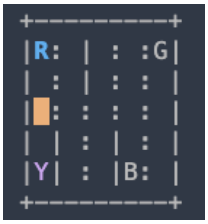
Introduction and Problem Formulation

In this project, we implement the taxi problem from the OpenAI Gym and utilize Q-learning, Monte Carlo and Deep Q-Network(DQN) methods. Then we compare and analyze their performance on the taxi problem. There are four designated locations in the grid world indicated by R(red), B(blue), G(green), and Y(yellow). One episode is defined as one sequence of states, actions, and rewards that end with a deterministic state.



The Taxi game map

When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination, and then drops off the passenger. Once the passenger is dropped off, the episode ends. The taxi cannot pass through a wall and the pick-up point and destination differ.

State Space

The state is defined by the position of the taxi on the grid (presented by a column and a row number between 0 and 4). The size of the state space is 25*5*4= 500, where 25 is the possible locations for the taxi in the 5x5 grid map, 5 is the possible locations for the passenger, and 4 is the number of possible destinations. The state space is discrete since there are a finite number of states that can be performed. In addition, the passenger can be in the location of R, G, Y, B in the taxi and the destination can be the location of R, G, Y, B.

Action Space

The action space is discrete since there are a finite number of actions that can be chosen. It can be defined as [0- move up; 1- move down; 2- move left; 3- move right; 4- pick up a passenger; 5- drop off passenger].

Reward Scheme

There will be a +20 reward for driving the passenger to the destination for a successful drop-off and a reward of -1 for each action. There will also be a -10 penalty for executing the “pickup” or “drop-off” actions illegally. The reward formulation can be defined as

r = { -1, for any actions from any state;

+20, for a successful dropoff;

-10, any illegal "pickup" and "dropoff" action}

Methodology and Algorithm Description

There are three algorithms implemented to solve the Taxi problem. Q- learning, Monte Carlo, and Deep Q-Network(DQN).

Q - Learning

The Q-Learning method is a value-based learning algorithm where the agent aims to optimize the value function to solve the problem. This technique requires 4 steps to implement the learning process: Initialization, Exploration, Observation and Updating the value function.

1. The initialization step: initializing all the Q values in the Q-table to 0.
2. Exploration and exploitation using epsilon greedy strategy

   First, we explore the action space by taking random actions with random probability restrained by comparing the value of epsilon with samples drawn from uniform distribution. Then we exploit the learned value by executing the action with the highest Q-value at the state the agent is located.
3. Observe the reward obtained through exploration.
4. Update the value function: taking the rewards observed in the previous step and updating the Q-table accordingly following the equation below.

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\Big(reward + \gamma \max_{a} Q(next\ state, all\ actions)\Big)$$

Figure: Equation for updating the Q table

Selection of Hyperparameters

We choose epsilon decay to determine the best set of values of parameters. Compared with epsilon as 0.1 and 0.5 shown below, epsilon 0.9 generates the highest average reward and the least timesteps taken. Therefore, we set the epsilon value as 0.9 which introduces a randomness of 0.9 for the agent to select random actions without referring to the Q table. This allows the agent to explore the action space further with exploitation tradeoff.

For the other 2 hyperparameters in this algorithm, the discounted rate gamma and the learning rate alpha, we initialize them as gamma = 0.9, alpha = 0.9 which guarantees convergence while maintaining a fast learning rate.

DQN

Deep Q-Network uses the epsilon-greedy exploration algorithm to explore and construct a memory-like object first, and then exploit the information that is gathered. Specifically, the learning network model consists of one input layer which instantiates a Keras tensor of state size of 500 and passes through three hidden dense layers with ReLU activation which feed all outputs from the previous layer to the next layer. Finally, a final dense layer is utilized by activating the linear activation function which has the same dimension as the action space. After building the network, the agent explores the states where they randomly take actions and get rewards and calculate the errors by MSE. The experience is recorded and then fed to exploit the training model to update the weight while training. After many trials and errors, the gamma is set as 0.95, the alpha is set as 0.1 and the epsilon is set as 0.02 which can produce the optimal result.

Monte Carlo

The Monte Carlo algorithm is a model used to predict the probability of a variety of outcomes when the potential for random variables is present. As the taxi problem can contain random variables, implementing the Monte Carlo algorithm, specifically the Monte Carlo Control Algorithm, to solve the outlined problem can be beneficial. The Monte Carlo method is a computational algorithm that relies on repeated random sampling to obtain numerical results, as such, many episodes are required. The code that

implements this solution consists of a class dedicated to the agent, which contains functions that randomly select an action and a function that implements each step and stores the resulting values accordingly. The code also contains a function outside of the agent class which implements the Monte Carlo Control method for each episode up until the specified total.

Results

Q - Learning

After the training, the Q-table is updated with different values generated for each state over the 1000 episodes as shown below.

Q Table	Actions					
	Up	Down	Left	Right	Pickup	Dropoff
State 0	0	0	0	0	0	0
State 499	-1.719	-1.719	-1.719	13.59	-9.0	-9.0

Figure: Q-Table after Training

In the rewards for episodes over time figure below, we can tell that the agent starts to generate good rewards after about 300 episodes. However, since it starts with random actions in the beginning episodes resulting in extremely low rewards of about -600, the average reward is -15.66 over the 1000 episodes. For penalty, since the agent learns to find the optimal path, the penalty is zero. Furthermore, it takes 28872 timesteps in total.

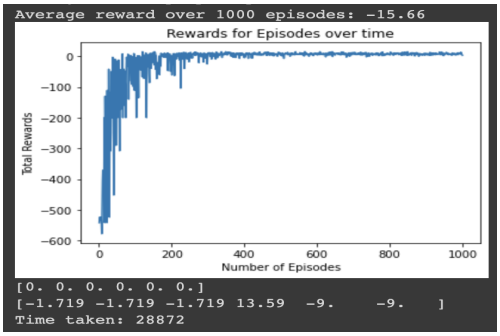


Figure: Performance with Epsilon = 0.9

Furthermore, if we set epsilon as 0.1 or 0.5, they will take more time to generate the result. As shown below, they also result in relatively lower average reward compared to the epsilon we choose 0.9.

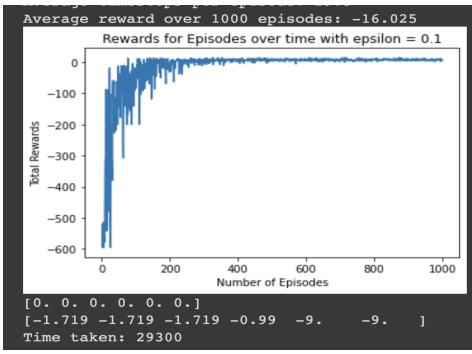


Figure: Performance with Epsilon = 0.1

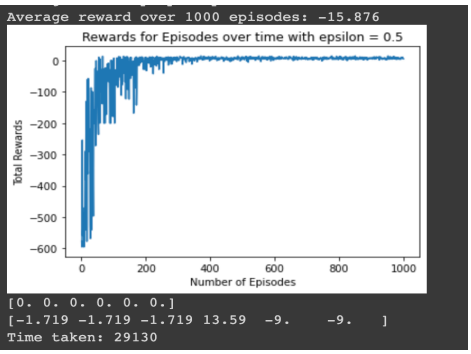


Figure: Performance with Epsilon = 0.5

DQN

After training the agent with 1000 episodes, most of the maximum rewards are stuck at -102 which means the agent learned not to make illegal actions and avoid a -10 reward. However, the agent failed in successfully dropping off the passenger to gain 20 rewards at the end of the episode which produced huge negative rewards. After trying hundreds of times including changing network structure, loss function, weight update and hyperparameters like gamma, learning rate, alpha, and size of the network, I still cannot get a good result. The following two figures show sets of alpha/gamma/epsilon values used: 0.1/0.95/0.02 and 0.9/0.9/0.2. The structure of the network or the weight update function may lead the agent to find the local minimum which needs future improvement.

Monte Carlo (Control)

When implementing the Monte Carlo Control method to solve the taxi problem, the following

```
Episode 1000 | Total point average of the last 100 episodes: -102.52
```

```
Episode 1000 | Total point average of the last 100 episodes: -103.33
```

sets of alpha/gamma/epsilon values were used: 0.5/1/0.01 and 0.01/0.9/0.01. The results for the first set of values (0.5/1/0.01) resulted in the following, which displaying an average reward of -763.49 for the 1000 total episodes.

```
Episode:   1000/1000 || Average reward: -763.49 || eps: 0.01
```

When running the second set of values (0.01/0.9/0.01) for a total of 1000 episodes, we see the following results, displaying an average reward of -738.92. This shows that the second set of values

```
Episode:   1000/1000 || Average reward: -738.92 || eps: 0.01
```

results in a more optimized end policy, as the average reward is better than the average reward of the first set of values.

Discussion

By comparing the performance of these 3 methods, we can tell that Q-Learning gives out the best performance by having the largest average reward over 1000 episodes and the Monte Carlo method performs the worst with the lowest average reward generated. For DQN, it does not outperform the Q-Learning because this problem environment is not ideal for DQN.

For the Q-Learning part, it was challenging to find a suitable tuning method for the hyperparameters. I tried to tune the parameters based on the ratio of reward over time steps. According to its performance in other related works, this method can generate very optimal solutions. However, it is very time consuming, which makes it not an ideal tuning method for this project. I end up with the epsilon decay method that generates a fair solution and is convenient in times taken. In the future, I will investigate state of art methods used for similar projects and improve my method accordingly.

For DQN, one of the challenges is the implementation of the coding. I experience several problems with libraries, systems, and kernel crashes. There was a time I tried to execute my code and the kernel crashed the whole day so I can do nothing to work on my project. I spent the whole day searching for a solution, even reinstalling all the environments. Finally, I restarted my laptop and found out that insufficient laptop memory led to the kernel crash. Finding an optimal hyperparameter is also another challenge I faced that I need for future learning.

For the Monte Carlo algorithm implementation, one of the challenges was getting the gym environment operational. For some reason, my environment was not set up properly and thus, I was unable to test the Monte Carlo implementation. To overcome this challenge, Songyu graciously offered to test the implementation in my stead and show me the results for me to complete the implementation.

References

Guillaume Androz. (2021). Deep Q-Learning with Pytorch and OpenAI-gym: The Taxi-cab puzzle. MLearning.ai

Sarkar, A. (2020, September 15). Reinforcement learning for taxi-V2. Medium. Retrieved December 7, 2022, from https://medium.com/@anirbans17/reinforcement-learning-for-taxi-v2-edd7c5b76869

Doshi, K. (2021, February 14). Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step. Medium. https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Chen, A. (2021, January 22). *Reinforcement learning and Q learning -an example of the 'taxi problem' in python*. Medium. Retrieved December 8, 2022, from https://towardsdatascience.com/reinforcement-learning-and-q-learning-an-example-of-the-taxi-problem-in-python-d8fd258d6d45

Mayers, G. (2020, July 30). *Reinforcement learning: Using Q-learning to drive a taxi!* Medium. Retrieved December 8, 2022, from https://medium.com/analytics-vidhya/reinforcement-learning-using-q-learning-to-drive-a-taxi-5720f7cf38df

Developer, A. S. K. S., Author: Brendan Martin Founder of LearnDataSci, Satwik Kansal Software DeveloperSoftware Developer experienced with Data Science and Decentralized Applications, & Brendan Martin Founder of LearnDataSciAuthor and Editor at LearnDataSci. Python development and data science consultant. (n.d.). Reinforcement Q-learning from scratch in Python with Openai Gym. Learn Data Science - Tutorials, Books, Courses, and More. Retrieved December 8, 2022, from https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/