Report On : Project – 4 Tom and Jerry in Reinforcement Learning

Neeraj Ajit Abhyankar **UB Person No. : 50290958** **UBID : nabyank**

Objective :

In this project we have to combine the reinforcement learning and deep learning models. The task is to teach the model / agent to reach its goal in the grid-world environment i.e. the cartoon character TOM catches the character JERRY. The shortest path is to found such that TOM reaches JERRY, also the initial positions will be changed at every reset of the game. The Deep Q-Network (DQN) approach successfully works for this task. The DQN model combines the deep learning with the reinforcement learning.

Key Concepts :

Environment : The environment is a 5x5 grid-world in which the agent and the goal are set with changing positions at every reset. The agent is TOM cat and the goal is JERRY mouse, the task in this environment is to train the model using DQN such that the agent reaches the goal.

Agent : An Agent is the entity that interacts with the environment to train itself to find the shortest path to the goal. In our case the agent is TOM cat which interacts with the environment to reach the goal JERRY mouse. The agent is a green square in the environment of 5x5 grid, the agent learns and trains with the rewards it acquires with each episode.

Goal : The Goal in our environment is the JERRY mouse, which the agent has to reach and train the model on the rewards achieved with each episode with initial positions changed after every reset.

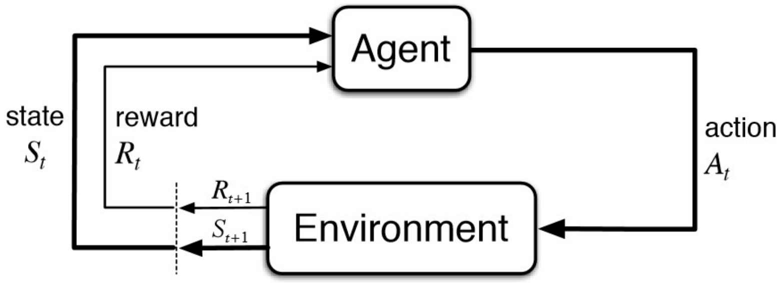
Discount Factor : The **discount factor** is multiplied by future rewards as discovered by the agent in order to dampen these rewards effect on the agent's choice of action.

Reward : A **reward** is the feedback by which we measure the success or failure of an agent's actions.

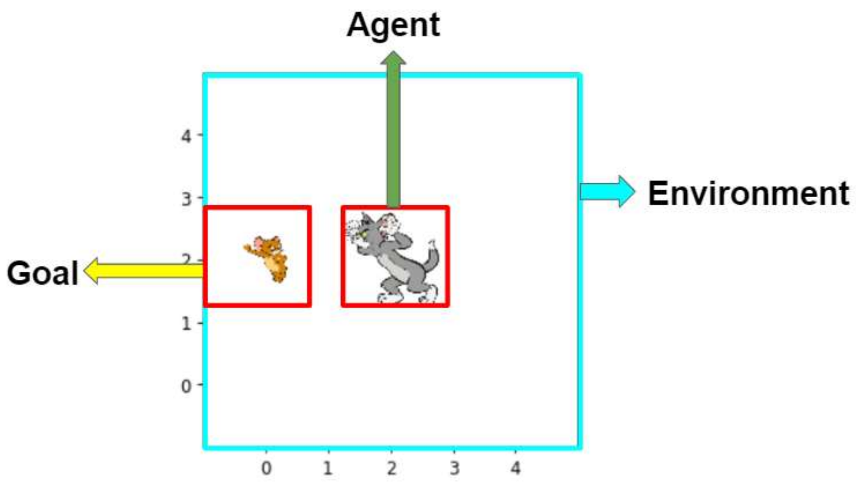
Q-value : **Q-value** is long term value with the discount, except that it takes an extra parameter, the current action. $Q\pi(s, a)$ refers to the long-term return of the current state s, taking action a under policy π. Q maps state-action pairs to rewards.

Reinforcement Learning : Reinforcement Learning is basically model with a goal and an agent in an environment where the training takes place on the action and its result basis, i.e. the agent performs an action and then sees its results and learns from the acquired values or data values observed.

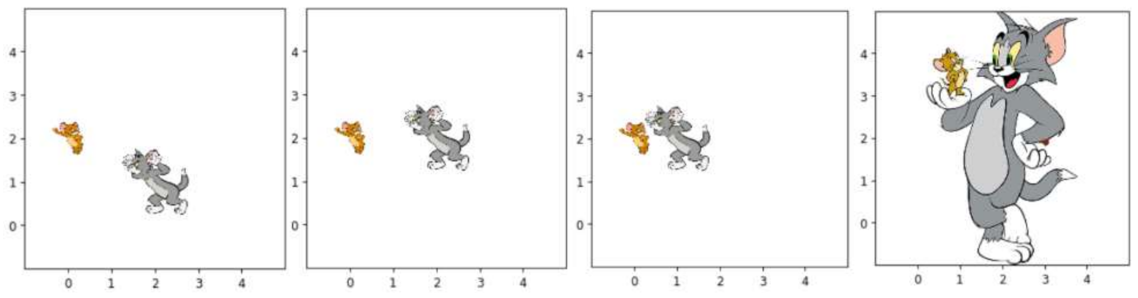
Markov Decision Process : The Markov Process plays an important role in the reinforcement learning model, a Markov Decision Process is a 5-Tuple wherein the actions, states, predictions, discounting factor are used to determine the next set of actions, states, predictions, discounting factor.



About the Environment and Model :
The Environment as mentioned above a 5x5 grid with two squares i.e. Green – Agent and Yellow – Goal; the main task in the environment is for the green square to reach the yellow square which marks the completion of one episode, we have to iterate the model for many such episodes to train the model and make it learn the game. TOM cat has to catch JERRY mouse is the game, in this with each episode TOM cat or our agent learns the shortest path to get to the goal or the JERRY mouse. The Model will be training on the DQN algorithm, which successful integrates the Deep Learning and Reinforcement Learning and yields better results.

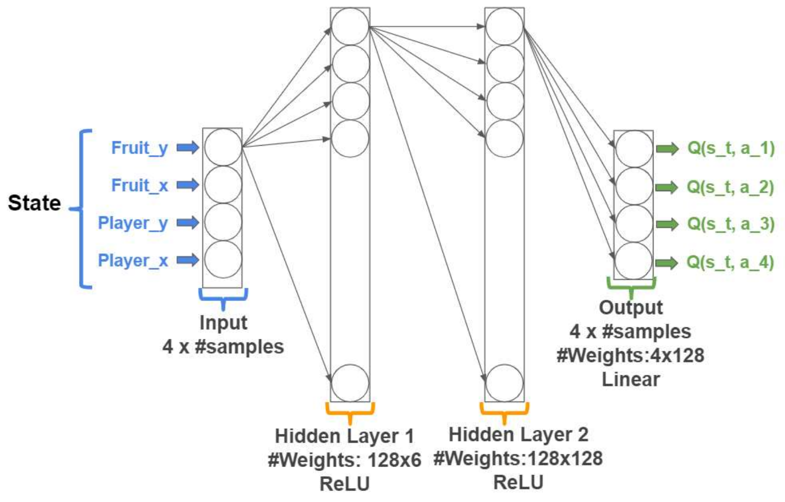


The below images show that the agent step by step can reach the goal i.e. TOM cat our agent reaches the JERRY mouse our goal. For every step the agent is given a reward of one point so that it knows that it is going towards the goal and can efficiently find the shortest path.



The Brain & The Memory :

The Brain of the agent in our environment is the part where the model is created and stored. The Brain is where the model trains and learns the environment and the memory is where it stores the learned data from the model where all the shortest paths learned are stored.



The above image shows the Neural Network Structure in which there are 3- hidden layers with activation functions LINEAR -> RELU -> RELU -> LINEAR with state_dim = 128, action_dim = 4 and 128 nodes.

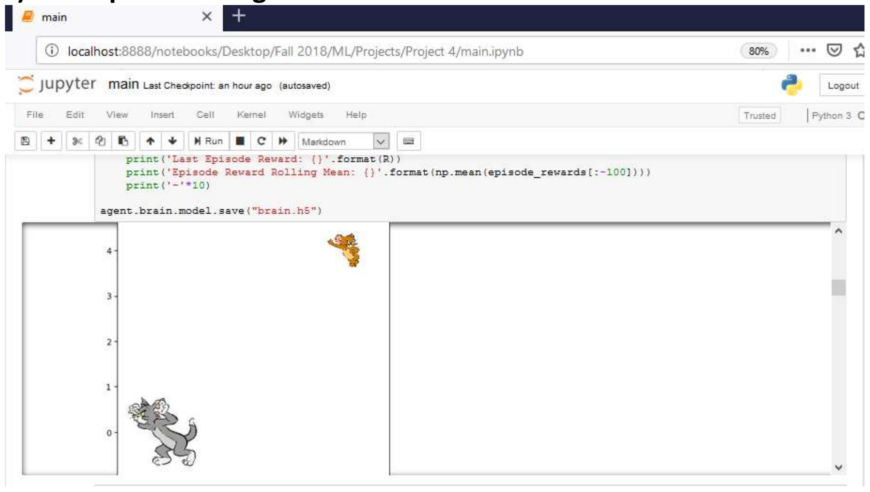
The Memory stores every experience which the agent has in the past with the rewards and the loss of points where it does not take the right path towards the goal. The memory also serves as a database from which the agent can extract the shortest path when it finds the path to be one with previous experience.

Hyper - Parameter Changes :

Parameter Set 1 :

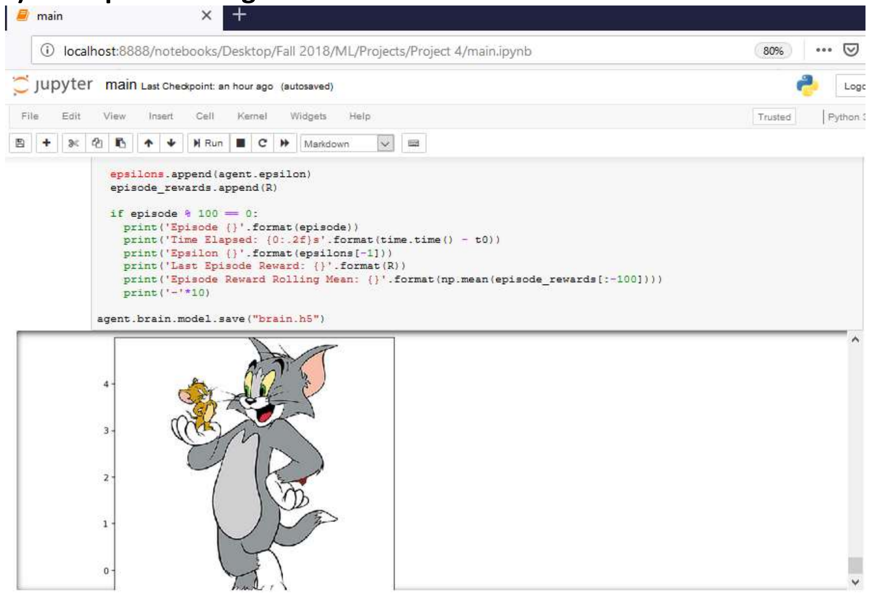
max_epsilon = 1 ; min_epsilon = 0.05 ; lambda = 0.00005 ; num_episodes = 10000

1) First Episode Image :


main
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

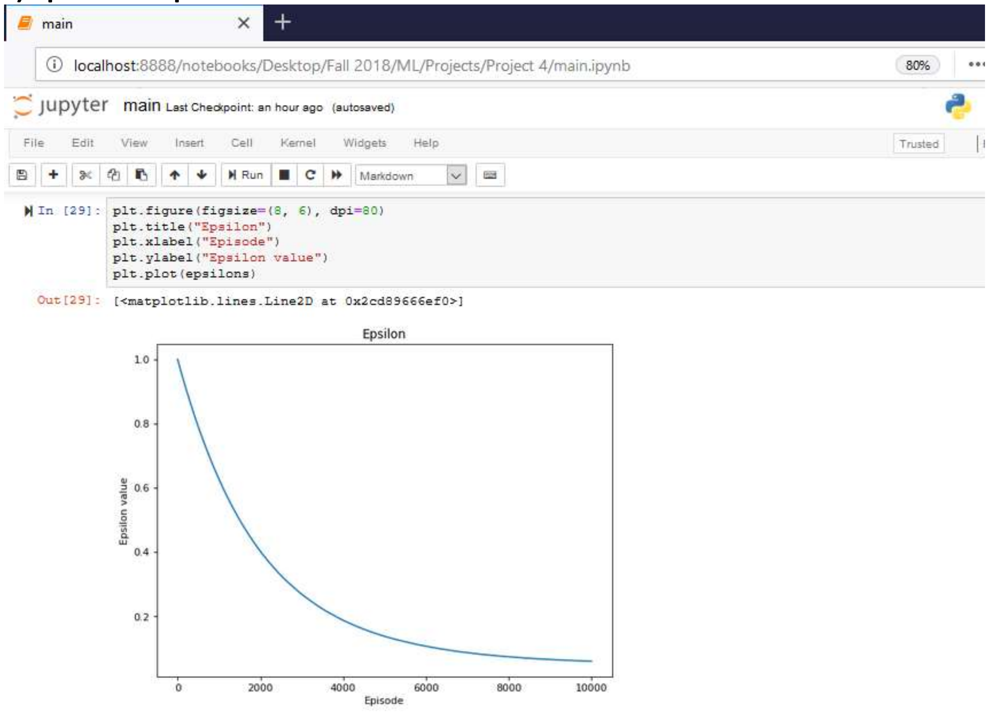
```
    print('Last Episode Reward: {}'.format(R))
    print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
    print('-'*10)

agent.brain.model.save("brain.h5")
```

2) Last Episode Image :


main
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Markdown

```
  epsilons.append(agent.epsilon)
  episode_rewards.append(R)

  if episode % 100 == 0:
    print('Episode {}'.format(episode))
    print('Time Elapsed: {0:.2f}s'.format(time.time() - t0))
    print('Epsilon {}'.format(epsilons[-1]))
    print('Last Episode Reward: {}'.format(R))
    print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
    print('-'*10)

agent.brain.model.save("brain.h5")
```

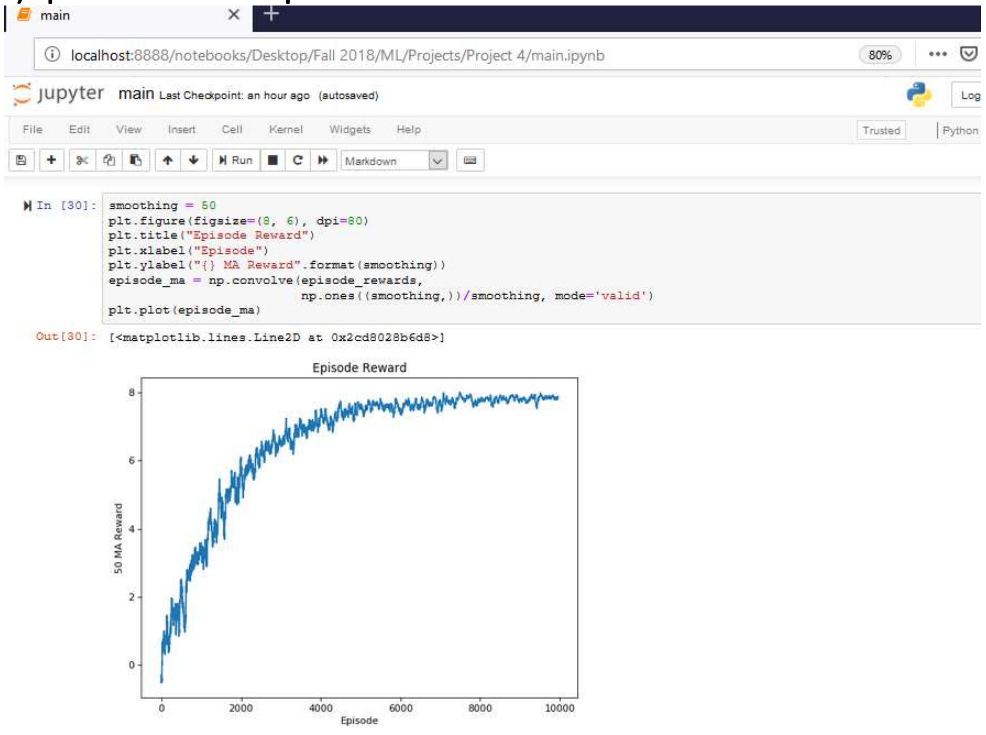
3) Epsilon Graph :


main
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Markdown

```
In [29]: plt.figure(figsize=(8, 6), dpi=80)
         plt.title("Epsilon")
         plt.xlabel("Episode")
         plt.ylabel("Epsilon value")
         plt.plot(epsilons)

Out[29]: [<matplotlib.lines.Line2D at 0x2cd89666ef0>]
```

Epsilon
Epsilon value
1.0
0.8
0.6
0.4
0.2
0
2000
4000
6000
8000
10000
Episode

4) Episode Reward Graph :


main
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: an hour ago (autosaved)
Log
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Markdown

```
In [30]: smoothing = 50
         plt.figure(figsize=(8, 6), dpi=80)
         plt.title("Episode Reward")
         plt.xlabel("Episode")
         plt.ylabel("{} MA Reward".format(smoothing))
         episode_ma = np.convolve(episode_rewards,
                                  np.ones((smoothing,))/smoothing, mode='valid')
         plt.plot(episode_ma)

Out[30]: [<matplotlib.lines.Line2D at 0x2cd8028b6d8>]
```

Episode Reward
50 MA Reward
8
6
4
2
0
0
2000
4000
6000
8000
10000
Episode

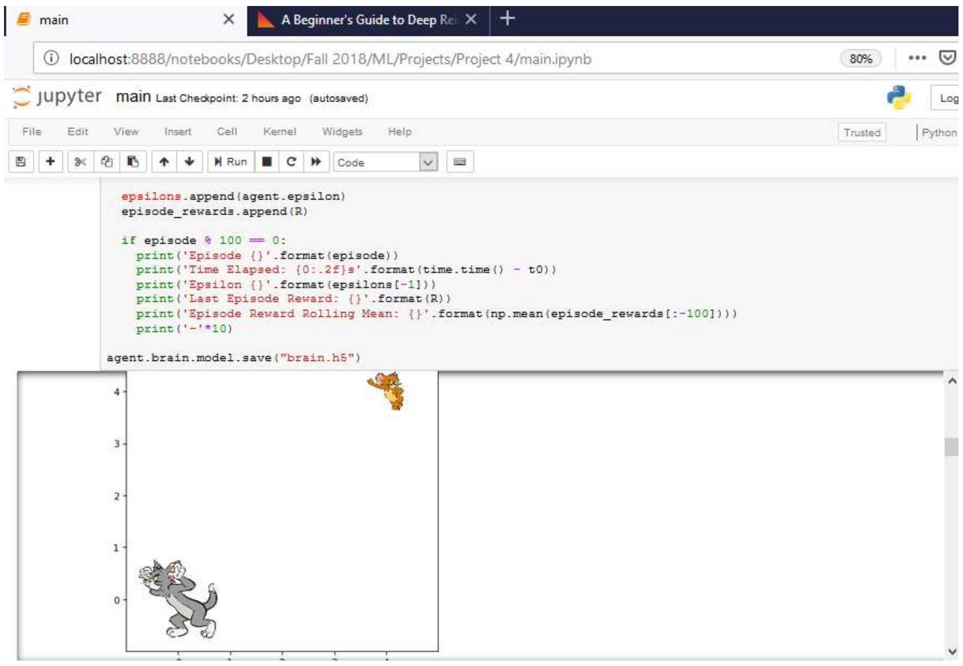
Graph Analysis :

For parameter set 1 the graph analysis shows that the epsilon value goes on decreasing exponentially as the number of episodes increase. Also as the number of episodes increase the reward graph shows an exponential increase, this seems to happen because of the brain. The brain stores the location of the memory / memorizes the episodes in which the agent reaches the goal. Thus, improving the performance of the model as it trains the agent to find the shortest path to reach the goal. The first image shows the initial position of the agent and the goal where the model training begins, where as the last image shows the end of the model training where the agent has reached the goal finding the shortest path.

Parameter Set 2 :

max_epsilon = 1 ; min_epsilon = 0.1 ; lambda = 0.0001 ; num_episodes = 5000 ; DenseLayer2 = softmax

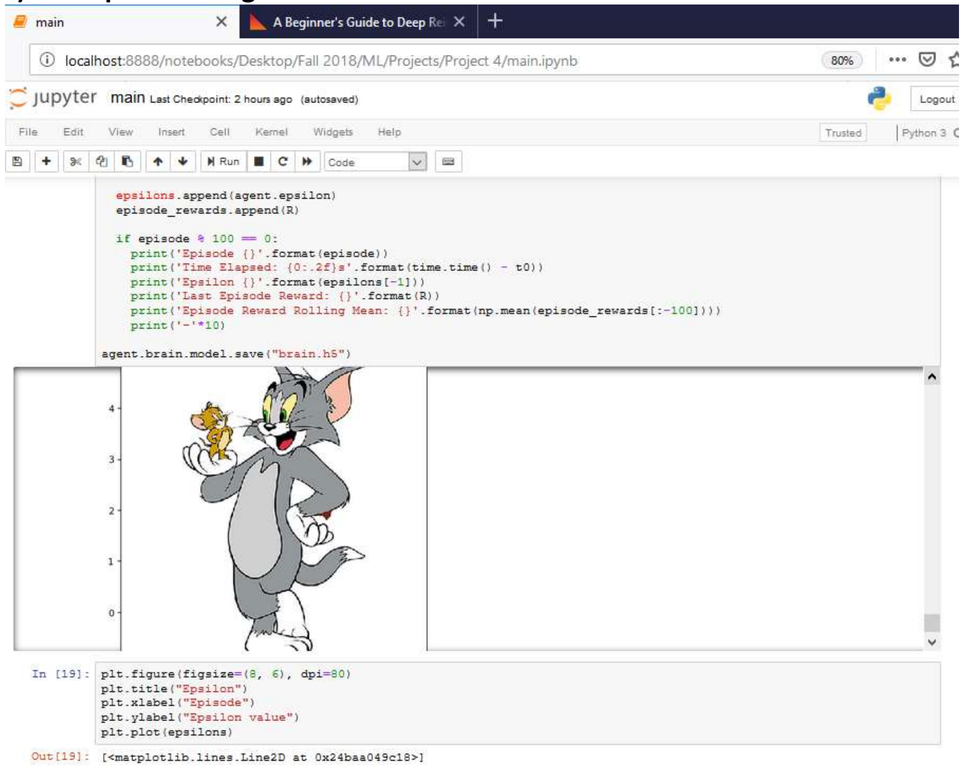
1) First Episode Image :



main
A Beginner's Guide to Deep Re
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: 2 hours ago (autosaved)
Log
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code
4
3
2
1
0

```
 epsilons.append(agent.epsilon)
 episode_rewards.append(R)

 if episode % 100 == 0:
   print('Episode {}'.format(episode))
   print('Time Elapsed: {0:.2f}s'.format(time.time() - t0))
   print('Epsilon {}'.format(epsilons[-1]))
   print('Last Episode Reward: {}'.format(R))
   print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
   print('-'*10)

agent.brain.model.save("brain.h5")
```

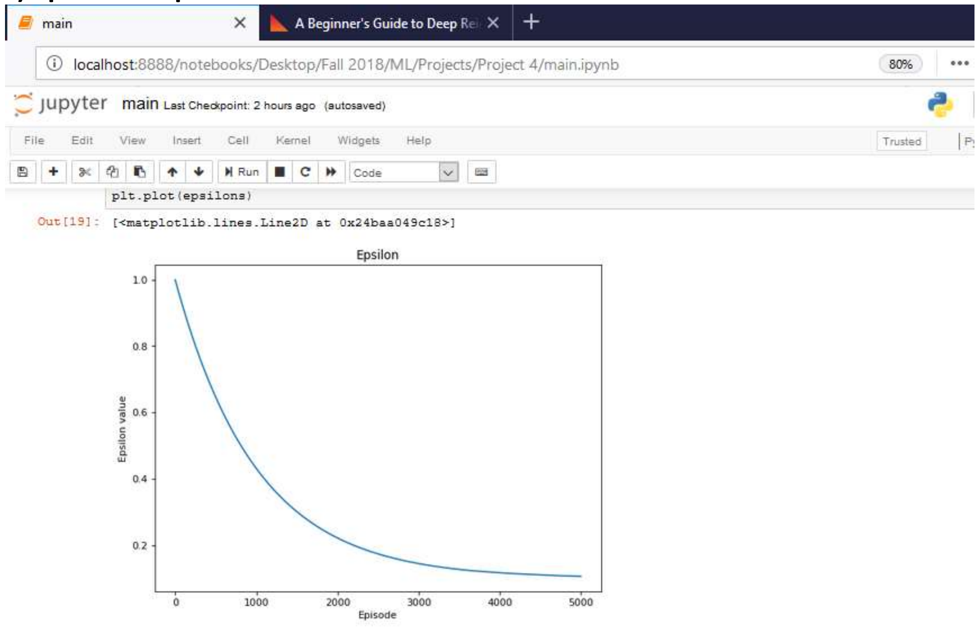
2) Last Episode Image :



```
  epsilons.append(agent.epsilon)
  episode_rewards.append(R)

  if episode % 100 == 0:
    print('Episode {}'.format(episode))
    print('Time Elapsed: {0:.2f}s'.format(time.time() - t0))
    print('Epsilon {}'.format(epsilons[-1]))
    print('Last Episode Reward: {}'.format(R))
    print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
    print('-'*10)

agent.brain.model.save("brain.h5")
```

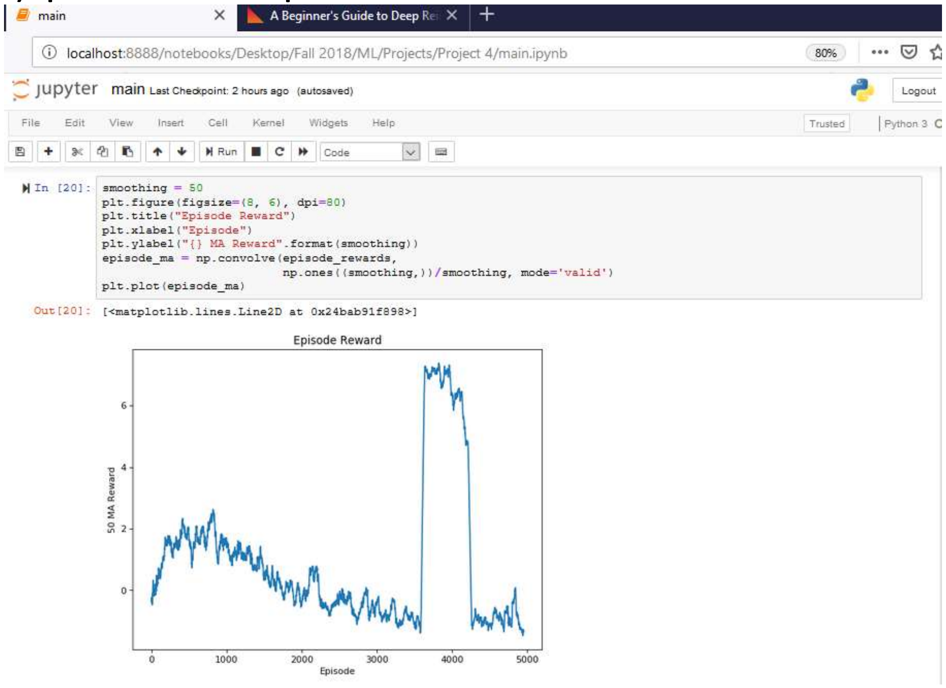
```
In [19]: plt.figure(figsize=(8, 6), dpi=80)
         plt.title("Epsilon")
         plt.xlabel("Episode")
         plt.ylabel("Epsilon value")
         plt.plot(epsilons)

Out[19]: [<matplotlib.lines.Line2D at 0x24baa049c18>]
```

3) Epsilon Graph :



```
plt.plot(epsilons)

Out[19]: [<matplotlib.lines.Line2D at 0x24baa049c18>]
```

4) Episode Reward Graph :


main
A Beginner's Guide to Deep Re
localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Episode Reward
50 MA Reward
Episode
0
2
4
6
0
1000
2000
3000
4000
5000

```
In [20]: smoothing = 50
         plt.figure(figsize=(8, 6), dpi=80)
         plt.title("Episode Reward")
         plt.xlabel("Episode")
         plt.ylabel("{} MA Reward".format(smoothing))
         episode_ma = np.convolve(episode_rewards,
                                  np.ones((smoothing,))/smoothing, mode='valid')
         plt.plot(episode_ma)

Out[20]: [<matplotlib.lines.Line2D at 0x24bab91f898>]
```

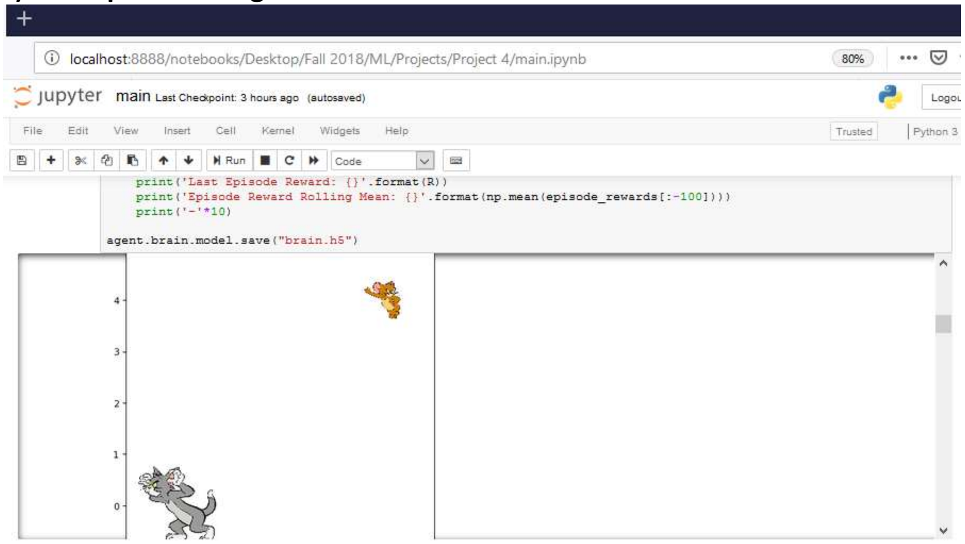
Graph Analysis :

For parameter set 2 there was a change made in the second dense layer of the neural network, wherein the activation function was changed to 'softmax'. This resulted in the same exponential decrease in the epsilon graph but the reward graph was disturbed as the agent could not reach the goal as many times as in parameter set 1. Thus the accuracy of the model was severely degraded. Also the first image shows the initial position of the agent and the goal where the model training begins, where as the last image shows the end of the model training where the agent has finally reached the goal finding the shortest path even after severe degradation in the model. This shows that the training of the model resulted in the decreased accuracy for agent to reach the goal but the training of the model was successful in making the agent learn to reach its goal.

Parameter Set 3 :

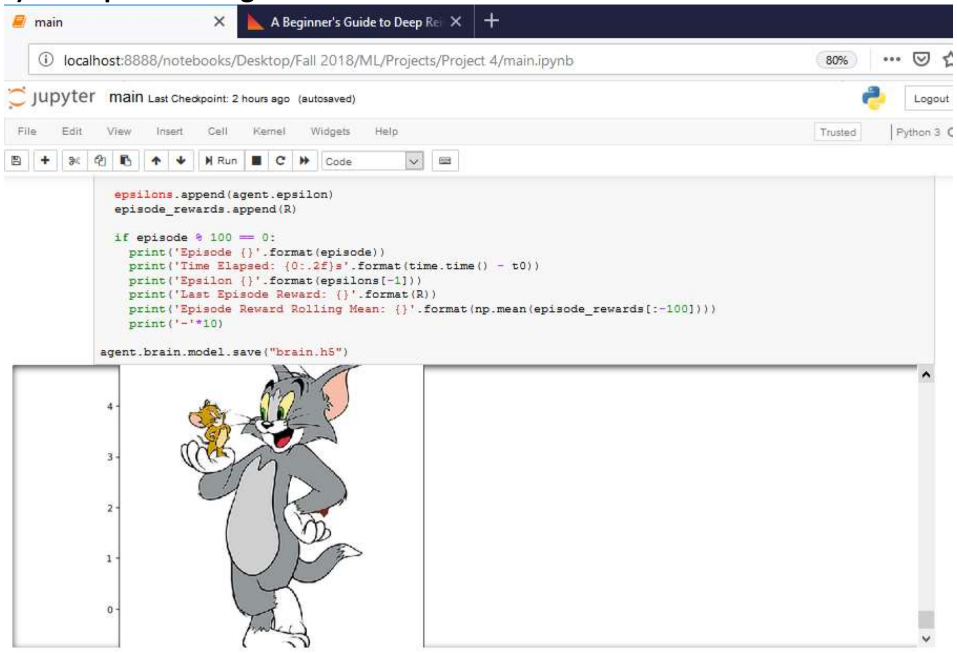
max_epsilon = 1 ; min_epsilon = 0.002 ; lambda = 0.000005 ; num_episodes = 20000

1) First Episode Image :



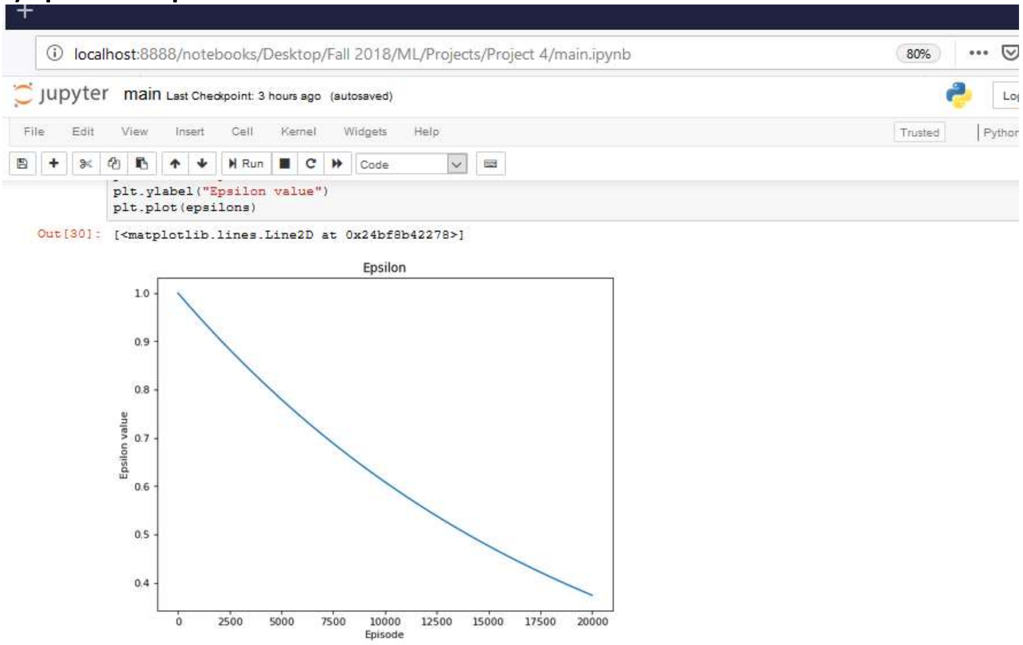
```
    print('Last Episode Reward: {}'.format(R))
    print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
    print('-'*10)

agent.brain.model.save("brain.h5")
```

2) Last Episode Image :



```
  epsilons.append(agent.epsilon)
  episode_rewards.append(R)

  if episode % 100 == 0:
    print('Episode {}'.format(episode))
    print('Time Elapsed: {0:.2f}s'.format(time.time() - t0))
    print('Epsilon {}'.format(epsilons[-1]))
    print('Last Episode Reward: {}'.format(R))
    print('Episode Reward Rolling Mean: {}'.format(np.mean(episode_rewards[:-100])))
    print('-'*10)

agent.brain.model.save("brain.h5")
```

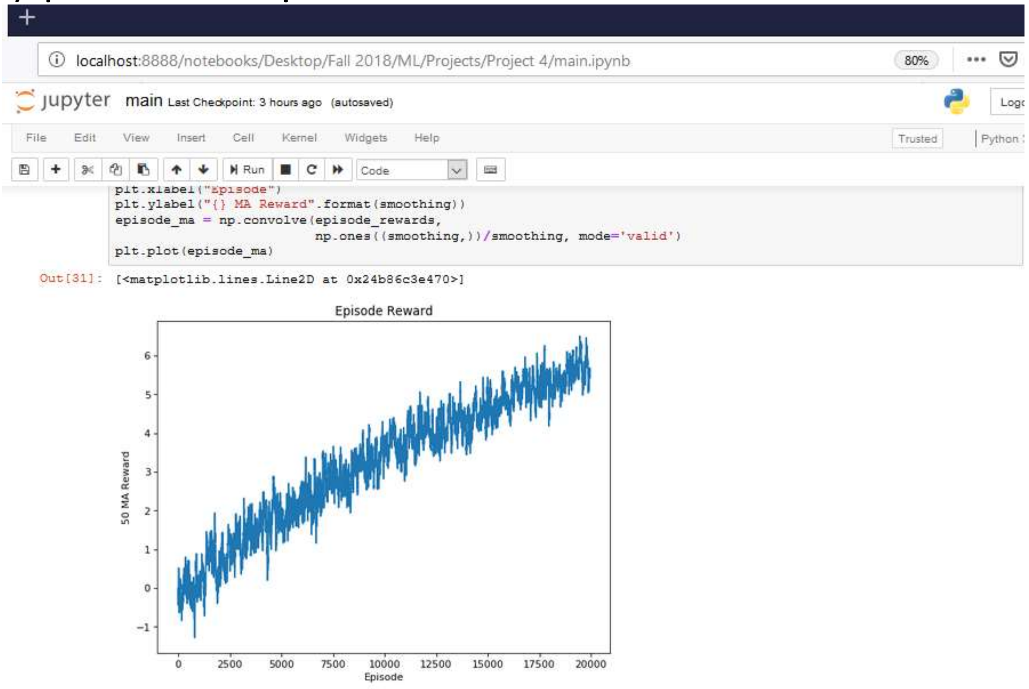
3) Epsilon Graph :


localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: 3 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code
Epsilon
Epsilon value
Episode
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
2500
5000
7500
10000
12500
15000
17500
20000

```
plt.ylabel("Epsilon value")
plt.plot(epsilons)
```

```
Out[30]: [<matplotlib.lines.Line2D at 0x24bf8b42278>]
```

4) Episode Reward Graph :


localhost:8888/notebooks/Desktop/Fall 2018/ML/Projects/Project 4/main.ipynb
80%
jupyter main Last Checkpoint: 3 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code
Episode Reward
50 MA Reward
Episode
6
5
4
3
2
1
0
−1
0
2500
5000
7500
10000
12500
15000
17500
20000

```
plt.xlabel("Episode")
plt.ylabel("{} MA Reward".format(smoothing))
episode_ma = np.convolve(episode_rewards,
                         np.ones((smoothing,))/smoothing, mode='valid')
plt.plot(episode_ma)
```

```
Out[31]: [<matplotlib.lines.Line2D at 0x24b86c3e470>]
```

Graph Analysis :

For parameter set 3 it was observed that there was more accuracy compared to parameter set 2 and similar accuracy as parameter set 1 as the number of episodes was increased to 20,000 with other hyper-parameters. The epsilon decrease was less exponential compared to the parameter set 1 and the reward graph was much denser as more number of episodes were used to train the model. The first image shows the initial position of the agent and the goal where the model training begins, where as the last image shows the end of the model training where the agent has reached the goal finding the shortest path successfully and more accurately with more number of episodes and hyper-parameter changes.

Question Answers :

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

A—

The Basic definition of the Q-value is that it the reward measured and gained after performing an action in a particular state. To understand how the model will work when maximum Q-value is selected at each state; consider the agent initializes with a random reset position and starts exploring with each random action taken to reach towards the goal / target in which the maximum reward is achieved at each step near the goal / target. Say we make the agent choose the maximum Q-value by choosing maximum reward at each step, the probability or chances to reach the goal / target increases more since the model has learned and trained and will utilize the data from brain, also the agent will have more chances of reaching the goal in lesser number of steps than the estimated.
The disadvantage of choosing maximum Q-value / Reward at each step will result in the agent to exploit more and explore less i.e. the agent will stop exploring new paths towards the goal / target. Also it misses the chance of greater rewards with other paths which it will not explore when it will utilize the memorized paths instead of new shortest paths. Also if we make the agent choose on a random basis the steps and actions to be taken, this way the agent will explore more instead of exploiting and will reach the goal / target slower than in the maximizing situation.

Two possible ways to force the Agent to Explore :

1) We can increase the discount factor gamma in the equation of the Q-values, as the discount factor increases the reward will decrease and this will make the agent explore more. This can also result in the agent to not follow optimum path but it will reach the goal / target ultimately. Also it can result in the agent to find a better optimum / shortest path to the goal / target.

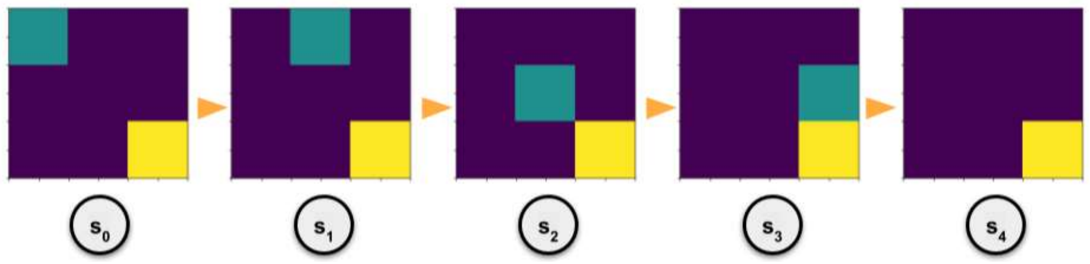
2) We can also decrease the value of epsilon i.e. the exploration rate as it will in turn decrease the brain dependency i.e. we have to decrease the value of epsilon gradually instead of exponentially decreasing graph. This will make the agent explore more instead of exploiting and find new shortest paths and reach the goal / target slower nut eventually it will. Thus we can say that decreasing the value of discounting factor / gamma will result in the agent to explore the environment and find new paths to the goal / target by reducing the brain dependency.

Q2. Calculate Q-values and provide all the calculation steps.

Consider an environment which is a 3x3 grid, where one space of the grid is occupied by the agent (green square) and another is occupied by a goal (yellow square). The agent's action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT. The goal is to have the agent move onto the space that the goal is occupying in as little moves as possible. Initially, the agent is set to be in the upper-left corner and the goal is in the lower-right corner. The agent receives a reward of:

1 when it moves closer to the goal
-1 when it moves away from the goal
0 when it does not move at all (e.g., tries to move into an edge)

Consider the following possible optimal set of actions and their resulting states, that reach the goal in the smallest number of steps:



A—

Given : $Q(s_t; a_t) = r_t + \text{gamma} * \max Q(s_t+1; a)$ **where gamma = 0.99**

The Q-Table for Reinforcement Learning Task :

State	Up	Down	Right	Left
0	3.90	3.94	3.94	3.90
1	2.94	2.97	2.97	2.90
2	1.94	1.99	1.99	1.94
3	0.97	1	0.99	0.97
4	0	0	0	0

For Each Individual state the Q-table value has to be calculated :

The Symmetric Matrix is given as below for the Q-tale calculation :

X0	X1B	YB
X1A	X2	X3B
YA	X3A	G

Where :

X0 – is the initial position of the agent,
X1A & X1B – Similar states in the matrix due to symmetry and X1A will e considered as X1B with max q-values.
X3A & X3B – Similar states in the matrix due to symmetry and X3A will e considered as X3B with max q-values**.**
YA & YB – Similar states in the matrix due to symmetry and YA will e considered as YB with max q-values.
X2 – State in the middle of the matrix.
G – This the Goal where the agent has to reach.

After State **X3** when we want to go to state **X4** the reward will be zero if we go in any direction therefore the table is filled with zeros.

1) Say for State **X3B :**

Action to go Up :	$Q_t3 = -1 + 0.99 * \max (Q_tY) = -1 + 0.99*1.99 = -1 + 1.97 = 0.97$
Action to go Down :	$Q_t3 = 1 + 0.99 * \max (Q_tG) = 1$
Action to go Left :	$Q_t3 = -1 + 0.99 * \max (Q_t2) = - 1 + 0.99*1.99 = 0.97$
Action to go Right :	$Q_t3 = 0 + 0.99 * \max(Q_t3) = 0.99$

2) Say for State **X2 :**

Action to go Up :	$Q_t2 = -1 + 0.99 * \max (Q_t1) = -1 + 0.99 *2.97 = 1.94$
Action to go Down :	$Q_t2 = 1 + 0.99 * \max (Q_t3) = 1.99$
Action to go Left :	$Q_t2 = -1 + 0.99 * \max (Q_t1) = -1 + 0.99 * 2.97 = 1.94$
Action to go Right :	$Q_t2 = 1 + 0.99 * \max (Q_t3) = 1.99$

3) Say for State **X1B :**

Action to go Up :	$Q_t1 = 0 + 0.99 * \max (Q_t1) = 0.99 *2.97 = 2.94$
Action to go Down :	$Q_t1 = 1 + 0.99 * \max (Q_t2) = 1+ 0.99*1.99 = 1+1.97 = 2.97$
Action to go Left :	$Q_t1 = -1 + 0.99 * \max (Q_t0) = -1 + .99*3.94 = 2.90$
Action to go Right :	$Q_t1 = 1 + 0.99 * \max (Q_tP) = 1+ 0.99*1.99 = 1+1.97 = 2.97$

4) Say for State **YB :**

Action to go Up :	$Q_tP = 0 + 0.99 * \max (Q_tP) = 1.97$
Action to go Down :	$Q_tP = 1 + 0.99 * \max (Q_t3) = 1.99$
Action to go Left :	$Q_tP = 0 + 0.99 * \max (Q_tP) = 1.97$
Action to go Right :	$Q_tP = -1 + 0.99 * \max (Q_t1) = -1 + 0.99*2.97 = 1.94$

5) Say for State **X0 :**

Action to go Up :	$Q_t0 = 0 + 0.99 * \max (Q_t0) = 3.90$
Action to go Down :	$Q_t0 = 1 + 0.99 * \max (Q_t1) = 1+ 0.99*2.97 = 3.94$
Action to go Left :	$Q_t0 = 1 + 0.99 * \max (Q_t1) =1+ 0.99*2.97 = 3.94$
Action to go Right :	$Q_t0 = 0 + 0.99 * \max (Q_t0) = 3.90$

Key Imports :

1) import random, math, time - Import random numbers library function, math functions library and time functions library.

2) import numpy as np – Import scientific computing library offered by python as np.

3) from keras.models import Sequential – Import sequential model from keras libraries.

4) from keras.layers import * - Import keras functional layer library.

5) from keras.optimizers import * - Import keras optimization functions library.

6) import matplotlib – Import plotting library of python

7) import matplotlib.pyplot as plt - Import plotting library as matlab plots as plt

8) from matplotlib.image import imread – import imread as a pyplot function to read images as in matlab.

9) from matplotlib import rc, animation – Import font styles and animation for plotting graphs.

10) from IPython import display – Import display function to show images from a file.

11) from IPython.display import HTML - Import embed HTML function library for python output.

Conclusion :

From the above, Graphs and Outputs we can conclude that our model works successfully as the agent reaches the goal when trained with the optimum set of hyper-parameters. The agent trains and learns the shortest path to the goal using the memorized paths and the current saved paths and reaches the goal. Also varying and tuning different parameters showed how the performance and be increased or decreased for the model, also DQN works successfully as it integrates both the deep learning and reinforcement learning to yield optimum performance. Also it was observed that the decreasing value of epsilon gradually resulted in slow learning compared to other high values where there an exponential decrease in the graphs.

References :

https://skymind.ai/wiki/deep-reinforcement-learning