EE5904 Part II

Project 2: Q-Learning for World Grid Navigation

Student Name : Emerson Raja Jose Peeterson

Student number: A0195021N/E0383686

Email: e0383686@u.nus.edu

Project Description:

The objective of this project is to implement a Q-learning algorithm to enable an robot/agent to traverse a 10x10 grid from the start state to Stop state following an optimal policy. The optimal policy is the set of actions to be taken from an initial start state that will maximize the total reward of the trip/run when finishing at the final stop state. Unlike other learning techniques, Q-learning only uses reward information when moving to a new state and observing the state transition for an action. Other information like state transition probabilities are not required.

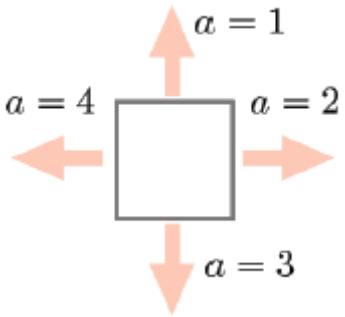


Figure 1: Illustration of possible actions from an inner state

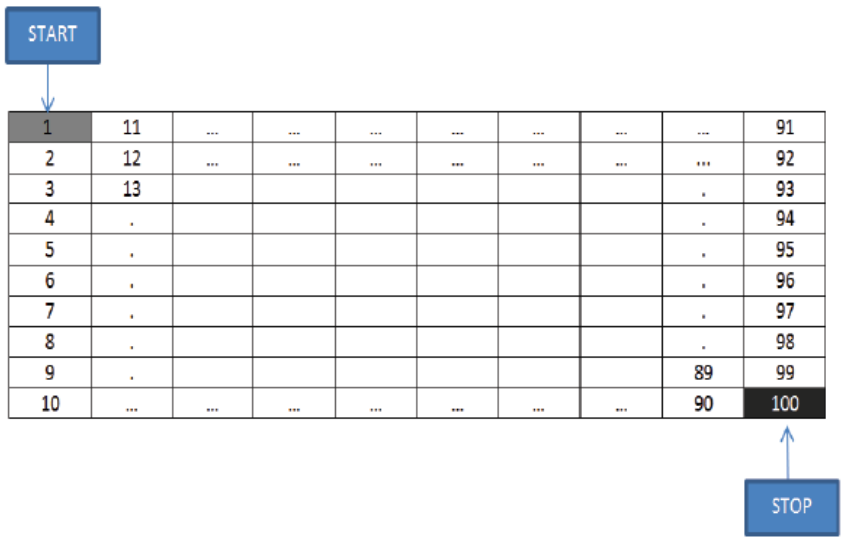


Figure 2: The 10x10 world grid view

Tasks to be performed:

Two main tasks to be completed in this project

Task 1: Using MATLAB program to implement Q-learning using the ε-greedy exploration algorithm using the reward function in task1.mat :

- For two settings of discount factor γ = 0.5 & 0.9 as specified in Table I in the results of Q-Learning section below.

- Run Matlab Q-Learning program 10 times (i.e., 10 runs) for each ε_k and α_k.

- The maximum number of trials in each run is set to 3000 i.e. maximum 3000 state transitions.

- The average program execution time of the "goal-reaching" runs is to be calculated

- Output the trajectory of optimal policy when the goal state is reached and the final reward associated with this optimal policy.

- Describe the implementation and the results.

Task 2: Experiment with a few different values of γ, ε_k and α_k and select the parameter values γ, ε_k and α_k that reach the goal in the shortest amount of execution time.

- Find an optimal policy based on the reward function specified in a file qeval.mat.
- Use the variable name "qevalreward" with dimension 100 x 4 with each column corresponding to an action and each row to a state
- generate as fast as possible a column vector named "qevalstates" that contains the states visited in the goal reached run.
- output a 10 x 10 grid showing the trajectory of the robot, along with the total reward.

Realizing Q-Learning in MATLAB

The principle of Q-Learning

The heart of Q-Learning is the update formula as shown below.

$$Q_{k+1}(s_k, a_k)$$

$$= \quad Q_k(s_k, a_k) + \alpha_k \left(\underbrace{r_{k+1} + \gamma \max_{a'} Q_k\left(s_{k+1}, a'\right)}_{\text{Estimate of } Q^*(s_k, a_k)} - Q_k(s_k, a_k) \right)$$

Where S_k is the current state, a_k is the action to take from current state. $Q_k(S_k,a_k)$ is the Q value from the previous step, k. $Q_{k+1}(S_k,a_k)$ is the updated Q value based on α_k, the learning rate, r_{k+1} reward for taking action a_k. γ is the discount factor for taking many steps, k. $Q_k(s_{k+1}, a')$ is the maximum Q value that can be possibly collected form the next state S_{k+1} and taking action a' and going to up to the terminal state or Stop state.

The Q value is iteratively updated for a state S_k and a_k. If we continue to iterate indefinitely, the Q value will converge to optimal Q value if the two following conditions are satisfied:

i) $$(1) \sum_{k=0}^{\infty} \alpha_k^2 < \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k \to \infty$$

Where α_k is the learning rate and α_0 is finite.

ii) All (S, a) pairs are (asymptotically) visited infinitely often. Agent has non-zero probability of selecting any available action in every state it visits.

In Q-Learning there are two strategies to reach the end goal or final Stop state. It is by:

1) Exploitation which uses greedy policy to select currently known best action

$$a_{k+1} = \max_{a'} Q_k\left(s_{k+1}, a'\right)$$

2) Exploration which tries action other than currently known best action

$$a_{k+1} \neq \max_{a'} Q_k\left(s_{k+1}, a'\right)$$

In this project we mainly use the Exploitation strategy to find the optimal policy which tries to maximise the returns.

The ε - Greedy pseudocode

The methodology described can be implemented through the ε-greedy algorithm below.

Inputs:

Discount factor: γ

Exploration probability; ε_k

Learning rate: α_k

Trials: 10

Function:

For Trial 1 to 10

Initialize Q-function, e.g., $Q_0 = 0$

Determine the initial state S_0 (Every trial starts from Start state = 1)

Parallel for loop for time step k, select action a_k according to:

a) Maximising the Q-value with a probability of 1- ε_k
b) uniformly randomly select an action from all other actions available at state, S_k with probability of ε_k

Apply action a_k, receive reward r_{k+1}, then observe next state S_{k+1}

Update Q-function with:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left(r_{k+1} + \gamma \max_{a'} Q_k \left(s_{k+1}, a' \right) - Q_k(s_k, a_k) \right)$$

Set k = k + 1 and repeat parallel for loop for the next time step.

Exit when trial = 10

Outputs:

Optimal Q with maximum value from start state to goal reached state (stop state) for a particular trial.

Thereby getting the optimal policy which maximises the reward.

Check the execution time for the goal reached states.

Matlab Implementation

The overview of each function in Task1 is give below. Please read the comments on source code for detailed understanding.

Task1.m

This is the main file. In this file the total reward/ return and optimal policy is calculated for the two value [0.5 0.9] of gamma and four methods of computing ε_k and α_k (also known as decay methods). The optimal policy for each of the decay methods as trajectories on a 10x10 grid. The number of goal reached runs and the execution times are output for each of the 4 decay methods.

Q_Trials.m

Starting with Q value of 0 this function carries out each run for 3000 trials or steps. Starting from state 1, a new Q value for states is calculated at the end of a trial. This function also checks if difference in Q values is small enough to be deemed reaching steady state or converged.

Calc_Q.m

Calc_Q function calculates the Q value according to the iterative update rule formula. The states and actions are initialised to 1. Several iterations are carried during which ε_k and α_k are computed using the ''calc_decay" function. The action to take is determined by the " explore_or_exploit.m" function. State transitions are captured by "transit.m" function. The Q value is updated as per the iterative update formula for Q-learning.

calc_decay.m

Calculates the ε_k and α_k learning rate for the current kth step according to the decay method.

Explore_or_exploit.m

Determines the likelihood of taking explorative actions. This function uses the negative 1 (-1) reward of actions at boundary states to determine the legitimate/legal action to take without going out of the grid. From the predefined reward matrix from task1.mat the action with max Q value is picked. If we are at the initial steps, k of the iteration there is a greater propensity to explore which is to take the any of the rewards other than the maximum reward. Therefore, the later iteration steps, k will produce the maximum total reward.

transit.m

This function converts linear index states from 1 - 100 to grid index subscript (i, j) and keeps track of transition to current state.

cal_Optim_Pol.m

Calculates the optimum policy for the given value of Q by taking the run that gave the maximum Total reward.

cal_reward.m

Sum the rewards associated with actions for all the states of the optimal policy due to a particular run.

Results of Q-Learning

Table 1: Results of Q-learning

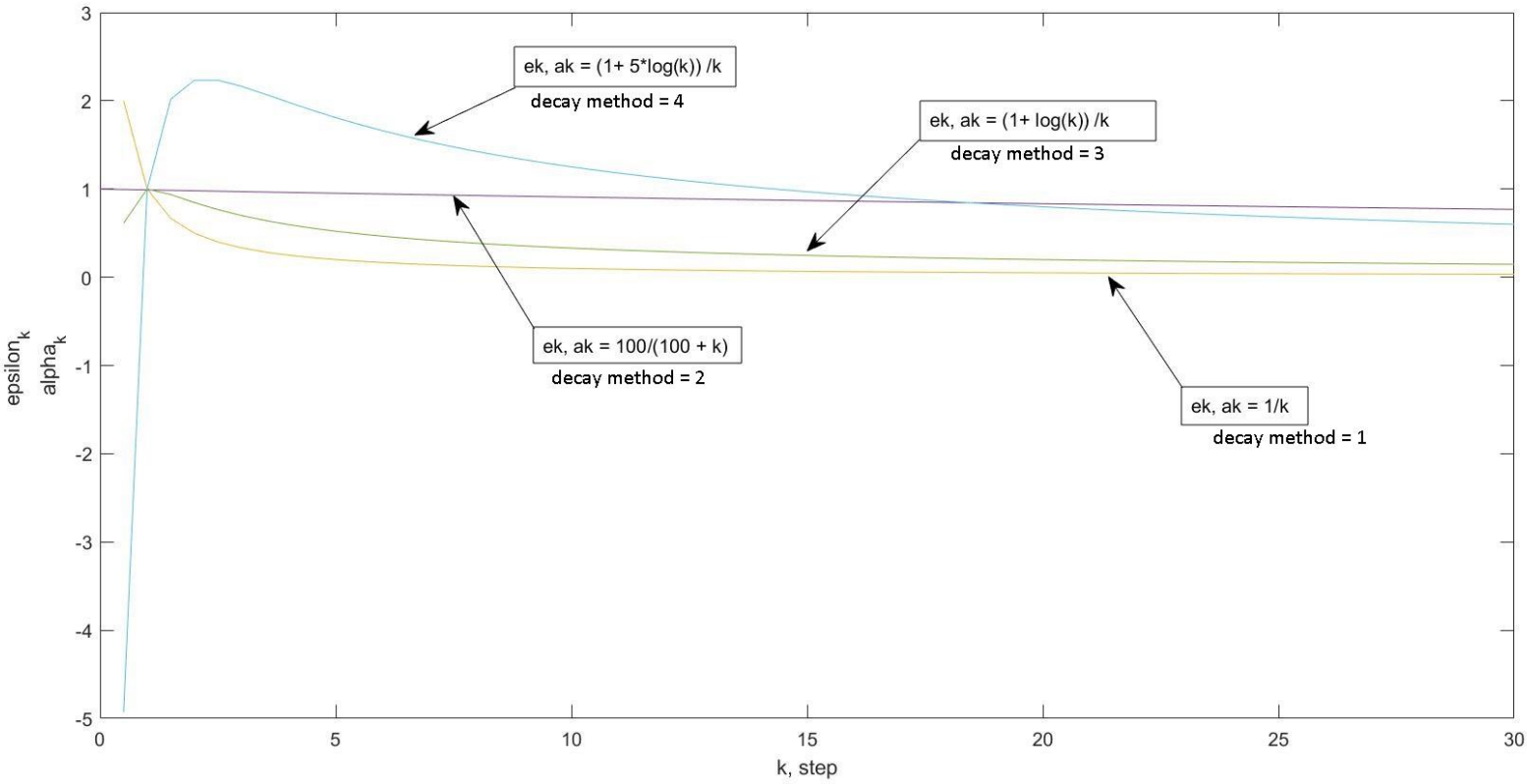
Decay Method	ε_k and α_k	No. of goal-reached runs		Execution time (sec.)	
		$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
1	$\frac{1}{k}$	0	0	Max iterations reached before reaching Stop state.	Max iterations reached before reaching Stop state.
2	$\frac{100}{100+k}$	0	10	Max iterations reached before reaching Stop state.	0.3064
3	$\frac{1+\log(k)}{k}$	0	1	Max iterations reached before reaching Stop state.	0.7451
4	$\frac{1+5*\log(k)}{k}$	0	10	Max iterations reached before reaching Stop state.	0.8058

The output from Matlab Command Window is shown below:

```
Table_1 =

         0         0       NaN       NaN
         0   10.0000       NaN    0.3064
         0    1.0000       NaN    0.7451
         0   10.0000       NaN    0.8058
```

From table 1 we can see that convergence occurs for decay methods 2,3 & 4 for gamma = 0.9. For decay method 3 convergence occurs only once whereas for 2 and 4 convergence occurs all 10 times. The results in decay methods 2 and 4 are able to find the optimal policy from all the 10 runs whereas the optimal policy for decay method 3 is from the only run that converged.

The execution times of the converged runs show that decay method 2 has the fastest average execution time compared to the decay methods 2 and 3. The graph below shows the decay Vs. steps for different decay methods. As the exploration probability, εk and learning rate, αk are the same, a balanced number between the highest and the lowest value of εk and αk from the graph above enables convergence to occur.



After about 18 steps the behaviour of decay methods 2 and 4 are similar. Decay method 1 has the lowest values so the learning rate is affected and rewards are not increased fast enough even though exploration probability is reduced. For decay method 3 the case is similar to decay method 1 but initially it is closer to decay method 2 which gives it a chance to converge during the earlier steps but if it is missed then it is difficult to converse as the function becomes similar to decay method 1 which has the least chance of convergence. For decay method 4, initially the robot spends more time exploring rather than maximising rewards so maximum final rewards are achieved in later runs.

Shown below are two other instances of robot traversing the grid.

```
Table_1 =

         0         0       NaN       NaN
         0   10.0000       NaN    0.7960
         0    1.0000       NaN    2.6211
         0   10.0000       NaN    2.0467
```

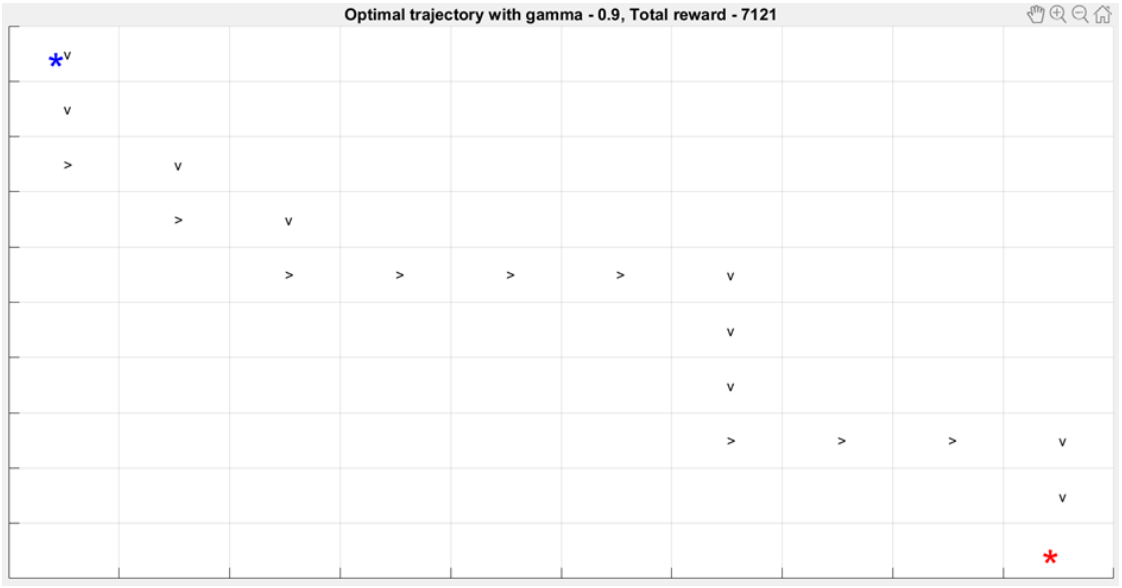
```
Table_1 =

         0    1.0000       NaN    2.0920
         0   10.0000       NaN    0.4581
         0         0       NaN       NaN
         0   10.0000       NaN    2.1707
```

The fact that convergence occurs for γ = 0.9 suggests that a myopic view for this relatively small grid of 100 states gives a better performance rather than a farsighted view of maximising reward by taking more steps.

Trajectory of optimal policy that maximises reward.

The maximum reward achieved is 7121 at the 100th state. It takes 18 state transitions or actions to move from state 0 to state 100. The best result with fastest execution time is for decay method 2 followed by the other decay methods. The trajectory is the same for the optimal policy in all 4 decay methods (as shown below) but the execution time and the number of goals reached are different.

Best Decay method, ε_k and α_k: $\frac{100}{100+k}$



Conclusion

Q-Learning allows an agent to exploit as well as explore the environment to as defined by the learning rate and exploration probability. It does not require knowledge of transitions probabilities. In this project we use rewards information to decide actions to take that maximises the total reward at the 100th state. The optimal parameters for this reward.mat was found to be decay method 2 and gamma = 0.9 which results in maximum reward of 7121 through 18 state transitions.