OpenAI Gym's Taxi-v3 task

Nguyen Tien Vuong

MSc Computer Science, Artificial Intelligence Specialization, Eötvös Loránd University
nguyentienvuong.scorpio13@gmail.com

1 Introduction

Reinforcement Learning is a technique of machine learning which the software agent learns to perform certain actions in an environment to maximum reward. It does thus by exploration and exploitation of data it learns by recurrent trials of maximizing the reward. To take a deeper look into this, 'Taxi-v3' is implemented using Q-learning to demonstrate how can reinforcement learning techniques be used to apply to the problem of simulation of self-driving cab. Moreover, by using Q-learning, a Q-table is created and it is where a Q-value for each state, action pair is stored. For each state, several actions can be done, which leads to multiples Q-value, and the goal is to choose the action with the best opportunity to get the maximum reward. In this work, the actions and the states of the taxi cab are the inputs for the algorithm. After Q-learning agent explored enough, it can maximize the rewards cleverly by making smart moves to drop off the passenger.

2 Related work

Beside Q-learning, many other Reinforcement Learning techniques have been applied to 'Taxi-v3' task. For example, Tom Roth has introduced other methods for this such as SARSA and Expected SARSA, and he has showed that Expected SARSA delivers better reward than Q-learning, while SARSA does not deliver better reward than Q-learning [1].

3 Detail of the task

The main task of the taxi cab in this project is to pick up the passenger at one location and drop them off in another. Furthermore, the taxi-cab is rewarded for each move, with a low penalty when it is traveling, with a large penalty if it picks up or drops off the passenger at the invalid location, also it will receive a big reward if succeeds.
In this work, a state is described by the location on the grid (a 5×5 matrix), a location to drop-off the passenger from four choices, and the passenger which may be in one of the four locations or inside the taxi. From that, the number of possible states can be calculated, which is $5 * 5 * 5 * 4 = 500$ possible states (five

rows, five columns, five passenger locations and four destinations).
Moreover, there are six possible actions. The taxi cab can move to the four directions: South, North, East or West also with two other actions which are pick up or drop off.

4 Method

4.1 Q-learning

The purpose of this method is to study a policy, which informs an agent what action to take beneath what circumstances. The model of the environment is not necessary, and it is possible to handle the issues with stochastic transitions and rewards.
For each Markov decision process, Q-learning detects a policy that is the best within the idea that it maximizes the expected value of the total reward over solid steps, starting from the present state. Q-learning can determine an optimal policy for any given Markov decision process.
There is a function within the algorithm which computes the condition of a state-action combination:

$$Q : S \times A \to \mathbb{R}$$

Before starting learning, Q is initialized to a possibly arbitrary fixed value. Then, at each time t an action a_t is selected, a reward r_t is observed, a new state s_{t+1} is entered (that may depend on both the previous state and the selected action), and Q is updated. The main idea of the algorithm is a simple value iteration update, using the weighted average of the old and the new information:

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where:$Q^{new}(s_t, a_t)$ is the updated value, $Q(s_t, a_t)$ is the current Q value, r_t is the reward for current state and action, $max_a Q(s_{t+1}, a)$ is the estimate of optimal future value[2].
An episode of the algorithm ends when state s_{t+1} is a final state.

4.2 Hyperparameters

Learning rate: The learning rate $0 \leq \alpha \leq 1$ decides to what proportions newly obtained information overrides old information. If α is 0, it will make the agent learn nothing and only exploit prior knowledge, if α is 1, it will make the agent consider only the most recent information and ignore the previous knowledge to explore possibilities[2].
Discount factor: The discount factor $0 \leq \gamma \leq 1$ decides the importance of future rewards. A factor of 0 will make the agent only learn about actions that produce an immediate reward, while a factor approaching 1 will make it strive for a long-term high reward[2].
Exploration vs Exploitation: There's a tradeoff between exploration (choosing a random action) and exploitation (choosing actions based on already learned

Q-values). It is necessary to prevent the action from always taking the same route, and possibly overfitting, so another parameter called ϵ will be used to take care this issue during training. If ϵ is set to 0, the object never explore but always exploit the knowledge it already has. On the contrary, having the ϵ set to 1 make the algorithm always take random actions and never use past knowledge[3].

5 Experiments/Results/Discussions

First of all, the hyperparameters are set as follow: $\alpha = 0.7, \gamma = 0.618$, and $\epsilon = 1$. Then, in the first step, the Q-table is created, which will be used to track states, actions, and rewards. The quantity and activity status in the Taxi environment determines the size of the table.
In the next step, choosing an action given the states based on a random number. If the random number is larger than epsilon, exploitation will be employed and the best action will be selected. Otherwise, exploration will be employed and a random action will be chosen.
For the next moves, the action is performed and the reward is measured. When choosing an action, we will continue it and measure the associated reward.This is done using the built-in function env.step (action) in Python to perform a one-step move. It returns the next state, the reward from the previous action, also indicates whether our agent has met the goal and diagnose the performance. If the agent reaches the goal, then the environment will be reset.
Finally, the collected information can be used to update the Q-table using the equation mentioned in Section 4.1. Furthermore, ϵ is reduced to cut down on exploration.
These steps will be performed 2000 times as training episode is set to be 2000. Here are the results:

Fig. 1.

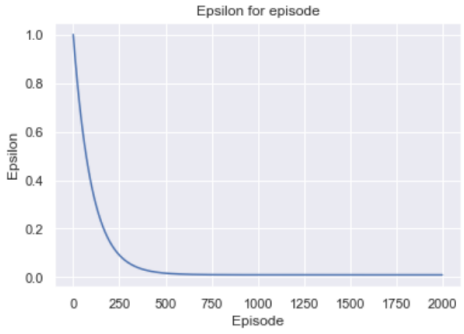

Fig. 2.

From Figure 1 and Figure 2, the agent is on the right track, which is getting rid of negative rewards pretty quickly, then become quite steady around 0. At the same time, it can be seen that the exploration rate ϵ decreases with learning. The agent learns to choose better action to get rewarded, as opposed to random behavior.
To highlight the performance of Q-learning, consider the case where the agent does not use the algorithm to determine the best action for the current state and act random instead. By this, the learning rate is kept but $\epsilon = 1$, in other words, there is no exploitation.

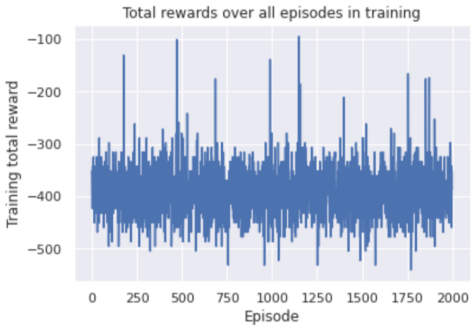

Fig. 3.

As can be seen in Figure 3, the agent does not improve its performance and the total rewards are around -390, while in Figure 1 with exploitation, the result is significantly better.

6 Conclusion

In this project, the objective is to understand how to implement 'Taxi-v3' task using Q-learning method, which is a technique that has been widely used in many works which required Reinforcement learning method. Using this algorithm, the taxi cab is trained 2000 times and the results have shown that the agent learns and the reward is enhanced kind of quickly.
Beside what has been shown above, Q-learning still has some limitation. One of them is it can only be applied to problems with finite states. If it encounters an environment with a continuous space, then Q-table cannot be defined. Luckily, in 'Taxi-v3', it only has finite number of states.

References

1. Tom Roth. *Sarsa, expected sarsa and Q-learning on the OpenAI taxi environment.* `https://tomroth.com.au/sarsa-qlearning/`
2. Wikipedia *Q-learning* `https://en.wikipedia.org/wiki/Q-learning`
3. Blockgeni *Reinforcement Q-learning in Python* `https://blockgeni.com/reinforcement-q-learning-in-python/`