NANYANG TECHNOLOGICAL UNIVERSITY

AI6101
REINFORCE LEARNING PROJECT REPORT

Kang Xiwen

G2202670K

School of Computer Science and Engineering

10.10.2022

Contents

Contents **i**

1 Introduction **1**

1.1 Q-learning Algorithm 1

1.2 Implementation 2

2 Result Display **3**

2.1 Learning Progress 3

2.2 Final Q-table 4

Chapter 1

Introduction

In the project, **Q-learning** method is implemented to train to find the best policies for the BoxPushing grid-world game. In this report, we will make description for the Q-learning method and display the learning process as well as the final Q-table.

1.1 Q-learning Algorithm

Q-Learning is a value-based algorithm in the reinforcement learning algorithm. Q represents $Q(s, a)$, which is the expectation value that taking action a in state s can obtain, and the environment will offer feedback rewards according to the agent's action. The main content of the algorithm is to construct state and action into a Q-table to store the expectation value of pairs of state and action, which can be referenced to select the action that can obtain the maximum benefit according to the Q-value. During the learning process, the updating rule of Q-values is shown as below:

$$Q_{new}(S_t, A_t) = Q_{old}(S_t, A_t) + \alpha * (R_{t+1} + \gamma * \max_a Q_{old}(S_{t+1}, A_t) - Q_{old}(S_t, A_t))$$

where α is the learning rate; γ is the discount factor; t is current time step.

The whole algorithm process is shown in ***Figure1.1***: we first initialize the Q-table with zero values. For each episode, we initialize the environment and state, and then

use $\epsilon - greedy$ action policy to take action to explore the environment and update the Q-table value for each step in the episode.

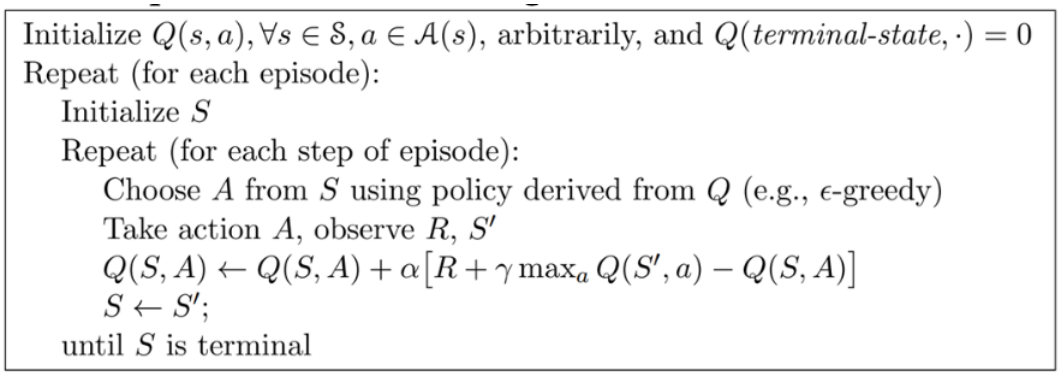

Figure 1.1: Pseudo code of Q-learning process

1.2 Implementation

According to the Q-value update equation in ***section1.1***, we implement the function in QAgent class as below:

```
def train(self, state, action, next_state, reward):
    self.Q[state][action - 1] = (1 - self.alpha) *
        self.Q[state][action - 1] + self.alpha * (reward +
            self.discount_factor * max(self.Q[next_state]))
```

Listing 1.1: Q-learning train fucntion

The episode given in the source code is 1000 which is too small for the learning process, so we reset it to 10000. After the Q-learning procedure, we can get the final Q-table value and export the data into .csv file and convert it into .xlsx file instead. Related code is in line 88 to 114 in **Q-learning.py**.

Chapter 2

Result Display

2.1 Learning Progress

During the learning process, we record the rewards for each step of the episode and after each episode we calculate the summation of step rewards. Thus we can display the plot for episodes rewards vs. episodes in ***Figure2.1***.

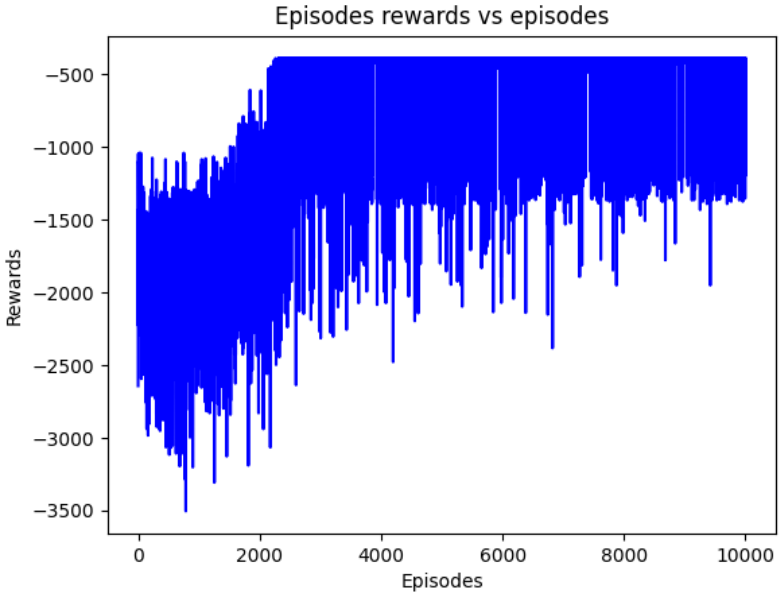

Figure 2.1: Episodes rewards vs. episodes

2.2 Final Q-table

The part of final Q-table is shown in ***Figure2.2***. Since the whole table is too big to show, we attach both the .csv and .xlsx file in the code directory. As is displayed in the Q-table, each state and action pair corresponds to an expectation value and the policy follows the rule that each step achieves max expectation value.

	STATE	ACTION				POLICY
		UP	DOWN	LEFT	RIGHT	
0	((5, 0), (4, 1))	-348.1549714	-347.3476932	-347.3475357	-335.7047686	RIGHT
1	((4, 0), (4, 1))	-339.3425603	-339.3383974	-341.6459461	-337.5959908	RIGHT
2	((3, 0), (4, 1))	-331.0450334	-341.6277413	-332.5311644	-332.2430561	UP
3	((2, 0), (4, 1))	-327.2644243	-324.5287915	-329.4794411	-329.7534058	DOWN
4	((1, 0), (4, 1))	-325.7819046	-322.2220165	-322.7324577	-321.9257403	RIGHT
5	((0, 0), (4, 1))	-319.6218916	-321.4038655	-319.6218916	-323.432491	UP
6	((5, 1), (4, 1))	-324.9543117	-335.7047635	-347.3477023	-347.3441902	UP
7	((4, 1), (3, 1))	-313.0851633	-337.7045674	-337.6944334	-337.6324072	UP
8	((3, 1), (2, 1))	-300.0860236	-326.8903989	-326.9514679	-326.9543031	UP
9	((2, 1), (1, 1))	-300.0859609	-315.0849717	-285.9454783	-314.9209641	LEFT

Figure 2.2: Final Q-table