Course: Introduction to Artificial Intelligence

Spring 2025

Institute of Business Administration, Karachi

Assignment 3

Q-learning for a Gymnasium Environment

Introduction: In this assignment, you will implement Tabular Q-Learning algorithm to learn the Gymnasium environment of Taxi. Go through this tutorial to learn about the OpenAI's gym.

https://www.datacamp.com/tutorial/reinforcement-learning-with-gymnasium

The Taxi environment is a 5x5 grid representing city streets. Your agent controls a taxi that needs to pick up a passenger from one of four specific locations (let's say a bank, a post office, a school, or a cafe) and drop them off at another of these locations. The taxi can move in four cardinal directions, pick up the passenger if at the correct location, and drop off the passenger if at the correct destination. Each move costs a small penalty, a large reward is given for a successful drop-off, and attempting invalid pickups or drop-offs incurs a larger penalty. The agent's task is to learn the optimal sequence of actions to minimize travel time and maximize rewards by efficiently transporting passengers to their destinations within this discrete state and action space. Full details of the environment is available here:

https://gymnasium.farama.org/environments/toy_text/taxi/

You will use the Tabular Q-Learning algorithm to learn the optimal policy for navigating through the Taxi environment. The Tabular Q-Learning algorithm works by maintaining a Q-Table, where the rows represent the states and the columns represent the possible actions. The values in the matrix represent the expected future rewards for taking a particular action in a particular state.

Part 1 (baseline):

Methodology:

Setting up the Environment: First, you will import the necessary libraries and create the Taxi environment. The environment has a 5x5 grid world.

Defining the Q-Table: Next, you will define the Q-Table with a size corresponding to the Taxi environment's state space (500 states) and action space (6 actions). The rows represent the states (taxi location, passenger location, destination), and the columns represent the actions (move north, south, east, west, pickup, dropoff). Initially, the values in the Q-Table will be set to zero.

Implementing the Q-Learning Algorithm: You will use the Q-Learning algorithm to update the Q-Table values based on the rewards obtained by the agent in each state. The Q-Learning algorithm works by selecting the action with the highest Q-value in the current state and updating the Q-Table based on the reward obtained and the maximum Q-value in the next state. The full algorithm is given in the lecture slides.

Running the Simulation: Finally, you will run the simulation for a fixed number of episodes (e.g., 5000 episodes) and test the agent's performance in the environment (for 100 episodes). You will plot the rewards obtained by the agent in each episode to visualize the learning process. Also, report the average reward over 100 testing episodes. Use a learning rate (α) = 0.8 and a discount factor (γ) = 0.95. In the ϵgreedy policy, the initial ϵ is 1, and after 2000 episodes, reduce ϵ by a factor of 0.999 after each episode (so $\epsilon=\epsilon*0.999$), but ϵ should not go below 0.01.

Results: Your Tabular Q-Learning algorithm should successfully learn a good policy for navigating the Taxi environment. The agent should learn to pick up passengers from their designated locations and drop them off at their correct destinations, minimizing the number of steps and penalties. The rewards obtained by the agent in each episode should show an increasing trend, indicating that the agent is learning and improving its performance. Your plot should show an upward trend in rewards over the training episodes.

Also, the average reward in the testing episodes should be a positive value close to the maximum possible reward per episode (which is +20 for a successful drop-off minus the cost of the steps taken). This will show that the agent has learned to act effectively in this domain. Briefly report your results in a document.

Part 2 (hyperparameter tuning):

Improve the performance of your algorithm by changing the values of the learning rate (α), discount factor (γ), and ϵ in the ϵ-greedy policy, as well as the number of episodes with the initial value of ϵ. Find the values of these hyperparameters that will reduce the training time (in terms of episodes) while maintaining a high average reward in the testing episodes. Compare your results with the baseline results you found in Part 1. Briefly report your findings in the document used for Part 1.

Part 3 (domain modification):

Repeat the procedure explained in Part 1, but this time, consider potential modifications to the environment or the reward structure (if you wish to explore further, go through the original Taxi environment code here and identify aspects of it which can be modified). Report your findings, including the hyperparameter values that result in the highest average reward in the testing episodes in your modified domain. You might need to adjust the number of training episodes and the ϵ decay schedule to achieve good performance.

Marks: 20 marks in total. 10 for Part 1, 5 for Part 2, and 5 for Part 3.

File format for submission: Submit your code in a file with this name format taxi-q-learning-00000.py, replace 00000 with your roll number. Similarly, the report should be named as taxi-q-learning00000.pdf, where 00000 gets replaced by your roll number.