CS 352 - Introduction to Reinforcement Learning

Course Project Presentation

Impact of Learning Rate in Q-Learning

Mohammed Haider Abbas
Muhammad Ahsan

Overview

- Q-Learning
- Taxi Problem
- Our Objective
- Results
- Conclusion
- GitHub

Q-Learning Algorithm

Q-Learning

- Q-Learning is a reinforcement learning algorithm that learns to make decisions by maximizing a reward signal.
- The goal of Q-Learning is to find an optimal policy for a given environment that maximizes the expected cumulative reward.
- The algorithm uses a Q-Table to represent the state-action values, where each entry corresponds to the expected reward for taking a particular action in a particular state.
- Q-Learning uses an iterative update rule to update the Q-Table based on the observed rewards and transitions in the environment.
- The update rule involves a learning rate (α), which controls the extent to which the agent updates its estimates based on new information.
- Q-Learning is an off-policy algorithm, meaning that it learns about the optimal policy while following a different policy, usually an ε-greedy policy to balance exploration and exploitation.
- Q-Learning has been successfully applied to a wide range of problems, including games, robotics, and control systems.

Taxi Problem

The Taxi Problem

- The Taxi Problem is a classical Reinforcement Learning problem
- Serves as a benchmark problem in Reinforcement Learning and is often used to test the performance of Q-learning and other RL algorithms such as SARSA.
- The goal of the agent is to pick up the passenger and deliver them to the destination
- There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.
- There are six available actions: LEFT, RIGHT, UP, DOWN, PICKUP and PUTDOWN
- Each step taken by the agent results in a reward of -1
- Upon successful delivery, the agent receives a reward of 20
- Performing illegal actions such as attempting to PUTDOWN without a passenger will result in a penalty of -10

Our Objective

Our Objective

- We focus on the impact of the learning rate parameter, α, on the performance of Q- learning.
- We compare the performance of Q-learning for various values of α. Specifically, we evaluate the impact of α on two key performance metrics: computational efficiency, i.e., the time taken to converge, and maximizing reward, i.e., the average reward obtained by running the learned policies for various starting states.
- We have implemented the Q-learning algorithm on the Taxi learning problem using OpenAI's Gym Toy Text Taxi environment.
- The α value that results in the highest average reward should be identified to determine which α value is more effective at maximizing reward.

Results

Results

- The choice of α had a significant impact on the agent’s performance.
- Choosing an appropriate value for α is important for the Q-learning algorithm to converge to the optimal policy.
- If α is too low, the agent may be slow to learn hence may require more time to converge or fail to converge.
- If α is too high, the agent may overreact to new information hence may require more time to converge.
- The learning rate at which we are getting the maximum reward is in the range of 0.4 to 0.6, mostly at 0.5, and at this range the agent is taking the least time to converge while maximizing reward.

Findings

- For low and high values of α, the agent did not maximize reward and was taking more time to converge.
- For α in range 0.4 to 0.6, mostly at 0.5, the agent is maximizing reward while taking least time to converge.
- $0.1 \leq \alpha \leq 0.9$ with increments of 0.1

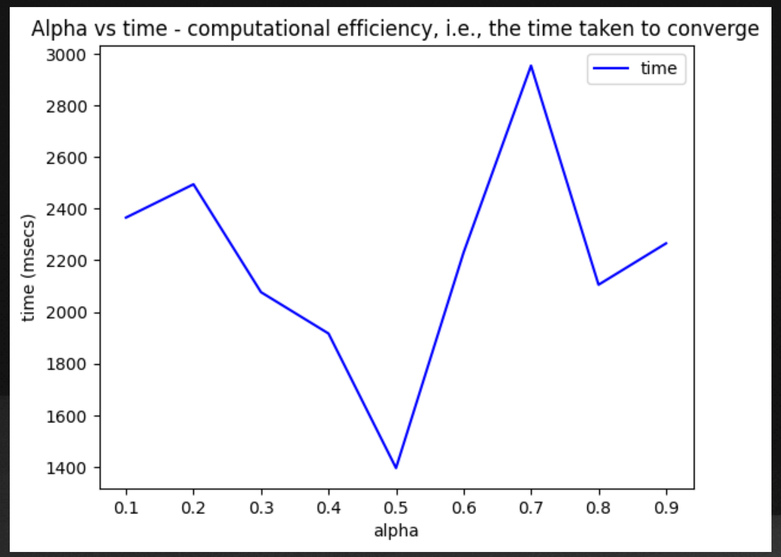


Figure 1

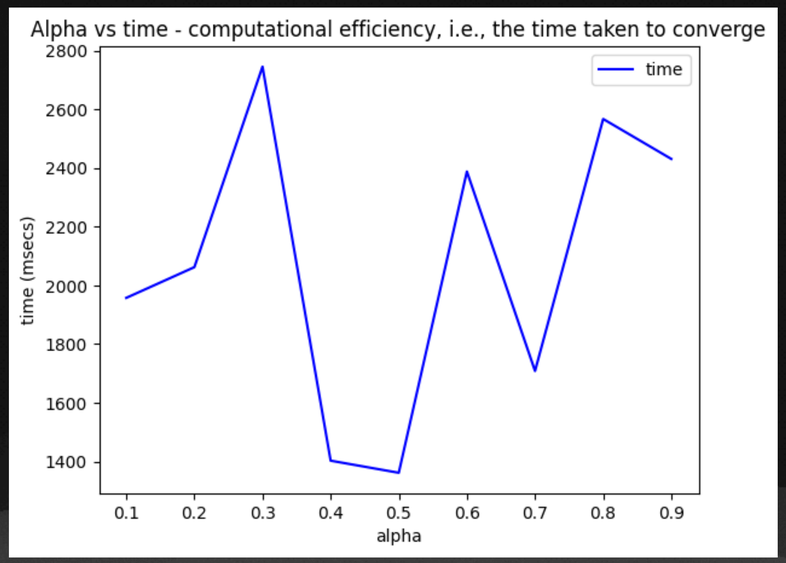
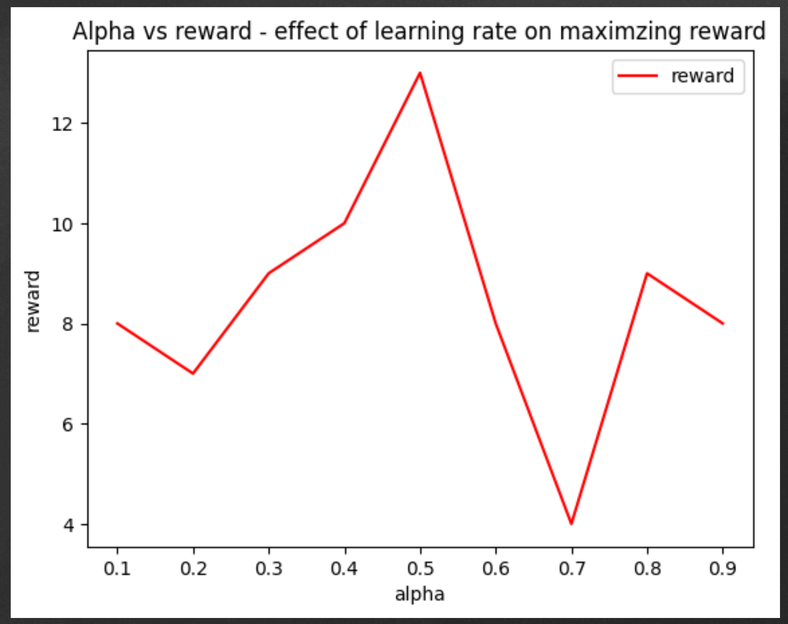
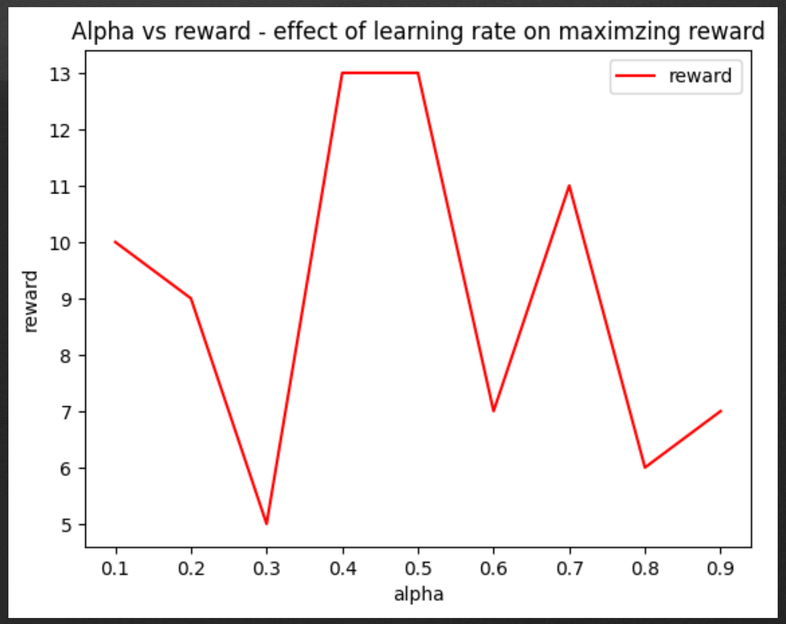


Figure 2





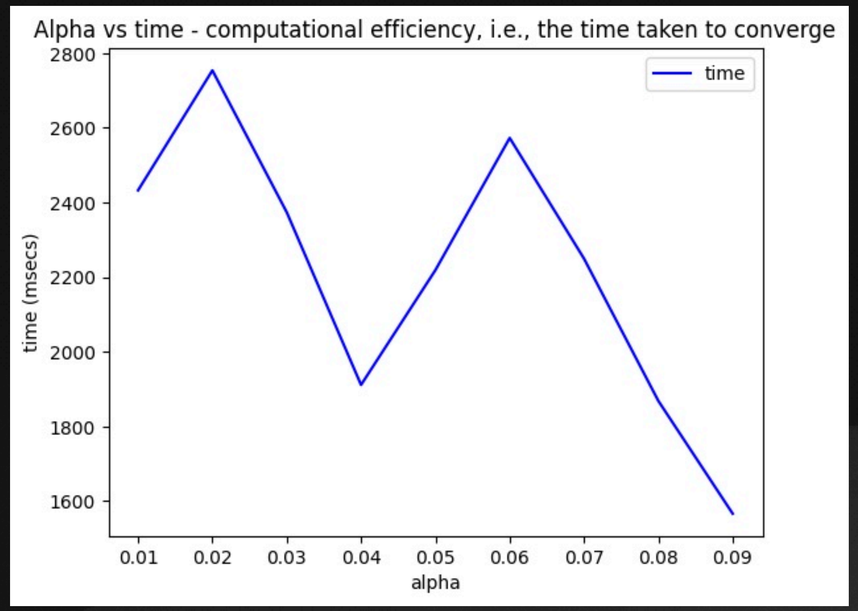
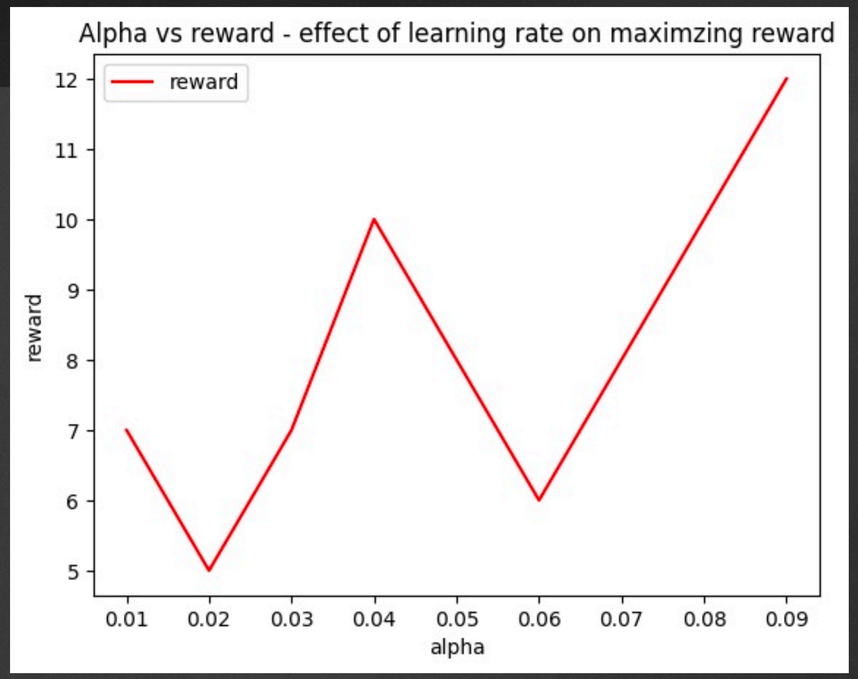


Figure 3



Very low or very High α values

- For cases such as very low or very high α values, the time taken to converge was higher than normal range of α values
- Shown plot is for very small alpha values ranging from 0.01 till 0.09 with increments of 0.01
- Here we can also notice that as we approach to 0.09, we are maximizing reward and getting less time to converge. This supports our stance that for very small α values, the agent will not maximize reward and will take longer to converge but as 0.09 is close to 0.1 which is relatively a larger value than all of the α values in the range therefore it will maximize reward and take least time to converge.

For $\alpha = 0$ or $\alpha = 1$

- For α = 0, our agent was failing to converge since it the program kept running which tells it did not converge. All runs for α = 0 were aborted after running for 5 minutes.
- For α equal to 1 our test went as expected and it took more time to converge and did not maximize reward, as shown in plots below.
- Below plot also highlights our finding that at 0.4 ≤ α ≤ 0.6 we are getting maximum reward and least time to converge.

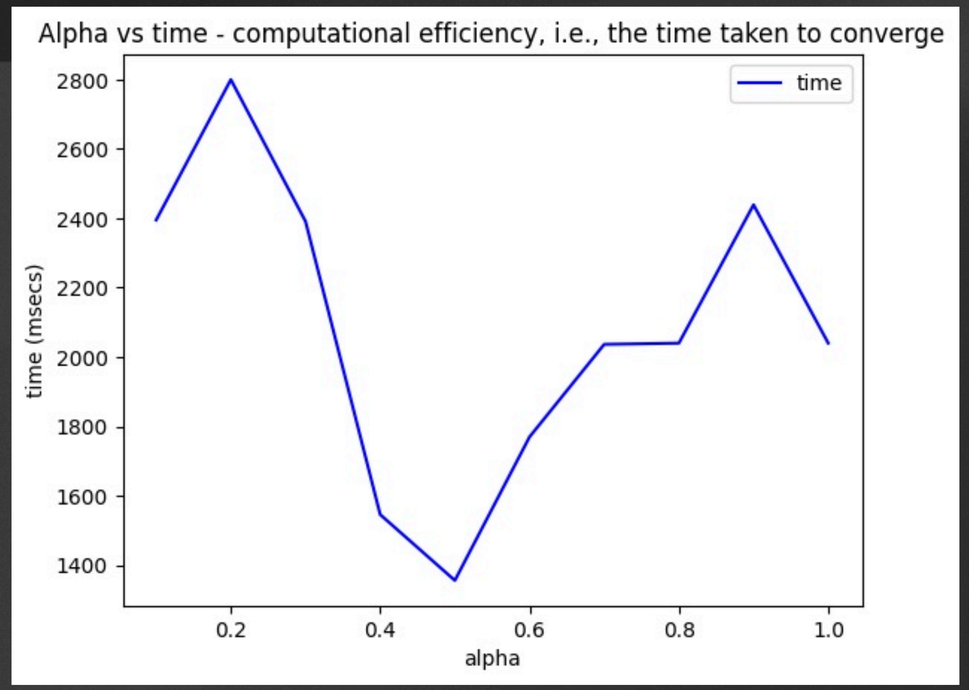
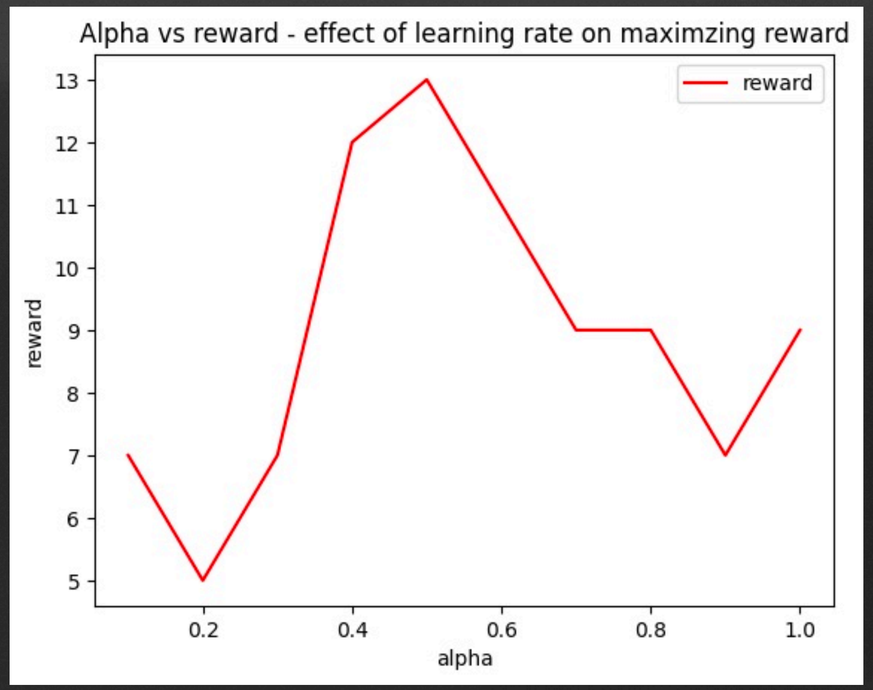
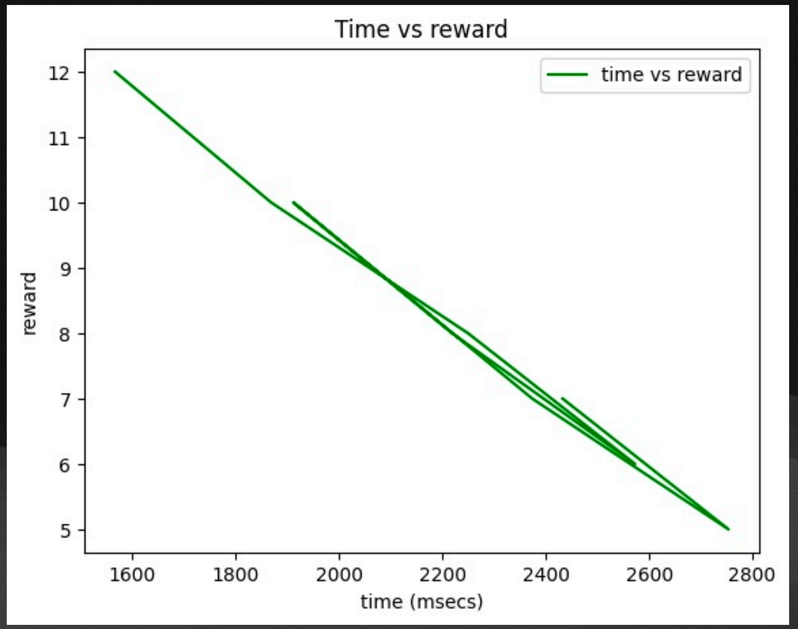


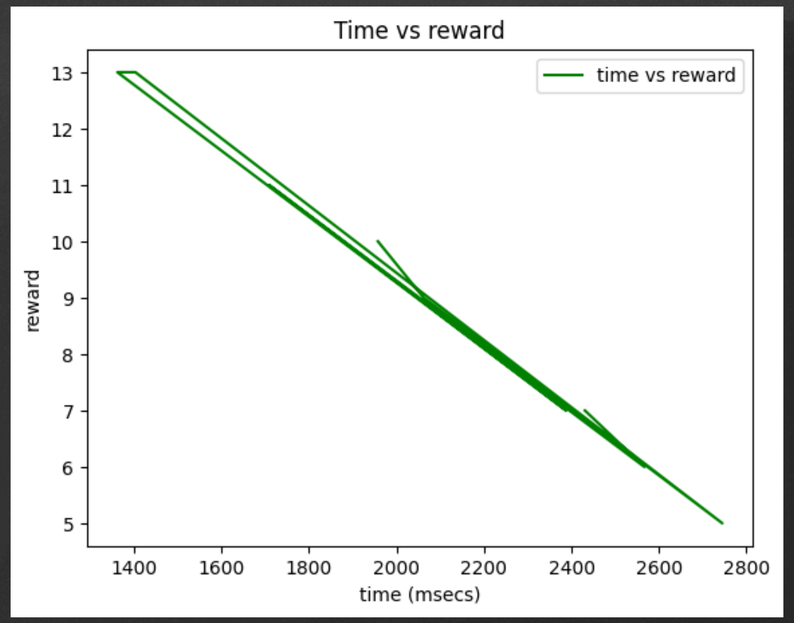
Figure 4

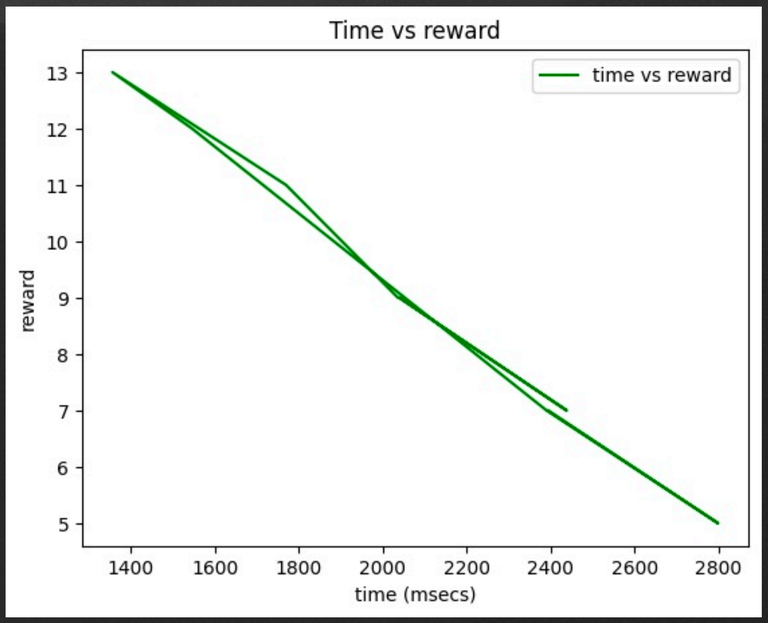


Relationship between Computational efficiency and Maximizing Reward

- Maximizing reward is inversely proportional to computational efficiency
- In other words, the time taken to converge is inversely proportional to the reward obtained
- An agent takes less time to obtain maximum reward and vice versa







Conclusion

Conclusion

- We analyzed the performance of Q-learning algorithm for various values of the learning rate parameter, α.
- We evaluated the performance of agent in terms of computational efficiency and the ability to maximize rewards.
- By selecting an appropriate α value within a certain range, we can optimize the performance of the Q-Learning algorithm
- For low and high values of α, the agent did not maximize reward and was taking more time to converge.
- Optimal range of α where it gave maximum reward and was computationally efficient i.e. faster convergence was in the range of $0.4 \leq \alpha \leq 0.6$ with the most optimal value of α being at 0.5
- Maximizing reward is inversely proportional to computational efficiency

GitHub Repository:

https://github.com/haider-06418/RL-Project

Thank you!