Project-3: REINFORCEMENT LEARNING USING Q LEARNING ALGORITHM

Shubhangi Mane
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
UB Id: 50295705
smane2@buffalo.edu

ABSTRACT

This project aims at building a reinforcement learning agent using Q learning algorithm. The agent navigates through 4X4 grid environment. The agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. In this project we perform three tasks.

1. Implementing the policy function
2. Updating the Q table
3. Implementing the training algorithm

Initially, the grid-world environment is defined. At first agent randomly selects the action by certain percentage of epsilon. But it is better for the agent to try all kinds of things before it starts to see the pattern. And eventually the agent picks up the action with highest reward, based on the methods reset, step and render and updates are done based on these actions.

INTRODUCTION:

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. It differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration and exploitation.

REINFORCEMENT LEARNING:

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. In reinforcement learning there's no answer key, but your reinforcement learning agent still has to decide how to act to perform its task. In the absence of existing training data, the agent learns from experience.

Q LEARNING ALGORITHM:

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

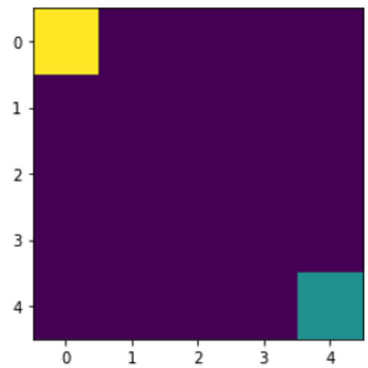
LEARNING SYSTEM:

ENVIRONMENT:

The environment is an 4X4 grid-world environment, so total of 16 steps are possible. The agent starts at the top left most corner and ends at the bottom right corner.

1. At each step, the agent has 4 possible actions including up, down, left and right. At each time step, the agent will take one action and move in the direction described by the action.

2. The agent will receive a reward of +1 for moving closer to the goal and −1 for moving away or remaining the same distance from the goal



Q LEARNING ALGORITHM:

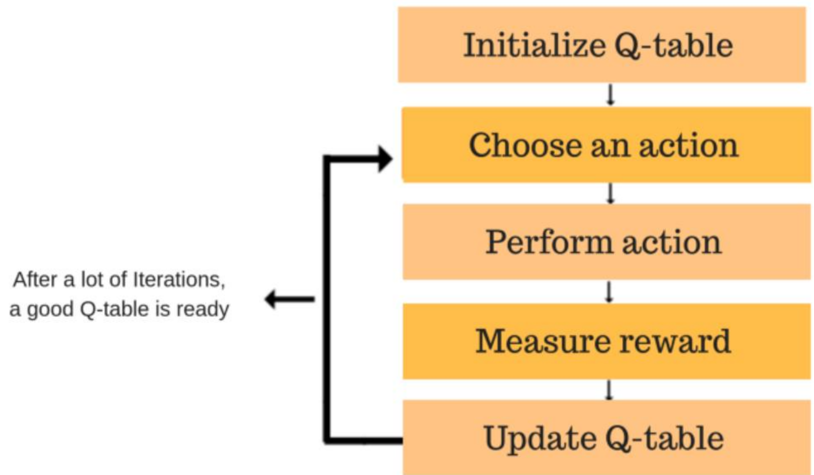
The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as exploiting since we use the information we have available to us to make a decision.The second way to take action is to act randomly. This is called exploring.

Q TABLE:

Q-Table is table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state. At each non edge the agent can move in 4ways up, down , right, left. In q table columns are the actions and rows are the states.The Q function takes two inputs state and the action.

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

Using the above function we calculate the values for each Q cell. Initially the Q table is set to all zeros and in an iterative process we update the values in Q table.



Step 1: We bulild a Q table of size nxm where n is number of actions and m is the number of states and we initialize it to zeros.

Step 2: We choose an action in state (s) based on the value in the Q table.

Step 3: We perform the choosen action in the previous step.

Step 4 and 5: We have taken the action and observed an outcome and reward and similarly update is made in Q table.

KEY COMPONENTS:

1. env.reset(): This method resets the environment to initial state.

2. env.render():For every action taken in a particular step a window will be rendered.

3. env.step(): This returns four values that include reward, observation, a Boolean value and information regarding the action taken.
By using all these methods we make the agent learn from the experiences and improve the performance by gaining high rewards.

Open AI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like pong or pinball. Gym is an open source interface to reinforcement learning tasks. Gym provides an environment and its is upto the developer to implement any reinforcement learning algorithms. Developers can write agent using existing numerical computation library, such as TensorFlow or Theano.

HYPER PARAMETERS:

Hyper-parameters are the using which we can improve the performance of the model or the agent. By changing these value we compute the reward, time taken to learn and then make the algorithm better. There are various hyper parameters that we used in the task. These include max/min epsilon, number of episodes, gamma, etc.

GAMMA: also called as discount rate. It is mainly used to calculate the future discounted reward. The discount factor is mainly used to control the agent to how much area it should explore. If there is no discount rate the agent simply reaches the goal but takes lot of time. If there is a discount rate associated it reaches goal in quick and efficient manner by exploring all the ways. The value of discount factor should be less than 1 and is better if it is at 0.9.

EPSILON: is the exploration/exploitation tradeoff. It is mainly used to control the amount the knowledge the agent gains. Initially we begin with larger value of epsilon by making the agent learn. Then we slowly decrease the value by changing to exploitation that is the agent has enough knowledge and now tries to maximize the reward.

EPISODE: is one play that is agent moving from source to destination once. By increasing the number of episodes the agent tries to learn more and gains high rewards. So, more episodes more the agent learn but at the same time with more episodes more time will be taken. Thereby improving performance of the agent and gaining the reward.

OBSERVATIONS:

By changing the value of hyper parameters we compute the reward and the total time taken. Finally we choose the values that give best reward.

The original result after completing the snippet is:

Episode	Gamma	Epsilon	Decay Rate
200	0.8	1.0	0.9

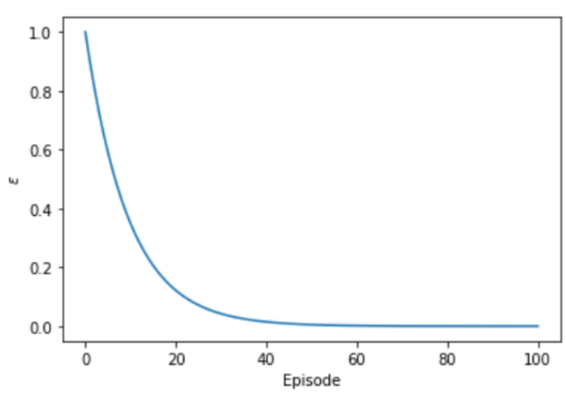


Figure 1 Plot showing value of epsilon versus each episode

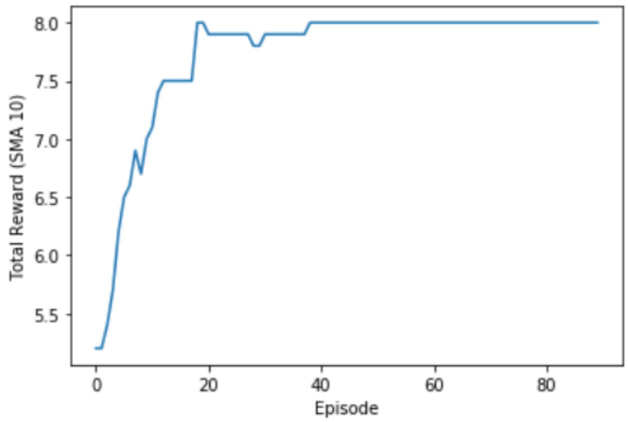


Figure 2 Plot showing Total rewards per episode

Q Table:

```
[[[ 0.473428   -0.16560252  5.18213486  0.07149384]
  [ 4.88168972  0.08070283  0.2881      0.0701552 ]
  [ 0.         -0.091       0.19       -0.23460914]
  [ 0.          0.          0.19       -0.091     ]
  [ 0.1        -0.1        -0.1        -0.091     ]]

 [[ 0.5155394  -0.07235712  0.          0.        ]
  [ 0.3439     -0.04591826  4.4908512  -0.06453641]
  [ 3.99778836 -0.181       0.         -0.16030604]
  [ 0.          0.          0.         -0.05706343]
  [ 0.          0.          0.         -0.1       ]]

 [[ 0.76755179  0.          0.          0.        ]
  [ 0.14811903 -0.11427949  0.         -0.09772884]
  [ 0.          0.          3.39899566 -0.1       ]
  [ 0.         -0.1        2.69735924 -0.0829    ]
  [ 1.89727698 -0.1        -0.1       -0.15751   ]]

 [[ 0.         -0.06453641  0.74026117 -0.1729    ]
  [ 0.68778308 -0.1         0.         -0.06460858]
  [ 0.          0.          0.         -0.091     ]
  [ 0.          0.          0.         -0.1       ]
  [ 0.99970031 -0.07561    -0.1        -0.1       ]]

 [[ 0.          0.          0.109      -0.1       ]
  [ 0.          0.          0.477559   -0.19      ]
  [ 0.         -0.1         0.19       -0.143659  ]
  [ 0.          0.          0.         -0.091     ]
  [ 0.          0.          0.          0.        ]]]
```

OBSERVATION 2:

Episode	Gamma	Epsilon	Decay Rate
150	0.9	1.0	0.9

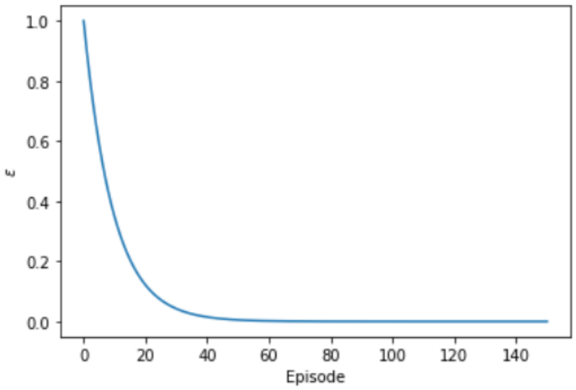


Figure 3 Plot showing value of epsilon versus episodes

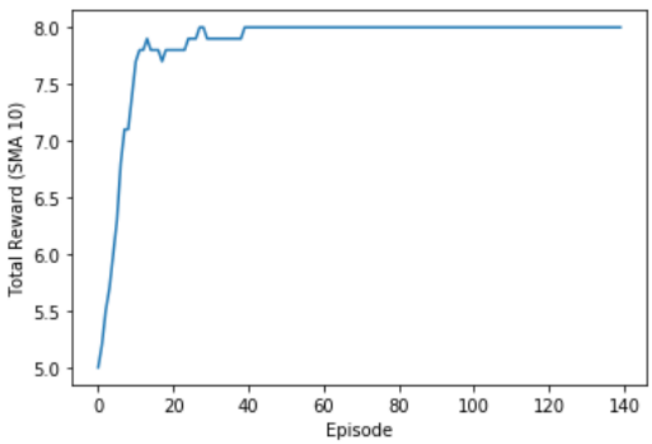
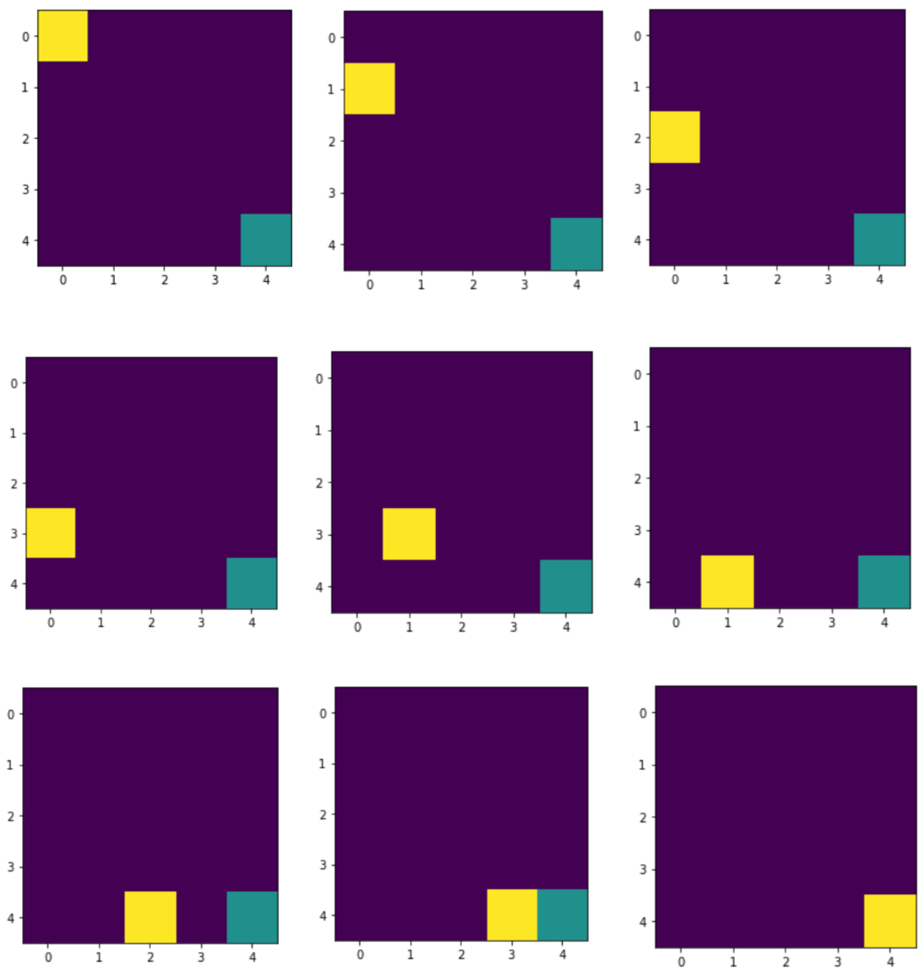


Figure 4 Plot showing Total rewards per episode

Q Table:

```
[[[ 0.43371049 -0.35565099  5.6508102  -0.26016791]
  [ 5.19545832 -0.27798155  0.28       -0.06460858]
  [ 0.          0.          0.40951    -0.147952  ]
  [ 0.          0.          0.         -0.204247  ]
  [ 0.          0.          0.          0.        ]]

 [[ 0.         -0.0901171   0.88491841 -0.18019   ]
  [ 4.67650774 -0.02677904  0.2881     -0.10895814]
  [ 0.         -0.1         0.3691     -0.06751   ]
  [ 0.19        0.          0.271      -0.091     ]
  [ 0.         -0.1        -0.1        -0.091     ]]

 [[ 0.30349    -0.1         0.         -0.091     ]
  [ 0.361       0.          4.09187712 -0.24499   ]
  [ 0.         -0.2377      3.43807642 -0.07561   ]
  [ 0.1        -0.0829      2.70979999 -0.14941   ]
  [ 1.89997095 -0.1        -0.1        -0.1670231 ]]

 [[ 0.3529     -0.1         0.         -0.1       ]
  [ 0.          0.          0.28       -0.1729    ]
  [ 0.19        0.          0.         -0.091     ]
  [ 0.          0.          0.16175705 -0.091     ]
  [ 0.99999788 -0.069049   -0.1        -0.1       ]]

 [[ 0.          0.          0.199      -0.1       ]
  [ 0.          0.          0.2071      0.        ]
  [ 0.          0.          0.271      -0.091     ]
  [ 0.          0.          0.         -0.091     ]
  [ 0.          0.          0.          0.        ]]]
```

The path the agent has taken:



CONCLUSION:

In this project, we built a reinforcement learning agent to navigate the classic 4x4 grid-world environment. The agent learnt an optimal policy through Q-Learning which allowed it to take actions to reach a goal while avoiding obstacles. The results obtaining by tuning different hyper parameters were explained in the above section.

REFERENCES:

1. https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14
2. https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
3. https://towardsdatascience.com/q-learning-54b841f3f9e4
4. https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

5. https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
6. https://www.datahubbs.com/intro-to-q-learning/