Neural Network Theory and Applications

Final Project Report

张灿 1090379019

1. Introduction

In this project, I will solve the maze problem using a well-known reinforcement learning algorithm called **Q-learning**. Reinforcement learning algorithm, known as an unsupervised learning, has been widely used for many applications such as robotics, multi-agent system, game, and etc. Reinforcement learning usually contains an **optimal policy**, a **reward function** and a **value function**; Q-learning applies an incremental dynamic-programming procedure that determines the **optimal policy** in a step-by-step manner. Also, the maze problem presented in this report is a little different from the original one, which has only one entry and one exit. In Part Two, I will illustrate the problem in a clear and distinct way.

2. Problem Description

In the maze problem, the robot starts from one room inside the maze, and the task is to find an acceptable path to get to the outside of the maze. By *acceptable*, I mean that the path found may not be shortest path, but it will be an accept result, such as the fifth shortest path. Also, there may be more than one path that leads to the outside of the maze.

The maze depicted in Figure 1 is one example of the maze problem. As shown in Figure 1, the width of the maze is 10, and the height is 8. And the maze is composed of 80 rooms, separated from others by walls and doors. In Figure 1, the walls are drawn as black lines and the doors as two separated short lines between two rooms. In addition, the room colored green is the start point of the robot, and the robot is drawn as a red point. One acceptable result of the maze problem is shown in Figure 2.

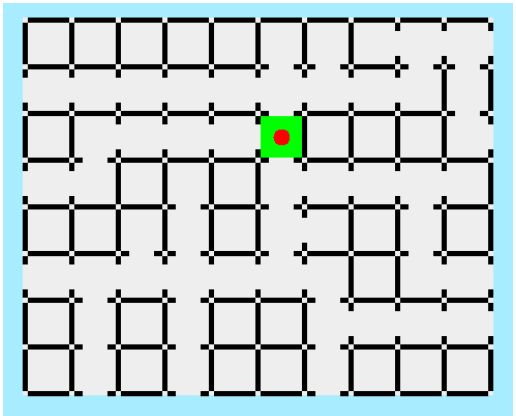

Figure 1. The maze problem

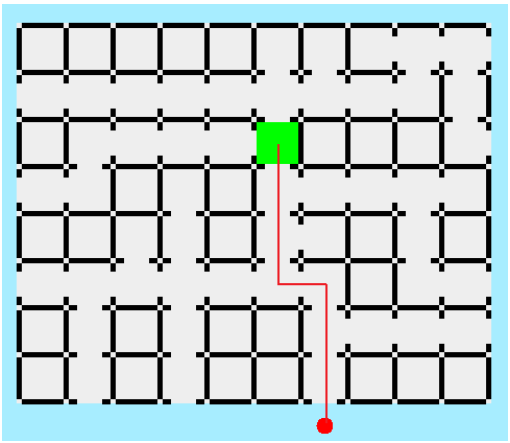

Figure 2. One result for the maze problem

3. Q-Learning Algorithm

Q-learning was first introduced by Watkins [1] in 1989, and the convergence proof has been presented later by Watkins and Dayan [2] in 1992. Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment.

In the Q-learning algorithm, the virtual agent will learn through experience without teacher (this is called unsupervised learning). The agent will explore state to state until it reaches the goal. We call each exploration as an *episode*. In one episode the agent will move from initial state until the goal state. Once the agent arrives at the goal state, program goes to the next episode.

The pseudo code of Q-learning Algorithm:

For each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$

Observe current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

Figure 3. The Q-Learning algorithm

4. Solution of the Maze Problem with Q-Learning

The whole solution for this maze problem is implemented in Java.

a) The Problem Model

The maze is modeled as two two-dimensional arrays in file *Maze.java*.

```
   // when there is a door between room[r][c] and room[r+1][c], set gridsV[r][c] as
true
   // when there is a door between room[r][c] and room[r][c + 1], set gridsH[r][c]
as true
   private boolean[][] gridsV;
   private boolean[][] gridsH;
```

Figure 4. The model of the maze

b) Elements in Q-Learning

The main implementation of the Q-learning algorithm is in the file *QLearningMazeSolver.java*. In this part, I will introduce the implementation with some key elements in the Q-learning algorithm.

i. R-values

The immediate reward values are stored in the three-dimensional array – *RMatrix*.

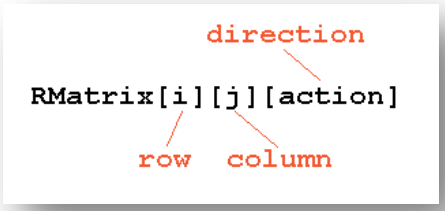

Figure 5. The RMatrix

As shown in Figure 5, the first dimension specifies *the row index* of the room and the second *the column index* and the third *the action*. For one room in the maze, the robot can only have the choice of moving through doors to the adjacent room, so the direction (namely EAST, SOUTH, WEST, and NORTH) is enough for specifying the action from current state. The RMatrix is initialized according to the following rule:

- -1 : where there's a wall
- 0 : where there's door
- 1000 : where the direction of the room is facing the outside

Thus, part of the RMatrix of the maze shown in Figure 6 will look like:

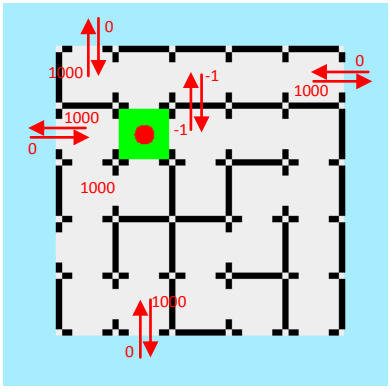

Figure 6. RMatrix for a simple maze

ii. Q-function

The Q-function for this maze problem is shown as below:

$$\mathbf{Q}(\text{state}, \text{action}) = \mathbf{R}(\text{state}, \text{action}) + \gamma \cdot \mathbf{Max}[\mathbf{Q}(\text{next state}, \text{all actions})]$$

Figure 7. Q-function for the maze problem

iii. Q-values

The q values are stored in a three-dimensional array – QMatrix. The QMatrix almost looks the same as the RMatrix. However, the QMatrix is initialized as a zero matrix, and its value will be updated with the Q-function after each iteration, while the RMatrix will stay unchanged.

At the begin, the QMatrix is full of zeros. Suppose $\gamma = 0.5$, after two iterations from *room A* to *the goal (the outside)*, the updated values in QMatrix are shown in Figure 9.

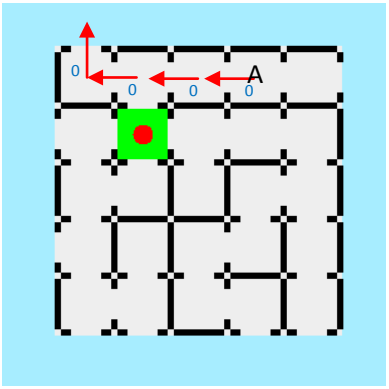

Figure 8. The initial QMatrix

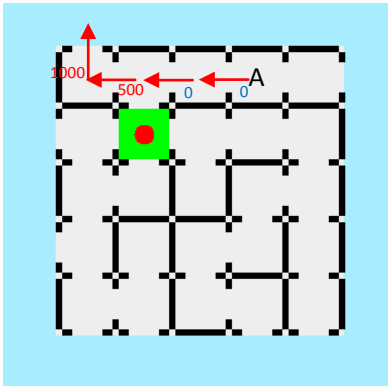

Figure 9. QMatrix after two iterations

iv. Learning Rate

The learning rate $\boldsymbol{\gamma}$ for the Q-function has range value of 0 to 1($0 \leq \gamma < 1$). If $\boldsymbol{\gamma}$ is closer to zero, the agent will tend to consider only immediate reward. If $\boldsymbol{\gamma}$ is closer to one, the agent will consider future reward with greater weight, willing to delay the reward.

5. Experiments

Results of four different experiments are illustrated in this part. In each experiment, different $\boldsymbol{\gamma}$ value will be used, and leads to different number of loops when the algorithm converges.

a) Experiment No.1

In this experiment, we deal with a simple maze depicted in Figure 10.

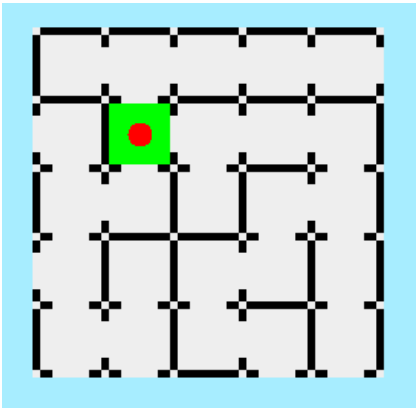

Figure 10. Maze for experiment No.1

$\boldsymbol{\gamma}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Convergence	**60**	**80**	**80**	**120**	**80**	**80**	**80**	**100**	**100**

Table 1. Convergence loops for different γ s

Solution for this maze:

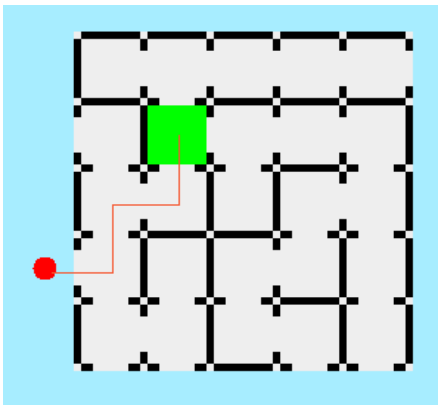

Figure 11. Solution for maze in experiment No.1

b) Experiment No.2

In this experiment, we deal with a complicated maze depicted in Figure 12.

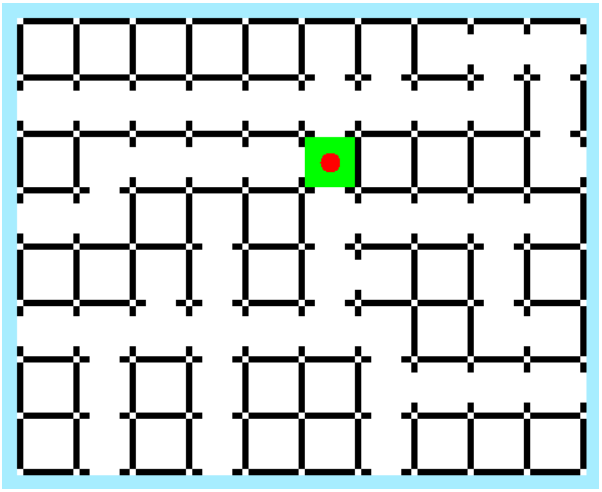

Figure 12. Maze for experiment No.2

γ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Convergence	-	-	**160**	**160**	**180**	**160**	**180**	**160**	**240**

Table 2. Convergence loops for different γ s

Solution for this maze:

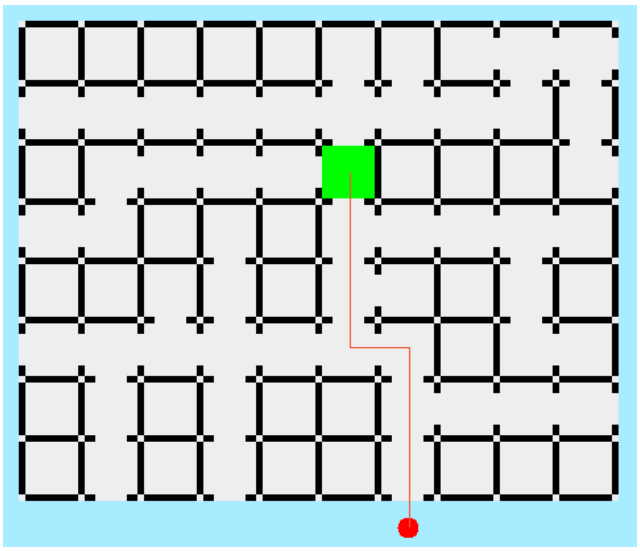

Figure 11. Solution for maze in experiment No.2

c) Experiment No.3

In this experiment, we deal with a more complicated maze depicted in Figure 14.

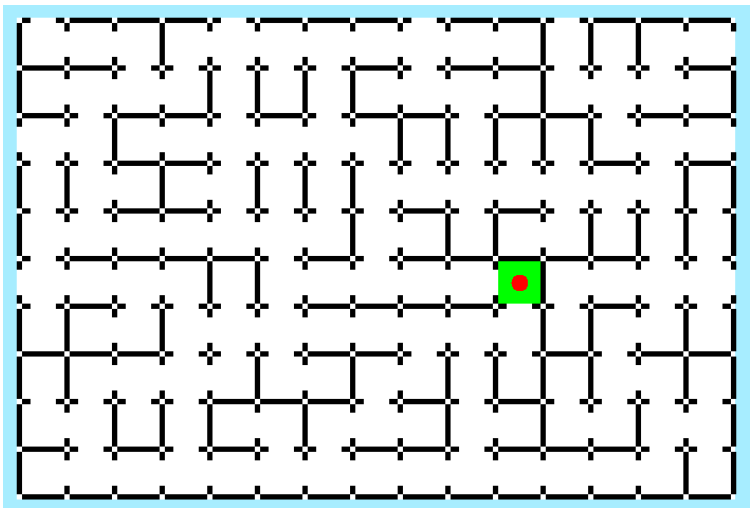

Figure 14. Maze for experiment No.3

γ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Convergence	-	-	-	-	-	-	**300**	**280**	**320**

Table 3. Convergence loops for different γ s

In Table 3, the symbol '-' represents that it cannot find a solution for the maze.

Solution for this maze:

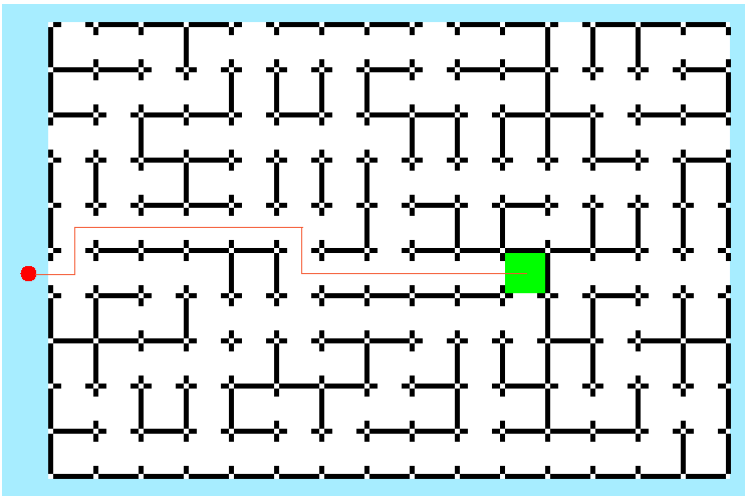

Figure 15. Solution for maze in experiment No.3

d) Experiment No.4

In this experiment, we deal with an 8*10 maze but with only one exit depicted in Figure 16.

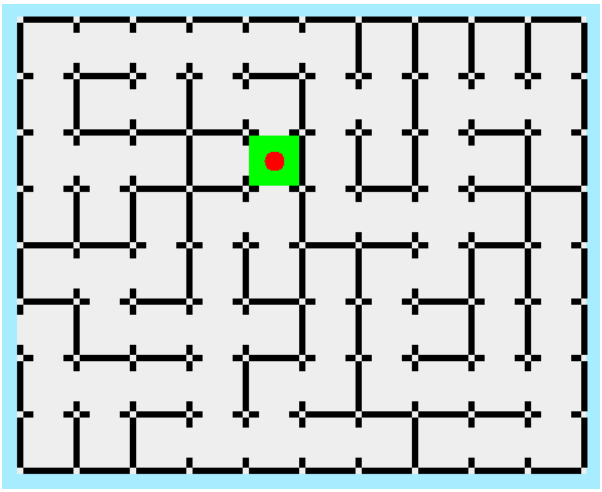

Figure 16. Maze for experiment No.4

γ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Convergence	-	-	-	-	-	-	-	-	-

Table 4. Convergence loops for different γ s

The algorithm cannot find a solution for such a maze, even it seems not as complicated as the maze depicted in Experiment No.3.

6. Conclusion

In this project, I try to solve the maze problem using Q-learning algorithm. After several experiments, I get to some conclusions concerning this maze problem:

First, Q-learning is really an effective algorithm to solve the maze problem. It converges fast even with different γ values.

Second, for some mazes (as depicted in Experiment No.2 and No.3), bigger γ values may be safer. By safer, I mean that the algorithm becomes more likely to converge with bigger γ values.

Finally, the more exits the maze has, the more likely the algorithm will converge. For those mazes who have few doors (as depicted in Experiment No.4), the algorithm may not find a solution.

Reference

[1] Watkins, C.J.C.H., (1989), Learning from Delayed Rewards. Ph.D. thesis, Cambridge University.

[2] Watkins and Dayan, C.J.C.H., (1992), 'Q-learning.Machine Learning', ISBN : 8:279-292

[3] http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/Q-Learning-Algorithm.htm