Delft University of Technology

Computational Intelligence

CSE 2530

Assignment 3:

Reinforcement Learning

Authors:

Akdemir, Rauf

Farooq, Shah Muhammad

Khan, Zeeshan

Sahin, Tamer

2.1

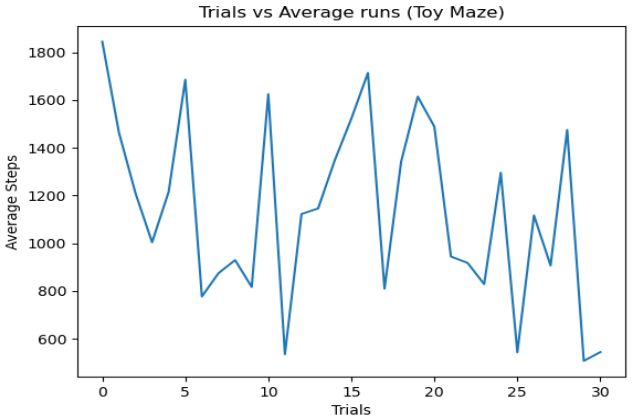
Ex 1.

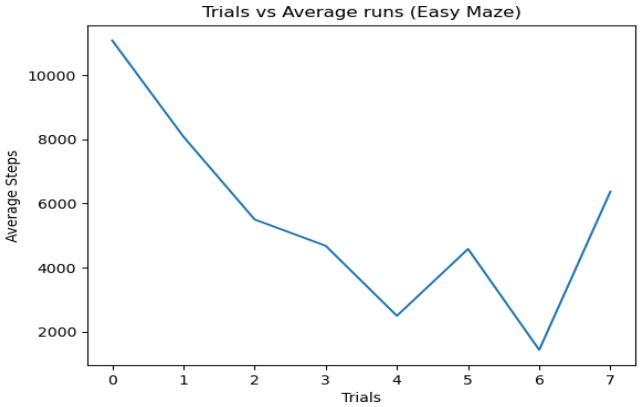
When the agent is not learning, all the action values remain zero. In our getBestAction implementation, we choose the best action from an arraylist that contains the possible actions in a state and their action values. The position of the best action in this arraylist is based on a comparison with the current maximum action value (starting out with a 0 value). If these action values are all equal to 0, this could result in the same action being selected over and over again from this list. To mitigate this risk of bias, our getBestAction algorithm selects a *random* action, as long as the action values are all zero.

Ex 2.
For the RunMe we have a while loop. Inside this loop we store the current state of the robot**.** Then the *selection.get_egreedy_action* method is called. This gives us either random action or a best action (determined probabilistically by the value of epsilon) to be performed by the robot. The algorithm then gets the next state of the robot by making it perform the action. The reward of this next state is then stored in a variable using the maze.get_reward function. The **learn.update_q function** is called to update the Q-table (Although this function does not update the Q-table yet). After all this is done, the robot checks if the reward equals the reward corresponding to the ending state. If it does, then this means a trial was completed, We then reset the robot's position and repeat the cycle. Otherwise the algorithm keeps on performing the steps mentioned above until the robot reaches the reward state.

Ex 3. See code

Ex 4.





The algorithm so far does not update the Q-table. The *get_best_action* function in MyEGreedy is implemented in such a way that if all the values in action_values are 0 then

we take a random step (see code). This means the robot is taking a random step always no matter what the value of epsilon is. This randomness is seen in graphs for both the easy maze (right) and the toy maze (left). The graph shows complete randomness in the average number of steps taken by the robot as the trials increase. It decreases and increases in a haphazard fashion.

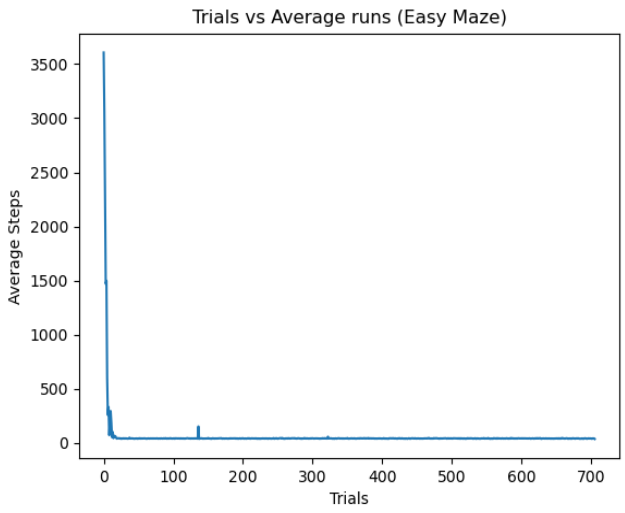
Ex 5.

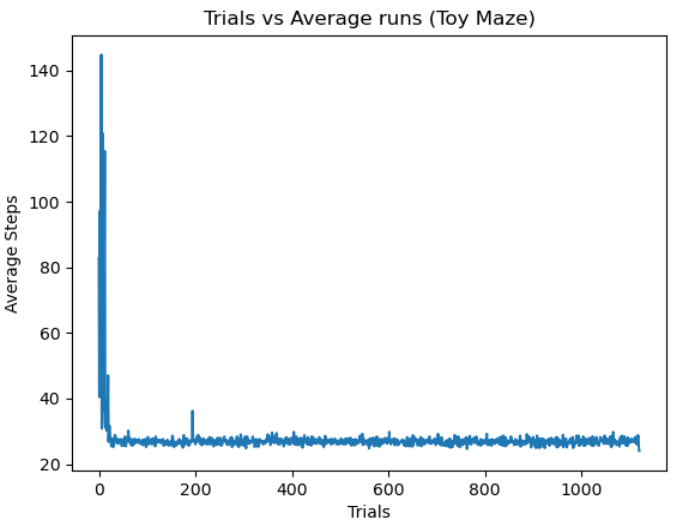
We use the following formula to update the Q-value in our Q-table:

$$Q(s,a)_{new} = Q(s,a)_{old} + \alpha(r + \gamma Q_{max}(s', a_{max}) - Q(s,a)_{old})$$

The function calculates the maximum expected future reward by using the new state(*state_next*) and all possible actions at that new state (*possible_actions*) and stores it in a variable. The new Q-value is then calculated using the above formula and the Q-table is then updated using the *self.setq* function.

Ex 6.

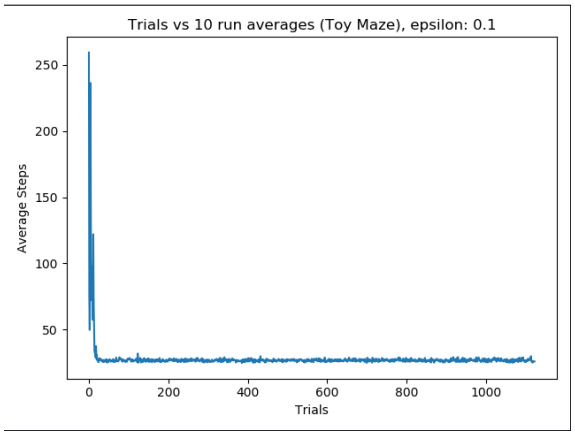


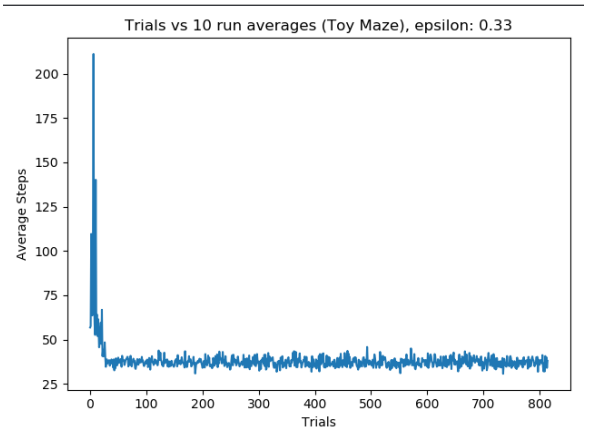


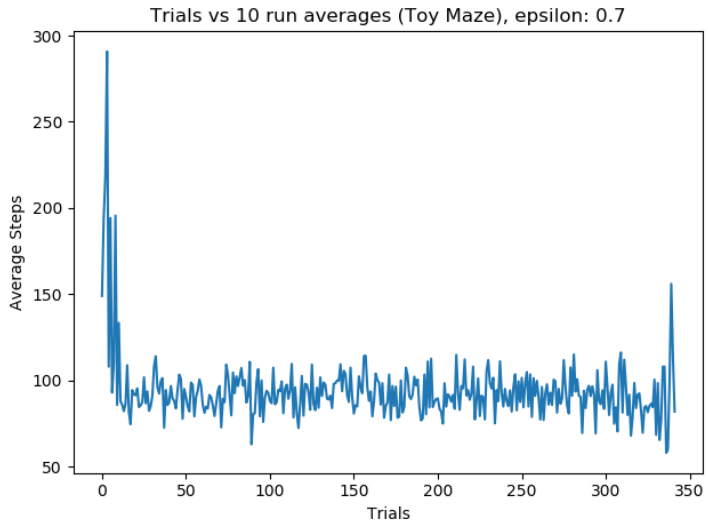
Now that we have implemented the updateQ() function. The algorithm updates the Q-table after each step taken by the robot. The robot uses this Q-table with a probability of 0.9 to take a step with the best action value. The robot also takes a random step in any direction with a probability of 0.1 sometimes. This allows the robot to exploit the best paths as well as explore new paths. The robot is thus able to learn the environment. This can be seen in the graphs for the easy maze(left) as well as the toy max (right). At earlier trials, the average number of steps is higher but as the trials increase, the robot is able to make an informed decision by making use of the Q-table and thus the number of average steps show a sharp decrease as the number of trials increase. After a certain number of trials, the average number of steps for each trial do not show any significant changes showing there is not much more improvement that could be made by updating the Q-table. The robot has found an optima (local or global).

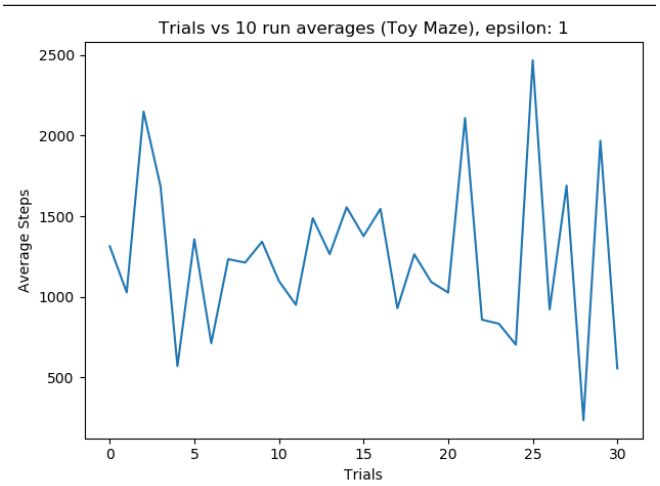
2.2

Ex 1.





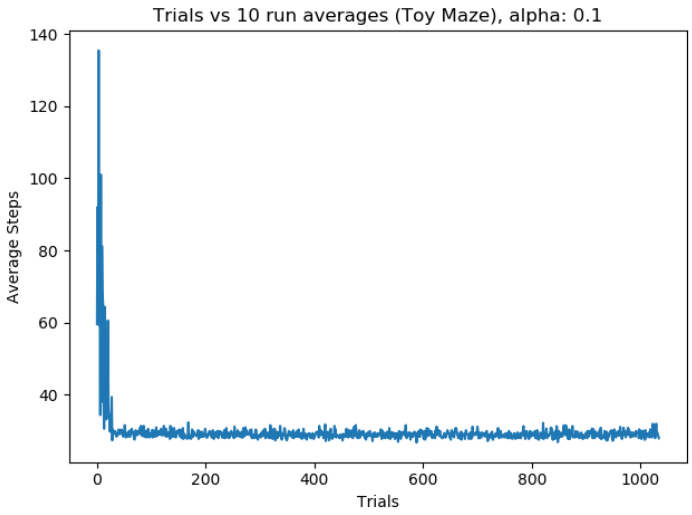


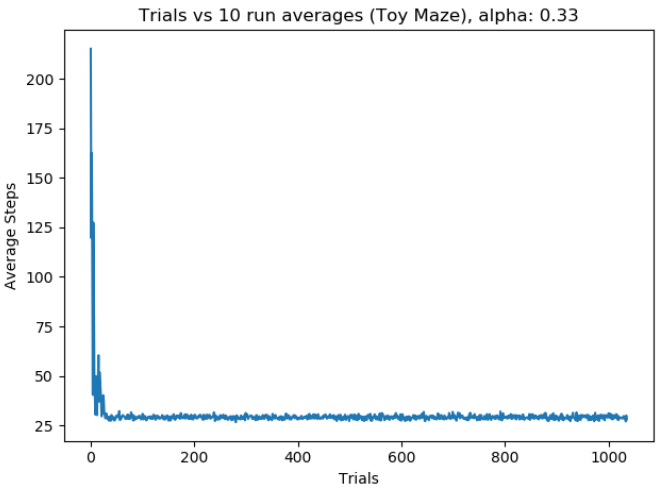


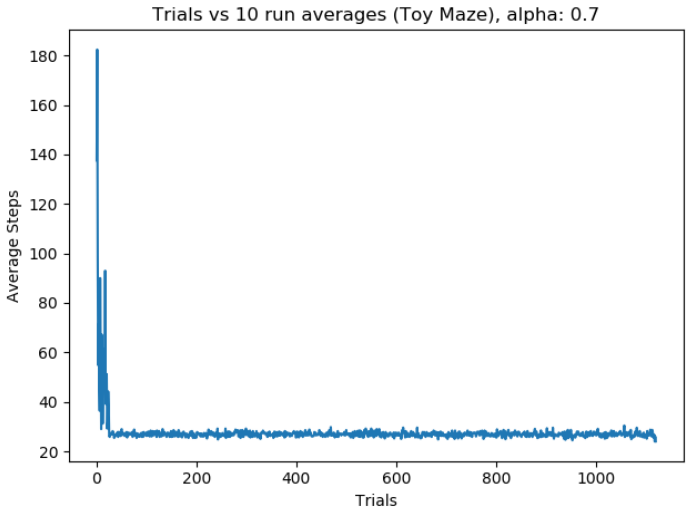
As we increase the value of epsilon towards 1 - increasing randomness in action selection, the number of average steps the robot takes decreases this means it is in a mode for more exploration than exploitation from previously learned actions.

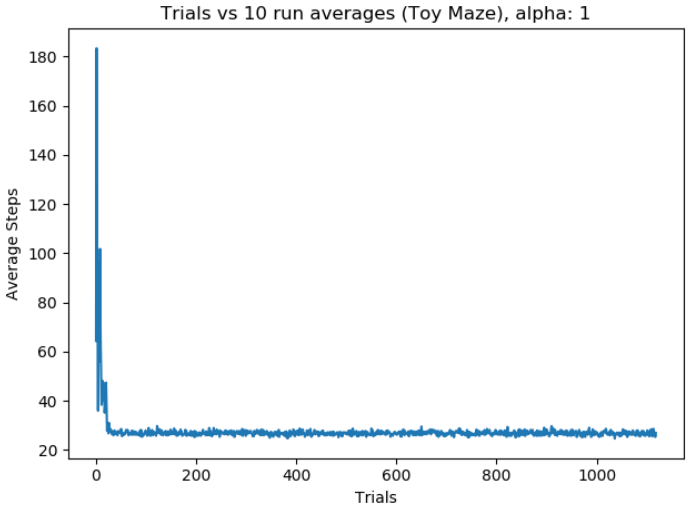
Smaller epsilon values such as 0.1 and 0.33 take some time when exploring and eventually converge to an optimal solution given enough trials with low noise.

Ex 2.









In this exercise, we kept the epsilon constant to 0.1. As shown in the graphs above,. The parameter alpha is our learning rate, and at various values at graphs seem identical. It is observed at low learning rates the speed of learning is lower and therefore it takes longer to converge to an optimal solution. However it may also lead to the agent getting stuck in a local minima due to the value being too small, contrarily a higher learning rate may quickly jump over the minima.

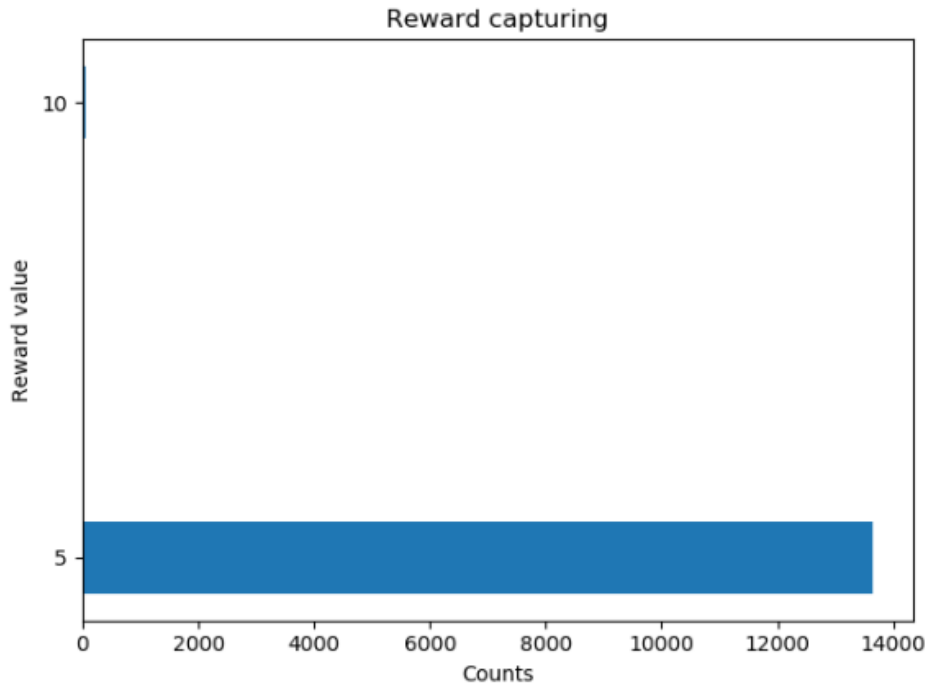
Ex 3.

- The trade-offs between a high and low epsilon value can be thought as the robot's learning trade-offs between exploration and exploitation. We need both to optimize the learning algorithm.

- Higher epsilon values will lead to the robot spending more time exploring the maze with random actions and is less likely to exploit information from learned actions, however it may still reach to the global minima.. Lower epsilon values will lead to less exploring and there are higher chances of the robot getting stuck in a local minima.

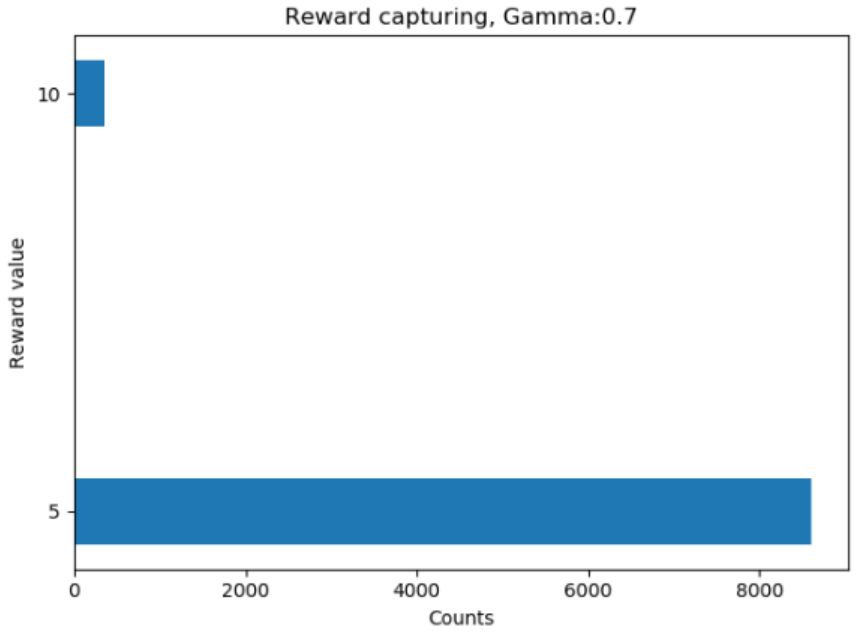
2.3

Ex 1.



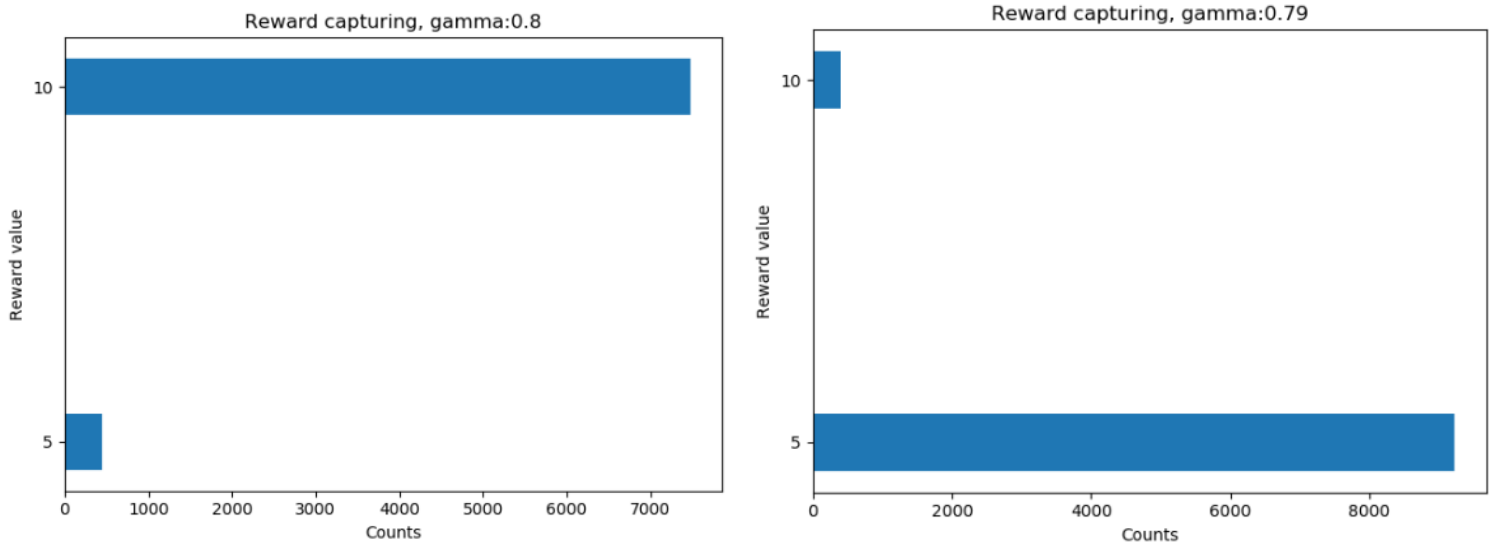
As seen from the graph above, once the new reward is set (9,0) the robot moves towards it and gets stuck in the local optimum this makes the agent select actions based on its goal to capture reward of value 5 instead exploring to look for rewards with higher values.

Ex 2.



Initial high exploration rate in earlier runs can help assure convergence to the optimal solution in the final run, therefore an epsilon of 1 is set for each run. The problem is then mitigated by reducing the epsilon value slightly for every trial this is done by multiplying by the beta constant of 0.99.
In the final run, the epsilon will have been reduced such that it may now take actions with confidence to converge to its global optimal solution.

Ex 3.



We can tune the values for gamma to give us to move towards smaller or higher rewards. As shown in the graphs above, values for gamma higher than 0.8 favor reaching for the reward of value 10 and lower values tend to favor absorbing the reward of value 5.