Tom and Jerry in Reinforcement Learning

Soumitra Alate
University at Buffalo
Buffalo, NY 14260
smalate@buffalo.edu

I. Overview:

The main task is to teach an agent to navigate in the grid-world environment and find the shortest path to reach the goal. This project combines reinforcement learning and deep learning.

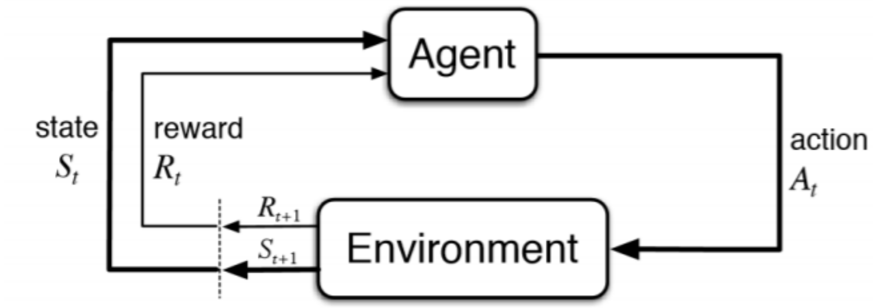
II. Libraries used in project:

1. Matplotlib: Matplotlib is a library used to plot mathematical functions to 2-dimensional graphs for representation.
2. Numpy: Numpy is used to calculate high level mathematical functions like linear algebra and Fourier transforms.
3. Keras: Keras is a high-level API used to build neural networks.
4. IPython: It is an interactive shell, that will help us to display our framework.
5. Time: It will be used to track how much time each computation takes.
6. Math- It provides access to big variety of mathematical functions.
7. Random- Generates a pseudo-random number.

III. Terms used in project:

1. Reinforcement Learning – It is direction in machine learning where an agent learns how to behave in an environment by performing actions and seeing the results.
2. Neural Network – Neural network is a machine learning model that involves intricate interconnection of nodes which are stacked in layers forming a sequential layered model.
3. Deep Learning– Deep learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural network.
4. Markov Decision Process: The Markov decision process, better known as MDP, is an approach in reinforcement learning to take decisions in a grid world environment. A grid world environment consists of states in the form of grids.

Markov Decision Process

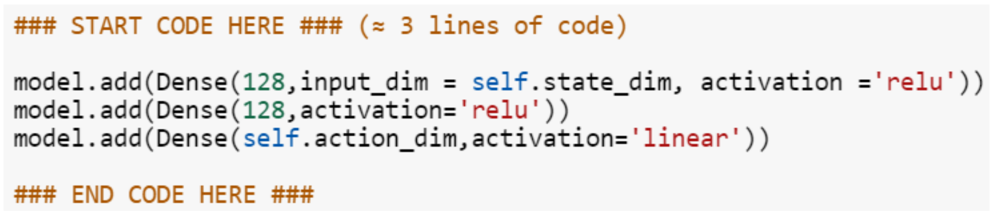


5. The discounting factor: (γ ∈ [0, 1]) penalize the rewards in the future. The future rewards do not provide immediate benefits
6. Exploration: Discovers better action selection. Improves knowledge about the environment.
7. Exploitation: Maximizes the reward based on what agent already knows.

IV. Process:

In this project we have a 5x5 environment where the Agent is Tom and target state is Jerry. The agent performs 3 methods act, observe and replay. The act method takes current state as input and returns the agent action. The action is either selected randomly or it is passed to the neural network based on the randomly generated number. If the generated number is less than epsilon, then random action is selected else it is given to neural network. The observe method saves the tuple (s, a, r, s_) s = state, a=action, r=rewards, s_=next state. The agent learns about the environment using the replay method in which some random samples are given to the neural network for training. Replay method helps the agent to keep track of recent as well as previous observations. Maximum reward the agent can get is 8. Initially the agent explores the environment. After gaining the knowledge the agent starts to exploit. Brain of the agent is where the neural network is implemented. Memory is used to save the experiences of agent.

V. Description of code snippet implementation:

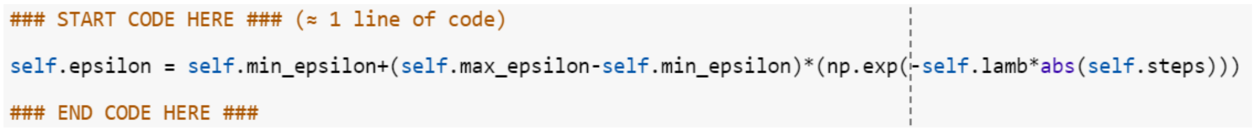


```
### START CODE HERE ### (≈ 3 lines of code)

model.add(Dense(128,input_dim = self.state_dim, activation ='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(self.action_dim,activation='linear'))

### END CODE HERE ###
```

Here 3-layer neural network is built using a keras library which acts as a brain of the agent. The model is created and held in the brain. We have implemented a three-layer neural network with two hidden layers.
Input to the neural network is state_dim which represents the co-ordinates of Tom and Jerry.
In the first hidden layer number of hidden nodes are 128 and activation function used is “Relu”
In the second layer the number of hidden nodes are 128 and activation function used is “Relu”
The output layer has 4 output nodes which represents the Q- values for each action possible in given state and the activation function used is “Linear” which represents the real numbers.

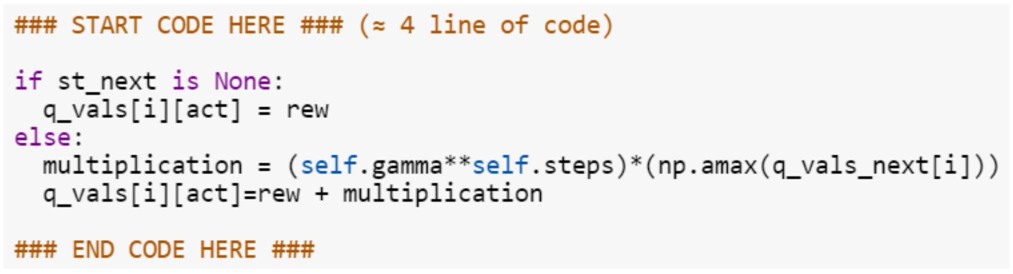


```
### START CODE HERE ### (≈ 1 line of code)

self.epsilon = self.min_epsilon+(self.max_epsilon-self.min_epsilon)*(np.exp(-self.lamb*abs(self.steps)))

### END CODE HERE ###
```

Here the epsilon decay formula for epsilon is implemented. After every iteration we decrease the epsilon value so the probability of agent exploiting the memory increases. We have set the epsilon_max to 1 and epsilon_min to 0.05.



```
### START CODE HERE ### (≈ 4 line of code)

if st_next is None:
  q_vals[i][act] = rew
else:
  multiplication = (self.gamma**self.steps)*(np.amax(q_vals_next[i]))
  q_vals[i][act]=rew + multiplication

### END CODE HERE ###
```

The above function states that if the next step is the goal state then we give the final reward else the
the bellman formula is used which tells us that the maximum future reward is the reward the agent received on entering the current state plus the maximum future reward for the next state.

$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

Question and Answers:

1) Role in training the agent:

The replay method in the code is used to train the agent. The role in training the agent is to not only learn from recent observation but also from previous observations. The replay method randomly samples the observation from memory and these are given to neural network for training so that agent can learn quickly and can reach the goal state in optimum steps.

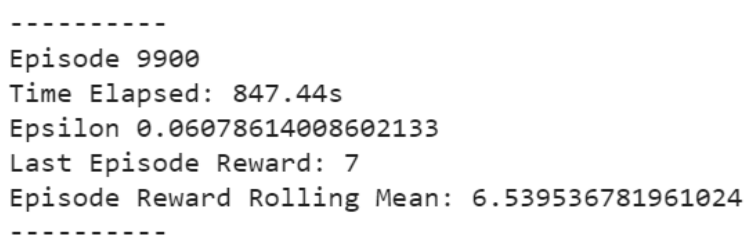
2) How quickly your agent was able to learn?

Using the default setting the agent got the mean average reward of 6.539 in 10000 steps and the time taken was 847.44 sec but after tweaking the hyper-parameters it was observed that the agent got a mean reward of 7.741 in 10000 and the time taken was 781.61 sec. So we can conclude it was able to learn quickly after tweaking the hyper-parameters.

VI. Network Settings:

Training Time (In Seconds)	**Episode Reward Rolling Mean**
847.44	6.539

Table 1: Default Settings (Given in code)



```
----------
Episode 9900
Time Elapsed: 847.44s
Epsilon 0.06078614008602133
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.539536781961024
----------
```

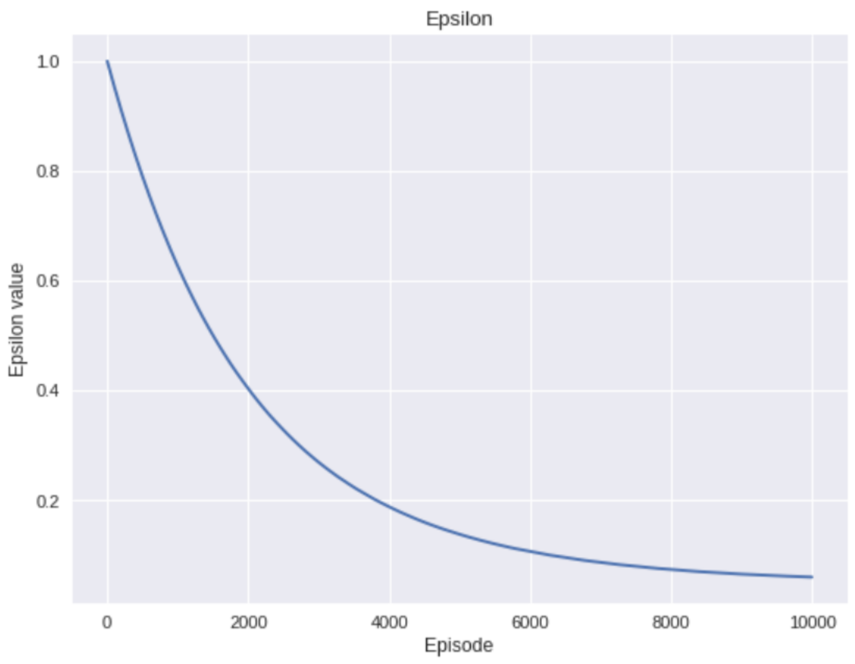


Figure: Episode vs Epsilon

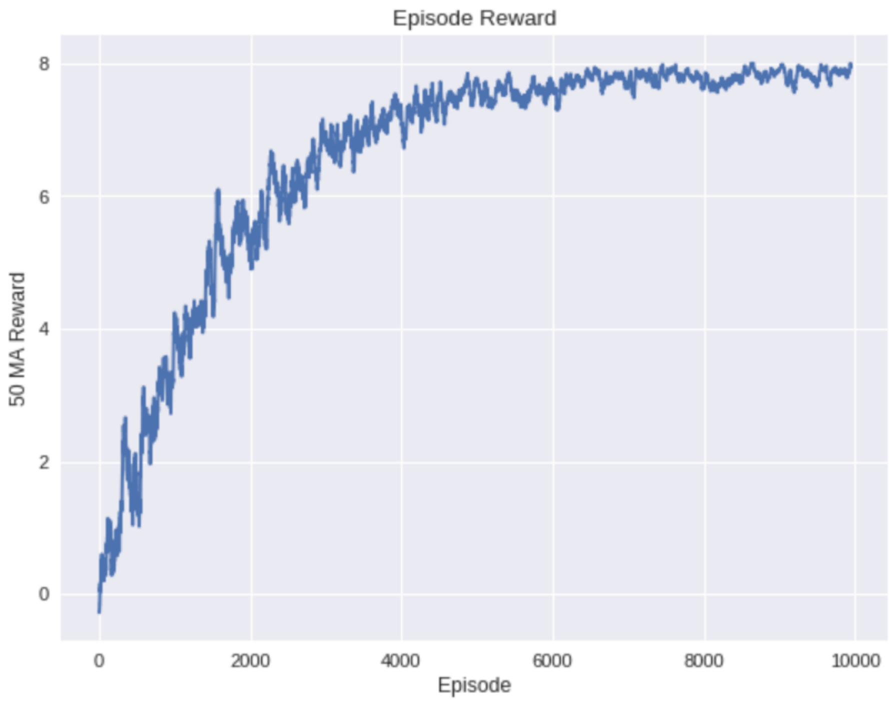
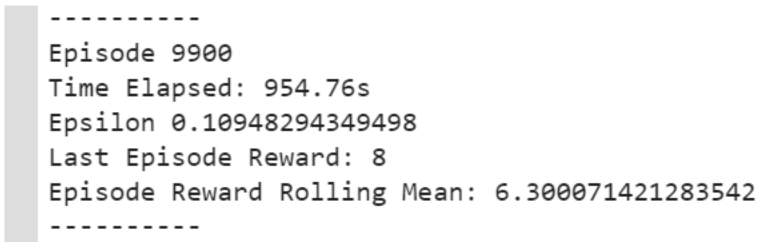


Figure: Episode Reward vs 50 MA Reward

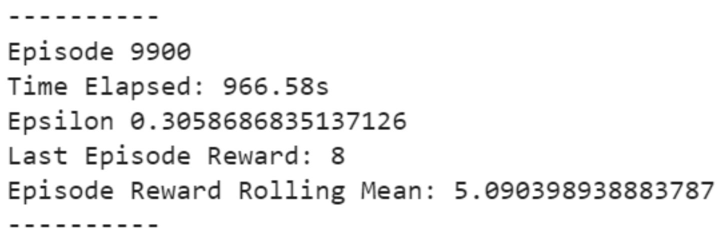
MIN_EPSILON	**Training Time (In Seconds)**	**Episode Reward Rolling Mean**
0.1	954.76	6.300
0.3	966.58	5.090
0.01	757.95	6.668

Table 2: Changing the Hyperparameter MIN_EPSILON

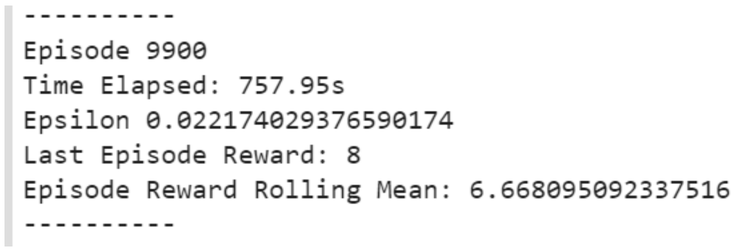
Screenshots of Results for the above table:



```
----------
Episode 9900
Time Elapsed: 954.76s
Epsilon 0.10948294349498
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.300071421283542
----------
```



```
----------
Episode 9900
Time Elapsed: 966.58s
Epsilon 0.3058686835137126
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.090398938883787
----------
```

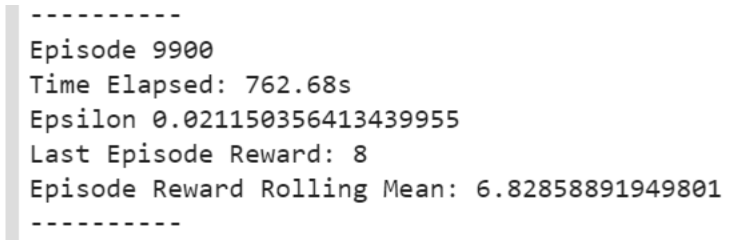


```
----------
Episode 9900
Time Elapsed: 757.95s
Epsilon 0.022174029376590174
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.668095092337516
----------
```

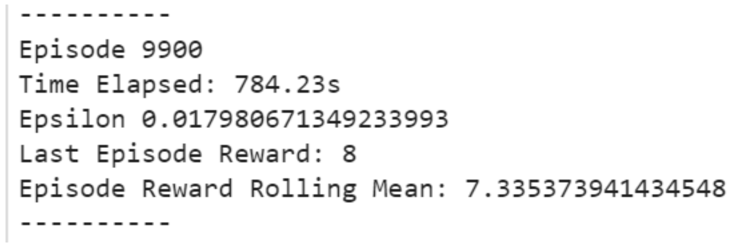
MAX_EPSILON	**Training Time (In Seconds)**	**Episode Reward Rolling Mean**
0.9	762.68	6.828
0.6	784.23	7.355
0.3	781.61	7.741

Table 3: Changing the Hyperparameter MAX_EPSILON

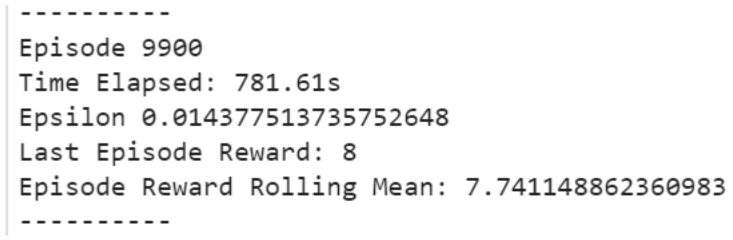
Screenshots of Results for the above table:



```
----------
Episode 9900
Time Elapsed: 762.68s
Epsilon 0.021150356413439955
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.82858891949801
----------
```



```
----------
Episode 9900
Time Elapsed: 784.23s
Epsilon 0.017980671349233993
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.335373941434548
----------
```

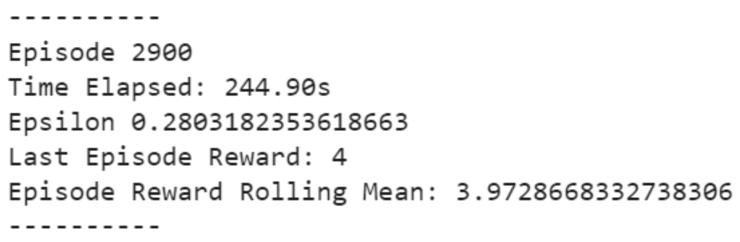


```
----------
Episode 9900
Time Elapsed: 781.61s
Epsilon 0.014377513735752648
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.741148862360983
----------
```

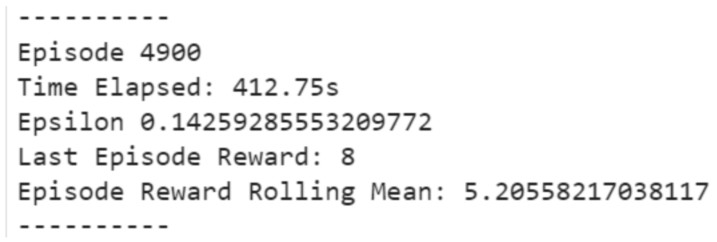
Number of Episodes	**Training Time (In Seconds)**	**Episode Reward Rolling Mean**
3000	244.90	3.972
5000	412.75	5.205
7000	574.95	5.893

Table 4: Changing the Hyperparameter: Number of Episodes

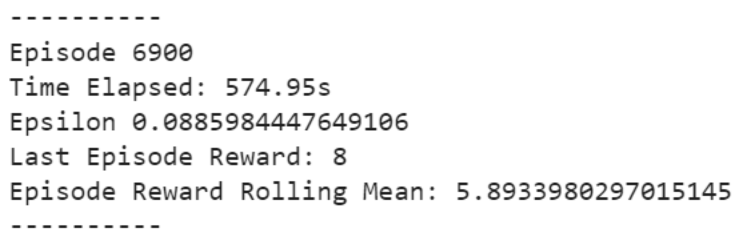
Screenshots of Results for the above table:



```
----------
Episode 2900
Time Elapsed: 244.90s
Epsilon 0.2803182353618663
Last Episode Reward: 4
Episode Reward Rolling Mean: 3.9728668332738306
----------
```



```
----------
Episode 4900
Time Elapsed: 412.75s
Epsilon 0.14259285553209772
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.20558217038117
----------
```

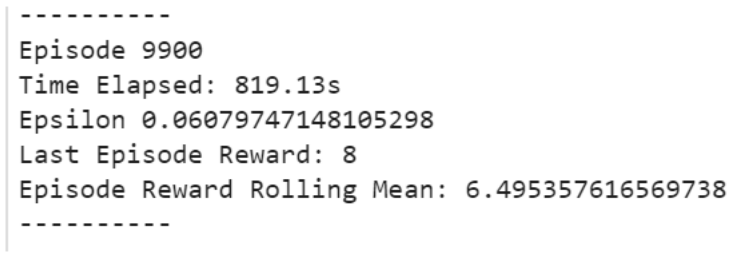


```
----------
Episode 6900
Time Elapsed: 574.95s
Epsilon 0.0885984447649106
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.8933980297015145
----------
```

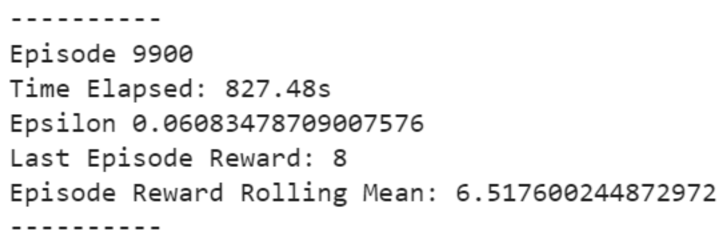
Gamma (Discounting Factor)	Training Time (In Seconds)	Episode Reward Rolling Mean
0.77	819.13	6.495
0.50	827.48	6.517
0.30	828.95	6.479

Table 5: Changing the Hyperparameter: Gamma (Discounting Factor)

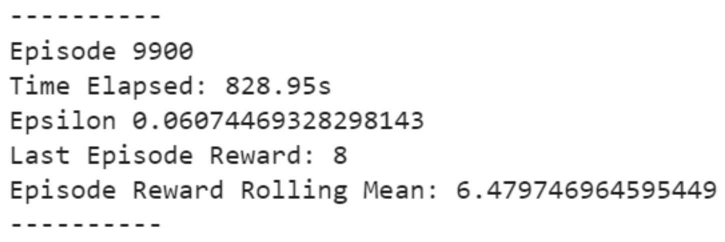
Screenshots of Results for the above table:



```
----------
Episode 9900
Time Elapsed: 819.13s
Epsilon 0.06079747148105298
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.495357616569738
----------
```



```
----------
Episode 9900
Time Elapsed: 827.48s
Epsilon 0.06083478709007576
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.517600244872972
----------
```



```
----------
Episode 9900
Time Elapsed: 828.95s
Epsilon 0.06074469328298143
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.479746964595449
----------
```

VII. Writing Task:

Q1) Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways that can force the agent to explore.

Solution:

If the agent always chooses the action that maximizes the Q-value, then it gets stuck in non-optimal policy wherein the agent does not try to explore the entire environment to find the best action for each state. Two ways that can force the agent to explore are:

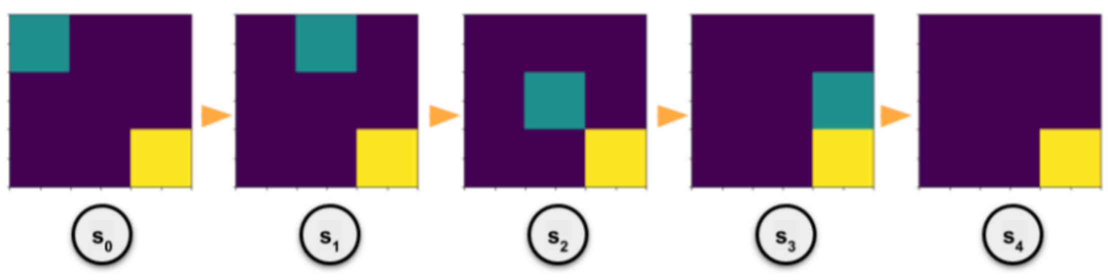
1) To explore the environment, the agent can pick some random actions as we have implemented in the code (if random value is less than epsilon random action is selected). Using such approach will help the agent to explore the environment more and will be able to perform better. If we don't select random actions the agent will not be able to learn about the environment.

2) We can also set the hyper parameters to high values such as gamma can be set to 0.99, epsilon_max = 1, epsilon_min = 0.1. Also the number of episodes can be increased so that the agent can explore the environment. It will help the agent to improve the knowledge about the environment and also it will come to know what actions it should take in a particular state. So setting the hyperparameters to a high value is another approach to explore the environment.

Q2) Calculate Q-value for the given states and provide all the calculation steps.

Solution:

- Environment grid: 3X3
- Actions: UP, DOWN, LEFT, RIGHT
- Rewards: 1 when it moves closer to goal. -1 when it moves away from the goal state. 0 when it does not move at all.
- Gamma (Discounting factor) = 0.99
- Bellman Equation: Q (st, at) = rt + γ ∗ max Q (st+1, a) where st = current state, at =action, rt = reward, γ = discounting factor, st+1 = next state
- One of the possible optimal action sequence.



Calculation Steps:

Initially we start calculating Q-function from the last state (i.e. S4)

S4: If the agent has already reached the target state then the Q-value in state S4 for all possible actions will be 0.

S3: Using bellman equation we will calculate Q values in all state for all possible actions.

Results of the actions (UP and LEFT) for state S3 will be same because of grid symmetry. MAX value for S3 will be obtained after moving down to state S4 so the maximum value for S3 is 1.

- (S3, UP) = -1 + 0.99 * max(S2) = -1 + 0.99*(1.99) = 0.97
- (S3, DOWN) = 1 + 0.99 * max(S4) = 1 + 0.99*(0) = 1
- (S3, LEFT) = -1 + 0.99 * max(S2) = -1 + 0.99*(1.99) = 0.97
- (S3, RIGHT) = 0 + 0.99 * max(S3) = 0 + 0.99*(1) = 0.99

S2: Results of the actions (UP and LEFT) for state S2 will be same and results of the actions (DOWN and RIGHT) for state S2 will be same because of grid symmetry

- (S2, UP) = -1 + 0.99 * max(S1) = -1 + 0.99*(2.97) = 1.94
- (S2, DOWN) = 1 + 0.99 * max(S3) = 1 + 0.99*(1) = 1.99
- (S2, LEFT) = -1 + 0.99 * max(S1) = -1 + 0.99*(2.97) = 1.94
- (S2, RIGHT) = 1 + 0.99 * max(S3) = 1+0.99*(1) = 1.99

S1: Results of the actions (DOWN and RIGHT) for the state S1 will be same because of grid symmetry.

- (S1, UP) = 0 + 0.99 * max(S1) = 0 + 0.99*(2.97) = 2.94
- (S1, DOWN) = 1 + 0.99 * max(S2) = 1 + 0.99*(1.99) = 2.97
- (S1, LEFT) = -1 + 0.99 * max(S0) = -1 + 0.99*(3.94) = 2.90
- (S1, RIGHT) = 1 + 0.99 * max(S2) = 1 + 0.99*(1.99) = 2.97

S0: Results of the actions (UP and LEFT) for state S0 will be same and results of actions (DOWN and RIGHT) for state S0 will be same because of grid symmetry.

- (S0, UP) = 0 + 0.99 * max(S0) = 0 + 0.99*(3.94) = 3.90
- (S0, DOWN) = 1 + 0.99 * max(S1) = 1 + 0.99*(2.97) = 3.94
- (S0, LEFT) = 0 + 0.99 * max(S0) = 0 + 0.99*(3.94) = 3.90
- (S0, RIGHT) = 1 + 0.99 * max(S1) = 1 + 0.99*(2.97) = 3.94

STATE	UP	DOWN	LEFT	RIGHT
0	3.90	3.94	3.90	3.94
1	2.94	2.97	2.90	2.97
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

Table 6: Q-Table

VIII. Conclusion:

After tweaking the hyper-parameters it was observed that the agent got the highest mean reward for following setting and the time taken was 781 secs.

- MAX_EPSILON = 0.3
- MIN_EPSILON = 0.1
- Episodes =10000
- Gamma = 0.99

IX. Works Cited:

1. https://medium.com/@curiousily/solving-an-mdp-with-q-learning-from-scratch-deep-reinforcement-learning-for-hackers-part-1-45d1d360c120
2. https://medium.com/init27-labs/understanding-q-learning-the-cliff-walking-problem-80198921abbc
3. https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47
4. https://medium.com/@m.alzantot/deep-reinforcement-learning-demystified-episode-0-2198c05a6124
5. http://mnemstudio.org/path-finding-q-learning-tutorial.htm
6. https://stable-baselines.readthedocs.io/en/master/modules/dqn.html
7. https://medium.com/coinmonks/implement-reinforcement-learning-using-markov-decision-process-tutorial-272012fdae51