Analysis of Q-learning Exploration-Exploitation Trade-off in Taxi Problem

Sofia Teeriaho
s3887626
Artificial Intelligence
s.teeriaho@student.rug.nl

Mohamed Sherif Osama Gamil
S3897605
Artificial Intelligence
m.gamil@student.rug.nl

Abstract

Reinforcement learning and specifically Q-learning can be seen to be a prominent candidate in optimising decision-making while maneuvering a partially observable environment. This could be credited to it being capable of comparing future actions and states and update its knowledge based on previous experiences before making each step. We employed Q-learning to analyze its effect on the performance of an agent in the Taxi Environment. Using the epsilon-greedy exploration method, we were also able to have an insight into the exploration-exploitation trade-off faced by the agent in the environment.

1 Problem Description

1.1 OpenAI Taxi Environment

For our project we decided to use OpenAI's Gym environment to model the Q-learning algorithm. More specifically we use 'Taxi-v3' environment to illustrate how we can apply reinforcement learning in combination with different exploration/exploitation methods. The taxi problem is episodic meaning the taxi is placed in a different state for each experiment and is given a specific amount of steps it is allowed to take each experiment to reach its goal.

1.1.1 States

The environment is a 5x5 table, meaning there are 25 squares the taxi can exist on at any time. Four of these squares are specified locations. Each experiment, two of these locations are randomly assigned as pickup and drop-off locations. There is a passenger at the pickup location that should then be transported to the drop-off location. The experiment comes to an end when the taxi drops off the passenger at the drop-off location or when the taxi has taken the maximum amount of time steps(actions). Given the amount of (specified)locations we can calculate the amount of possible states. Given that a passenger can exist in one of the four locations including inside the taxi, we can say there are 5 locations in total in which 4 of them are destinations. As can be seen in figure 1, the four destinations are represented with letters.

A letter is colored blue if that is the pickup location of the passenger. A purple letter represents the drop-off location. The yellow square represents the moving taxi, this changes to green once the taxi has a passenger on board, indicating the 5th possible location.

1.1.2 Actions

There are six primitive actions the agent can take: four navigational actions that move the taxi one square North, South, East, or West and a pick up and drop off action.

Preprint. Under review.

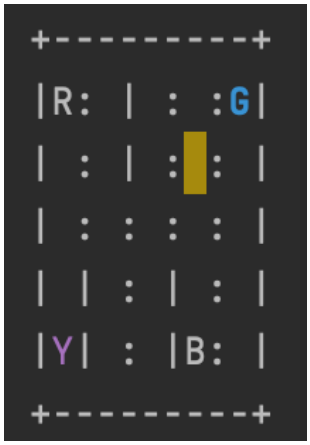

Figure 1: Taxi-v3 environment example

1.1.3 Rewards

These deterministic actions lead to different rewards. Each navigational action is worth -1 points, a successful drop-off action, meaning the agent has reached the goal is worth +20 points. An illegal drop-off or pickup action at the 4 destinations gives a penalty of -10 points (Molina et. al, 2020).

2 Q-learning

Q-learning is an off-policy control algorithm, which explores a non-deterministic environment using a trial and error approach and then chooses the best action based on previous knowledge. The trial and error phase allows for the agent to update a kind of mind-map of received rewards or penalties from previous actions. Storing knowledge from past experiences allows the agent to improve its performance in the future (Chin et. al, 2011).

In order to store learned knowledge of rewards and penalties a "Q-table" is used. This is a matrix pairing between every state and action. Each cell in the table refers to an action state pair and can be identified as a Q-value.

The Q-algorithm evaluates each Q-value in the Q-table. An agent will receive a reward or penalty for each taken action a in state s. When an agent chooses to exploit, the maximum q-value at state s is selected (Bellinger et. al, 2020). the Q-value is updated using the following reward-value function.

$$Q(S_t, A_t) ::= Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big].$$

2.1 Epsilon-greedy Selection

Given that Q-learning is an off-policy learning algorithm, it uses the epsilon-greedy exploration method to determine an agent's action according to the q-table of previous state-action pairs. The epsilon value determines the probability of exploiting, choosing the action with highest expected reward, or exploring, choosing a random action in the set of actions. Exploration allows the agent to gain more knowledge about its environment and gain higher rewards in the long run. Exploitation on the other hand just gives immediate rewards. However, the problem with this is that we can not guarantee that the same action or set of actions always give the highest estimated reward. Depending on the problem at hand, it is critical to assess the best epsilon value, this ensures the correct balance between exploration and exploitation. A smaller epsilon value results in more exploitation as opposed to a high ϵ that focuses more on exploration.

2.2 Learning Rate and Discount Factor

The learning rate (α) and discount factor (γ) influence how fast the algorithm is able to converge. Both the learning rate and discount factor range between 1 and 0. The learning value determines how much a new learned value will be taken into account. A higher learning rate results in the agent considering more of the currently gained knowledge compared to a low learning rate that acts more from previous knowledge. The discount factor helps determine the importance of future states and their rewards. A discounting factor closer to 1 means the algorithm values future gain, looking for high rewards in the long term rather than a low value that considers immediate rewards (Bellinger et. al, 2020).

Parameters	
experiments	1000
steps per experiment	100
ϵ-greedy values	[0.0, 0.05, 0.1]
learning rate (α)	0.8
discount factor (γ)	0.8

2.3 Algorithm outline

Algorithm 1 Q-learning (Sutton,2018)

Input: <nr of experiments, step size, epsilon>
Outout: Q-table
Initialize Q-table(states,actions) with 0
for each experiment **do**
    reset state
    total rewards = 0
    for each step in experiment **do**
        choose action from states using ϵ-greedy policy
        take action, get reward and new state
        $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a) - Q(s_t, a_t))$
        update state
        update total reward
    end for
    (terminate if goal is reached)
end for
return: Q-table

3 Results

3.1 Figure 2

Figure 2 shows the accumulated reward each experiment. To make the graph more clear, we avoided including the total reward of each experiment and instead took the average of every 10 reward values. For example, the average reward value from the 1st to the 10th experiment is -270. By looking at y-axis we can see that the reward values are mostly negative values, this is because the agent is given penalties more than rewards, see section 1.1.3.

Now looking back at the exploration-exploitation trade-off for the ϵ-greedy method, we know that the agent chooses the highest value when it chooses to exploit. We initialize the Q-table with zeros, this means that our agent goes through all the action-state pairs because it chooses the unexplored action-state pairs with a value of 0 since they are greater than -1 , which is what a state would receive after exploring a location for the first time one of the navigational actions. It determines which actions it should not take in order to reach the goal, this means learning early on that pickup and drop-off actions lead to larger penalties, therefore avoiding them when exploiting. We can observe from our

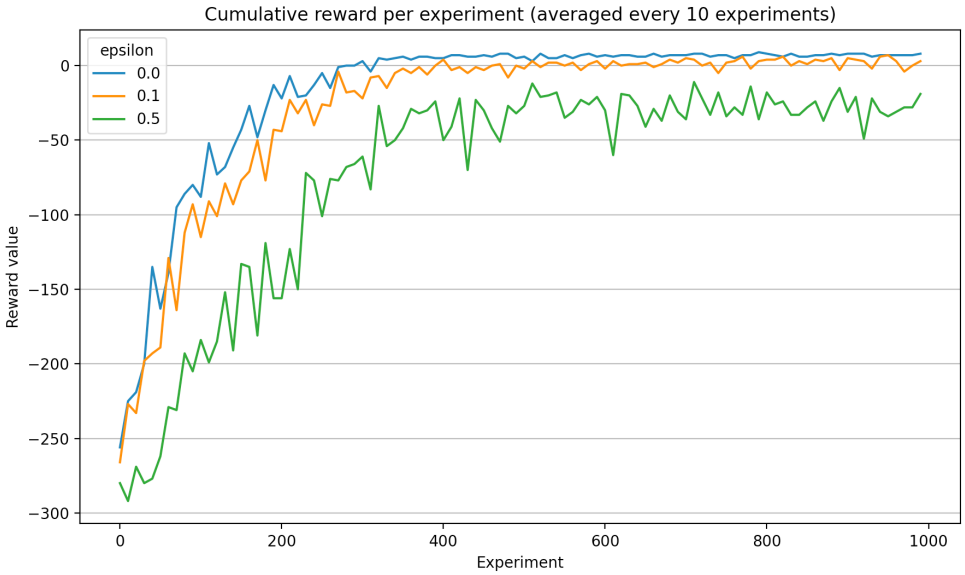

Figure 2: Total rewards averaged every 10 experiments

plot (figure 2) that an epsilon value of 0 reaches the highest cumulative reward compared to higher epsilon values. This is due to the fact that some degree of exploration allows for the agent to explore previous actions that have given a penalty rather than ignoring them completely or coming back to them once other states have been visited in the case of $\epsilon = 0.0$.

We also decided to include an epsilon value of $\epsilon = 0.5$ to observe how our taxi performs when it explores 50% and exploits 50%. We can observe that the cumulative reward value never reaches a positive reward, this is because our agent knows exactly what path to take to reach the given destination, however, it occasionally (in this case 50% of the time) chooses a random action, resulting in more visited states and a lower cumulative reward value. The more apparent fluctuations can be explained by the difference in the time in which the agent explores. If the agent explores early on then this means the majority of the 50% it should exploit is used in later time steps resulting in a higher reward while when it exploits early on it has to explore later in the time steps when it should actually be taking advantage of its previous knowledge rather than choosing arbitrary states that most likely will inhibit reaching the goal.

3.2 Figure 3

Figure 3 shows how many steps each experiment took. This plot demonstrates how the agent trains, by decreasing the number of steps it needs to take each experiment to reach the goal. We can see that approximately the first 50 experiments the agent most of the time reaches the limit of time steps, stopping before it can reach a goal. It is purely there to explore the environment and update the Q-table, the passenger does not get dropped off or picked up successfully. Compared to figure 2, we can see that this plot fluctuates much more. This can be explained by the randomness of the Taxi-v3 environment. Each experiment the starting position, pickup and drop-off location change, which means that the minimum amount of steps is independent from the training method. For example, the taxi might start off at the leftmost corner of the environment to then have to pickup and drop-off on the other side. This is also why it is harder to observe a difference between the epsilon values. Like the previous plot, the highest epsilon value performs poorly. It can not reach a the minimum amount of steps that can be taken to reach the goal. It takes alternative routes and makes "irrational" mistakes while exploring.

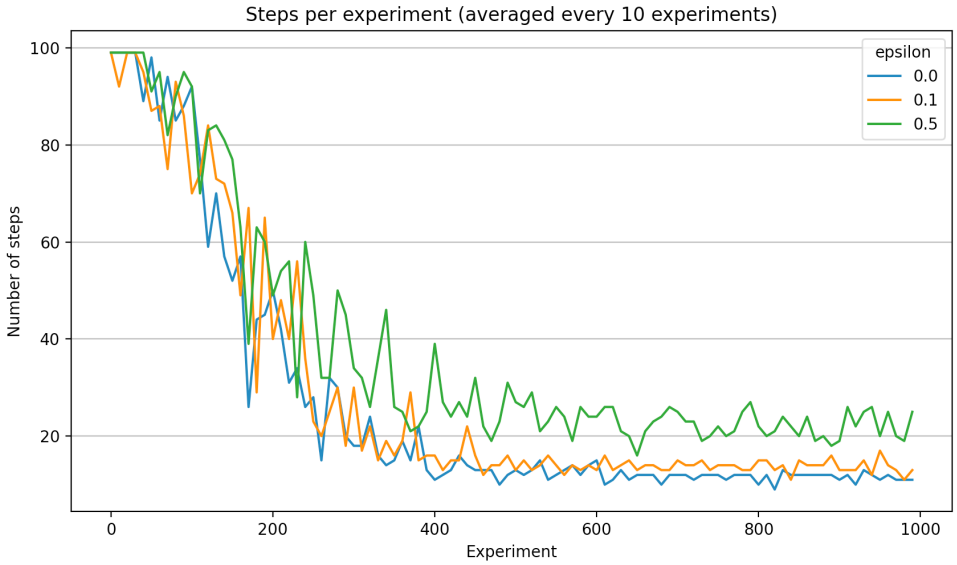

Figure 3: Total time steps averaged every 10 experiments

4 Discussion

The main aim of this experiment is to set a suitable domain to observe the effect of Q-learning on an autonomous agent in a realistic environment. It could be compared to a real life taxi driver with a goal of gaining knowledge while minimising the cost of exploration. However, other forms of Reinforcement Learning exist that could be tested in this setting. One example would be Deep Q-learning where, instead of a q-table, a neural network is created. The input nodes of this neural network represent the input states and the output node is a set of action paired with their Q-values. Implementing such algorithm in the same environment could give us a better insight in using reinforcement learning to solving to developing and optimising autonomous systems.

References

[1] Bellinger, C., Coles, R., Crowley, M., Tamblyn, I. (2020). Active Measure Reinforcement Learning for Observation Cost Minimization. arXiv preprint arXiv:2005.12697.

[2] Chin, Y. K., Bolong, N., Kiring, A., Yang, S. S., Teo, K. T. K. (2011). Q-learning based traffic optimization in management of signal timing plan. International Journal of Simulation, Systems, Science Technology, 12(3), 29-35.

[3] Molina, A. M., Avelino, I. F., Morales, E. F., Sucar, L. E. (2020). Causal Based Q-Learning. Research in Computing Science, 149, 95-104.

[4] Sutton, R. S., Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[5] Yang, H. (2021). An Independent Study of Reinforcement Learning and Autonomous Driving. arXiv preprint arXiv:2110.07729.

[6] Zhang, J. (n.d.). Tutorial: An Introduction to Reinforcement Learning Using OpenAI Gym. Gocoder.one. Retrieved January 28, 2022, from https://www.gocoder.one/blog/rl-tutorial-with-openai-gym