University of Maryland, College Park

A James Clarke school of engineering

MAZE SOLVING USING Q- LEARNING

Final Project report

ENPM-690, Robot Learning

Submitted to Dr. Donald Sofge

Professor, University of Maryland

Table of Contents

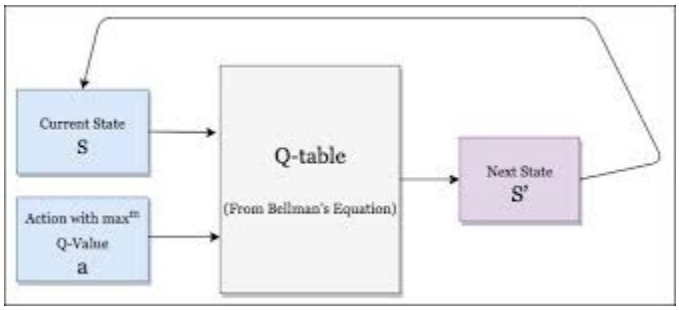
Abstract:

Machine learning is very important in several fields ranging from data mining to biological sciences. This project implements the Q -learning algorithm for abstract graph models with maze solving. Here, the maze matrix is converted into Q learning reward matrix taking the states and action into consideration. This implementation is on higher level abstraction, so other representation such as deep q learning, neural networks can also be used. In this project, the agent is fixed in the top left corner and the goal is defined based on the user inputs.

Introduction:

Everyone in their lives, have played a very common and obvious Maze game once in their lives. I've come up a similar interesting problem where the agent trains itself to reach the goal position in a discrete action space using the famous Q learning algorithm.

Reinforcement learning is one of the several machine learning techniques of artificial intelligence. The main system components that are involved in this project include states, action and reward system. Agent makes an action and it is rewarded and based on this reward it is possible to model any maze problem of similar configuration using reinforcement learning technique.



The system that is trained through this method will learn the optimal method of making decisions by performing interaction with its environment and receiving feedback.

The motivation is developed mainly from the definition of reinforcement learning. With the concept of reinforcement learning, there is no need of modelling the dynamic systems and this method can be implemented for path planning of mobile robots in a constrained environment.

Approach:

I've created an own environment to train the maze in a different grid space of 4X4, 5X5 and 10x10. In this environment the maze is hindered with the obstacles namely walls and pits. The goal is defined based on a user defined position and the aim is to find the optimal path to the end of the mazegame.

In this game, the states are defined as the position of the agent in the environment and the agent is assumed to traverse along the four directions which UP, down, left and right. So, when the agent is in process of reaching a goal, it perhaps collides with the walls and pits. Whenever, the agent gets into obstacles we reward based on the performance of the action.

1. Taking actions:

The action that agent takes in a Q learning is done in two possible ways. They are, Exploring and Exploiting.

Exploiting: With the Q table as reference, the agent tries to make a move in all possible actions. The agent then selects the action in a way where it can maximize the reward.

Exploring: During this action, the agent explores all the states randomly by taking a random action. This makes the agent to discover the new states, where it isn't possible during exploiting.

We choose a single set of action from exploiting and exploring by selecting the epsilon decay and tuning it based on the observed outcomes. In our case, the epsilon greedy is assumed as 0.9.

$$Action = \begin{cases} \max_a Q(s,a), & R > \varepsilon \\ Random\ a, & R \leq \varepsilon \end{cases}$$

R is random number between 0 and 1, ε is the exploration factor between 0 and 1. If ε is 0.1, then 10% of the times, the algorithm will select a random action to explore corresponding rewards.

2. Q Learning:

Based on the actions, we generate a new action and new state for every state. Firstly, we fix the q table to zeros. Then after each action a new state, corresponding reward for the action is also generated. Then, Q table gets update based on the Bellman's equation

Source: https://en.wikipedia.org/wiki/Q-learning

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Cell	Reward
Goal	+50
Pits	-10
Walls	-1
Free cells	0

This Q learning follows an off-policy temporal difference control algorithm. Since it is an off-policy algorithm, it does not follow the policy; instead, it directly approximates the optimal state-action value. It starts in a state-action pair (S, A) and ends up in a new state S'. Instead of using epsilon-greedy policy, it directly chooses the best possible state-action value for state S' by just doing a max. Next, it updates the state-action value Q (S, A) by applying Bellman update. . Then, it calculate the TD error by subtracting old estimated value Q(S, A) from the TD target, which is Next, it applies the Bellman update to the old estimated Q (S, A) by multiplying TD error and the learning rate and add it to the old estimated Q (S, A). After applying the Bellman update, it updates the current state S to the next state S' and the current action A to a new action chosen using epsilon-greedy policy on the state-action values for S'. And then, it goes into the next iteration and do the bellman update iteratively. Finally, the state-action value Q will converge to the optimal policy.

3. Implementation of the Q learning algorithm:

The pseudo code for the Q learning algorithm is depicted below, the code is fully developed and available in python3.

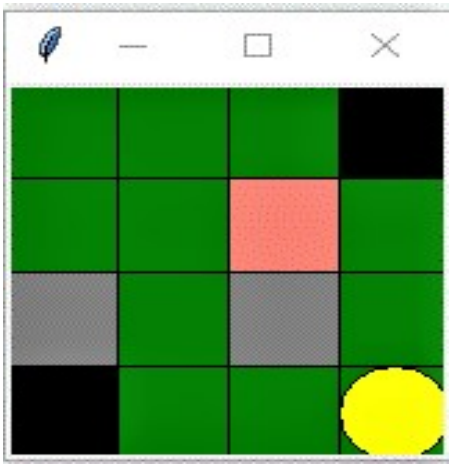
1. Initialize Q to zeros
2. For each episode select a random state then select an action based on the state using the epsilon greedy approach.
3. Travel to the next state and obtain the new state space and optimal action for the corresponding state.
4. Check for the states if it is already in the Q table else append the new state
5. Update the Q table based on the bellman's equation by calculating the error between the Q predict and Q estimate.
6. Make the initial state as the new state and loop for 'N' desired episodes

Choose the parameters for the algorithm appropriately so that, the agent tries to converge as fast as possible in the most optimal path.

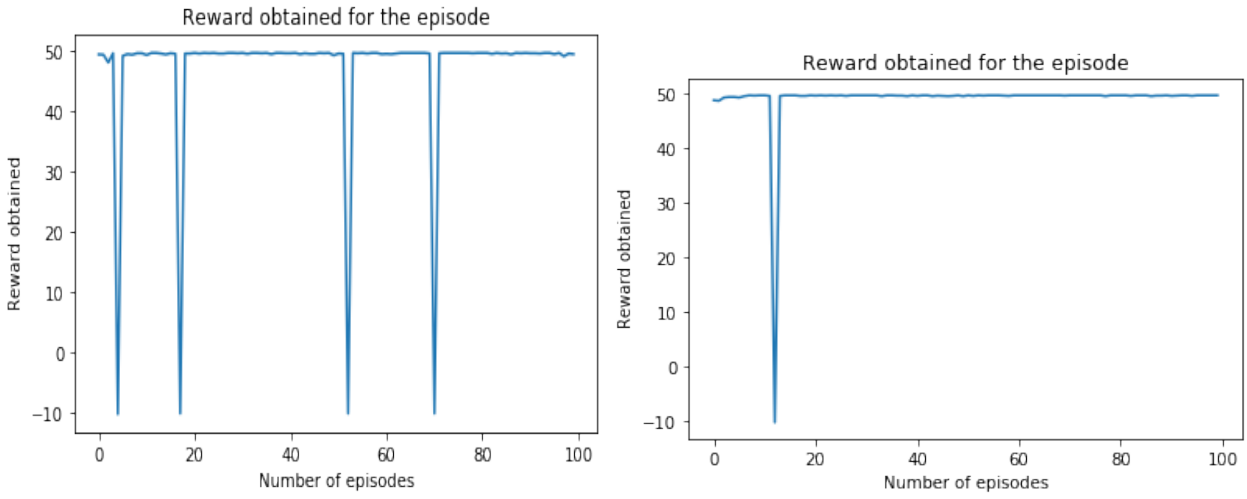
Actions	Encoded
UP	0
Down	1
Left	2
Right	3

Results:

4X4 grid:



Here is the visual representation of my 4x4 grid. The walls are colored with grey, with a reward of -1 and pits are colored as grey with reward of -10 and the goal is oval with reward of 50. Here the agent is orange colored. The algorithm is ran for a 100 episodes and from the graph we can conclude that it takes an average of 25 iterations to train.



As you can observe, the second plot is smooth which contrasts with the first plot is mainly because of tuning the learning rate

5X5 GRID: Output (https://youtu.be/rWHgy_lKe4Q)

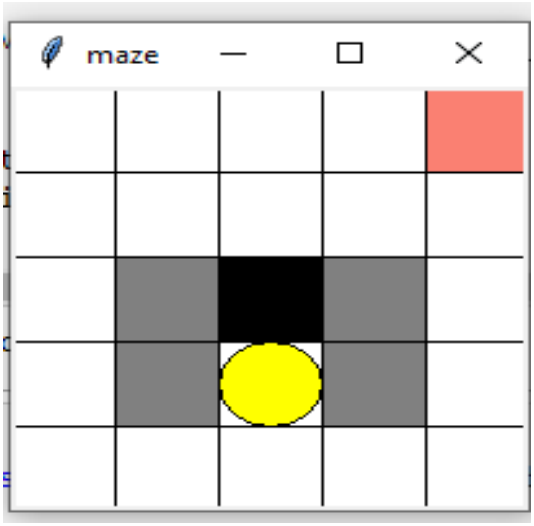
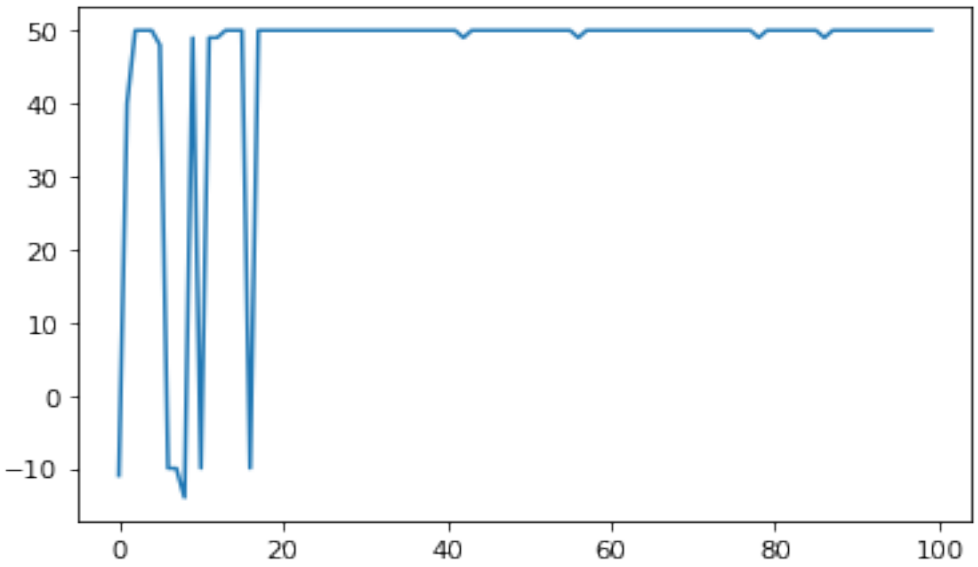
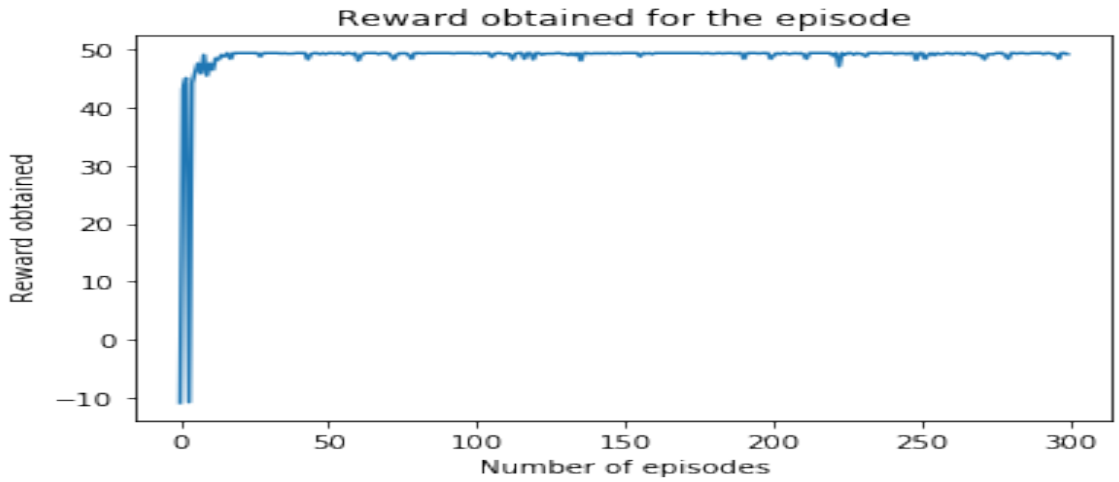


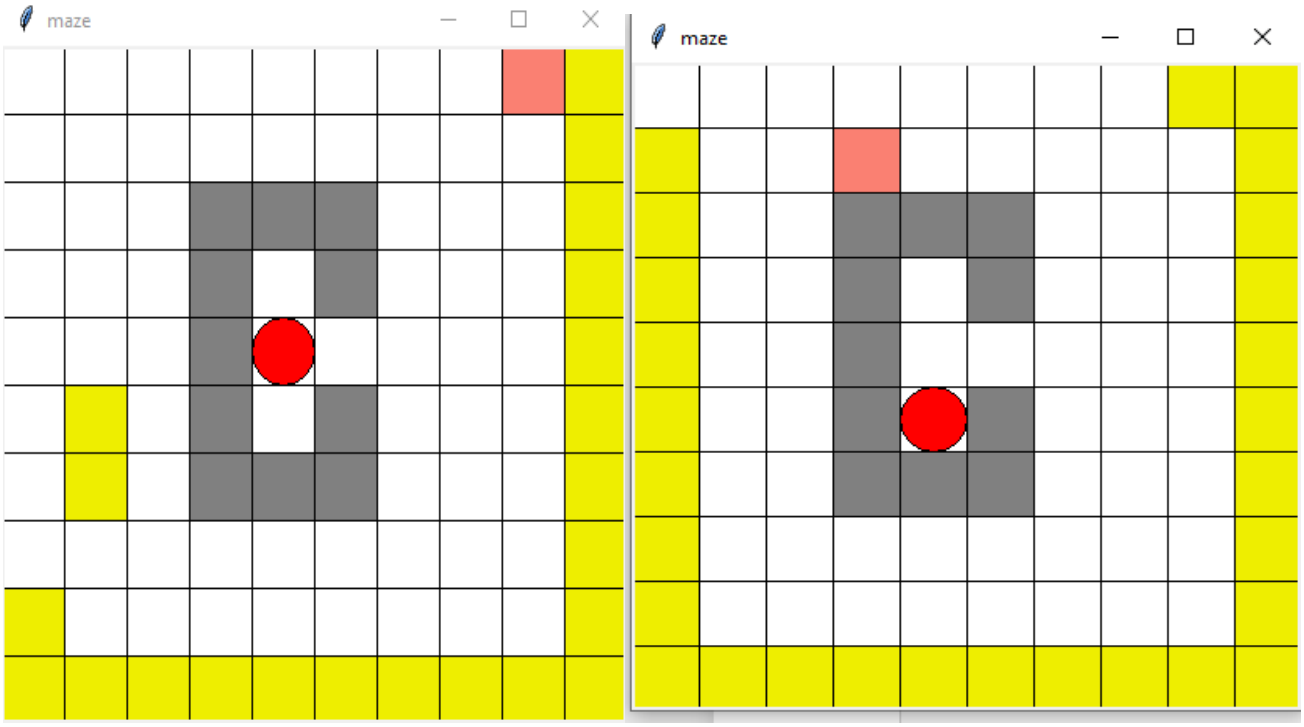
Fig1. 5X5 GRID

Here is the visual representation of my 5x5 grid. The walls are colored with grey, with a reward of -1 and pits are colored as grey with reward of -10 and the goal is oval with reward of 50. Here the agent is orange colored. The parameters are tuned learning rate = 0.4, gamma = 0.9 and epsilon = 0.9. The algorithm is ran for a 300 episodes and from the graph we can conclude that it takes an average of 15 iterations to converge.



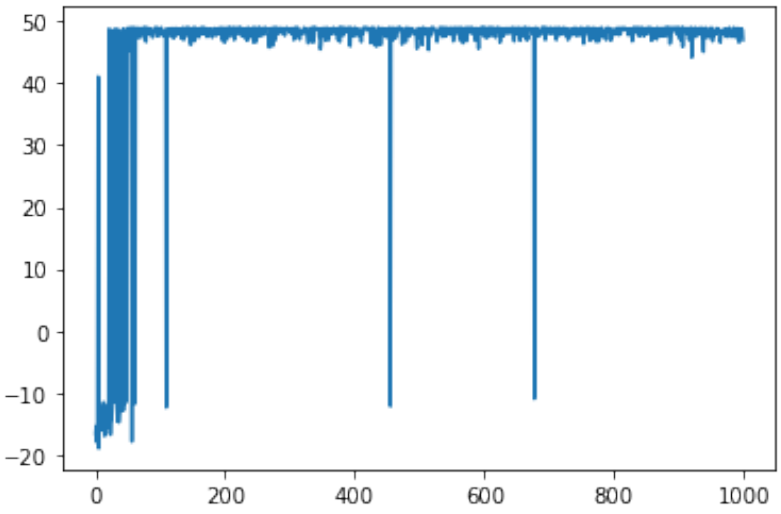


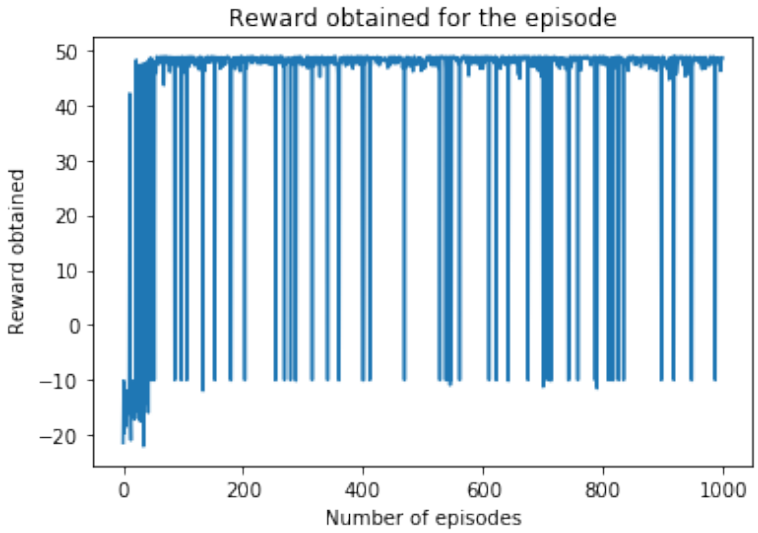
10X10 GRID: https://youtu.be/WB0WOTdUrF4, https://youtu.be/EyV5nVylUCU



Here is the visual representation of my 10x10 grid. The walls are colored with grey, with a reward of -1 and pits are colored as yellow with reward of -10 and the goal is oval with reward of 50. Here

the agent is orange colored. The parameters are learning rate = 0.4, gamma = 0.9 and epsilon = 0.8. The algorithm is ran for a 400 episodes and from the graph we can conclude that it takes an average of 160 iterations to converge.





Analysis:

Upon detailed analysis of the, two graphs for each grid size which are contrast to each other, it is observed that the learning parameters make a huge impact on making a decision and the epsilon greedy method explores all the states in the grid then choose a optimal path, this can be overcome with the Markov decision process where the probability of the reward is predicted predefine before the next action although it's not accurate at least to the smaller training data.

Q table for 10x10 grid for two episodes

States Actions	0	1	2	3
[0.0, 0.0, 40.0, 40.0]	-0.02	-0.02	-0.02	-0.02
[40.0, 0.0, 80.0, 40.0]	0	-0.02	0	0
[40.0, 40.0, 80.0, 80.0]	-0.0392	-0.05936	-0.032	-0.05504
[80.0, 40.0, 120.0, 80.0]	-0.0392	-0.04	-0.02	-0.0392
[120.0, 40.0, 160.0, 80.0]	0	0	0	-0.02
[40.0, 80.0, 80.0, 120.0]	-0.04352	-0.04352	-0.04	-0.0392

[40.0, 120.0, 80.0, 160.0]	-0.04611	-0.032	7.168	-0.02
[80.0, 120.0, 120.0, 160.0]	-0.04	0	32	0
[120.0, 120.0, 160.0, 160.0]	0	0	0	0
[0.0, 40.0, 40.0, 80.0]	-0.02	-0.04	-0.03152	0

Future work:

Based on the observed output, we can conclude that the memory usage is to store the q matrix is too high and for larger grids, the computational time would be very large as this explores all the possible states. The agent takes the optimal path once trained, that allows to follow the same path for every iteration after trained. This would not work if the goal is changed for each iteration. So, to make the system robust we predict the moves based on the probability using the deep convolution networks, this approach is called Deep Q learning. All these improvements can be done as future work to leverage the optimality and related factors

Conclusion:

In this project we developed maze solving algorithm based on the Q-Learning algorithm. While implementing this project we discovered some benefits and draw backs of Q learning to solve the Maze learning problem.

The code for the Project is available at https://github.com/Nishanth2708/Q_Learning_Maze

You can watch the output for the Videos at

1. Output for 10x10 : https://youtu.be/WB0WOTdUrF4
2. Output for 5x5 : https://youtu.be/rWHgy_lKe4Q
3. Output for 10x10_Complicated : https://youtu.be/EyV5nVylUCU

Bibliography

1. D. OsmanKovic, S. K. (n.d.). Implementation of Q learning for Solving Maze problem. *IEEE*.

2. Das, A. (2017). *Medium*. Retrieved from Introduction to Q learning: https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c

References:

1. https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/
2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5967320
3. https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c