BUILD A REINFORCEMENT LEARNING AGENT TO NAVIGATE THE CLASSIC 4X4 GRID-WORLD ENVIRONMENT

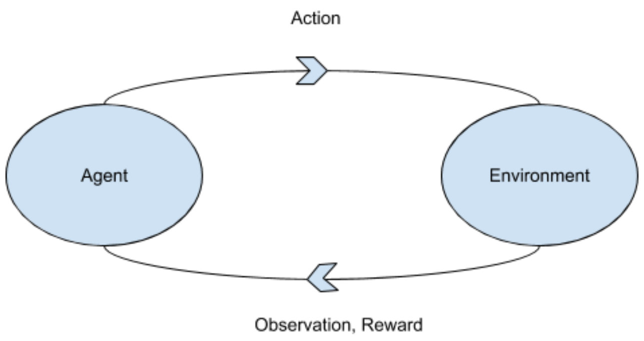
Rohith Kumar Gadalay
Department of Computer Science and Engineering
University at Buffalo
rohithku@buffalo.edu

ABSTRACT

The task of the project is to build reinforcement learning agent to navigate the classic 4x4 grid-world 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. The working environment and agent provided will be built to be compatible with Open-AI Gym environments and will run effectively on computationally limited machines. The project involves implementing three tasks. The first one is implementing the policy function, second is updating Q-table and next is to implement the training algorithm.

1 INTRODUCTION

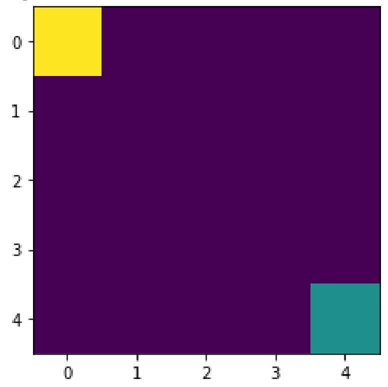
Reinforcement learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment so as to maximize some reward. Learning to control agents directly from high dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL). Most successful RL applications that operate these domains have relied on hand crafted features combined with linear value functions or policy representations. Clearly the performance of such systems heavily relies on the quality of the feature representation. Open-AI Gym is a toolkit for developing and comparing reinforcement learning algorithms. This is the gym open-source library, which gives the access to a standardized set of environments. Open AI Gym has an environment-agent arrangement. It simply means Gym gives the access to an "agent" which can perform specific actions in an "environment". In return, it gets the observation and reward as a consequence of performing a particular action in the environment.



2 ENVIRONMENT DETAILS

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations etc. The reinforcement learning community has developed a standard of how such environments should be designed, and the library which facilitates this is Open-AI's Gym(https://gym.openai.com/).

The initial state of our basic grid world environment is



The environment we provide is a basic deterministic n x n grid-world environment (the initial state for an 4x4 grid-world is shown in above Figure) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.

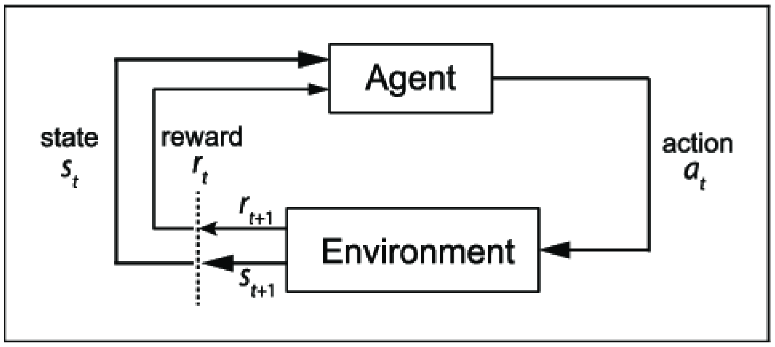
The environment's state space will be described as an n x n matrix with real values on the interval [0,1] to designate different features and their positions. The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. 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.

.

3 ARCHITECTURE

3.1 Reinforcement Learning

Reinforcement learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment to maximize some reward. This is typically modeled as a Markov decision process (MDP) as illustrated in the below figure.



An MDP is a 4-tuple (S, A, P,R), where

- S is the set of all possible states for the environment.
- A is the set of all possible actions the agent can take.
- P = Pr ($s_{t+1} = s' \mid s_t = s, a_t = a$) is the state transition probability function.
- R : S x A x S → R is the reward function.

Our task is find a policy π:S→ R which our agent will use to take actions in the environment which maximize cumulative reward, i.e.,

$$\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})$$

where $\gamma \in [0, 1]$ is a discounting factor (used to give more weight to more immediate rewards), s_t is the state at time step t, a_t is the action the agent took at time step t, and s_{t+1} is the state which the environment transitioned to after the agent took the action.

3.2 Q-Learning

Q-Learning is a process in which we train some function Q : S x A→R, parameterized by θ, to learn a mapping from state-action pairs to their Q-value, which is the expected discounted reward for following the following policy π_θ:

$$\pi(s_t) = \underset{a \in A}{\operatorname{argmax}}\, Q_\theta(s_t, a)$$

In words, the function Q_ will tell us which action will lead to which expected cumulative discounted reward, and our policy π will choose the action a, which, ideally, will lead to the maximum such value given the current state s_t.

Originally, Q-Learning was done in a tabular fashion. Here, we would create an |S| x |A| array, our Q-Table, which would have entries $q_{i,j}$ where i corresponds to the ith state (the row) and j corresponds to the jth action (the column), so that if s_t is located in the ith row and a_t is the jth column, $Q(s_t, a_t) = q_{i,j}$. We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

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

We are using our Q-function recursively to match (following our policy π) in order to calculate the discounted cumulative total reward. We initialize the table with all 0 values, and as we explore the environment (e.g., take random actions), collect our trajectories, [$s_0, a_0, r_0, s_1, a_2, r_2 \ldots\ldots s_T, a_T, r_T$] and use these values to update our corresponding entries in the Q-table.

During training, the agent will need to explore the environment in order to learn which actions will lead to maximal future discounted rewards. Agent's often begin exploring the environment by taking random actions at each state. Doing so, however, poses a problem: the agent may not reach states which lead to optimal rewards since they may not take the optimal sequence of actions to get there. In order to account for this, we will slowly encourage the agent to follow its policy in order to take actions it believes will lead to maximal rewards. This is called exploitation and striking a balance between exploration and exploitation is key to properly training an agent to learn to navigate an environment by itself.

To facilitate this, we have our agent follow what is called an Є-greedy strategy. Here, we introduce a parameter Є and set it initially to 1. At every training step, we decrease its value gradually (e.g., linearly) until we've reached some predetermined minimal value (e.g., 0.1). In this case, we are annealing the value of Є linearly so that it follows a schedule.

During each time step, we have our agent either choose an action according to it's policy or take a random action by taking a sampling a single value on the uniform distribution over [0, 1] and selecting a random action if that sampled value is than Є otherwise taking an action following the agent's policy.

4 CHALLENGE

4.1 Task-1(Implement Policy Function)

- The agent selects the action randomly by a certain percentage called epsilon.
- The random number is selected and if the selected number is less than epsilon then return the random choice action space.
- When the action is not decided randomly, the agent will predict the reward value based on the current state and pick the action that gives the highest reward.
- Return the policy value.

4.2 Task-2(Update Q-table)

- The updates occur after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent.
- The agent will not perform much learn after a single episode but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values.
- The steps involved are:
  - Agent starts in a state(s1) takes an action(a1) and receives the reward(r1).
  - Agents selects action by referencing Q-table with highest value(max) OR by random (epsilon, Є).
  - Update q-values.

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

α-learning rate often referred to as alpha or α, can simply be defined as how much we accept the new value vs the old value.
γ-is a discount factor which is used to balance immediate and future reward.

Reward-is the value received after completing a certain action at a given state.
Max- np.max() uses the numpy library and is taking the maximum of the future award and applying it to the reward of the current state.

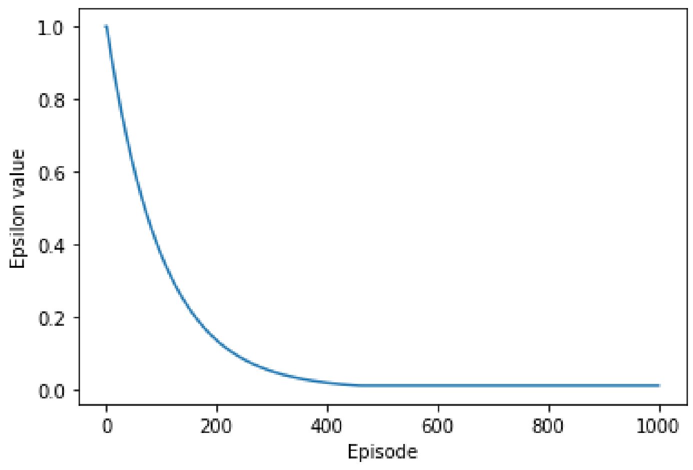
4.3 Task-3(Implement the training algorithm)

- The environment is initialized and it has three main methods reset(),step() and render(). We use reset() and step() methods only.
- When reset() is called, the environment is initialized with a fresh episode. This allows to effectively run through the episodes.
- The step() method accepts an action as a parameter, processes the action and returns the new state, the reward for performing the action and a Boolean indicating if the run is completed or not.
- When the agent is initialized, the type of environment should be passed into the Q-Learning Agent function.

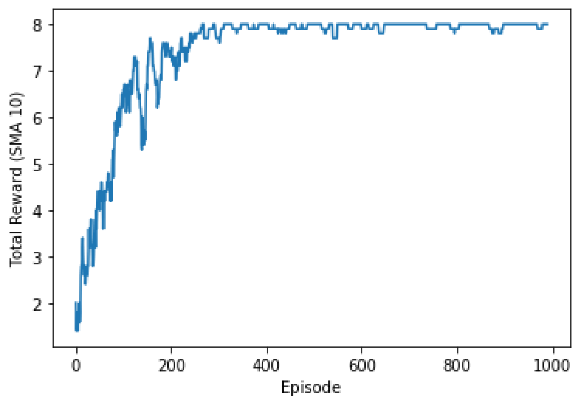
5 RESULTS

Initially the number of episodes is initialized to **1000** and decay is 0.99

The graph for Episode vs Epsilon value is

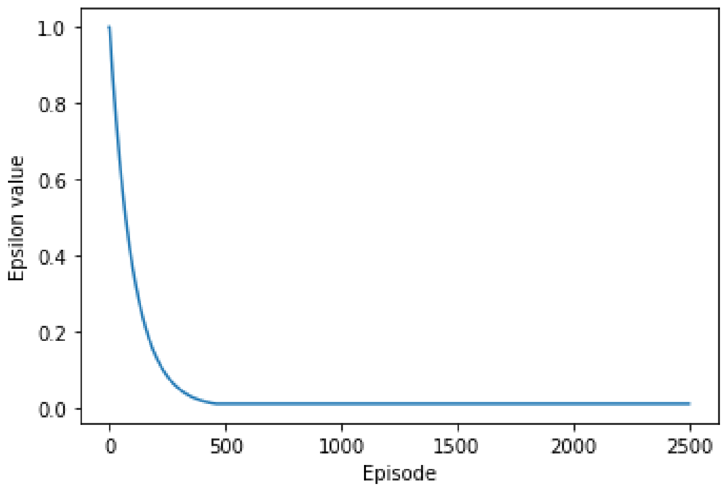


The graph for Episode vs Total Rewards is

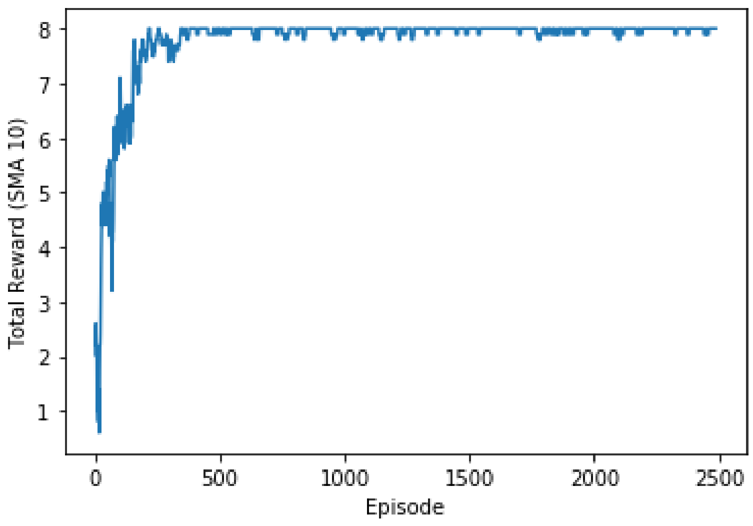


The number of episodes is initialized to **2500** and decay is 0.99

The graph for Episode vs Epsilon value is

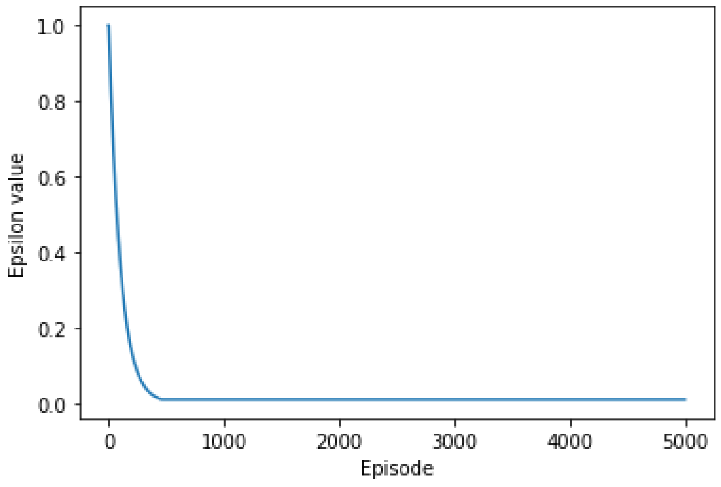


The graph for Episode vs Total Rewards is

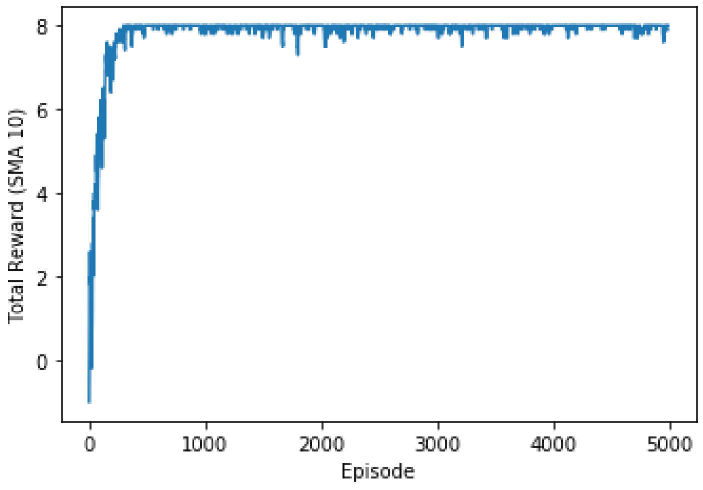


The number of episodes is initialized to **5000** and decay is 0.99

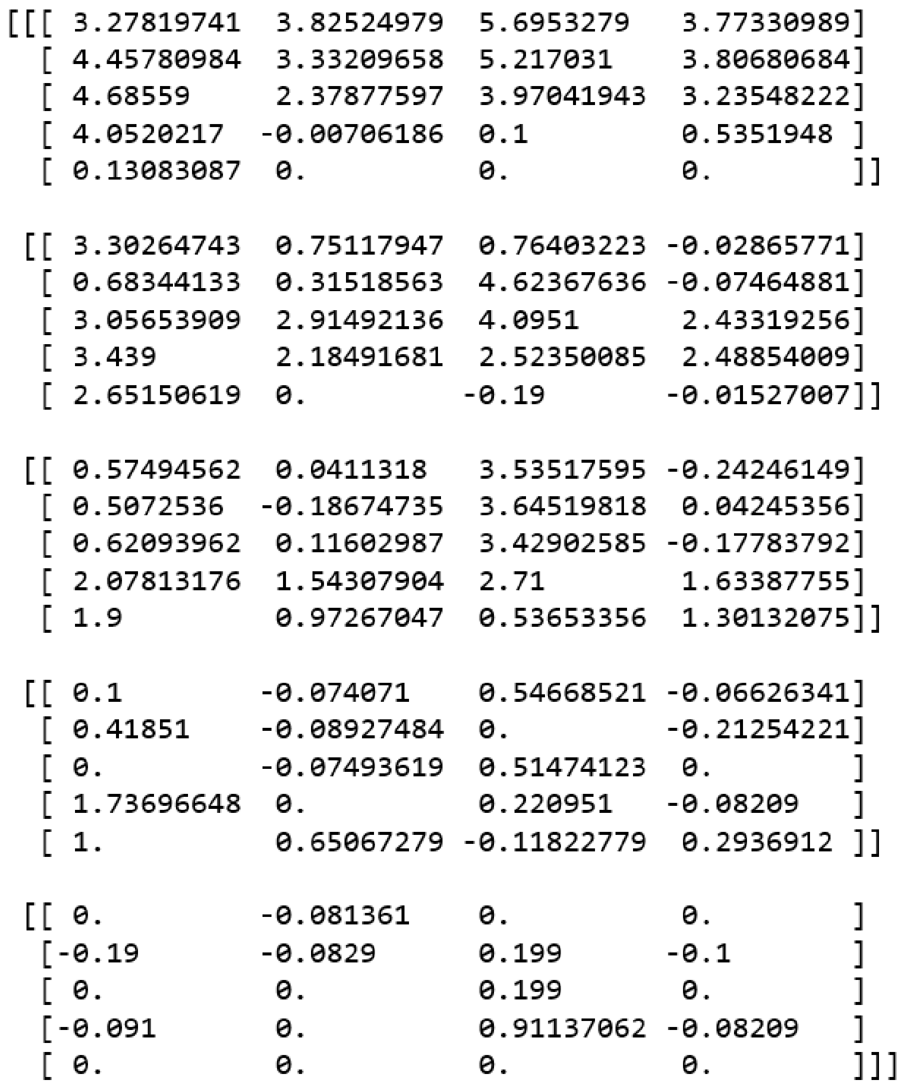
The graph for Episode vs Epsilon value is



The graph for Episode vs Total Rewards is



The q-table obtained is:

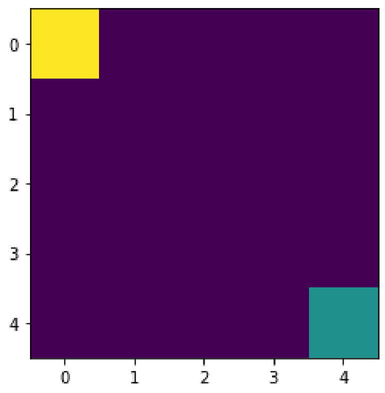


```
[[[ 3.27819741  3.82524979  5.6953279   3.77330989]
  [ 4.45780984  3.33209658  5.217031    3.80680684]
  [ 4.68559     2.37877597  3.97041943  3.23548222]
  [ 4.0520217  -0.00706186  0.1         0.5351948 ]
  [ 0.13083087  0.          0.          0.        ]]

 [[ 3.30264743  0.75117947  0.76403223 -0.02865771]
  [ 0.68344133  0.31518563  4.62367636 -0.07464881]
  [ 3.05653909  2.91492136  4.0951      2.43319256]
  [ 3.439       2.18491681  2.52350085  2.48854009]
  [ 2.65150619  0.         -0.19       -0.01527007]]

 [[ 0.57494562  0.0411318   3.53517595 -0.24246149]
  [ 0.5072536  -0.18674735  3.64519818  0.04245356]
  [ 0.62093962  0.11602987  3.42902585 -0.17783792]
  [ 2.07813176  1.54307904  2.71        1.63387755]
  [ 1.9         0.97267047  0.53653356  1.30132075]]

 [[ 0.1        -0.074071    0.54668521 -0.06626341]
  [ 0.41851    -0.08927484  0.         -0.21254221]
  [ 0.         -0.07493619  0.51474123  0.        ]
  [ 1.73696648  0.          0.220951   -0.08209   ]
  [ 1.          0.65067279 -0.11822779  0.2936912 ]]

 [[ 0.         -0.081361    0.          0.        ]
  [-0.19       -0.0829      0.199      -0.1       ]
  [ 0.          0.          0.199       0.        ]
  [-0.091       0.          0.91137062 -0.08209   ]
  [ 0.          0.          0.          0.        ]]]
```

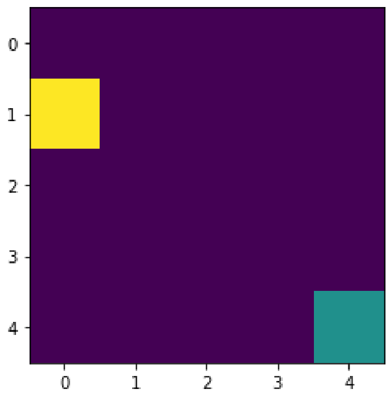
The number of steps to find the shortest path and reach the goal when Q-Learning Agent used is 8.

The flow is:

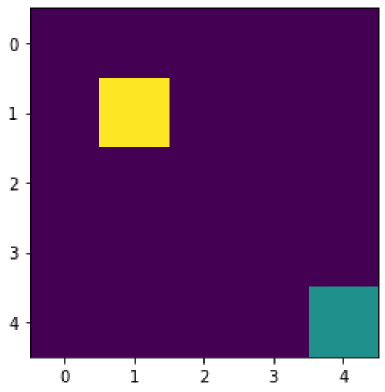
Initial step



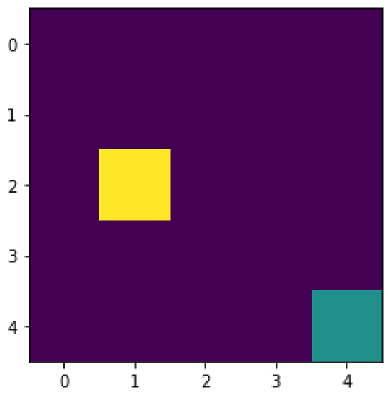
step 1



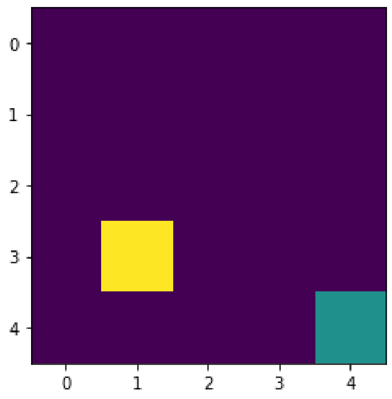
step 2



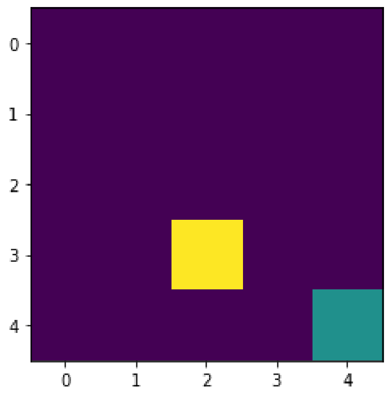
step 3

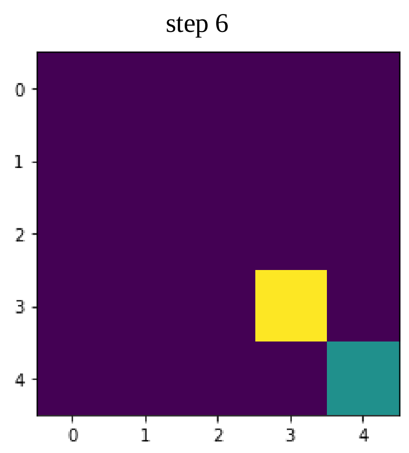


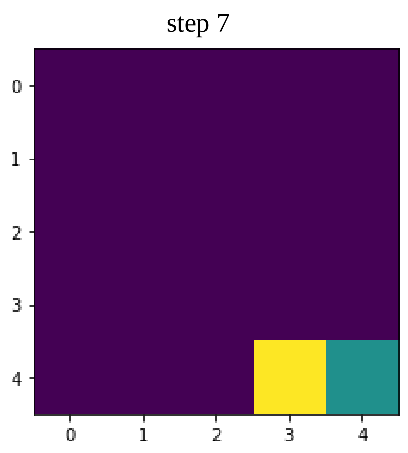
step 4

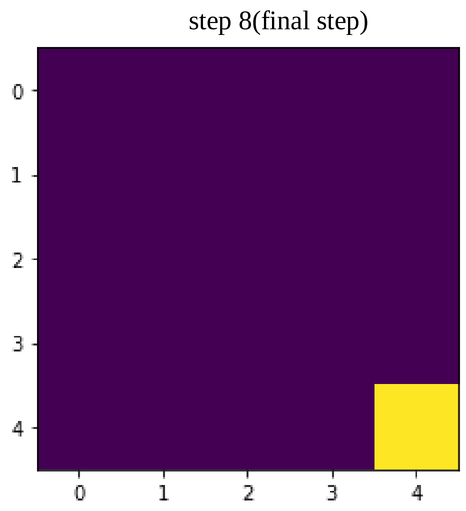


step 5









6 CONCLUSION

The task of building a reinforcement learning agent to navigate the classic 4x4 grid-world environment is achieved. The tasks of Implementing the policy function, Updating the Q-table and implementing the training algorithm is accomplished and it is observed that the minimum number of steps to find the shortest path on the grid environment and reach the goal is 8. The graph of Episode vs Epsilon value and Epsilon vs total rewards is plotted. It is observed that the graphs vary with different values of episodes. The updated Q-table is obtained when the agent runs successfully and the different iterations in which the games are played is also represented.

7 References

* https://nips.cc/Conferences/2015/PaperInformation/StyleFiles

* https://towardsdatascience.com/reinforcement-learning-tutorial-with-open-ai-gym-9b11f4e3c204

* https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2

*https://medium.com/velotio-perspectives/exploring-openai-gym-a-platform-for-reinforcement-learning-algorithms-380beef446dc

* https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

* https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265