EE5904R: Neural Networks

Q-Learning for World Grid Navigation

CHI JI

A0138141N

E0001795@u.nus.edu

National University of Singapore

1. IMPLEMENTATION & README

The task 1 consists of 4 main parts, as implemented in *task1_mian.m*, *Q_Learning.m, Plot_trajectory.m* and *Dynamic_programming.m* respectively.

For ***Q_Learning.m***, this function is to do the Q-learning process.

1) initialize the parameters.

2) For each trial: select action, apply action and then update Q-value matrix.

3) Continue training until the terminates condition is fulfilled.

For ***Plot_trajectory.m,*** this function is to extract optimal policy and then plot trajectory of the optimal policy.

For ***task1_mian.m,*** this script is the main script for above-mentioned two functions, and it calls them to execute and receive some values from them, such as the Q-value matix Qo and the total execution time. All output trajectories will be saved to the folder called '/result' and each figure will be named by the information such as its execution time, total reward etc. For example, 'Tpye2~870ms~54256r.png'.

For ***Dynamic_programming.m,*** this function is to verify the results given by Q-Learning algorithm using dynamic programming method, the result of this function will also be shown in RESULTS & ANALYSIS part.

The main script for task 2 is named as ***RL_main.m,*** you can run it for testing the performance of my code. The plot figure will be shown to you after running, and it will also be saved to '/result' folder named with the information about execution time and total reward. For example, 'qevaltest2~17246ms~54256reward.png'.

2. RESULTS & ANALYSIS

2.1 TASK 1

After running the scripts, we have results as follows.

ϵ_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
$\frac{1}{k}$	0	0	N.A	N.A
$\frac{100}{100+k}$	0	10	N.A	8.4
$\frac{1+\log(k)}{k}$	0	0	N.A	N.A
$\frac{1+5\log(k)}{k}$	0	4	N.A	19.2

For ϵ_k as well as α_k equals to decay function $\frac{100}{100+k}$ and γ equals to 0.9, trajectory of the optimal policy is as follow. In this case, the total reward is 54257.

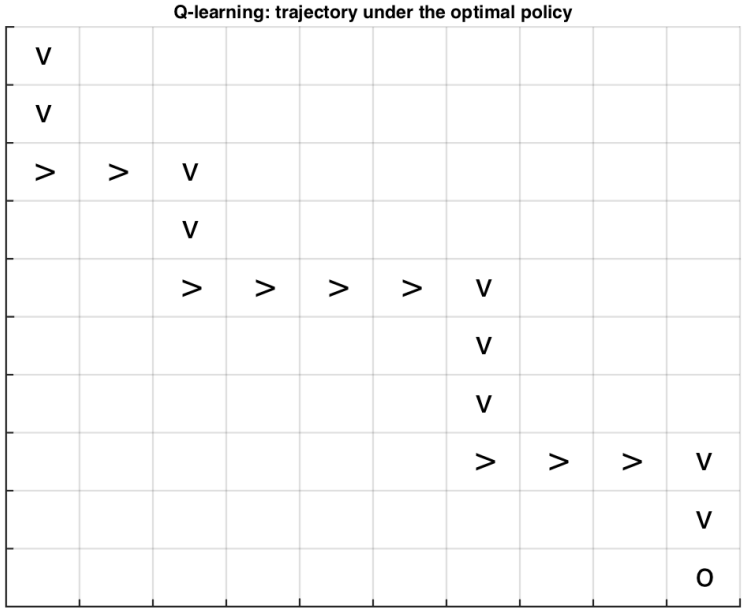


Fig.1 trajectory of the optimal policy ($\epsilon_k = \alpha_k = \frac{100}{100+k}$, $\gamma = 0.9$)

For ϵ_k as well as α_k equals to decay function $\frac{1+5\log(k)}{k}$ and γ equals to 0.9, trajectory of the optimal policy is as follow. In this case, the total reward is 54257.

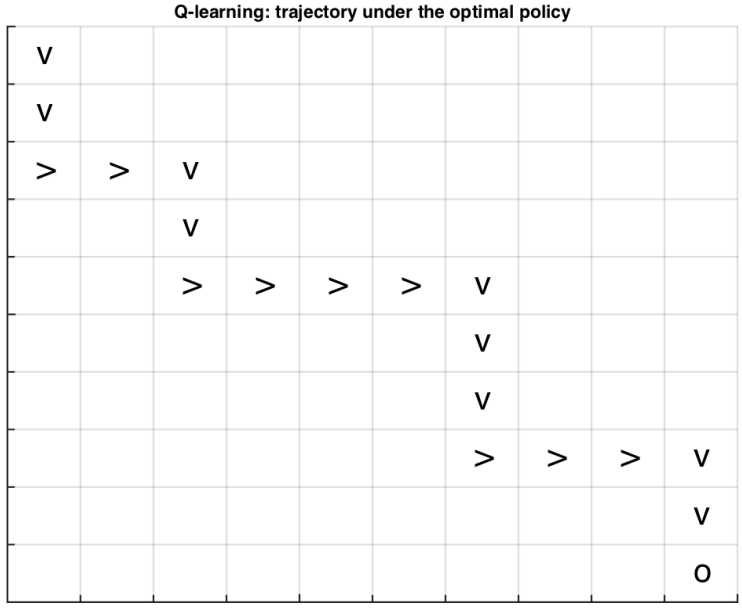


Fig.2 trajectory of the optimal policy ($\epsilon_k = \alpha_k = \frac{1+5\log(k)}{k}$, $\gamma = 0.9$)

For **dynamic programming** method with γ equaling to 0.9, trajectory of the optimal policy is as follow. In this case, the total reward is 236818, and the total execution time is around 1.3s.

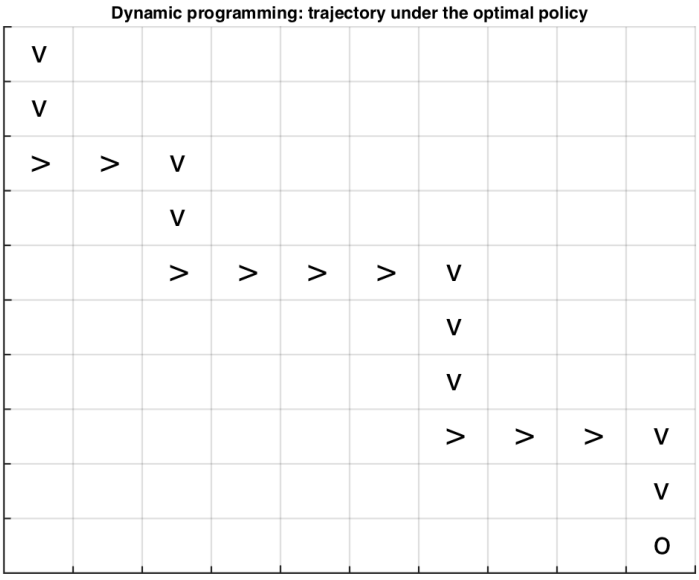
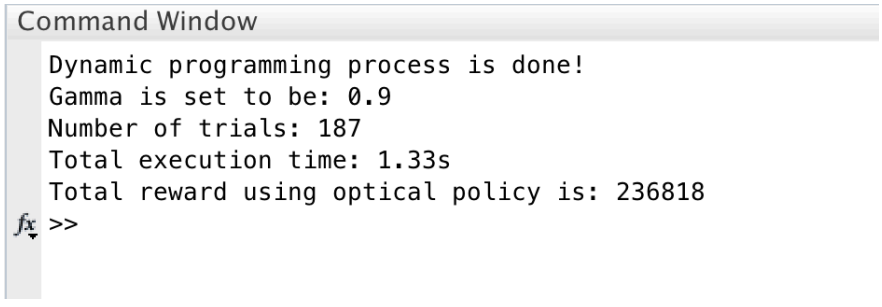


Fig.3 trajectory of the optimal policy (**dynamic programming**, $\gamma = 0.9$)

2.2 TASK 2

As mentioned in README part, you can run the script named '***RL_main.m***' for tset my code. The plot figure will be shown to you after running, and it will also be saved to '/result' folder named with the information about execution time and total reward. For example, 'qevaltest2~17246ms~54256reward.png'. The parameters/decay functions are set as: $\epsilon_k = \alpha_k = \frac{100}{100+k}$, $\gamma = 0.9$. For the reason I choose above-mentioned value and function, please refer to the ANALYSIS part. And the command window will show some detailed information as well. Below is an example.



2.3 ANALYSIS

Firstly, I'll talk about the impact of alpha and epsilon.

When learning rate α_k is smaller than 0.005, we assume that the update/change in Q-value matrix will become very tiny for every trial, so we break from that trial and go into next new trial. Hence, the choice of α_k determines the maximum steps in a single trial.

As for $\in_k$, it is the probability that we choose a direction with non-maximum Q-value/reward, this can be simply explained as the probability of exploration on every step in a trial. Hence the choice of $\in_k$ determines the 'degree' or 'percentage' for the exploration steps compared with exploitation steps, under the circumstance that the maximum number of steps in a trial is already determined by α_k.

Four types of decay functions are shown below, with the function's maximum trail-break steps and the number of steps that $\in$-greedy algorithm has a larger probability to explore rather than to exploit.

```
%Functions              % Maximum trail-break steps   %Exploration steps*
Types{1,1}='f=1/k';             % k(f<=0.005)=20,          k(f>=0.5)=2
Types{1,2}='f=100/(100+k)';    %k(f<=0.005)~=2000,        k(f>=0.5)=100
Types{1,3}='f=(1+log(k))/k';    % k(f<=0.005)~=1500,      k(f>=0.5)=5
Types{1,4}='f=(1+5*log(k))/k'; % k(f<=0.005)~=10000,     k(f>=0.5)=38

NOTE: Exploration steps* means the number of steps that epsilon-greedy
algorithm has larger probability to explore rather than exploit
```

Table.1 four types of decay functions with their characristics

As can be seen from the table, when we choose function 1 or 3, the number of exploration steps in every trial is very small, which leads to a typical error as shown below.

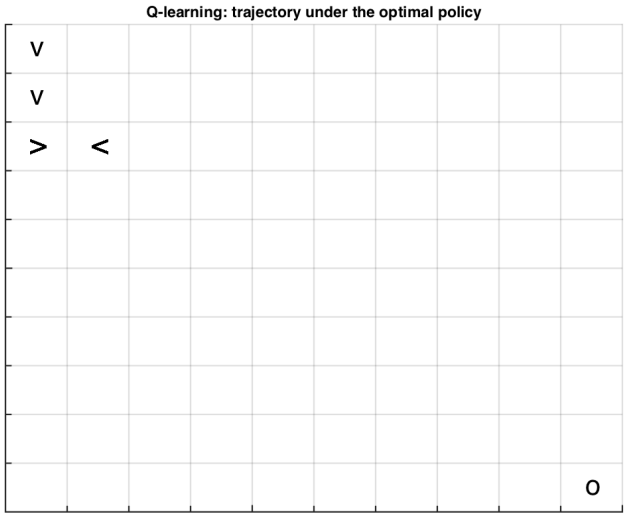


Fig.4 a typical wrong trajectory with the 'optimal' policy ($\in_k = \alpha_k = \frac{1}{k}, \gamma = 0.9$)

The reason for this error is that the training process is lack of exploration, hence the change/update on the value of Q-value matrix is mainly determined by a greedy policy. Just like 3->13->3->13…, the learning process has little chance to break out if the value of $\in$ drop significantly within a few steps. By contrast, a larger $\in$ here will give the algorithm much more chances to break out this infinite loops. Hence when we try function 1, 3, the algorithm never worked out any optimal policy.

Secondly, I'll talk about the influence of the discounted factor γ. After several runs of experiment, I find that there was no optimal policy generated when γ set to be 0.5, even using the dynamical programming. And under this circumstance, the typical error as shown in Fig.4 also occurred.

The reason for the error or the bad result is that, a small γ forces agent focus more on immediate reward from next few steps and heavily discount rewards from future steps. While a large γ forces agent to take into account future rewards more strongly, i.e., agent becomes more farsighted. Let's see the original rewards matrix, as shown in Fig.5,

	1	2	3	4
1	-1	1	44	-1
2	6	1	230	-1
3	31	45	1	-1
4	225	1	36	-1
5	1	1	31	-1
6	36	7	42	-1
7	33	48	45	-1
8	3	34	51	-1
9	5	25	2	-1
10	17	33	-1	-1
11	-1	29	1	51
12	1	1	51	28
13	1	229	1	237

Fig.5 part of the original reward matrix

The original reward value of state 3 direction 2, namely (3,2), is 45, and (13,4) is 237, which is larger than (13,2). Hence, if the agent starts from state 2, the best choice is to move to state 13 with direction 2, and when it reaches state 13, the best choice is to move to state 3 with direction 4. Hence there will be infinite loops here. To solve this kind of problem, we need to set γ to a bigger value so that the agent will focus more on future reward when it come to updating Q-value matrix. And the optimal policy also shows we that after updating, Qo(13,2) will be finally larger than Qo(13,4), which demonstrates the importance of a relatively big γ.

So, when γ set to be 0.9, decay function 2 (i.e. $\frac{100}{100+k}$) generated optimal policy every time and decay function 4 (i.e. $\frac{1+5\log(k)}{k}$) generated optimal policy 4 out of 10 times.