# EE5904

# Learning for World Grid Navigation

Li Xinrui

A0279734M

[E1143547@u.nus.edu](mailto:E1143547@u.nus.edu)

## Task1 Q-learning with E-greedy algorithm

The task is to find a best way to link the steps together and walk to the destination in a 10x10 map. There are 4 actions, going up, going down, moving right and moving left, each is represented by the column. The given reward data format is a 100x4 matrix as each row indicates a grid and the elements in each row are 4 reward values of different actions at the current grid. The value -1 means that this action at current grid is blocked. Column 1 is moving left, column 2 is moving down, column 3 is moving right and column 4 is moving up.

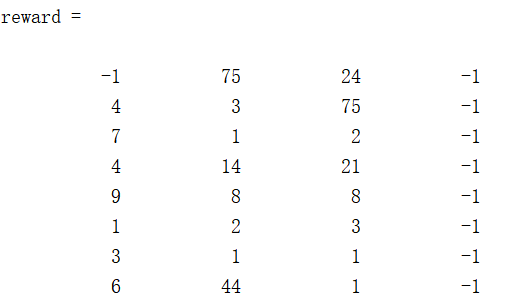


Fig. Visualization of the given reward data

It is required to run the program for 10 times and iterate the episode for maximum 3000 times. The dynamic programming model is conducted in this program. The main training algorithm follows the following flow:

**For** run =1:10

Initialize Q function

**Start timer**

**For** epoch = 1:3000

Initialize state s at the starting grid and k

**While** not reach the destination:

Update the exploration rate and Q update rate ɛk and ɑk

If ɛk very small, that is very few probability that different actions would be selected at current state, the action decision is fixed, jump to the next epoch.

Select action a for the current state s and Update Q function

Update k by 1, update s to the next state by the decided action.

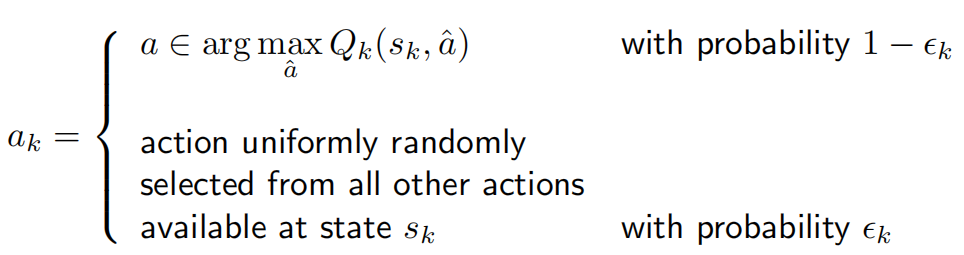
If the Q function optimization is converge, jump to the next run.

Update optimal policy and optimal results.

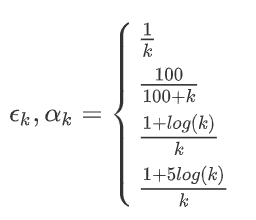
**End timer**

Check whether the destination is reached in this run, if yes, record the execution time and update the number of reaching run.

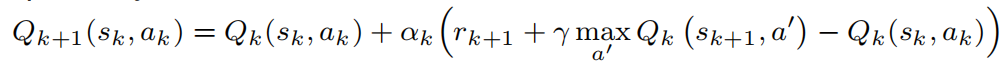
The Epsilon Greedy algorithm is a strategy that strikes a balance between exploration and exploitation. The algorithm makes a trade-off between exploring new strategies and exploiting known optimal strategies by controlling the proportion of randomly selecting actions or selecting the currently known optimal action at each decision. Exploitation: Use greedy policy to select currently known best action. Exploration: Try action other than currently known best action. It is important to take trade-off between exploration and exploitation for action a at the current state s by the following decision rule:



There are several rules to decrease Ɛk and ɑk by iteration:



In the project, a random number is generated and compared with Different equations indicate the probability for action exploration will decrease in different rates. Once the action at current state is chosen, the Q function is updated by the following equation:



There exists the possibility that the step chain may not reach the destination within one epoch. To break out of the infinite loop, the probability Ɛk is checked at each step to decide whether the walking should be paused and broken. In the project, the probability threshold is set to be 0.05, indicating that when the probability for exploration is extremely small, it is assumed that no other actions at current state would be selected, then it is time to break the current walking event and start a new walk at the beginning grid. If the current walking event is not stopped, then move to the next state by action and do the above process. Since the map is a 10x10 geometric, the state update rules is that:

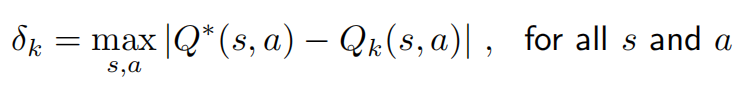
Action 1 = s-1

Action 2 = s+10

Action 3 = s+1

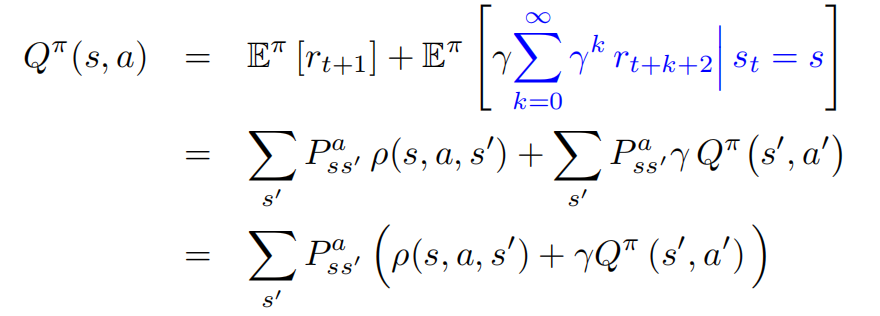
Action 4 = s-10

The difference between Q\* and Q is defined as :



It has been proved that the δk keep decreasing to 0 as the step k goes to infinity, hence value iteration of Q will converge to the optimal Q\*(s,a). in the dynamic programming, the optimal value Q\* is always the latest computed Qk+1(s,a), so δk is the maximal difference between the current computed Qk+1(s,a) and the previous computed Qk(s,a). In the project, the convergence is checked by the value of δk, if δk is smaller than the setting threshold, the value iteration converge, then the epoch iteration stops and the single run comes to an end.

After the ending of epoch, the optimal policy can be processed. Firstly, the action with maximal reward for each state is extracted and saved as a column vector, then motion of state is update by the selected action and the next action with maximal reward is chosen at the current state. The optimal policy calculation follows the Bellman Equation:



In this process, it is also checked whether the destination is reached, returning a flag. If the destination is reached, the goal-run times would be added by 1 and the execution time would be recorded.

Different parameters are set to run the model several times and the goal-reach run and execution time have been recorded.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ɛk and ɑk | No. Of goal-reach | | Execution time / s | |
| Gamma = 0.5 | Gamma = 0.9 | Gamma = 0.5 | Gamma = 0.9 |
| 1/k | 0 | 0 | nan | nan |
| 100 / 100+ k | 0 | 10 | nan | 1.288 |
| 1+log k / k | 0 | 0 | nan | nan |
| 1+5log k / k | 0 | 6 | nan | 3.968 |

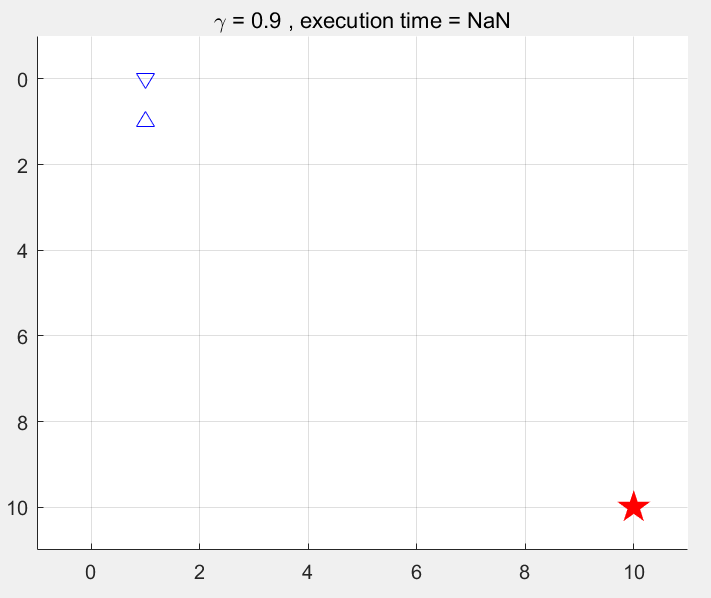
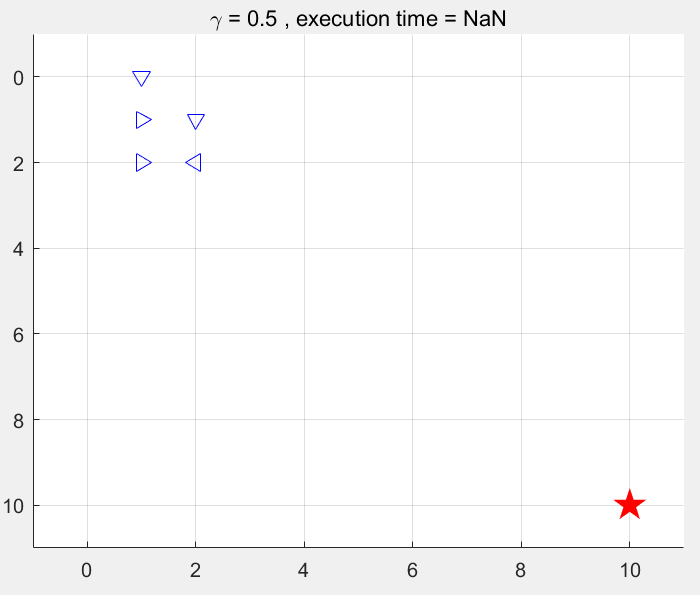


Fig. Decrease rate = 1/k, gamma = 0.5 and gamma = 0.9

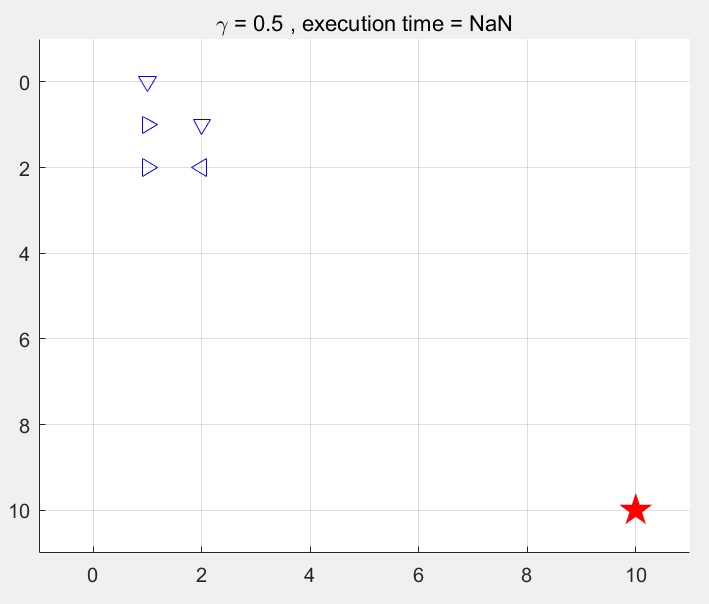
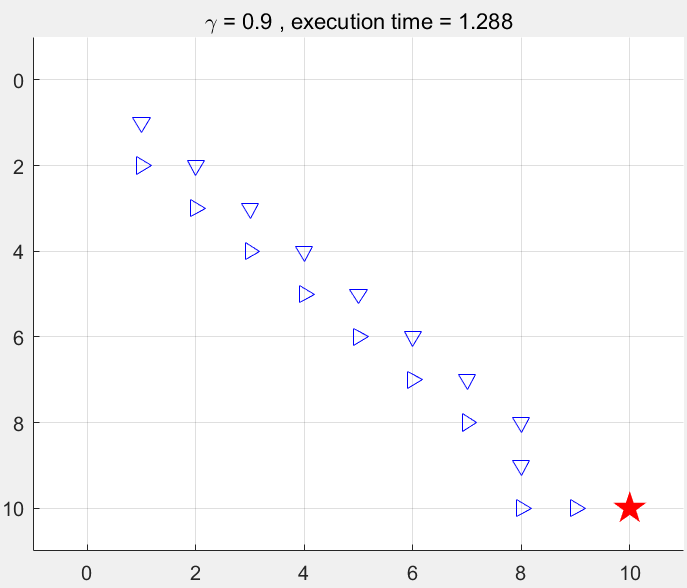


Fig. Decrease rate = 100 / (100 + k), y = 0.9 and y = 0.5

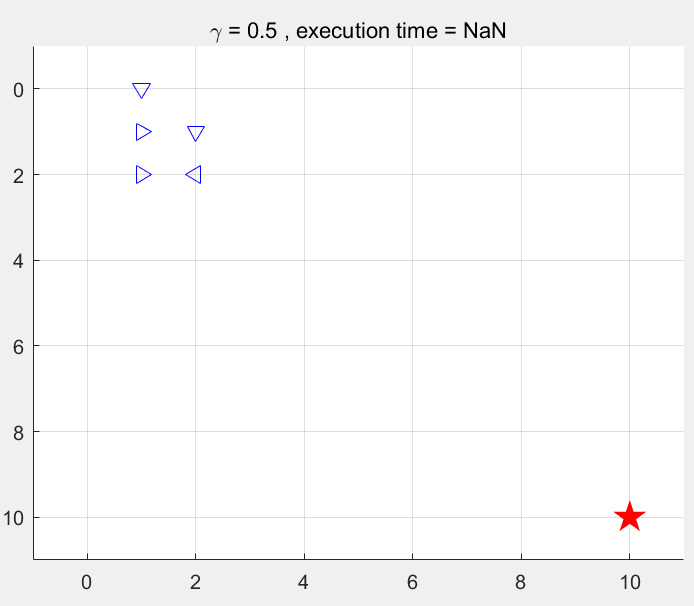
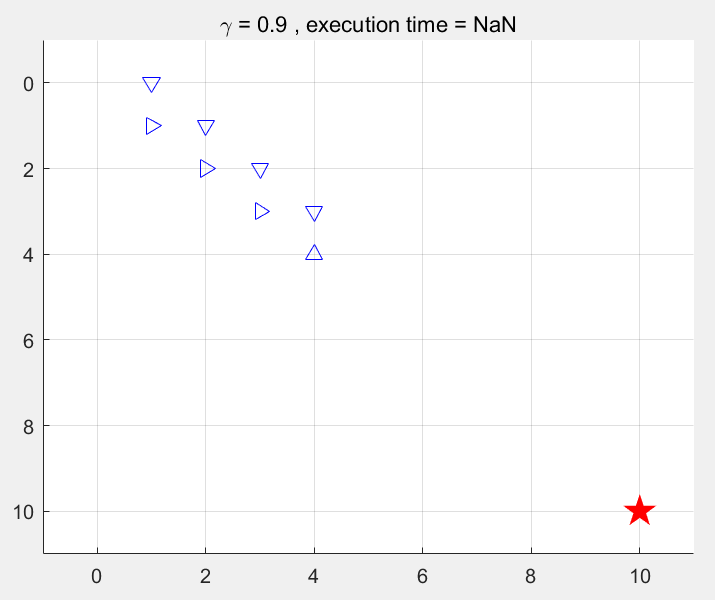


Fig. Decrease rate = 1+log(k) / k, gamma = 0.9 and 0.5

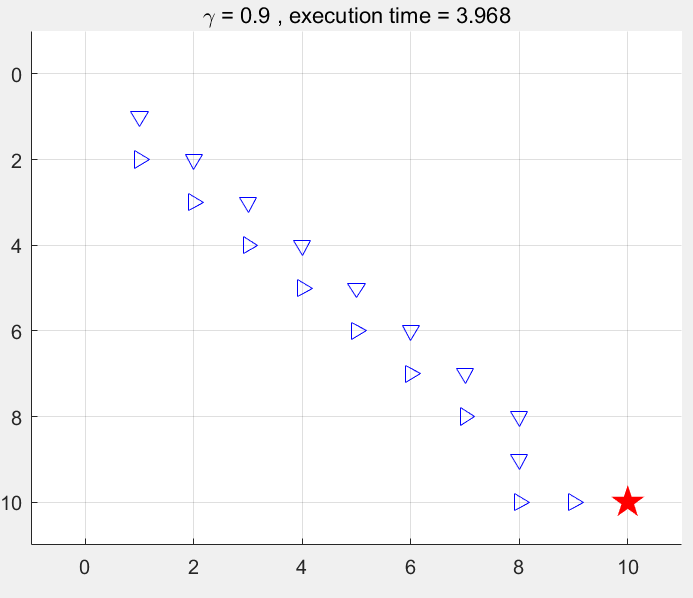
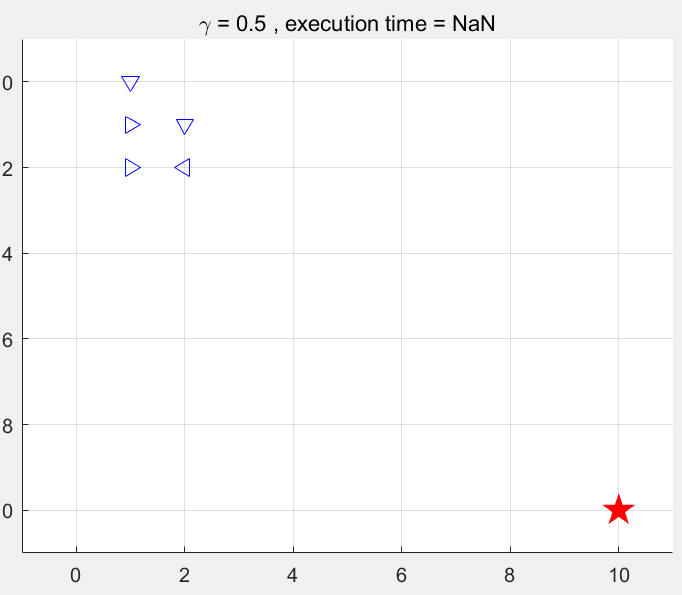


Fig. Decrease rate = = 1+ 5log(k) / k, gamma = 0.5 & 0.9

The column vector containing actions selected in named as “policy”, the result is illustrated below:

