CSE 574 PROJECT 4

Vishnu Varshath Harishankar
Person Number 50291399
ubit vharisha

1 Writing Tasks

1.1 Question 1

If the agent chooses the action that maximizes the Q value then it is looking for short term rewards rather than long term rewards which means it gives more importance to rewards that are obtained immediately rather than waiting for potential future rewards(which can be even bigger). We need to have a good exploration/exploitation trade-off (ie) tell the agent to explore more in the beginning and then when the agent is confident of the Q values then we can do exploitation.

Start Position			
Small Reward			
			Bigger Reward

In the above example if the agent decides to exploit rather than explore it will end up getting the smaller reward while if it explores then it can get the Bigger Reward. The Agent gets stuck performing non optimal actions.

There are two ways to make the agent explore:

We must initialize high Q values to encourage exploration. If the Q values are initialized to high values then the unexplored values(which look good) tempts the greedy search to explore more

To explore more, the agent has to pick random actions occasionally. In our code we pick random actions based on the value of epsilon. So we must make sure that epsilon value is high enough to pick random actions. So if we decrease the epsilon decay, epsilon stays high for some amount of time when the exploration happens.

1.2 Calculate Q value

Q Table after first iteration:

State	Up	Down	Left	Right
0	3.9008	?	3.9008	3.9403
1	2.940	2.9701	?	?
2	?	?	?	1.99
3	?	1	?	0.99
4	0	0	0	0

Calculations:

$Q(S_4, a) = 0$
Since this is the goal the agent has to reach

$Q(S_3, Down) = 1$
This is due to the fact that we can reach our terminal state S_4

$Q(S_3, Right) = 0 + 0.99\ max(Q(S_3, a))$ = 0 + 0.99(1) = 0.99

$Q(S_2, Right) = 1 + 0.99\ max(Q(S_3, a)) = 1 + 0.99(1) = 1.99$

$Q(S_1, Down) = 1 + 0.99\ max(Q(S_2, a)) = 1 + 0.99(1.99) = 1 + 1.97 = 2.9701$

$Q(S_1, Up) = 0 + 0.99\ max(Q(S_1, a)) = 2.940$

$Q(S_0, Right) = 1 + 0.99\ max(Q(S_1, a)) = 1 + 0.99(2.9701) = 1 + 2.9403 = 3.9403$

$Q(S_0, Left) = 0 + 0.99\ max(Q(S_0, a)) = 0 + 0.99(3.9403) = 3.9008$

$Q(S_0, Up) = 0 + 0.99\ max(Q(S_0, a)) = 0 + 0.99(3.9403) = 3.9008$

Q Table after Second Iteration:

State	Up	Down	Left	Right
0	3.9008	3.9403	3.9008	3.9403
1	2.940	2.9701	2.9008	2.9701
2	1.9403	1.99	1.9403	1.99
3	0.9701	1	0.9701	0.99
4	0	0	0	0

Calculations:

$Q(S_3, Left) = -1 + 0.99\ max(Q(S_2, a)) = -1 + 0.99(1.99) = -1 + 1.97 = 0.9701$

$Q(S_3, Up)$ is symmetric to $Q(S_3, Left)$ thus its equal to 0.9701

$Q(S_2, Down)$ is symmetric to $Q(S_2, Right)$ thus its equal to 1.99

$Q(S_2, Up) = -1 + 0.99\ max(Q(S_1, a)) = -1 + 0.99(2.9701) = -1 + 2.94 = 1.9403$

$Q(S_2, Left)$ is symmetric to $Q(S_2, Up)$ thus its equal to 1.9403

$Q(S_1, Left) = -1 + 0.99\ max(Q(S_0, a)) = -1 + 0.99(3.9403) = -1 + 3.90 = 2.9008$

$Q(S_1, Right)$ is symmetric to $Q(S_1, Down)$ thus its equal to 2.9701

$Q(S_0, Down)$ is symmetric to $Q(S_0, Right)$thus its equal to 3.9403

2 Project Report

2.1 Introduction

The project aims at Tom finding the shortest path to Jerry with the underlying assumption that both the positions of Tom and Jerry are deterministic. To solve this problem we use DQN which is also called Deep Q Network, a Deep Reinforcement Learning algorithm. Reinforcement Learning is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. The motivation behind this approach is to maximize the return from a starting state.

2.2 Code Snippets Implemented

1. Neural Network Implementation

```
def _createModel(self):
  # Creates a Sequential Keras model
  # This acts as the Deep Q-Network (DQN)

  model = Sequential()

  ### 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'))
```

The Neural Network forms the basis of our reinforcement learning algorithm. It is implemented using Keras Framework. We have a three layer network with the first layer having dimensions equal to the number of states which is nothing but Tom's and Jerry's location. The model will train and predict using the rewards it obtains from choosing any one of the four actions. The output layer is equal to number of actions because output is a vector of Q-values for each action possible in the given state.(the model gives us the reward for each action taken)

2. Implementation of exponential-decay for Epsilon

```
def observe(self, sample):  # in (s, a, r, s_) format
  """The agent observes an event.
  We pass a sample (state, action, reward, next state) to be stored in memory.
  We then increment the step count and adjust epsilon accordingly.
  """
  self.memory.add(sample)

  # slowly decrease Epsilon based on our eperience
  self.steps += 1

  ### START CODE HERE ### (≈ 1 line of code)
  self.epsilon=self.min_epsilon +(self.max_epsilon-self.min_epsilon) * math.exp(-self.lamb*self.steps)
  ### END CODE HERE ###

  self.epsilons.append(self.epsilon)
```

The above code snippet is just an implementation of exponential decay for epsilon.
We specify an exploration rate **Epsilon** which we set to 1 in the beginning. This is the rate of steps that the agent will do randomly. In the beginning, this rate must be at its highest value, because the agent does not know anything about the values in Q-table. This means the agent needs to do a lot of exploration, by randomly choosing its actions.

Exploitation is about using what you know, whereas Exploration is about gathering more data/information so that you can learn

We generate a random number. If this number is greater than epsilon, then we will do exploitation (this means we use what we already know to select the best action at each step). Else, we'll do exploration. As the agent progresses it starts learning and it becomes more confident of the Q values and hence the agent needs to start exploiting than exploring. This is where epsilon decay comes into picture. When the epsilon value

decreases with increase in time, the agent is confident of the Q values it can start to exploit.

3. Implementation of Q function

```
for i in range(batch_size):
  # Observation
  obs = batch[i]

  # State, Action, Reward, Next State
  st = obs[0]; act = obs[1]; rew = obs[2]; st_next = obs[3]

  # Estimated Q-Values for Observation
  t = q_vals[i]

  ### START CODE HERE ### (≈ 4 line of code)
  if np.array_equal(states_next[i],no_state):
    t[act]=rew
  else:
    t[act]=rew + self.gamma*max(q_vals_next[i])



  ### END CODE HERE ###

  # Set training data
  x[i] = st
  y[i] = t
```

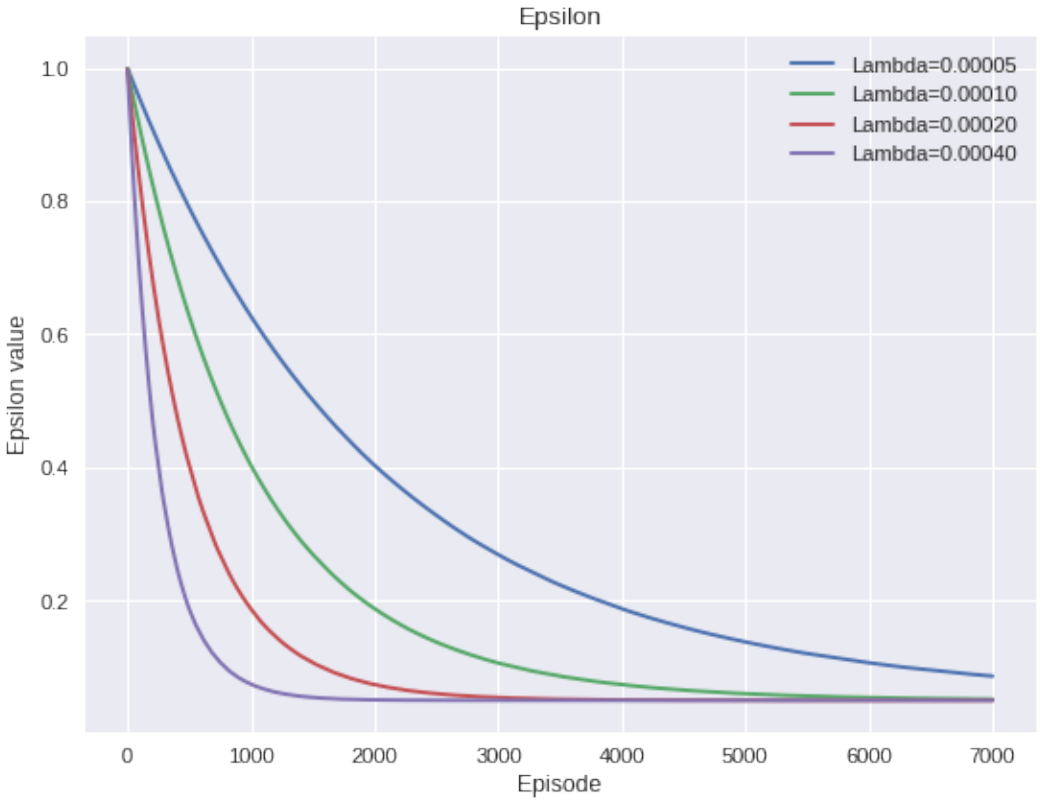
Q function is a function of state and action which provides us with a real number. Given that the agent is in state S and takes an action a, the Q function provides us with the expected total rewards received by the agent. Q function is a good indication of how good it is for an agent at state S to take an action a.

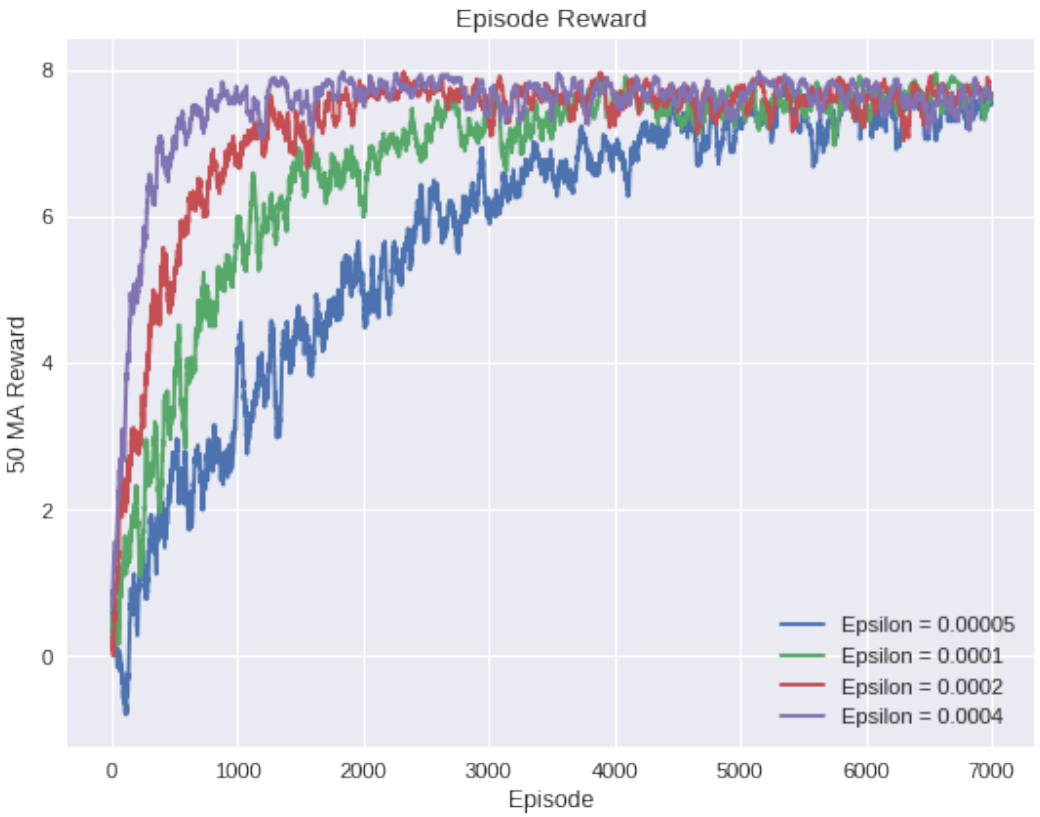
The implementation has two conditions.
1. If the episode terminates at $Step_{t+1}$ then the rewards is equal to the rewards at $Step_t$ because the agent cannot move further.
2. If the episode does **not** terminate at $Step_{t+1}$ then the expected reward is equal to reward at $Step_t$ plus the discounted expected future reward after transition to a next state $State_{t+1}$ provided by the Q-Function.

The Discounting Factor discount is a hyper-parameter that controls the importance of future rewards. The reason for using discount factor is to prevent the total reward from going to infinity. It also models the agent behavior when the agent prefers immediate rewards than rewards that are potentially received far away in the future when it set to a value close to zero. If set to a value close to 1 then the agent prioritizes the rewards in the distant future.

2.3 Hyper Parameter Tuning

1. Changing Hyper Parameter - **LAMBDA** As mentioned earlier in the report influences the Epsilon decay(exploration factor) which in turn influences the amount of random actions the agent chooses for exploration.

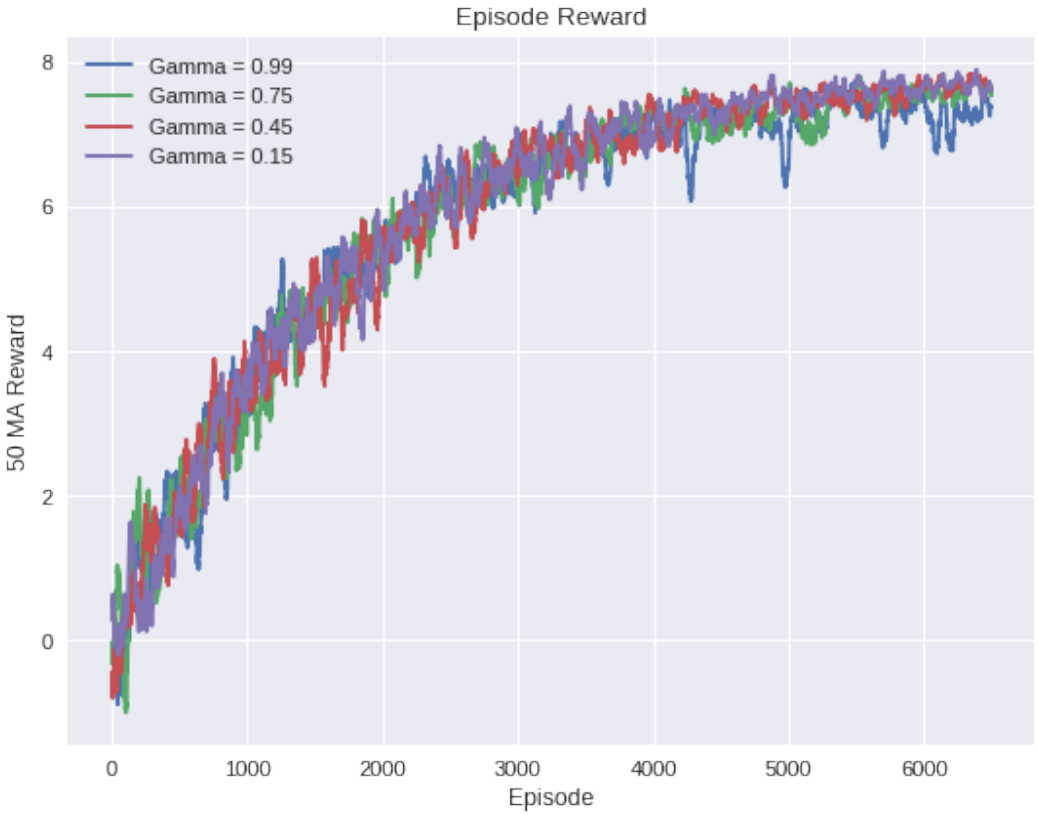




The Epsilon value mentioned in the graph is the Lambda value. The graph shows the variation of rewards with Lambda values of 0.00005, 0.00010, 0.00020, 0.00040.

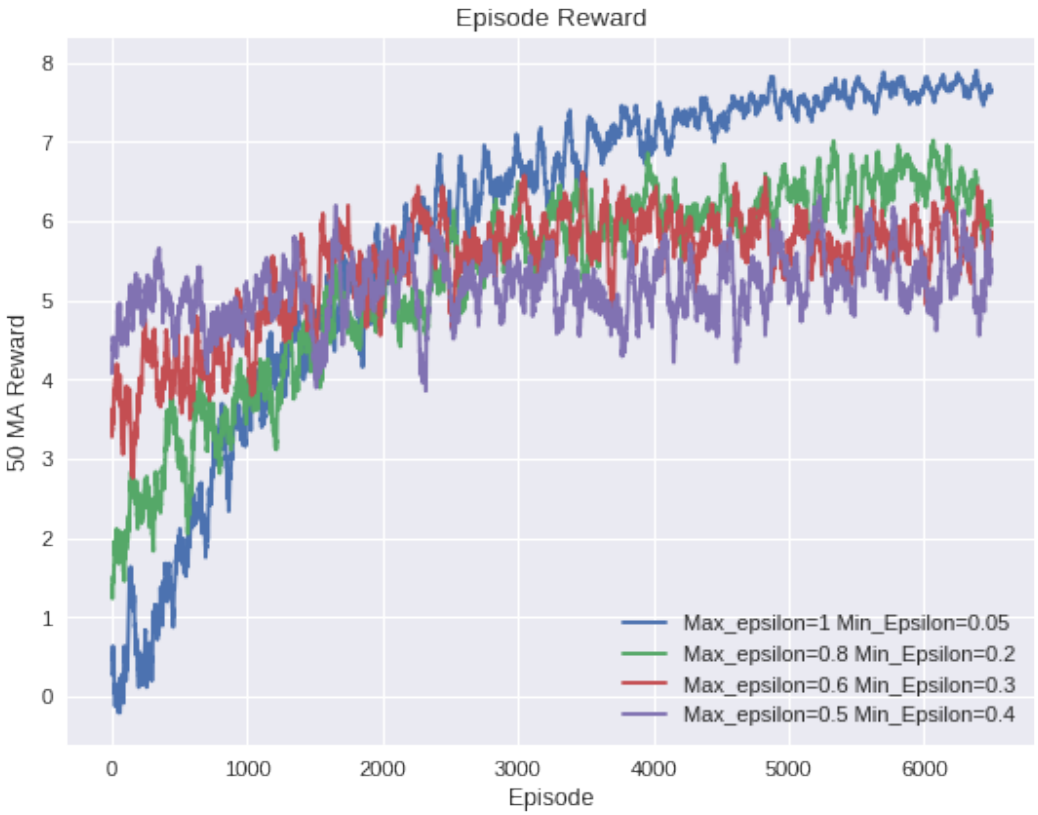
It can be seen from the graph that with higher values of Lambda which influences the epsilon decay, the fluctuations is more as it is exploiting than exploring. In contrast the curve is smoother for smaller epsilon decay values because the agent explores the environment more and it is able to learn in a stable manner. For a bigger and complex environment we need a good trade-off between exploration and exploitation.

2. Changing Hyper Parameter - **Gamma - Discounting Factor**



The graph shows the variation of Gamma vs rewards. Here in this problem there is not much of a variation but in complex and large problems gamma plays an important role because it controls the level of importance given to future rewards.

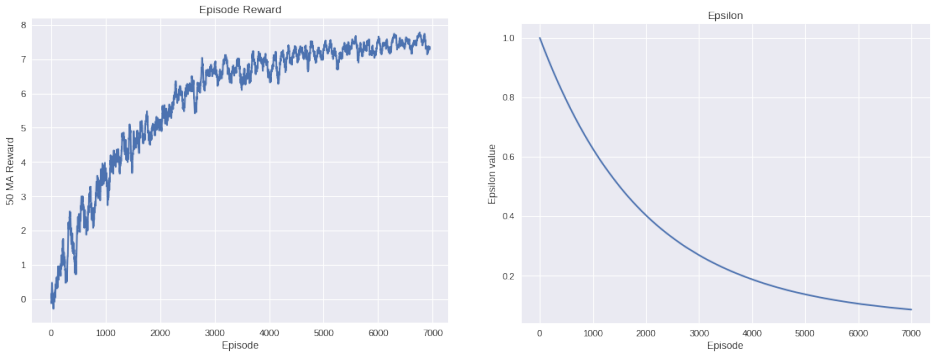
3. Changing Hyper Parameter - **Changing Maximum and Minimum Epsilon Values**



We see from the above image that there are more fluctuations when the difference between Minimum Epsilon and Maximum Epsilon is less and around 0.5 This is because Agent does not have a clear idea as to explore or exploit. We see the curve is smoother for Max Epsilon=1 and Min Epsilon=0.05 because there was a gradual epsilon decay and the Agent was able to explore in the beginning and exploit in the end.

2.4 Optimal Hyper-Parameters

The Optimal Hyper-paramters were: Lambda : 0.00005
Minimum Epsilon = 0.05
Maximum Epsilon = 0.99
Gamma(Discounting Factor)=0.99
Number of Episodes=7000



2.5 Observations and Inferences

1. I implemented the Neural Network part, the Epsilon Decay and Q Function.
2. The roles played by my snippet implementations are given below:
   Neural Network: The Neural Network forms the brain of the model which is used to train the agent by observations. It takes in the state and the provides the Q Values.
   Epsilon Decay: Epsilon Decay is important because it controls the nature of Agent (ie) to explore more or exploit more. Ideally we need a good trade-off because if exploitation is more the Agent keeps performing non-optimal activities and if exploration is more , there is a chance that the optimal activity is overlooked.
   Q-Function: Q function is important because given that the agent is in state S and takes an action a, the Q function provides us with the expected total rewards received by the agent. Q function is a good indication of how good it is for an agent at state S to take an action a.
3. There are some aspects of the code that can be better implemented. For Example we are considering only the next state. We can take into account the states in t+2,t+3... steps just as in Bellman Equation.
4. Just as I mentioned above in my optimal hyper-parameter section, the model learns the best at around 6500-7000 episodes. With a good discounting factor of 0.99 and a good lambda value of 0.00005 the model performs the best.