Neuroscience of Learning, Memory and Cognition

Reinforcement Learning Project Report

Mohammad Nourbakhsh marvast

Student ID: 401200482

January 2024

1 Algorithm

In this section we are going to explain the algorithm and through the python code. The code can be accessed through THE JUPITER NOTEBOOK.

The game is simple: In each block of the Grid file (grid.xlsx), we have the choice between two moves: go either upward or rightward for passing from the southeastern angle to the northwestern angle. Though, we are aiming to find the route with the most reward.
For this problem, we introduce these sets:

$$\textsc{States}: \quad \mathcal{S} := \{\text{Block} : \forall\ \text{Block} \in \text{Grid}\} \tag{1}$$

$$\textsc{Actions}: \quad \mathcal{A} := \{\text{Up},\ \text{Right}\} \tag{2}$$

Definition. For a sequence $s = \{s_i\}_{i=1}^{N} \subset \mathcal{S}$ of successive states, we define the reward of s to be $r(s) = \sum_{i=1}^{N} r(s_i)$. Where $r(s_i)$ is the reward we receive for visiting the i-th block and is given in the Grid file.
The problem is now reduced to finding the most reward-receiving sequence of successive states among all possible sequences.

Definition. We call a sequence of state-actions $s_0, a_0, s_1, a_1, \ldots, s_N, a_N$ for $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}$, an episode.

Definition. Q-table is simply a matrix where rows represent states and columns represent actions. The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. In simple words, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

At the first glance our Q-table is filled with zeros since the agent took no action.

Then, we construct episodes and we update the Q-table.

For constructing the first episode:

1. We are standing at the southeastern block and we know it's reward.

2. We are suppose to take an action! But how?!

   we use the ϵ-greedy policy:

   (a) We take an $0 \leq \epsilon \leq 1$

   (b) In our Q-table, we observe the action prices of our current state. We choose the action with the highest reward!

   (c) Choose a random number between 0 and 1.

   (d) If the ϵ is greater that our random number, we change our action. Else, we remain by our recent choice.

```
def epsilon_greedy_policy(Qtable, state, epsilon):
action = np.argmax(Qtable[state])
random_int = random.uniform(0,1)

## action = 0: go rightward
## action = 1: go up

if random_int < epsilon:
action = 1 - action

return action
```

3. After taking action and moving to the new state s^{new}, we can observe our reward value due to the Grid file (i.e. $r(s^{\text{new}})$).

4. Now we are in a place to update our Q-table due to the Bellman Optimality Equation:

$$Q^{\text{new}}(s,a) = (1-\alpha)Q(s,a) + \alpha\Big(r(s^{\text{new}}) + \gamma \max_{a'} Q(s^{\text{new}}, a')\Big)$$

   Where $0 \leq \alpha \leq 1$ is our learning rate and $0 \leq \gamma \leq 1$ serves as the discount factor determining the importance of future rewards and may also be interpreted as the probability to succeed at every step.

5. Now we should proceed to the new state and repeat the same for updating the whole Q-table and complete the single episode.

After completing an episode, we should move to a new episode by the Q-table established by the previous episode. You can see the code in the next subsection.

After having our Q-table updated, we have sufficient tools for finding the most-rewarding road.

1. We are standing at the southeastern block.
2. For choosing action, we refer to the updated Q-table. We choose the action that have the greater reward in our Q-table.
3. After moving to the next state, we can repeat the procedure.

2 Experiments & The Optimal Route

By the code, we have the optimal route (1) with the total reward of 385:

```
data =  pd.read_excel('Grid.xlsx', header = None)
grid = np.array(data)


road, rewards, max_road_reward = optimal_road(grid, ...)


plot_optimal_road(grid, ...)
```

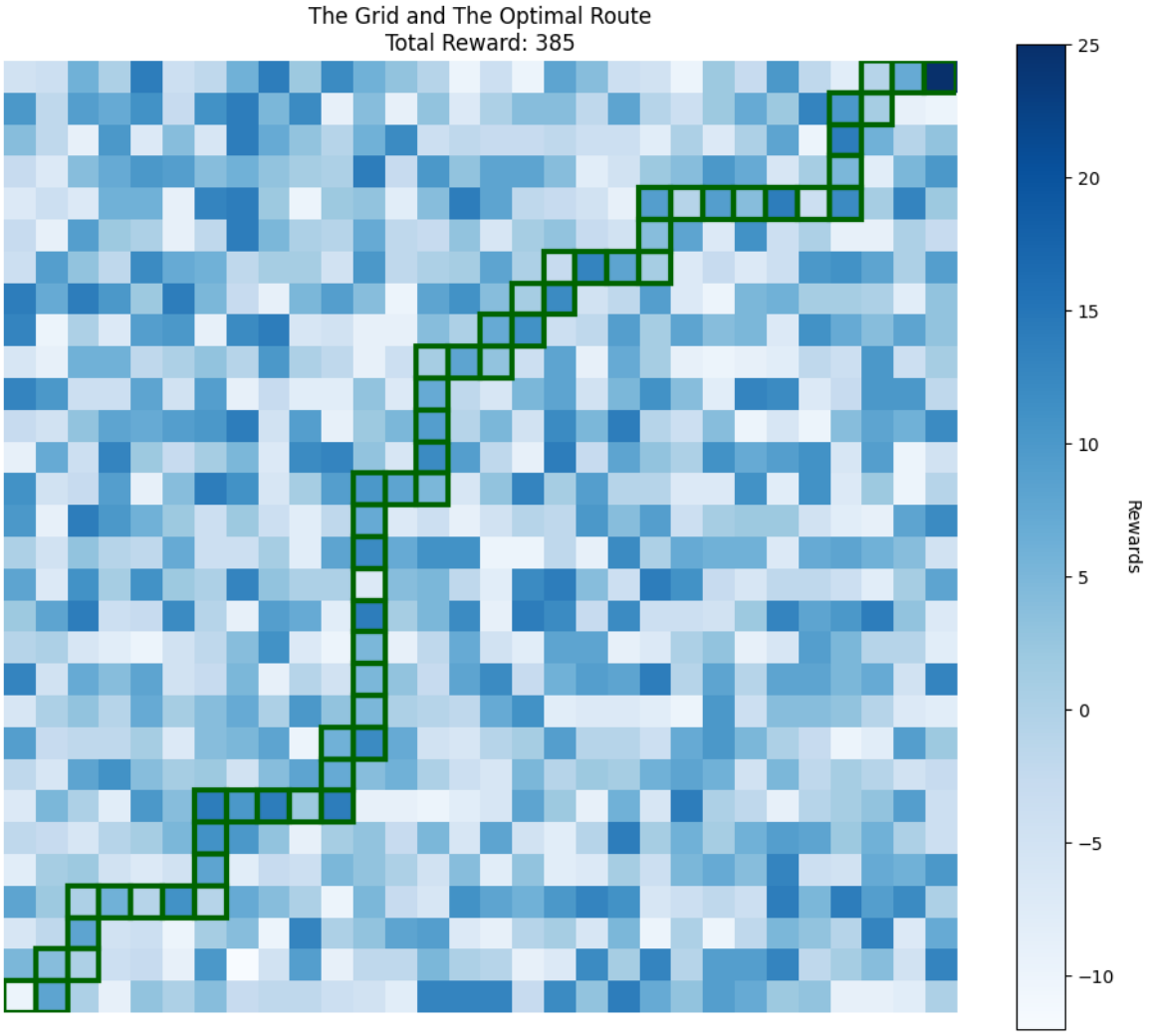
The Reward vs episodes plot can be seen at (2).

```
plot_dynamics(rewards)
```

Before Using this 30×30 grid, I test the code with 3, 3×5, and several random matrices. (3).

```
##fack data set
n = 3
m = 3
M = np.random.normal(0, 20, size=(n, m))
pd.DataFrame(M)


fack_road, fack_rewards, max_road_reward = optimal_road(M, ...)
```



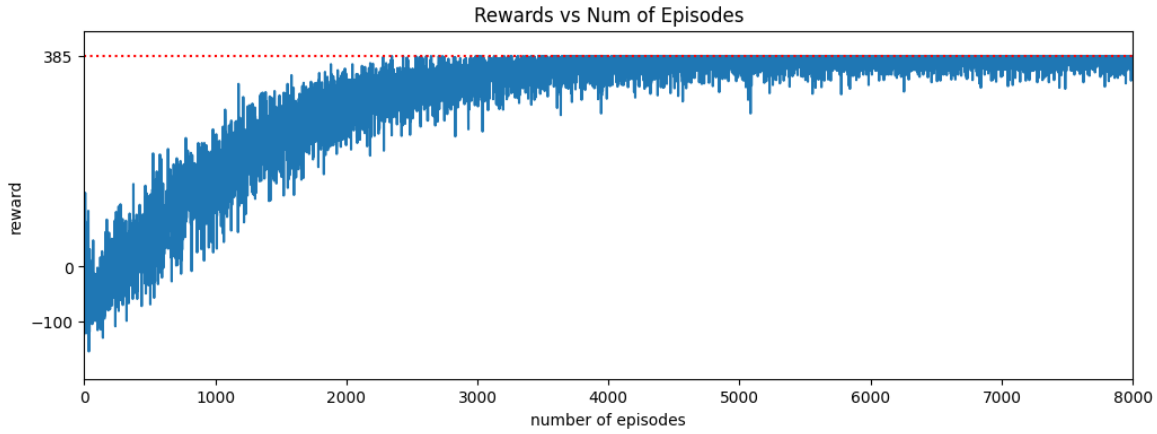
Most-rewarding path

```
plot_optimal_road(M, ...)
```

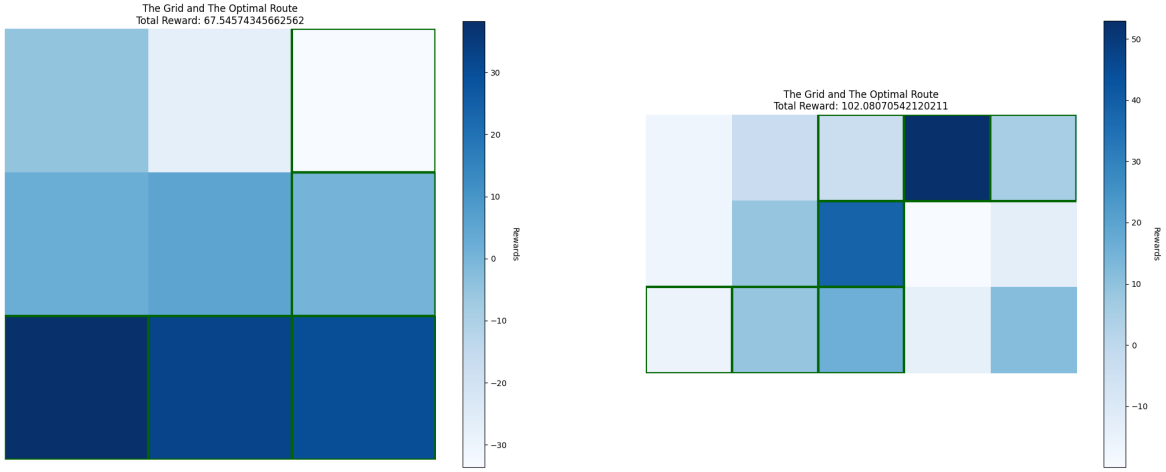
```
##fack data set
n = 3
m = 5
M = np.random.normal(0, 20, size=(n, m))
pd.DataFrame(M)

fack_road, fack_rewards, max_road_reward = optimal_road(M, ...)

plot_optimal_road(M, ...)
```



Rewards vs Episodes



Some Fake Data

I figured out that **exploration decay rate** is the most crucial parameter for updating our Q-table. With sufficiently small decay rate, with even low learning rate (about 0.4) we have the desired result! For the ϵ-greedy policy, we use this decay rate to allow less and less exploration around and let the agent to relay more and more on its exploitation.