**Reinforcement Learning Game Design-Taxi**

Table of Contents

1. Introduction
2. Features

2.1 Taxi-v3 Environment

2.2 States

2.3 Action Space

2.4 Reward

1. Q Learning
2. Technical Details

4.1 Initialization

4.2 Training the Agent

4.3 Testing the Trained Agent

1. Results

**1. Introduction**

The project shows in detail how to build and train a reinforcement learning model in Python by introducing and implementing the Q-learning algorithm, using OpenAI Gym's Taxi-v3 environment as an example. The project covers the environment setup, state and action space definition, reward mechanism, and training process, and illustrates the effect of model learning by comparing the behavior of random and trained agents.

**2. Features**

2.1. Taxi-v3 Environment

The Taxi-v3 environment consists of a 5x5 grid map where the agent, represented as a taxi driver, must pick up a passenger at one of four designated locations (Y, R, G, B) and drop them off at their destination, which is also one of the four locations. The environment is designed to test the agent's ability to navigate the grid, pick up passengers, and deliver them to the correct destination while maximizing cumulative rewards.

2.2. States

In the Taxi-v3 environment, the states are defined by the following components:

* The taxi's position on the grid (x, y coordinates).
* The passenger's location (either at one of the four pickup locations or in the taxi).
* The passenger's destination.

The state space is represented by a vector: State=[*x*pos\_taxi​,*y*pos\_taxi​,pos\_passenger,dest\_passenger]

Given the grid size of 5x5, the possible positions for the taxi are 25. The passenger can be at one of the four pickup locations or in the taxi, resulting in 5 possible states. The destination can be one of the four locations. Therefore, the total number of states is 5×5×5×4=500. However, the actual number of usable states is slightly less due to the constraint that the pickup location and destination cannot be the same.

2.3. Action Space

The agent in the Taxi-v3 environment can perform the following discrete actions:

|  |  |
| --- | --- |
| Action | Action number |
| Move forward | 0 |
| Move backward | 1 |
| Move right | 2 |
| Move left | 3 |
| Pick up the passenger | 4 |
| Drop off the passenger | 5 |

2.4. Rewards

The reward structure for the agent's actions is as follows:

* Moving: -1 (a small penalty to encourage the shortest path).
* Incorrect drop-off: -10 (a larger penalty for delivering the passenger to the wrong location).
* Successful drop-off: +20 (a positive reward for completing the task successfully).

**3. Q-Learning**

Essentially, Q-learning lets an agent use the rewards from the environment to learn, over time, the best action to take in a given state.

In our Taxi environment, we have a reward table P that the agent will learn from. It does this by looking at the reward it gets for taking an action in the current state, and then updates the Q value to remember whether that action was beneficial or not.

The values ​​stored in the Q table are called Q-values, and they map to (state, action) combinations.

The Q-value for a particular state-action combination represents the "quality" of the action taken from that state. Better Q-values ​​mean a greater chance of getting a larger reward.

The Q-learning algorithm updates the Q-values based on the following equation:

Where:

* *α* is the learning rate.
* *γ* is the discount rate.

We initialize a Q-table with dimensions corresponding to the state and action spaces (500 states and 6 actions, resulting in a 500x6 table). The training process involves multiple episodes where the agent explores the environment and updates the Q-values based on the rewards received.

**4. Technical Details**

4.1. Initialization

To operationalize the Q-learning algorithm within the context of the Taxi-v3 environment, the initial preparatory step involves the installation and subsequent importation of requisite libraries into the Python programming environment. Foremost among these libraries is the OpenAI Gym toolkit, which furnishes the Taxi-v3 environment—a grid-based simulation designed to evaluate reinforcement learning algorithms. This environment serves as the experimental framework within which the agent's learning and decision-making processes are assessed.

In addition to the OpenAI Gym library, several ancillary libraries are enlisted to facilitate data manipulation and visualization. The NumPy library is utilized for array operations and mathematical functions。 The Matplotlib library is employed to generate graphical representations of the training process and the agent's performance metrics.

Furthermore, the random library is incorporated to introduce stochasticity in the agent's action selection process during the exploration phase. This stochastic element is crucial for enabling the agent to explore the state-action space thoroughly, thereby enhancing the robustness of the learned policy. Collectively, these libraries form the foundational infrastructure that supports the implementation, execution, and evaluation of the Q-learning algorithm in the Taxi-v3 environment, thereby ensuring a comprehensive and methodical approach to reinforcement learning.

4.2. Training the Agent

The training process of the Q-learning agent in the Taxi-v3 environment is meticulously designed to optimize decision-making through iterative learning. A Q-table, initialized with zeros, serves as the core data structure to store the expected utility of taking specific actions in given states. The training commences with the specification of critical hyperparameters: a learning rate (α) of 0.1, which controls the extent to which newly acquired information overrides existing knowledge; a discount rate (γ) of 1.0, which determines the importance of future rewards; and an exploration rate (ϵ) of 0.1, which balances the trade-off between exploration of new actions and exploitation of known optimal actions.

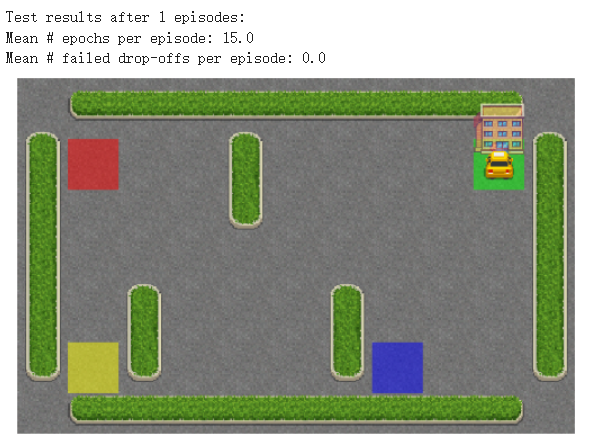
Over the course of 10,000 episodes, the agent engages in a dynamic interaction with the environment. At the onset of each episode, the environment is reset to its initial state. The agent then proceeds to take actions, either by randomly sampling from the action space to explore new possibilities or by selecting the action with the highest Q-value to exploit known information, contingent upon the exploration rate. Following each action, the environment provides feedback in the form of the next state, a reward signal, and a termination flag indicating whether the episode has concluded.

Throughout the training, the cumulative rewards and the number of epochs (steps) taken in each episode are meticulously recorded. Periodic monitoring of the training progress is facilitated by printing the cumulative rewards every 100 episodes. Upon completion of the training regimen, the agent's learning trajectory is visually assessed through two plots: one depicting the cumulative rewards per episode, which reflects the agent's increasing proficiency in maximizing rewards, and the other illustrating the number of epochs per episode, which indicates the agent's enhanced efficiency in navigating the environment with fewer steps.

4.3. Testing the Trained Agent

To validate the training results, we test the trained agent by running it through the environment and observing its performance. After executing enough iterations, we can find that the taxi always drives directly to the passenger, takes the shortest way to the destination, and successfully drops the passenger off.

**5. Result**

****

Appendix: (Code in Google Colab)

