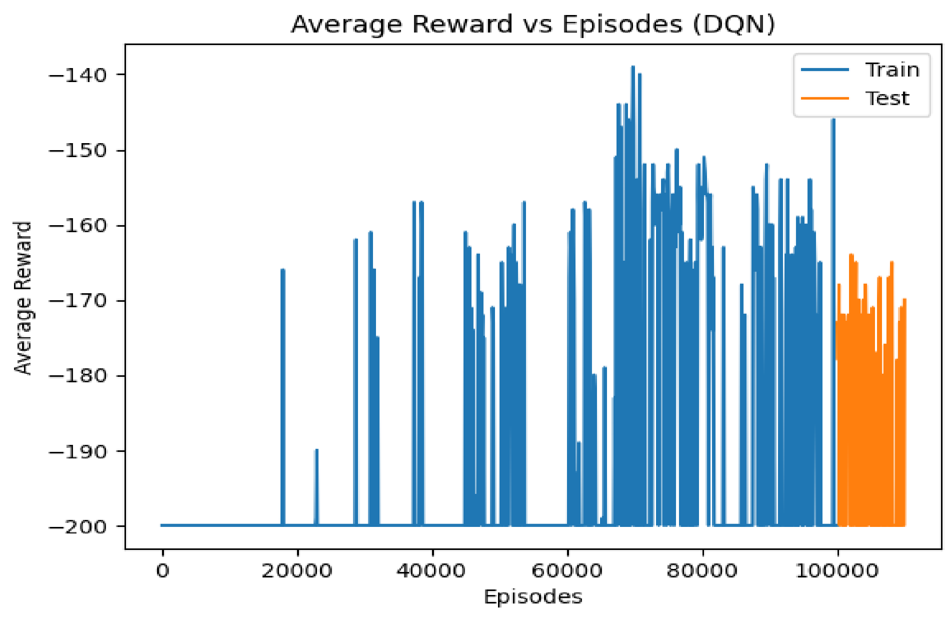


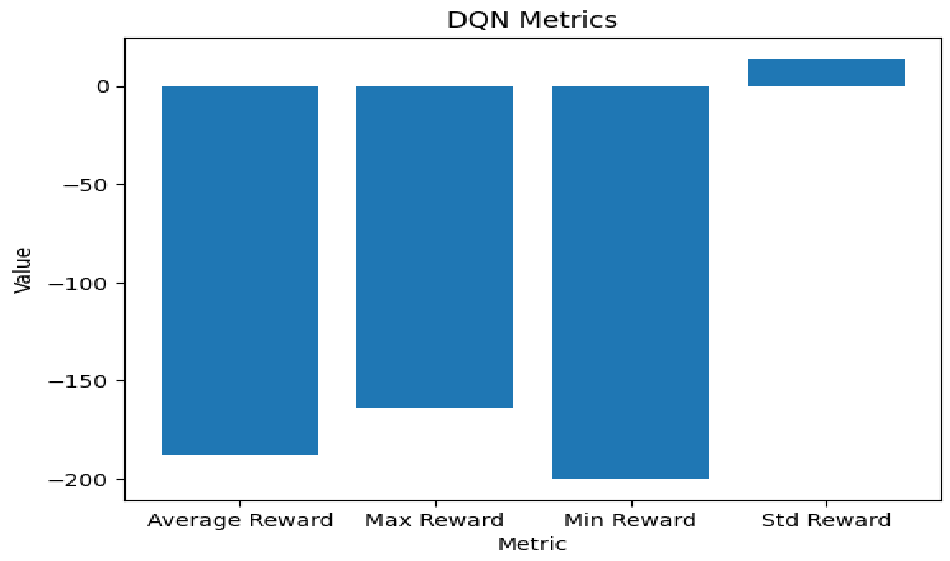
Observation: This trial is to get the best hyperparameter using grid search. The rewards were better than the previous trials. However, the agent was stuck in a loop and never actually learnt.

DQN

1st Trial: (small number of episodes)

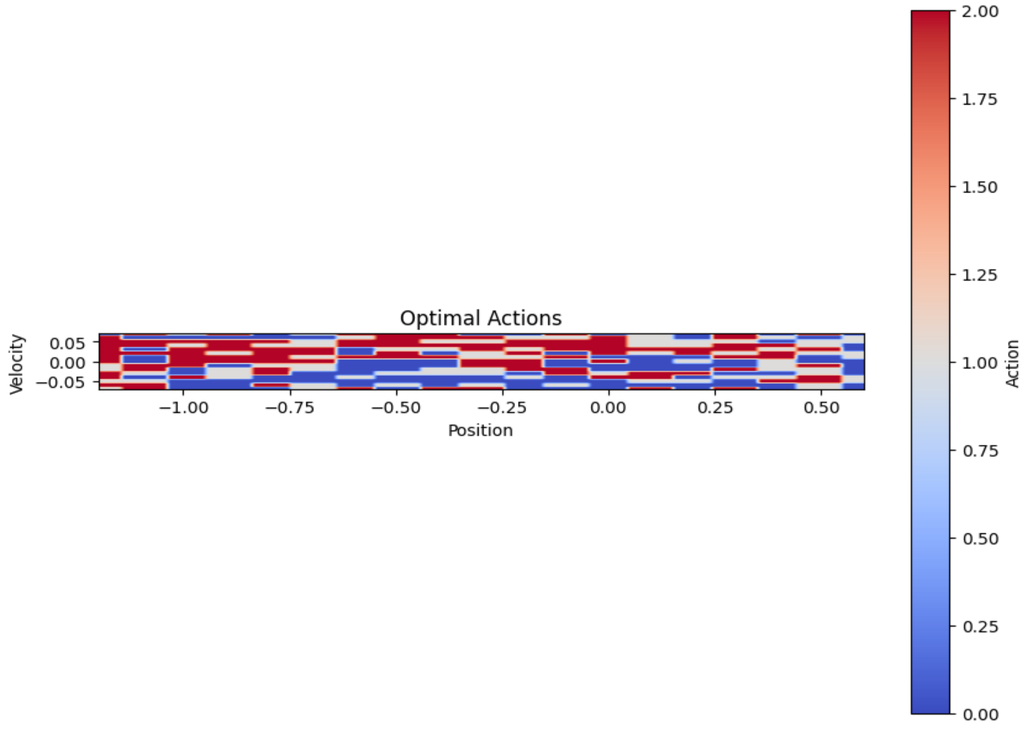
- Cumulative Rewards



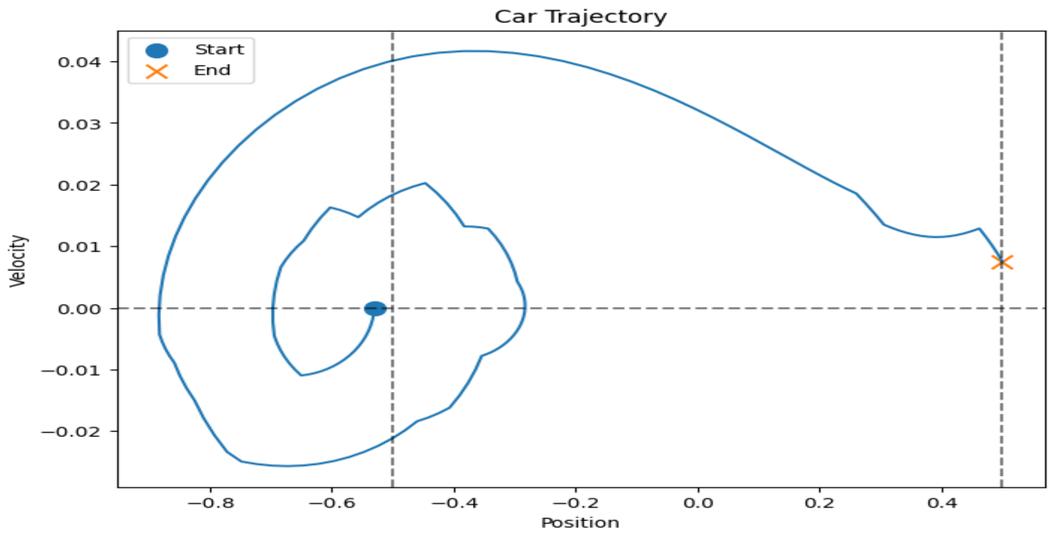


DQN Metrics:
Average Reward: -188.13
Max Reward: -164.0
Min Reward: -200.0
Std Reward: 13.862651261573307

-Optimal Actions



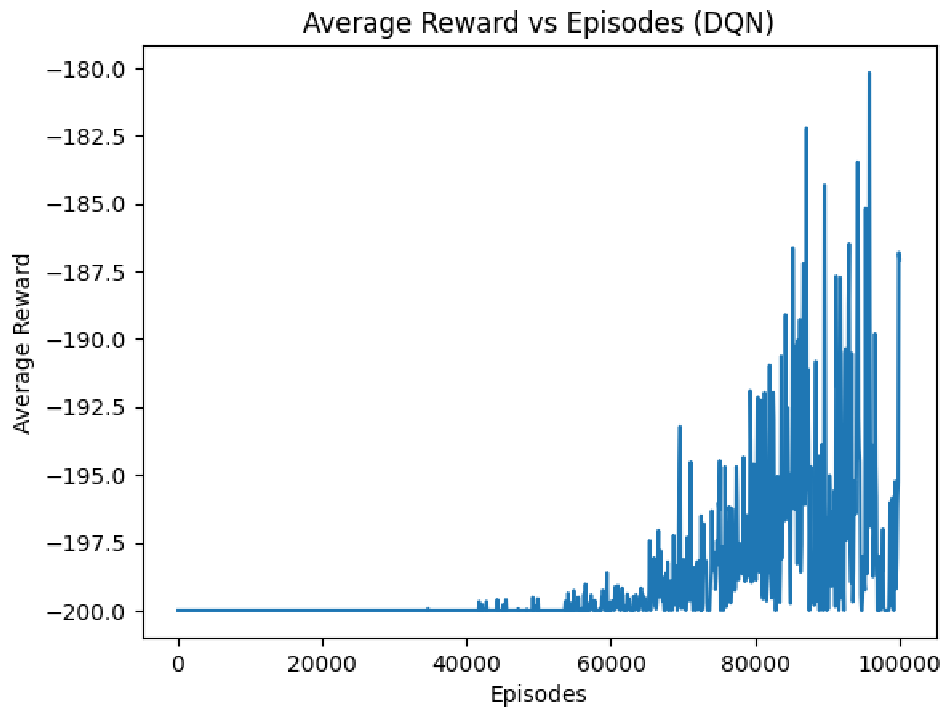
-Trajectory

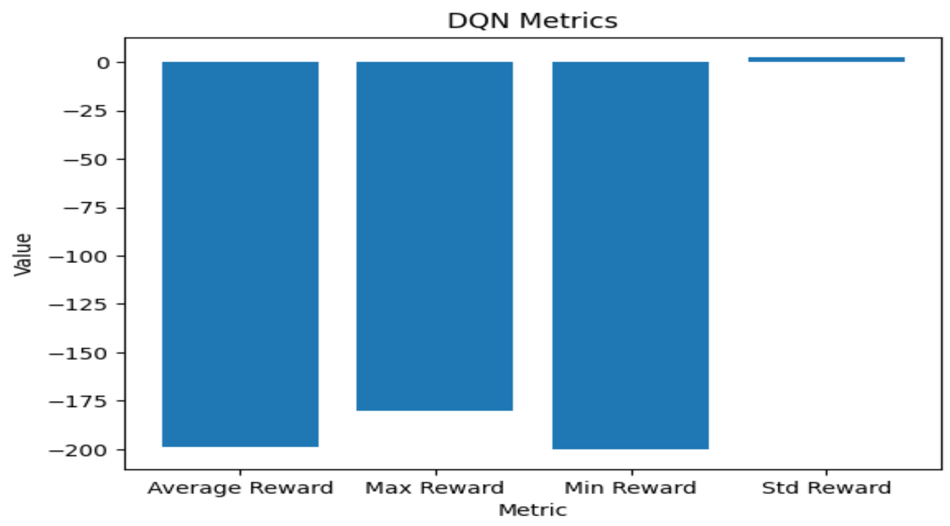


Observation: Rewards were not that good. However, the trajectory shows the success of reaching the mountain. The figure shows that 0.5 on x-axis the car reached the top at velocity of 0.01.

2nd Trial: (Higher number of episodes)

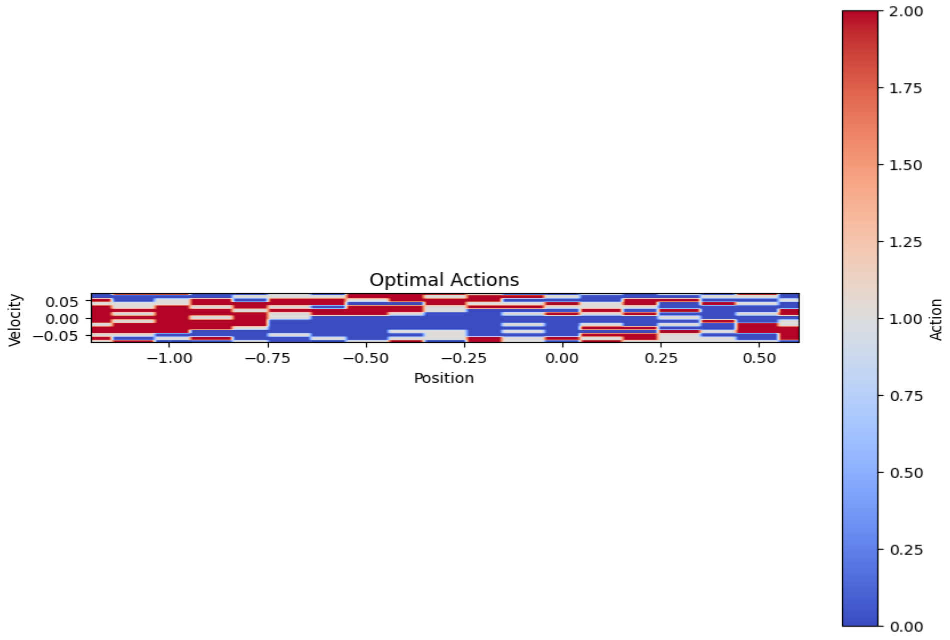
- Cumulative Rewards



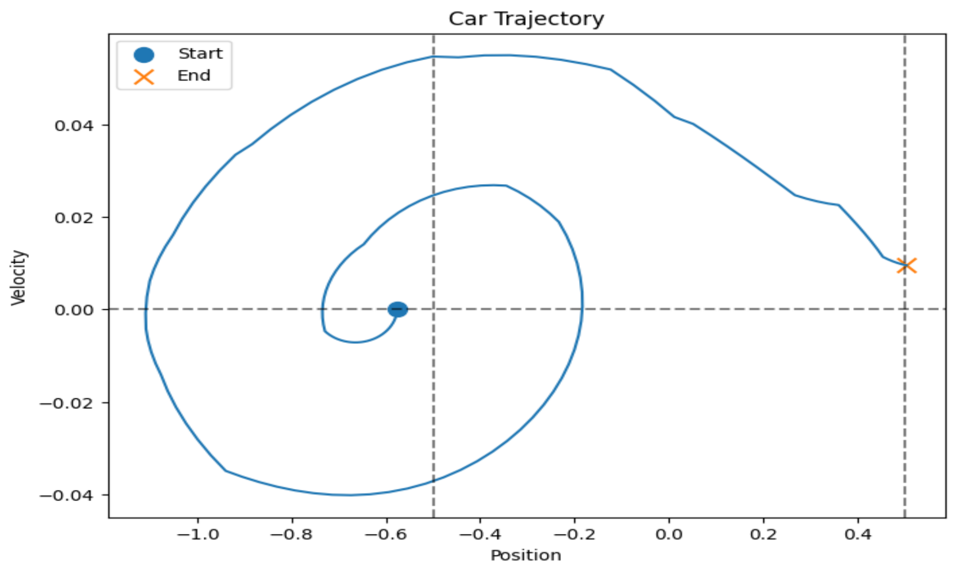


```
DQN Metrics:
Average Reward: -198.90596
Max Reward: -180.18
Min Reward: -200.0
Std Reward: 2.5125464927837657
```

- Optimal Actions:



- Trajectory:



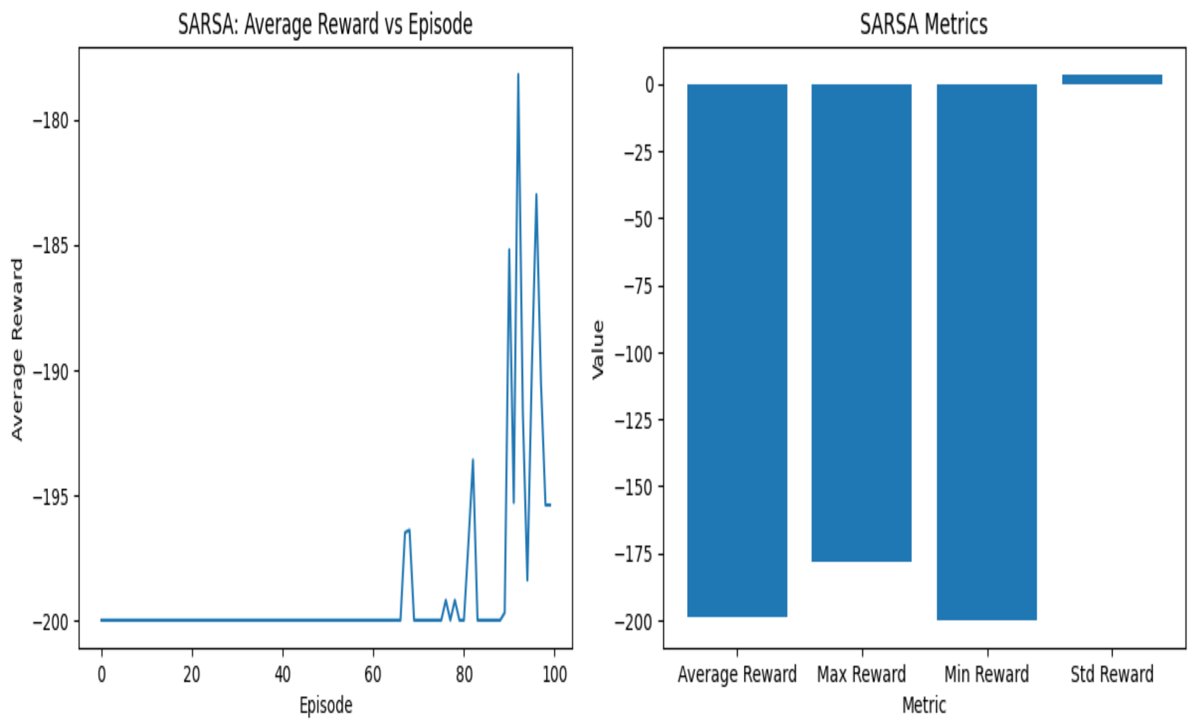
Observations: The rewards show a particularly satisfactory performance with a low variability in rewards std. Trajectory shows the success of reaching the mountain at position 0.5 and velocity around 0.01. This illustrates the effect of the higher number of episodes and confirms the accountability of DQN in this problem.

3rd and 4th trials: These were trials to use DQN with random search for hyperparameter tuning. The code crashed in both trials due to memory performance and the limitations of GPU on Colab.

SARSA

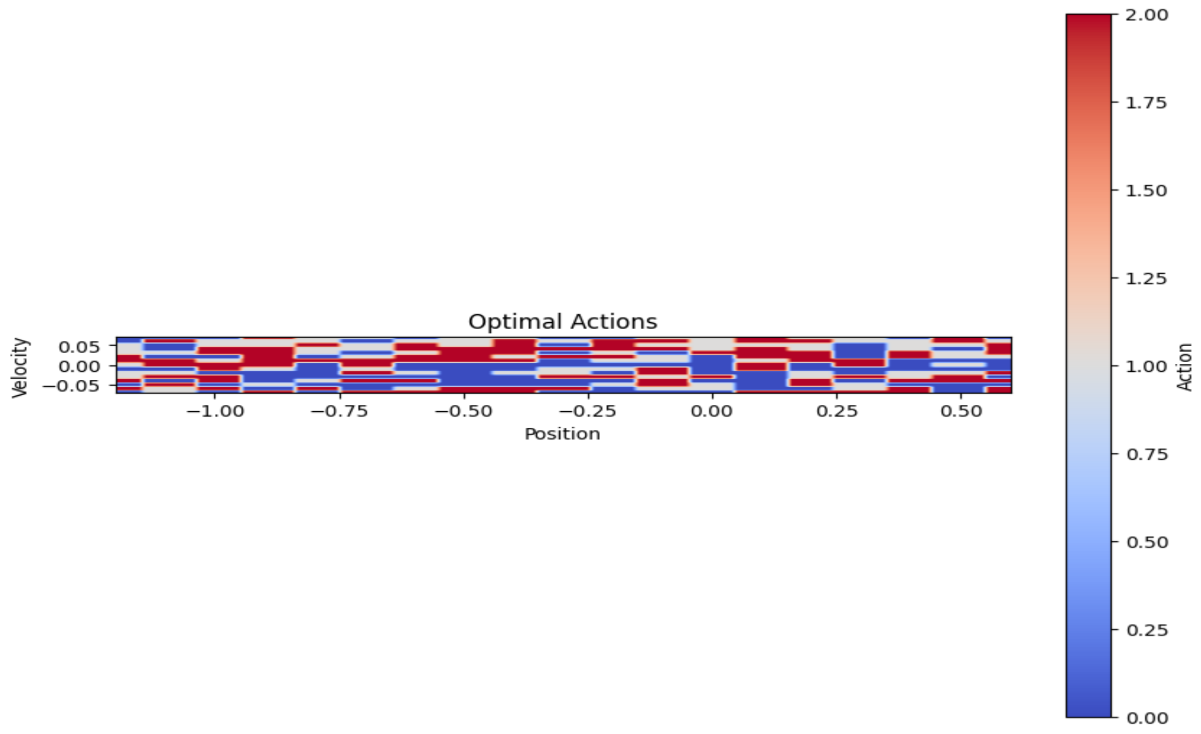
1st Trial: (small number of episodes)

- Cumulative Rewards

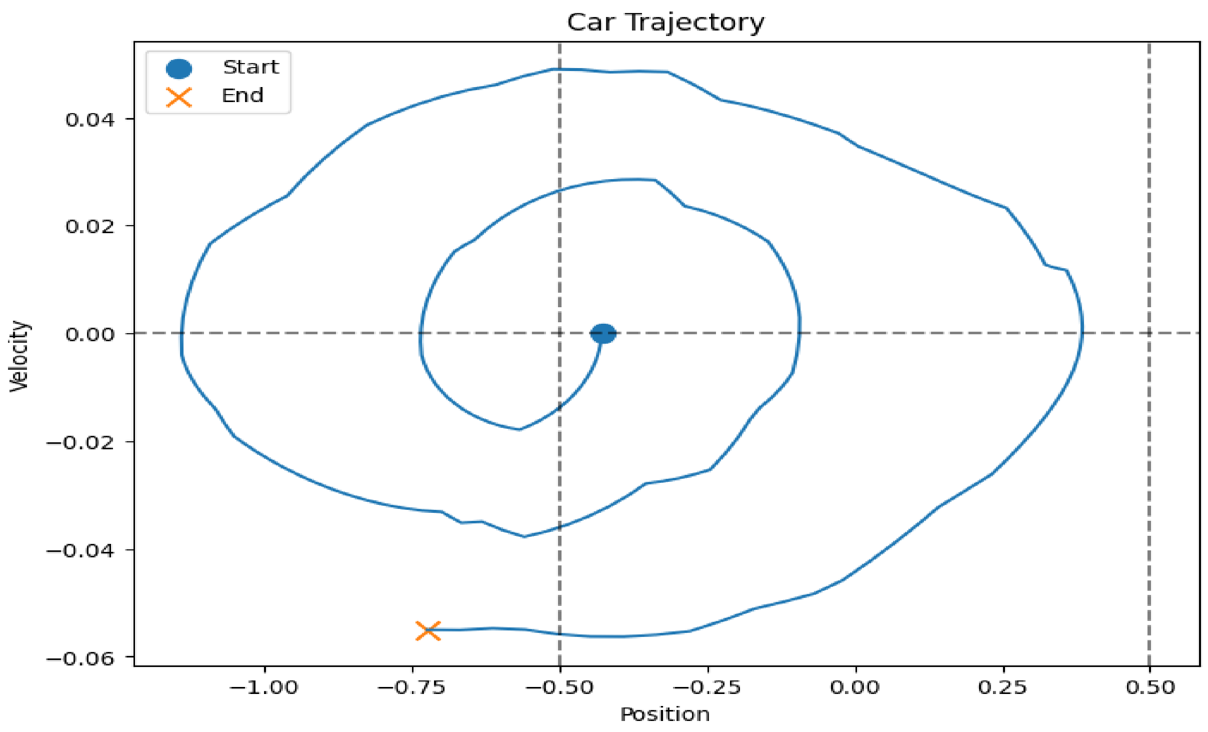


```
SARSA Metrics:
Average Reward: -198.84500000000003
Max Reward: -178.2
Min Reward: -200.0
Std Reward: 3.5428060912220425
```

- Optimal Actions



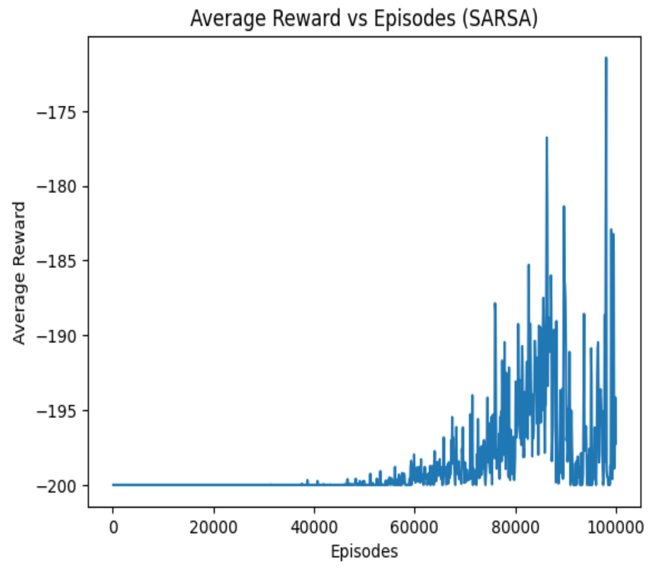
- Trajectory

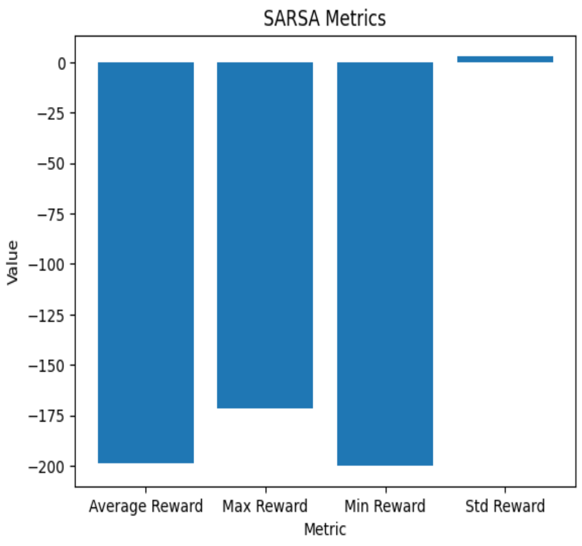


Observations: The performance rewards were good with good variability of rewards std. However, the trajectory shows failure to reach the mountain top.

2nd Trial: **(Higher number of episodes)**

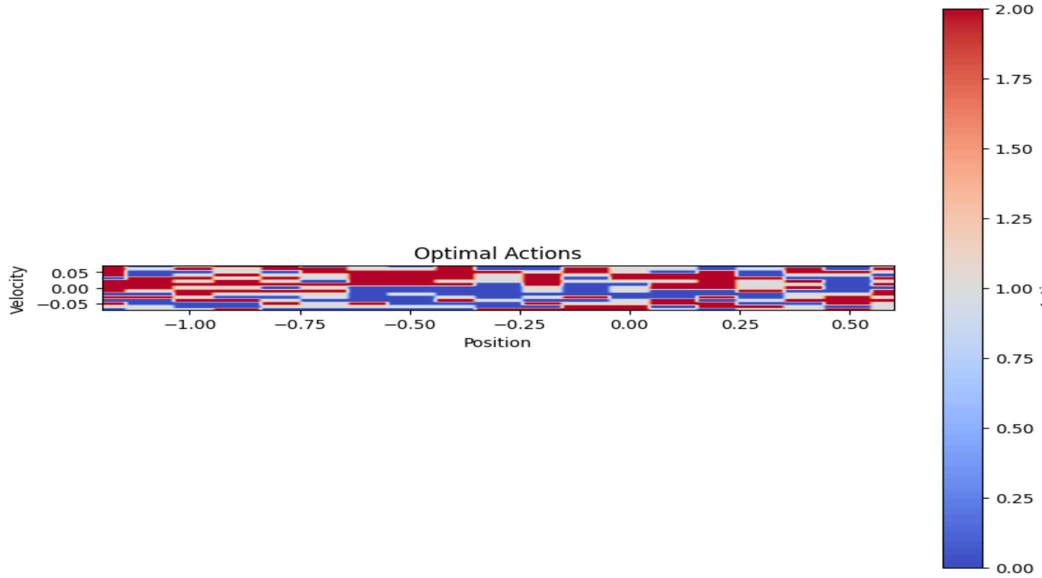
- Cumulative Rewards



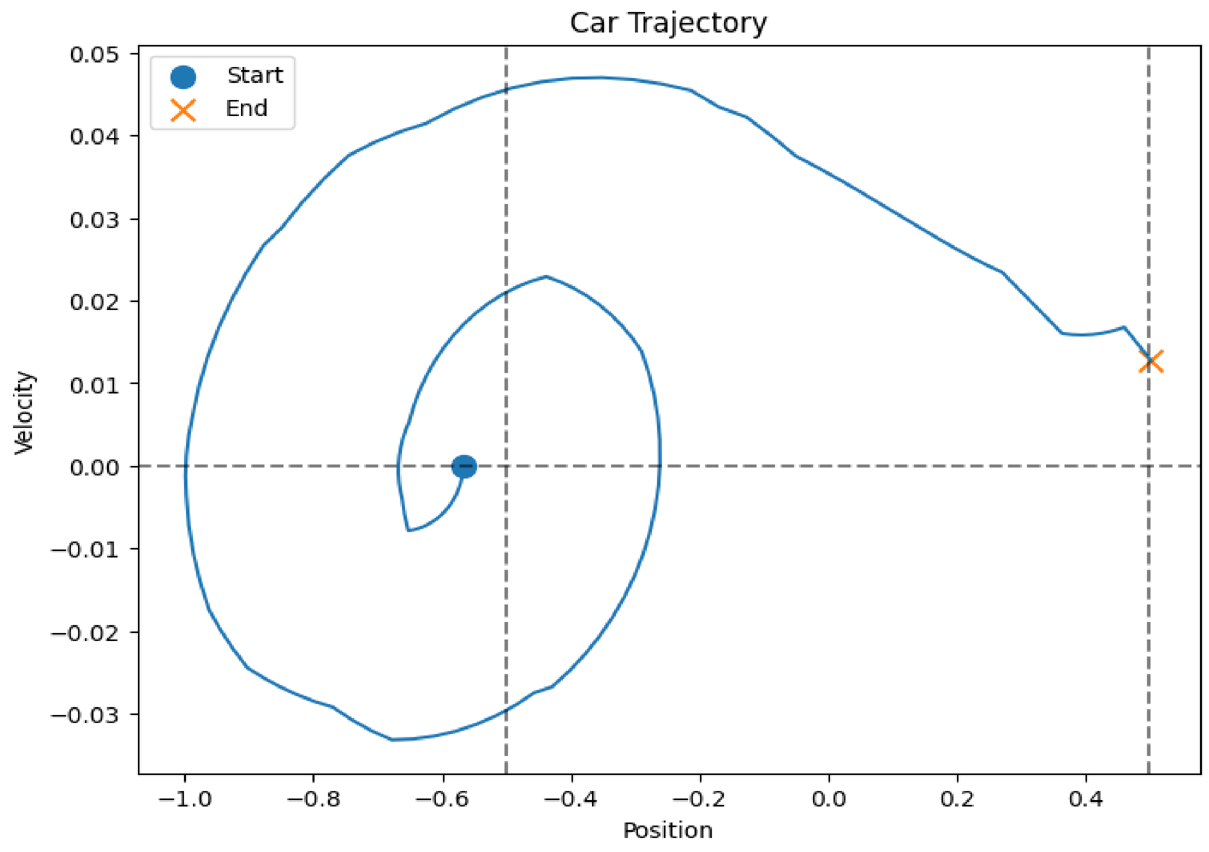


SARSA Metrics:
Average Reward: -198.61824
Max Reward: -171.44
Min Reward: -200.0
Std Reward: 3.048464318702123

- Optimal Actions:



- Trajectory



Observations:

The performance rewards were satisfactory, and the std variability is satisfactory, and the agent reached the top of the mountain at position 0.5 and velocity 0.01.

Performance Summary:

	Rewards STD	**Trajectory**	**Number of Episodes**
Q-Learning 1st trial	6.916	Failed	Small
Q-Learning 2nd trial	4.133	Failed	High
DQN 1st trial	13.862	Succeeded	Small
DQN 2nd trial	2.512	Succeeded	High
SARSA 1st trial	3.542	Failed	Small
SARSA 2nd trial	3.048	Succeeded	High

The lower reward variability observed in the DQN algorithm can be attributed to its ability to effectively approximate the Q-function using a neural network. The DQN algorithm utilizes a deep neural network as a function approximator, which enables it to handle complex state-action spaces more efficiently. This capability allows DQN to converge to a more stable and optimal policy, resulting in a narrower range of reward values.

On the other hand, SARSA algorithm exhibited higher reward variability, indicating a greater exploration-exploitation trade-off during the learning process. SARSA updates its Q-values based on the next action chosen by the current policy, which leads to more exploration of the state-action space. This exploration can result in more diverse rewards, but it may also lead to longer convergence times and suboptimal policies.

Q-learning showed intermediate reward variability, striking a balance between exploration and exploitation. Q-learning updates its Q-values based on the maximum Q-value of the next state, effectively selecting the greedy action. This approach tends to converge faster than SARSA but may overlook certain exploration opportunities.

Conclusion:

This project aims to apply RL algorithms to the MountainCar problem and analyze their performance in terms of convergence speed and optimal solutions. By doing so, we hope to gain insight into the effectiveness of different RL algorithms and their hyperparameters in solving this problem. After testing the 3 algorithms and making a trial for each with different number of episodes, DQN 2nd trial showed the best performance with respect to rewards variability and trajectory success. After that, SARSA 2nd trial showed the 2nd best performance with respect to rewards variability and trajectory success. What is common between them both was the high number of episodes. Last was the DQN 1st trail, trajectory was successful, however, variability was higher than the previously stated trials.

Lesson learned: DQN and SARSA show better performance than Q-Learning as they provide more stable performance than Q-Learning. Also, the higher the episodes number the better performance achieved.

References:

[1] Hayes, G. (2019, February 22). Getting Started with Reinforcement Learning and Open AI Gym. Towards Data Science. Retrieved from https://towardsdatascience.com/getting-started-with-reinforcement-learning-and-open-ai-gym-c289aca874f ↗

[2] Solving the Mountain Car Problem Using Reinforcement Learning. (2017, December 2). Retrieved from https://www.youtube.com/watch?v=KzsBaqYzNLc ↗