Brian Formento

A0196190Y

brianformento@hotmail.com

Neural Networks EE5904/ME5404

Project 2: Q-Learning for World Grid Navigation

Task 1

In this task, the Q-learning algorithm has been implemented in python. To complete this task it is required to fill in a table that highlights the algorithm's performance for different values of the discount factor, learning rate and exploration probability.

Implementation

The **.mat** file containing the reward function has been opened in python using the scipy.io library. This matrix together with the discount factor, learning rate and exploration probability were initialised in a class. The algorithm runs for 10 runs and for each run 3000 trials are done. Firstly both 'a' and the reward need to be sourced. This happens in a while loop that makes sure the reward is not -1 since this would mean the chosen action 'a' would lead to an edge. The 'a' choice is chosen either to be under an exploration or exploitation policy by:

$$a_k = \begin{cases} a \in \arg\max_{\hat{a}} Q_k(s_k, \hat{a}) & \text{with probability } 1 - \epsilon_k \\ \text{action uniformly randomly} \\ \text{selected from all other actions} \\ \text{available at state } s_k & \text{with probability } \epsilon_k \end{cases}$$

With a valid 'a' the second state can be predicted by adding or subtracting a constant number from the current state (1,-1,10,-10) as the Q-function is a list of 100 values. This movement transition is recorded for later. Now the probability of the current action 'a' also needs to be calculated using a similar process as before.

From this stage all the values needed to fill the Q-function are present. This is carried out using the following equation:

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

Where $Q_k(s_k, a_k)$ is the value of the Q-function at the previous iteration step, therefore a mirror Q-function (in the previous iteration step) needs to be accounted for and updated with $Q_{k+1}(s_k, a_k)$ straight after using equation (1).

After the above a policy is generated. This policy is optimized using a hill-climber (we keep the current best and only update until a better one is generated). Therefore the performance is additive to the current best one. Because of this trait, the algorithm is heavily dependent on the starting path and can easily get stuck in a local optimum. However when a good path

is found the algorithm works hard to optimise it. Hence if a reward of 0 is detected at the end of the 10 runs the algorithm is restarted.

The cumulative reward was calculated with the following equation:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Given the reward landscape, the algorithm should be short-sighted to maximise this reward as for example just after 1 step, a ‘230’ reward is present. Therefore the best strategy for the robot would be to take the initial step, move back and move back again to the 230 reward spot iteratively until the gain of the reward is below a certain threshold. After this, the robot should maximise the exploration to find the exit.

Results

This section will highlight the results achieved by running the algorithm.

ek, αk	**No. of goal-reached runs**		**Execution time (sec.)**	
	γ = 0.5	γ = 0.9	γ = 0.5	γ = 0.9
1/k	0	0	N/a	N/a
100/100+k	3/10	0	0.026	N/a
(1+log(k))/k	3/10	0	0.24	N/a
(1+5*log(k))/k	3/10	3/10	0.065	0.24

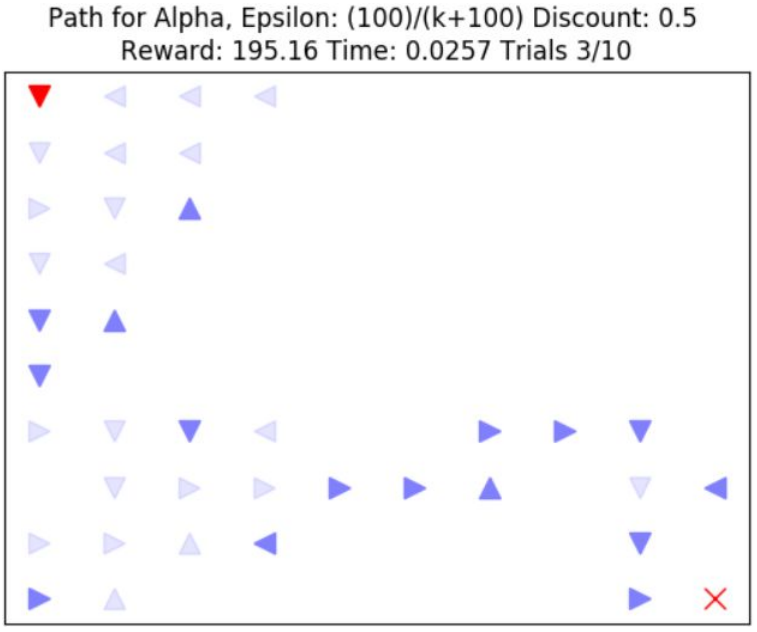
Table 1: parameter values and performance of Q-learning.

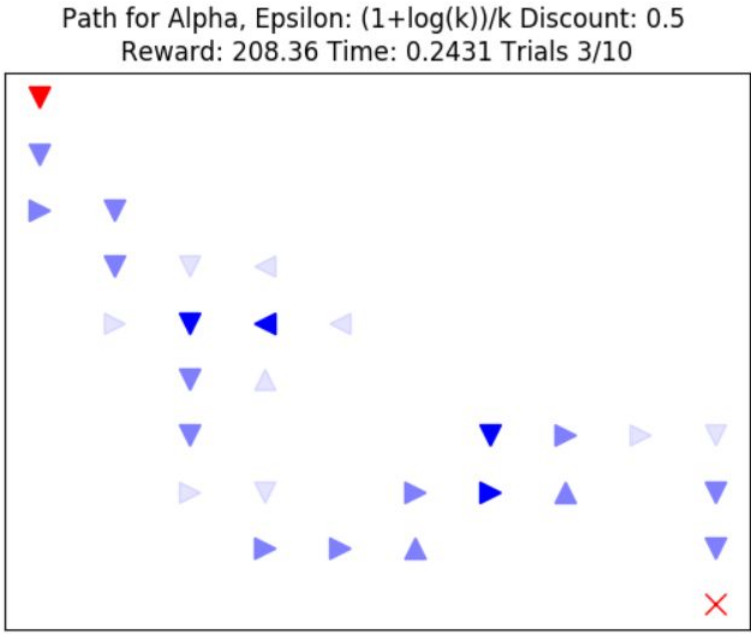
Every run has a chance of failing at first. However, there are some exploitation policies and learning rates that always fail to reach the target. This is because the probability to get an action to explore decays too quickly. Therefore the robot gets stuck in the middle of the landscape exploiting. Although this works well in terms of achieving a high reward it doesn't fit the overall goal **to reach the finishing point as quickly as possible while achieving the maximum amount of reward.**

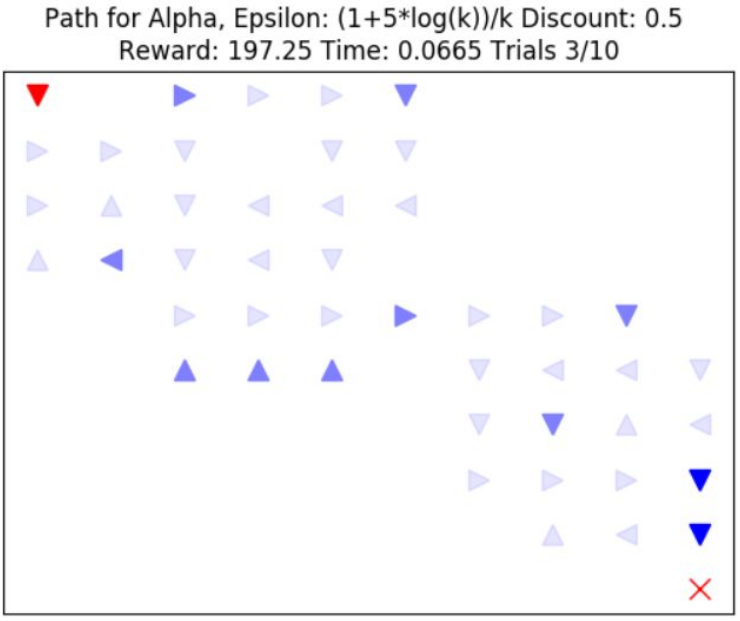
Below are listed some of the successful runs. Where it is important to note **how the results have been displayed.**

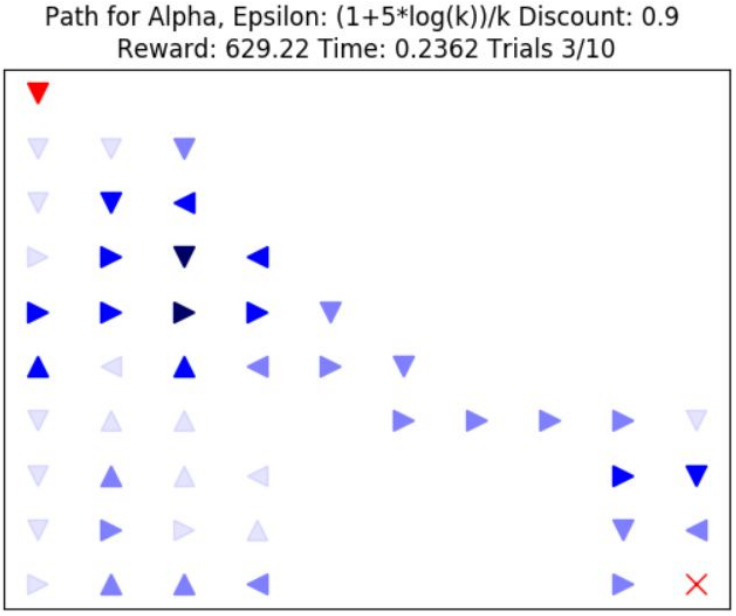
▼	Start node	Start
▼	Landed on node only once	Normal movement (exploring)
▼	Landed on node between 1 and 10 times	Exploiting while exploring
▼	Landed on node between 10 and 100 times	Medium level of exploiting
▼	Landed on node more than 100 times	Heavy level of exploitation (between adjaciant nodes)
✕	End node	End

With some effort, it is possible to follow the ultimate path that the agent has taken. It was difficult to come up with a method to show how the agent moved without showing over 1000 movements that intertwined with each other while still showing the level of exploitation occurring. In fact, it was often the case that the agent would leave a path to then re-enter at an earlier already visited node. This would have made following a path almost impossible therefore when this happens the latest node gets overwritten. Hence the directions are all connected from the start node **however this does not show the whole path the agent took** but it shows the critical path, where all other markers give intuition on the exploration path.









Conclusion

the main objective of this algorithm is to get to a certain state while maximising the reward along the way. The purpose is not to get to the state as quickly as possible or by doing the least amount of iterations. To this sense the algorithm, when using certain parameters works well. The run that achieved the best reward was using a discount factor of 0.9 and (1+5*log(k))/k for the exploration policy and learning rate. This suited this task as the exploration policy doesn't degrade too quickly and is able to almost always find the exit node. The main success and how it was able to achieve such a high reward was by exploiting early on to later move to the finishing line without worrying too much about exploiting. Obviously, this takes a longer period of time than other runs as a lot of iterations have been used to exploit the landscape.

Task 2

For this task (100)/100+k has been kept as the learning rate and exploration probability. This is because it has been found to be the most robust It might fail, but only for a couple of times. However, using (1+5*log(k))/k should output the best result. These values have been shown to achieve the best policy for the lowest amount of time. However, for the latter case, the algorithm might occasionally fail and has to be restarted, at its worst it might take 20 tyres. The algorithm might take a long time to run due to all the extra steps such as writing to a text file the 'qevalstates' and plotting an empty map. Overall it might take 5 min to run. The algorithm should also work when no go zones are placed within the grid, for example, a -1 at qeval[5][5].

The algorithm will only work if the end state is at 100.