Q-Learning for World Grid Navigation

Project 2 of EE5904/ME5404 Part II

Jia Yansong
A0263119H
jiayansong@u.nus.edu

Module: EE5904/ME5404 Neural Network

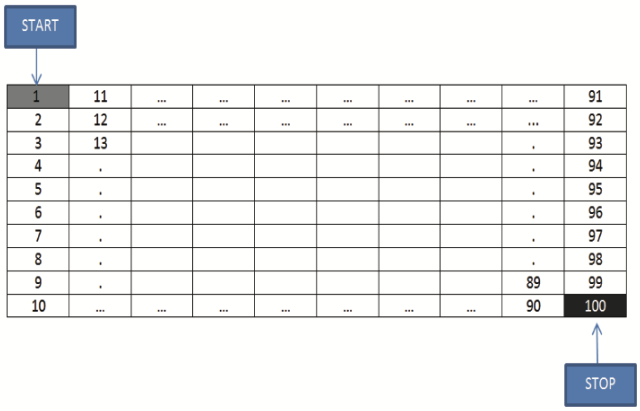
Department of Mechanical Engineering
National University of Singapore
Singapore
2023.4.11

Abstract

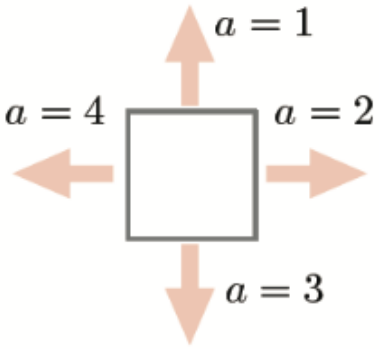
This project is the second project of EE5904/ME5404 Neural Network Part II. This project comprises two primary sub-tasks. The first sub-task involves the use of the Q-Learning algorithm to perform reinforcement learning, with the objective of identifying the optimal policy for a robot to navigate from the top left corner of a 2D grid to reach the goal state situated at the bottom right corner. In the first sub-task, various parameter sets are employed, and a reward map is provided to facilitate the learning process. We run the program 10 times for each set of parameters and set the maximum number of steps per episode to 3000. The aim of this task is to analyze and compare the outcomes (number of times the goal was achieved and average execution time) of different parameter sets. The second task involves testing our program using an unknown reward map. For this task, we use a parameter set where epsilon (exploration rate) and alpha (learning rate) are both equal to $\frac{300}{300+k}$. The code is available: https://github.com/JYS997760473/NUS-ME5404-EE5904-Projects/tree/main/RL.

1 Task Description

The task is to program a robot to move on a grid of size 10×10, starting from the top-left cell and reaching the bottom-right cell, as shown in Figure 1a.



(a) Illustration of a 10×10 world grid with a start state and goal state. The index of each cell follows the MATLAB column-wise convention.



(b) Possible actions of the robot at a given state.

Figure 1: Task description.

The objective of the robot is to optimize the total reward it receives during its journey, ultimately reaching the goal state. At any given state on the grid, the robot has four available actions, as illustrated in Figure 1b. The actions correspond to moving the robot deterministically either up (a = 1), right (a = 2), down (a = 3), or left (a = 4) to reach the adjacent state.

The learning process involves multiple trials where a robot begins at the initial state (s = 1) and moves to different states using the Q-learning algorithm with $\epsilon - greedy$ exploration until it reaches the goal state (s = 100). The trial ends once the robot reaches the goal state, and this process is repeated until the Q-function values converge to their optimal values. After convergence, an optimal policy can be derived from the Q-function values.

1.1 Task 1

The reward function is given in $task1.mat$ and the parameter sets ϵ_k, α_k, and γ are also specified. The program is run 10 times for each set of parameter values, and the number of times the goal state is reached is also recorded. The maximum number of trials in each run is 3000. The parameter sets are shown in Table 1, and we need to run the program and fulfill this table.

Table 1: Parameter values and performance of Q-Learning

ϵ_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
$\frac{1}{k}$	?	?	?	?
$\frac{100}{100+k}$	?	?	?	?
$\frac{1+log(k)}{k}$	?	?	?	?
$\frac{1+5log(k)}{k}$	?	?	?	?

1.2 Task 2

The reward function is currently unspecified, and we also should manually configure the program's parameters, ϵ_k, α_k, and γ.

2 Implementation of Task

2.1 Q-Learning

The pseudocode of the Q-Learning algorithm is shown in Figure 2.

Parameters Step size $\alpha \in (0, 1]$, and small $0 < \epsilon < 1$
$Q(s, a) = 0$ Set arbitrary initial Q values for non-terminal states
Q values are zero for all terminal states by definition

Loop for each episode
  Initialize S
  Loop for each step of episode
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A; receive R and observe S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_{a'} Q(S', a') - Q(S, A)\big]$
    $S \leftarrow S'$
until $S \in \hat{\mathcal{S}}$, with $\hat{\mathcal{S}}$ being the terminal state set

Figure 2: Pseudocode of Q-Learning.

2.2 ϵ-greedy Exploration

Action with $max\ q_\pi(s, a)$ is called the greedy action at s

$$A^* \triangleq \underset{a}{argmax}\ q_\pi(s, a)\ \in A(s) \tag{1}$$

$\epsilon-$greedy policy is

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{for } a = A^* \\ \frac{\epsilon}{|A(s)|} & \text{for } a \neq A^* \end{cases} \tag{2}$$

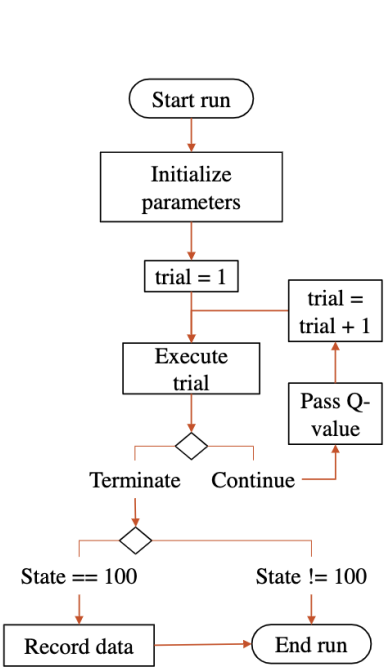
where ϵ is the hyperparameter that controls the balance between exploration and exploitation during the learning process.

2.3 Task 1

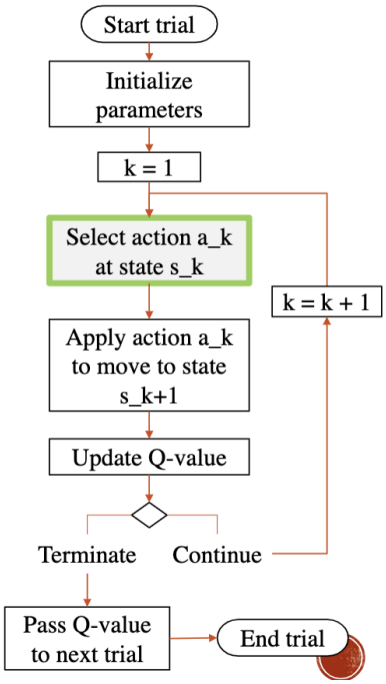
The main program of task 1 is named *RL_task1.m*. According to the requirement of the project, the maximum number of trials of each run is set to 3000, and the number of run times is set to 10. For each run, it calls a helper function file named *Qlearning.m* with four input variables: *oldQtable*, *reward*, *epsilon_type*, and *gamma* which returns six variables: *reach_goal*, *reachOpt*, *execution_time*, *newQtable*, *newTrials* and *totalReward*, which represent whether this run reaches the goal state, whether this run obtains an optimal policy, the execution time of this run, updated Q-table, number of trials in this run, and maximum reward get in this run respectively.

2.3.1 Flow Chart

The flow chart of task 1 is shown in Figure 3, the flow chat of each run is shown in Figure 3a, and the flow chart of each trail in each run is shown in Figure 3b.



(a) Flow chart of each run.



(b) .

Figure 3: Flow chart of each trail.

2.3.2 Execution Time

In the *Qlearning.m* function file, the MATLAB built-in function *tic* and *toc* are used to calculate the execution time of each run.

2.3.3 Termination Condition

The termination condition for each trial is either when the robot reaches the goal state of $s_k = 100$ or when the learning rate α_k falls below a threshold of 0.005. This threshold is used because if the learning rate becomes too small, it will have no significant impact on updating the robot's performance.

The termination condition for each run is either the Q-table converges to the optimal values or the maximum number of trials is reached. A threshold value of 0.05 is utilized to determine whether the difference between two Q-tables has decreased to a level that can be considered indicative of convergence towards an optimal policy. This threshold serves as a criterion for assessing the degree of similarity between the Q-tables, with values below this threshold being indicative of a high degree of similarity or convergence.

2.3.4 Render Optimal Policy and Optimal Path

The render function file for rendering optimal policy is named *drawOptPolicy.m* and that for rendering optimal path is named *drawOptPath.m*. These two files are already implemented in the *RL_task1.m* file.

2.4 Task 2

All the parts of the algorithm and the implementation are the same as task 1 except the designed parameters, ϵ_k, α_k, and γ. The MATLAB file of Task 2 is named *RL_main.m*.

2.4.1 Designed Parameters

After testing all the sets of parameters in Table 1, I find that $gamma = 0.9$, $\epsilon_k = \alpha_k = \frac{100}{100+k}$ is the best, so the parameter set designed by myself is $gamma = 0.9$, $\epsilon_k = \alpha_k = \frac{300}{300+k}$.

3 Results and Analysis

3.1 Task 1

3.1.1 Results

After running the program for all the sets of parameters, the results of Task 1 are shown in Table 2.

Table 2: Parameter values and performance of Q-Learning

ϵ_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
$\frac{1}{k}$	1	1	1.9469	1.8029
$\frac{100}{100+k}$	10	10	0.0647	0.1118
$\frac{1+log(k)}{k}$	10	4	1.1632	9.4301
$\frac{1+5log(k)}{k}$	10	10	0.1128	0.5591

From Table 2 we can see that when the ϵ_k and α_k are equal to $\frac{1}{k}$, the numbers of goal-reached runs are both only 1, means that this set of parameters is not able to reach the optimal policy.

When the ϵ_k and α_k are equal to $\frac{100}{100+k}$, the numbers of goal-reached runs are both 10, and the execution time of $\gamma = 0.5$ and $\gamma = 0.9$ are $0.0647s$ and $0.1118s$ respectively.

When the ϵ_k and α_k are equal to $\frac{1+log(k)}{k}$, the numbers of goal-reached runs of $\gamma = 0.5$ and $\gamma = 0.9$ are 10 and 4 respectively, and the execution time of $\gamma = 0.5$ and $\gamma = 0.9$ are $1.1632s$ and $9.4301s$ respectively.

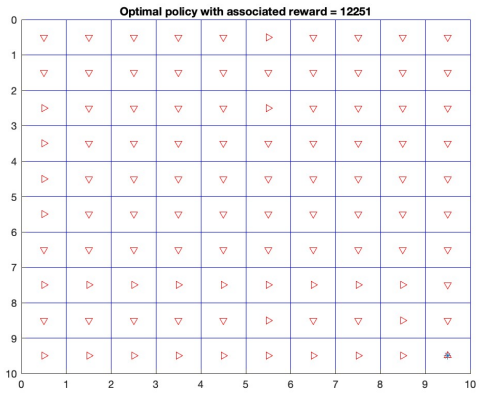
When the ϵ_k and α_k are equal to $\frac{1+5log(k)}{k}$, the numbers of goal-reached runs of $\gamma = 0.5$ and $\gamma = 0.9$ are both 10, and the execution time of $\gamma = 0.5$ and $\gamma = 0.9$ are $0.1128s$ and $0.5591s$ respectively.

3.1.2 Results and Analysis

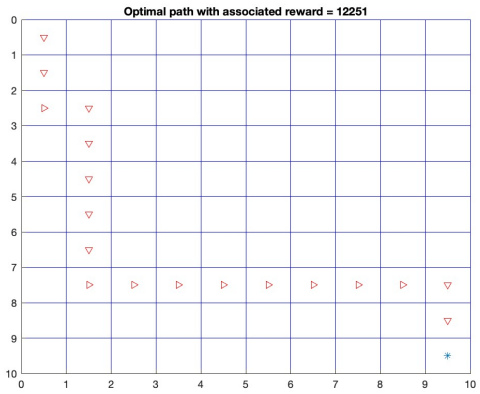
Although every set of parameters can reach the goal state, and some sets can reach the goal in every run, it does not mean that all these set reaches the global optimal policy.

After rendering the optimal policy map, it is found that only two sets of parameters reach the global optimal policy: $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.9$ and $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$ $\gamma = 0.9$.

The optimal policy and optimal path of $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.9$ is shown in Figure 4.



(a) Optimal Policy.

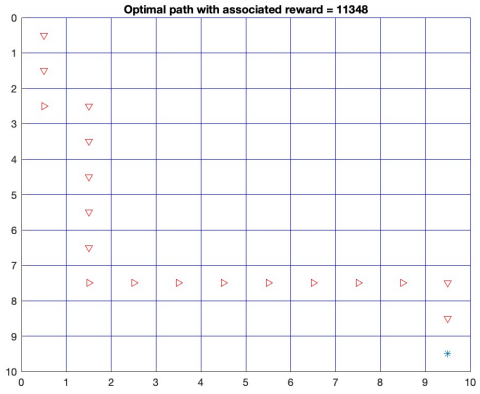


(b) Optimal Path.

Figure 4: Optimal Policy and Optimal Path of $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.9$.

The optimal policy and optimal path of $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$ $\gamma = 0.9$ is shown in Figure 5.

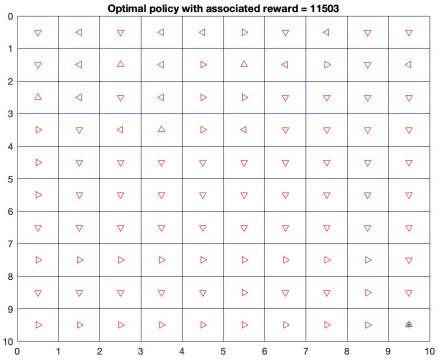
(a) Optimal Policy.



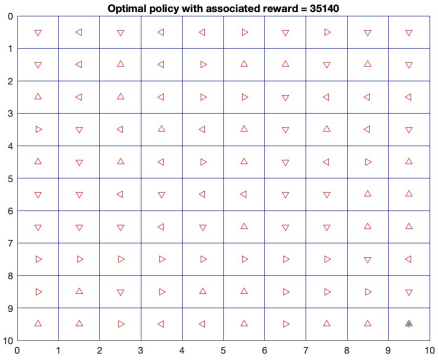
(b) Optimal Path.

Figure 5: Optimal Policy and Optimal Path of $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$ $\gamma = 0.9$.

Other sets of parameters can only reach the local optimal policy. For example, the optimal policy of $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$ $\gamma = 0.9$ is shown in Figure 6a and the optimal policy of $\epsilon_k = \alpha_k = \frac{1+log(k)}{k}$ $\gamma = 0.5$ is shown in Figure 6b.



(a) Optimal policy of $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.5$.



(b) Optimal policy of $\epsilon_k = \alpha_k = \frac{1+log(k)}{k}$ $\gamma = 0.5$.

Figure 6: Optimal policies of $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.5$ and $\epsilon_k = \alpha_k = \frac{1+log(k)}{k}$ $\gamma = 0.5$.

From Figure 6a and Figure 6b we can see that the third state's optimal action are both up, it can be concluded that though these two sets of parameters are able to reach the goal state and the optimal policy, they do not reach the global optimal policy but the local optimal policy.

3.2 Task 2

From Task 1's results, it can be concluded that $gamma = 0.9$ $\epsilon_k = \alpha_k = \frac{100}{100+k}$ is the best set of parameters because it can reach the global optimal policy and its execution time is less than that of $gamma = 0.9$ $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$.

I plan to maintain a discount rate γ of 0.9 but increase the values of ϵ_k and α_k slightly to improve the performance of the algorithm. The ϵ_k and α_k are set to:

$$\epsilon_k = \alpha_k = \frac{300}{300+k} \tag{3}$$

After creating a dummy *qevalreward* reward function and testing it, the result is better than any results of the sets of parameters in Task 1.

4 Summary

In this project, I run the Q-Learning algorithm for different sets of parameters and compare their results, finally find that only two sets of parameters reach the global optimal policy: $\epsilon_k = \alpha_k = \frac{100}{100+k}$ $\gamma = 0.9$ and $\epsilon_k = \alpha_k = \frac{1+5log(k)}{k}$ $\gamma = 0.9$.

Then in Task 2, I design the parameters to

$$\epsilon_k = \alpha_k = \frac{300}{300+k},\ \gamma = 0.9 \tag{4}$$

, by creating a dummy reward function *qevalreward*, it is found that the result of these designed parameters is better than any of those in Task 1.