Abstract

Small changes in the balance between exploration and exploitation for an agent can have significant effects on their performance. We will investigate and examine how the exploration rate affects the mean cumulative reward and speed of learning of an agent in an environment, and thereby determine an optimal exploration rate. By creating an Epsilon-greedy algorithm to solve the 'Taxi Problem', we will compare the performance and speed of learning of the agent when it is trained using different exploration rates for a various number of training episodes. From our study, it was concluded that the optimal exploration rate for a particular amount of training was 30%, and that all Epsilon-greedy algorithms eventually learned the optimal policy no matter the exploration rate. Our findings can both be used to discuss how the algorithm we implemented could affect real-life taxi driving, and imply that the exploration rate has a minimal affect on the reward achieved by an agent after a certain number of training episodes.

Introduction

Q-learning is a reinforcement learning algorithm that finds the optimal next action for a given state, that yields the greatest reward. Tabular Q-learning is where the values obtained from the quality function are stored and updated in a table. When training an agent using Q-learning, it can either explore or exploit the environment. Exploring refers to an agent randomly choosing an action to gain more knowledge of the environment, whereas exploiting is where an agent repeatedly chooses the best option to its knowledge. It is best that an agent maintains a balance between its exploration and exploitation to receive the greatest possible reward. This is allowed by the Epsilon-greedy method, where epsilon refers to the exploration rate - how likely the agent is to explore rather than exploit the environment.

The purpose of this project is to answer the question: **"What is the optimal exploration rate of an agent that yields the greatest mean cumulative reward in a given environment?"**. This study was undertaken in order to apply theoretical knowledge to a simplified real-world situation and thus gain a greater understanding of reinforcement learning. For this investigation, the 'Taxi Problem' from open AI Gym, will be solved using tabular Q-learning to answer the research question. Our goal is to fully code and implement an Epsilon-greedy algorithm, which will allow us to solve the Taxi Problem and explore the effect of changing the number of training episodes on the mean cumulative reward, for various exploration rates. Furthermore, we will use our observations to examine how such an algorithm can be used to optimize real-life taxi driving.

Based on found literature and our intuition, our hypothesis is that after the agent has been trained for a relatively high number of episodes, all algorithms with different exploration rates will learn the optimal policy[1]. We predict this because the agent will have explored all possible actions by this point, and will know the optimal policy. Alongside this, we also predict that greater exploration rates will allow for a greater speed of learning, as it will explore the environment more often and will therefore gain knowledge at a higher rate. Hence, we predict that there will be an optimal exploration rate that optimizes exploration and exploitation.

Introduction to the Taxi Problem

In the 'Taxi Problem', a taxi (indicated by a red box) must find its way through a virtual environment to pick up a passenger from a randomly chosen location (indicated by a blue letter), and drop them off at another randomly

[1] Imad Dubbara, epsilon-Greedy algorithm, `https://imaddabbura.github.io/post/epsilon-greedy-algorithm/`

chosen location (indicated by a purple letter). Additionally, the environment contains walls (represented by solid lines) which the taxi cannot drive through. In order to optimize the taxi's route, every action taken gives a reward of -1. If the agent commits an illegal action, like drive through a wall or drop the passenger off at the wrong location, it receives a reward of -10. If the taxi successfully complete its mission, it receives a reward of +20.

Methods

We implemented tabular Q-learning by the Epsilon-greedy algorithm in Python. We researched the environment that open AI gym had created where the article titled "Getting Started with Gym"[2] was very insightful and helpful. In the Python script, we applied the quality function formula and set discount to be 0.9. This value was chosen as it seemed like a reasonable amount for the quality of the best action in the new state to be discounted by. In addition, the Q-table was initialized with zeros to ensure that the first action was randomized. The script that we wrote (See Appendix) allowed us to both train and test the agent, and automatically record results for all the exploration rates in our chosen range (from 0.1 to 0.9).

Sample size

To determine the sample size for the number of test episodes that the agent had to complete after being trained for a various number of episodes, we conducted a pilot study. The aim was to determine an expected sample variance. This was calculated to be 2.53. Using Definition 2.11[3] and estimating a 5% margin of error, the required sample size was calculated to be 200. Note that we estimated a small margin of error based on our pilot study and intuition.

Experiment 1

To test our algorithm and investigate whether a smaller exploration rate results in a greater mean cumulative reward, which is expected, we conducted an initial experiment. We set the number of training episodes to 5000, as we wanted to measure the overall performance after a significantly large amount of training episodes. We kept the discount at 0.9 and then recorded the mean cumulative reward for exploration rates = {0.1 ,0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. Afterwards, a graph of the mean cumulative reward as a function of the exploration rate was plotted. A sample of the training data set used can be seen below.

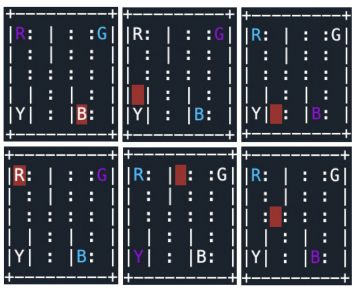

Figure 1: A sample of the agent's test training set. The grid is 5x5 and the red box is the taxi. The purple color indicates the drop-off location and the blue color indicates the pick-up location. The solid lines represent barriers.

[2]Gym, Getting Started with Gym, 2016, `https://gym.openai.com/docs/`
[3]DTU Learn, 2 Statistics, `https://learn.inside.dtu.dk/d2l/le/content/79546/viewContent/302976/View`

Experiment 2

To be able to determine an optimal exploration rate, we conducted another experiment. We increased the number of training episodes by intervals of 100, starting at 100, and recorded the mean cumulative reward. This process was repeated for epsilon = {0.1 ,0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9} and a graph of the mean cumulative reward as a function of number of training episodes for each epsilon value was plotted. The plot would provide insight into the speed of learning of each exploration rate as well as the agent's performance. A 95% confidence interval was then calculated for the number of episodes where there was a clear distinction between the data points. This would provide a statistical basis for whether there is an optimal exploration rate for this problem.

Results

Experiment 1

The graph we obtained after conducting Experiment 1 is given below. Note that this was for a very large number of training episodes, and therefore it is to be expected that a lower exploration rate would on average have a greater overall performance, and thus score a higher mean cumulative reward per run.

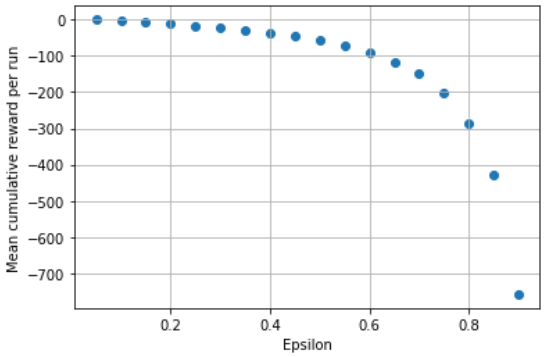

Figure 2: Mean cumulative reward per run for different exploration rates, for 5000 training episodes

The graph above shows that smaller exploration rates result in higher mean cumulative rewards per run. This will always be the case as greater exploitation on a large number of training episodes will yield a higher mean cumulative reward. The discovery is therefore quite trivial, but necessary. It suggested that the algorithm worked and was in line with our understanding of Q-learning.

Experiment 2

Figure 3 displays the results that we obtained after conducting Experiment 2. Note that the graph is scaled and starting from 400 episodes. This is due to the agent only being able to complete all 200 runs after at least 400 training episodes. To use this as a parameter of performance for all agents, results were only recorded in the range of 400 to 2000 training episodes.

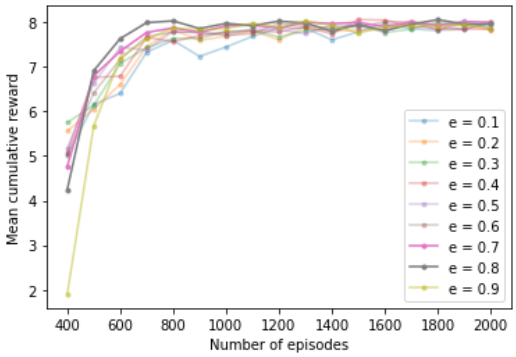

Figure 3: Mean cumulative reward per number of episodes for different exploration rates

Our results show that higher exploration rates clearly perform less successfully when the number of training episodes is relatively low. However, notice that when the agent is trained for approximately 1200 episodes or more, the data points for each epsilon value begin to converge. We noticed that for 400 episodes there was a clear difference in the mean cumulative reward for all of the different exploration rates. Hence, to provide a statistical basis for our observation, 95% confidence intervals were calculated. Given below in Table 1 is a table displaying the results we obtained for 400 training episodes, as well as the margins of error and confidence intervals that were calculated.

Epsilon	**Mean cumulative reward**	**Margin of error**	**Confidence interval**
10%	5.01	0.16	[4.85, 5.17]
20%	5.56	0.10	[5.46, 5.66]
30%	5.75	0.08	[5.67, 5.83]
40%	5.09	0.11	[4.98, 5.20]
50%	5.32	0.10	[5.22, 5.42]
60%	4.99	0.16	[4.83, 5.15]
70%	4.64	0.15	[4.49, 4.79]
80%	4.25	0.14	[4.11, 4.39]
90%	1.90	0.12	[1.78, 2.02]

Table 1: Results obtained while training the agent for 400 episodes with different exploration rates

From the table above, it can be seen that none of the intervals overlap. Therefore, we are able to conclude that the optimal exploration rate is 30%, as it has the greatest upper bound. Note that when the number of episodes is greater than 400, the confidence intervals overlap, as evident by the narrow distance between data points in Figure 3. This was verified by calculating 95% confidence intervals for a higher number of training episodes. Hence, no conclusions can be drawn for other episode totals.

Discussion

As there was no overlap between any of the confidence intervals in Table 1, it is possible to conclude with statistical significance that **for 400 episodes the optimal exploration rate is 30%**. This result is of great significance as it implies that in order for the agent to be most optimized, exploration is a significant factor, which supports our hypothesis. Despite having determined an optimal exploration rate, the mean cumulative reward is significantly lower than the convergence value found. Therefore, if the goal was to reach the greatest mean cumulative reward, then an optimal exploration rate cannot be determined. This is because the confidence

intervals for a greater number of training episodes overlapped. According to an academic journal[4] researchers found that lower exploration rates maximize the agent's performance and they determined that exploration rates less than 10% were most optimal. The Taxi Problem required a greater exploration rate as the penalties for illegal actions were more severe. There is therefore a distinct dependency between the exploration-exploitation trade off and the rewards within the problem being solved.

It has become apparent that larger exploration rates result in a higher speed of learning, and thus a greater rate of convergence. This is evident in Figure 3, as the curves representing large exploration rates improved the most as the number of training episodes increased. Moreover, the first curve to reach a mean cumulative reward of at least 8 is that of a high epsilon value, namely epsilon = 0.8. What is interesting to see, is that after around 1200 episodes the mean cumulative reward converges. In other words, it seems that after a certain amount of training, the epsilon value has a minimal effect on the performance of the agent. This supports our hypothesis and is inline with what researchers have found. This academic journal found that no matter the exploration rate, the agent always eventually learns the best option[5]. This result implies that even though the exploration rate may have an effect on how quickly the agent learns, after enough training the agent will learn the optimal policy no matter the chosen exploration rate. Hence, it can be suggested that, performance wise, the agent is fully optimized after approximately 1200 training episodes. We verified this by rendering every action the agent took, and saw that it followed the optimal policy without fault - the shortest distance from pickup to drop-off, avoiding any illegal actions.

Improved methods

One way we could improve our investigation and broaden our scope is by examining how epsilon-greedy decay impacts the performance of the agent in the same environment. One would presume that it would have a greater performance, as it will initially maximize exploration and then gradually maximize exploitation. One could then compare this to the optimistic initialization algorithm, where values greater than the mean reward are to be used. This would allow the agent to revisit many action-state pairs thus resulting in faster improvement. By comparing the different algorithms it would then give insight into which algorithm is more suitable. Another way that we could take our project further, is to increase the complexity of the taxi's mission. This could be done by, for example, having more than one passenger to pick-up and drop-off, or having randomly generated obstacles that would obstruct the optimum route. This would mean that the agent would have to make more conscious decisions in order to optimize its path. This would certainly require deep Q-learning and a very large data set. Once optimized, this could be applied to real-world taxi navigation. This leads us to ethical considerations concerning Flex Trafik.

Ethical considerations

According to recent news concerning Flex Trafik[6], the working conditions of drivers is inhumane. Workers receive significant fines for being late, and they are overworked. The study that we conducted could be used as a springboard towards optimizing Flex Trafik and thereby reducing working hours and required labour. Deep Q-learning could be utilized alongside Epsilon-greedy to determine both the optimal path and order of pickup and deliveries. However, this would likely result in Flex Trafik downsizing their workforce, due to increased optimization. Hence, reinforcement learning, specifically Q-learning, can both be a problem and solution to

[4]Semiu Ayobami Akanmu, Rakhen Garg, Abdul Rehman Gilal, Towards an Improved Strategy for Solving Multi-Armed Bandit Problem, 2019, `https://www.researchgate.net/publication/336650162_Towards_an_Improved_Strategy_for_Solving_Multi-_Armed_Bandit_Problem`

[5]Imad Dubbara, epsilon-Greedy algorithm, `https://imaddabbura.github.io/post/epsilon-greedy-algorithm/`

[6]Politikken, Her er 10 klager over trætte chauffører hos Flextrafik, 2022, `https://politiken.dk/forbrugogliv/forbrug/art8554846/Her-er-10-klager-over-tr-C3-A6tte-chauff-C3-B8rer-hos-Flextrafik`

many business issues. It is therefore very important that technology and innovation does not overshadow ethics and humanity.

References

Gym, Getting Start with Gym, 2016, https://gym.openai.com/docs/

DTU Learn, 2 Statistics, https://learn.inside.dtu.dk/d2l/le/content/79546/viewContent/302976/View

Semiu Ayobami Akanmu, Rakhen Garg, Abdul Rehman Gilal, Towards an Improved Strategy for Solving Multi-Armed Bandit Problem, 2019, https://www.researchgate.net/publication/336650162_Towards_an_Improved_Strategy_for_Solving_Multi-_Armed_Bandit_Problem

Politikken, Her er 10 klager over trætte chaufførerer hos Flextrafik, 2022, https://politiken.dk/forbrugogliv/forbrug/art8554846/Her-er-10-klager-over-tr-C3-A6tte-chauff-C3-B8rer-hos-Flextrafik
https://medium.com/@ashish_fagna/understanding-openai-gym-25c79c06eccb

Imad Dabbura, Epsilon-Greedy algorithm, 2018, https://imaddabbura.github.io/post/epsilon-greedy-algorithm/

Appendix

Responsibility

Introduction section: Mark Andrawes
Methods section: Albert Frisch Møller
Results section: Albert Frisch Møller (Figure 2 & Table 1) & Mark Andrawes (Figure 3 & Table 1)
Discussion section: Albert Frisch Møller (paragraph 1 & 3) & Mark Andrawes (paragraph 2 & 4)
Python code: Albert Frisch Møller & Mark Andrawes

Python Code

```
import gym
import numpy as np
import statistics
#-------------------------------------------------------------------#
#Training the agent
#Initializing conditions
gamma = 0.9
rewards_array = np.array([])
epsilon_array = np.array([])
var = np.array([])
epsilon = 0.10
```

```
while epsilon <= 0.90:

    #Initializing Q-table
    #It was given that there were 500 states and 6 possible actions

    V = np.zeros([500,6])

    env = gym.make("Taxi-v3").env

    rewards = np.array([])

    for i in range(1000):

        cumulative_reward = 0

        done = False

        state = env.reset()

        #Implementing epsilon greedy

        while done == False:

            p = np.random.rand(1,1)[0]

            if p < epsilon:

                action = env.action_space.sample()  #Exploration

            else:

                action = np.argmax(V[state]) #Exploitation

            observation, reward, done, info = env.step(action)

            cumulative_reward += reward

            if done == True:

                rewards = np.append(rewards, cumulative_reward)

            #Applying the quality function formula

            V[state,action] = reward + gamma*(np.max(V[observation]))

            state = observation

    epsilon_array = np.append(epsilon_array,epsilon)

    rewards_array = np.append(rewards_array,np.mean(rewards))

    var = np.append(var,statistics.variance(rewards))

    print(statistics.variance(rewards))

    print(np.mean(rewards))

    epsilon += 0.1

    #Testing the agent

    done_counter = 0

    for i in range(200):

        done = False

        state = env.reset()

        steps = 0
```

```
        while done == False:
            action = np.argmax(V[state])
            steps += 1
            observation, reward, done, info = env.step(action)
            state = observation
            if steps > 20:
                break
            if done == True:
                done_counter += 1
    print(done_counter)
```