CSE574 Assignment 4 Report

Tom and Jerry in Reinforcement Learning

Tenzin Norden
UB person# 50096989
December 05, 2018

1 Project Description

This project combines reinforcement learning and deep learning. We teach the agent to navigate in the grid world environment. In this model, Tom, a cat, will be navigating the grid to catch jerry, a mouse. The task here is for the Agent (Tom) to find the shortest path to the goal (Jerry) given that the initial positions of Tom and Jerry are deterministic.

The problem is solved using deep reinforcement learning algorithm-DQN (Deep Q-Network). The Model comprises of: an environment which is the grid world, Brain of the Agent where the model is created and held and Memory which stores the experiences of the agent.

Report on the coding task is written in section I and report on Writing Tasks is in section II.

Section I: Report on Coding Task

2 Model: Reinforcement Learning

Reinforcement Learning refers to goal-oriented algorithms, which learns how to attain an objective (goal). Here machines are given few inputs and actions, and then, reward them based on the output. The maximization of reward will be the end goal.

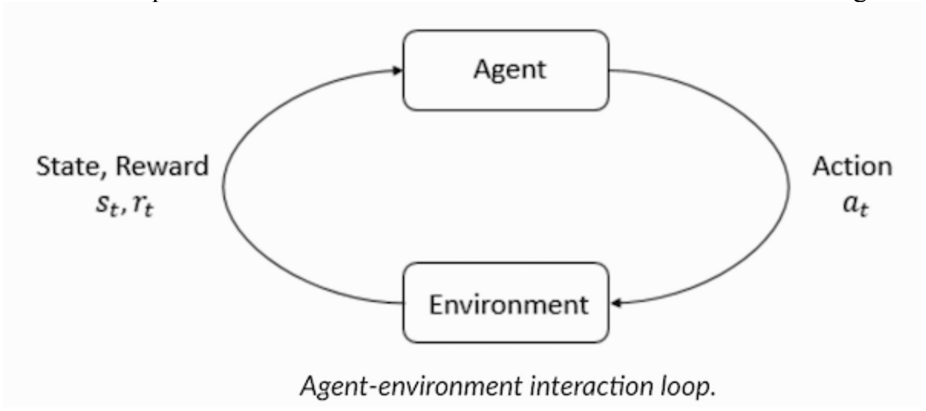


Figure 1: Schematic of reinforcement learning

a. Environment: Rewards and policies are based on the environment.
b. The Agent here is Tom (cat).
c. States: The state can be a position, a constant or a dynamic. States are grids in our environment in this project.
d. Action: Action are set of decisions an agent takes and it is usually based on the

environment. Different environments lead to different actions based on the agent. The actions are up, down, left and right.

e. Reward: Reinforcement Learning learns based off reward and it is tracked all the time. It plays vital role in tuning, optimizing the algorithm and stop training the algorithm. It depends on the present state, action and the future states.

f. Policies: policy is the rule used by an agent for choosing the next action, also called as agents brains.

Environment handles state and reward, and agent which decides which action to take given a particular state. An environment starts with some initial state S_0, which is passed to the agent. The agent passes an action a_o, based on the state S_o, back to the environment. The environment then reacts back to the agent, then passes the next state S_1, along with the resulting reward for taking the action, r_o, back to the agent. This process continues until the terminal state or the end of the episode is reached.

In this project, we used Neural Network for approximate mapping from state to action which attempts to maximize discounted accumulative reward.

3. Coding Task

3.1 Task 1: 3-layer Neural Network using Keras library

The brain of the agent is created using 3-layer neural network. It takes four inputs neurons and gives four outputs neurons. We have two hidden layers and the activation function implemented is ReLU. In the output layer, the Linear activation function is used as shown in figure 2.

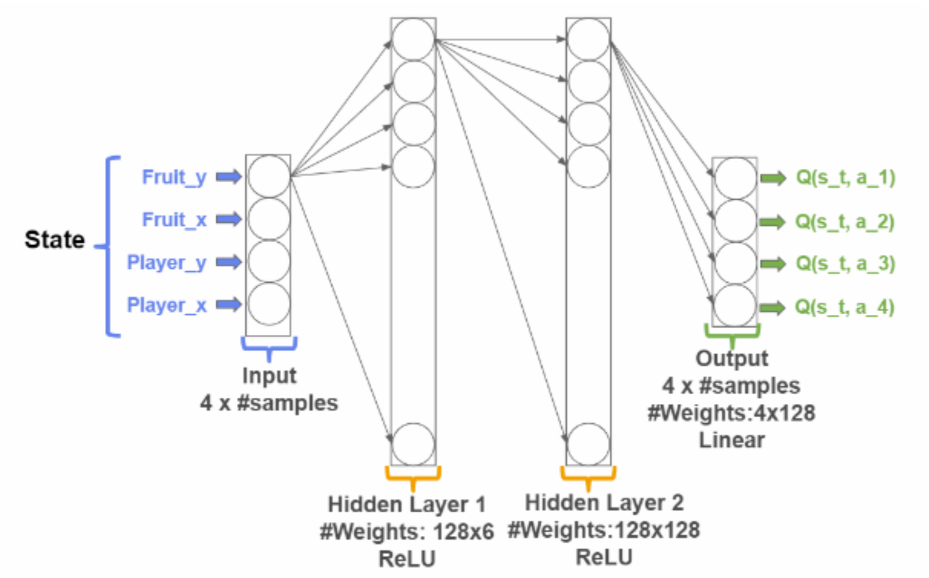
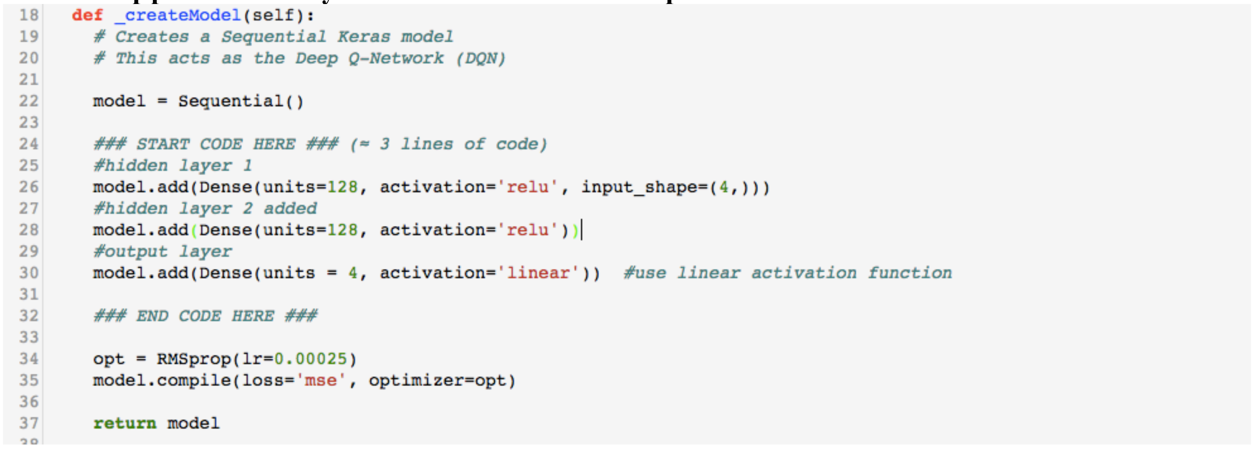


Figure 2. Schematic of two hidden layer neural network

Code snippet for 3-layer 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)
  #hidden layer 1
  model.add(Dense(units=128, activation='relu', input_shape=(4,)))
  #hidden layer 2 added
  model.add(Dense(units=128, activation='relu'))
  #output layer
  model.add(Dense(units = 4, activation='linear'))  #use linear activation function

  ### END CODE HERE ###

  opt = RMSprop(lr=0.00025)
  model.compile(loss='mse', optimizer=opt)

  return model
```

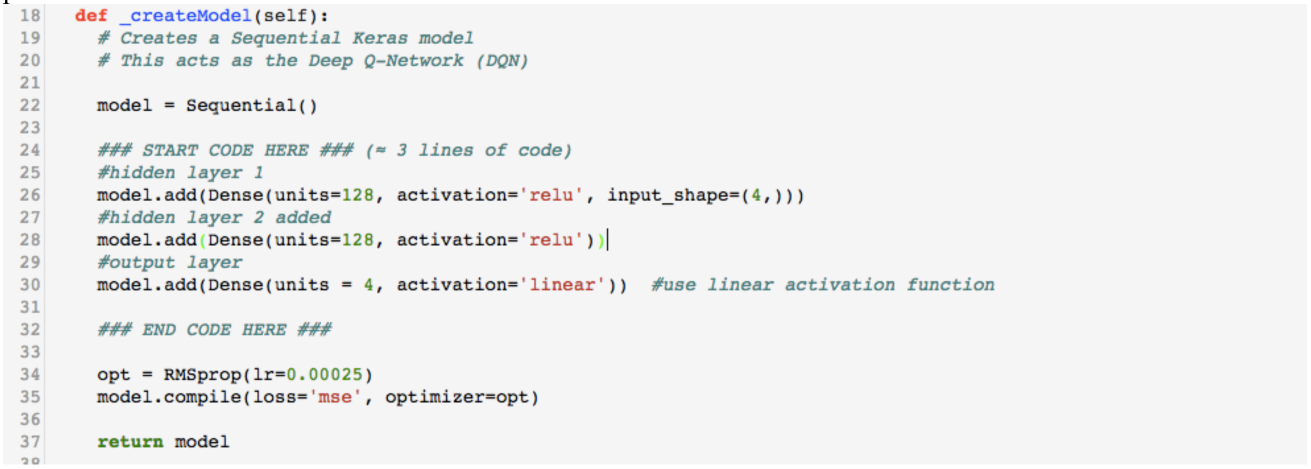
Number of hidden neurons for hidden layers is 128 and output is 4 which is equal to the number of actions.

3.2 Task 2: Epsilon Implementation

To force the agent to randomly explore the environment we use 'epsilon' or also called as 'exploration rate'. Epsilon is the percentage of which the agent will randomly select its action. At first, the agent tries all kinds of things before its starts to see the patterns. As the goes, we want to decrease the number of random action of our agent, so the exponential decay epsilon is introduced. When the agent is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|}$$

where ϵ_{min}, $\epsilon_{max} \in [0,1]$, λ is hyper-parameter for epsilon and $|S|$ is the total number of steps. The code for epsilon is shown in the code snippet. ϵ_{max} and ϵ_{min} are tunable hyper-parameters.



```
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)
  #hidden layer 1
  model.add(Dense(units=128, activation='relu', input_shape=(4,)))
  #hidden layer 2 added
  model.add(Dense(units=128, activation='relu'))
  #output layer
  model.add(Dense(units = 4, activation='linear'))  #use linear activation function

  ### END CODE HERE ###

  opt = RMSprop(lr=0.00025)
  model.compile(loss='mse', optimizer=opt)

  return model
```

3.3 Task 3: Q-function implementation: Deep Q-learning

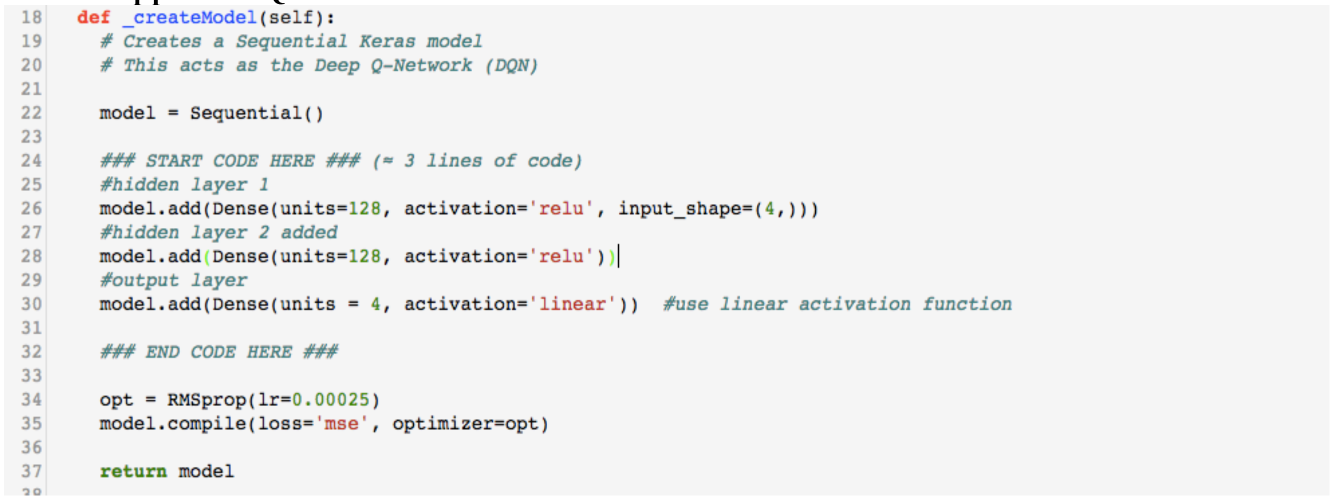
In reinforcement learning, the Q-learning algorithm uses the environment rewards to learn over time, the best action to take in a given state. The training of the agent is done by updating the q-values. Using the reward table, it chooses the next action based on whether its beneficial or not and then update a new value called Q-value. The new table created is called a Q-Table and they map to a combination called (state, Action) combination. The Q-function is given as follows

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

Where r_t is the reward and γ is the discount factor which controls the importance of the future rewards.

Q-value is equal to the reward if the episode is terminated at for the future step t+1. Otherwise, Q-value is the current reward plus the maximum Q-value possible of the future state and action, with the importance of the future reward controlled by the gamma function.

Code snippet for Q-function:



```
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)
  #hidden layer 1
  model.add(Dense(units=128, activation='relu', input_shape=(4,)))
  #hidden layer 2 added
  model.add(Dense(units=128, activation='relu'))
  #output layer
  model.add(Dense(units = 4, activation='linear'))  #use linear activation function

  ### END CODE HERE ###

  opt = RMSprop(lr=0.00025)
  model.compile(loss='mse', optimizer=opt)

  return model
```

4 Experiment

4.1 Gamma

Gamma is the discount factor which models the fact that future rewards is worth less than the immediate reward. Gamma needs to be less than 1 for the model to converge. In the following experiment, I tested reward output for three different gamma values: 0.1, 0.5 and 1.1. epsilon maximum and minimum values were kept constant at 1 and 0.05.

As seen in figure 3b, gamma = 0.1 has the highest mean reward of 8 after 10,000 episodes of training. Figure 3d plots gamma = 0.5, and also shows a mean reward of about 8. However, in figure 3f for gamma = 1.1, the reward goes to a negative number. Which tells that gamma value has to be less than 1 for the model to converge.

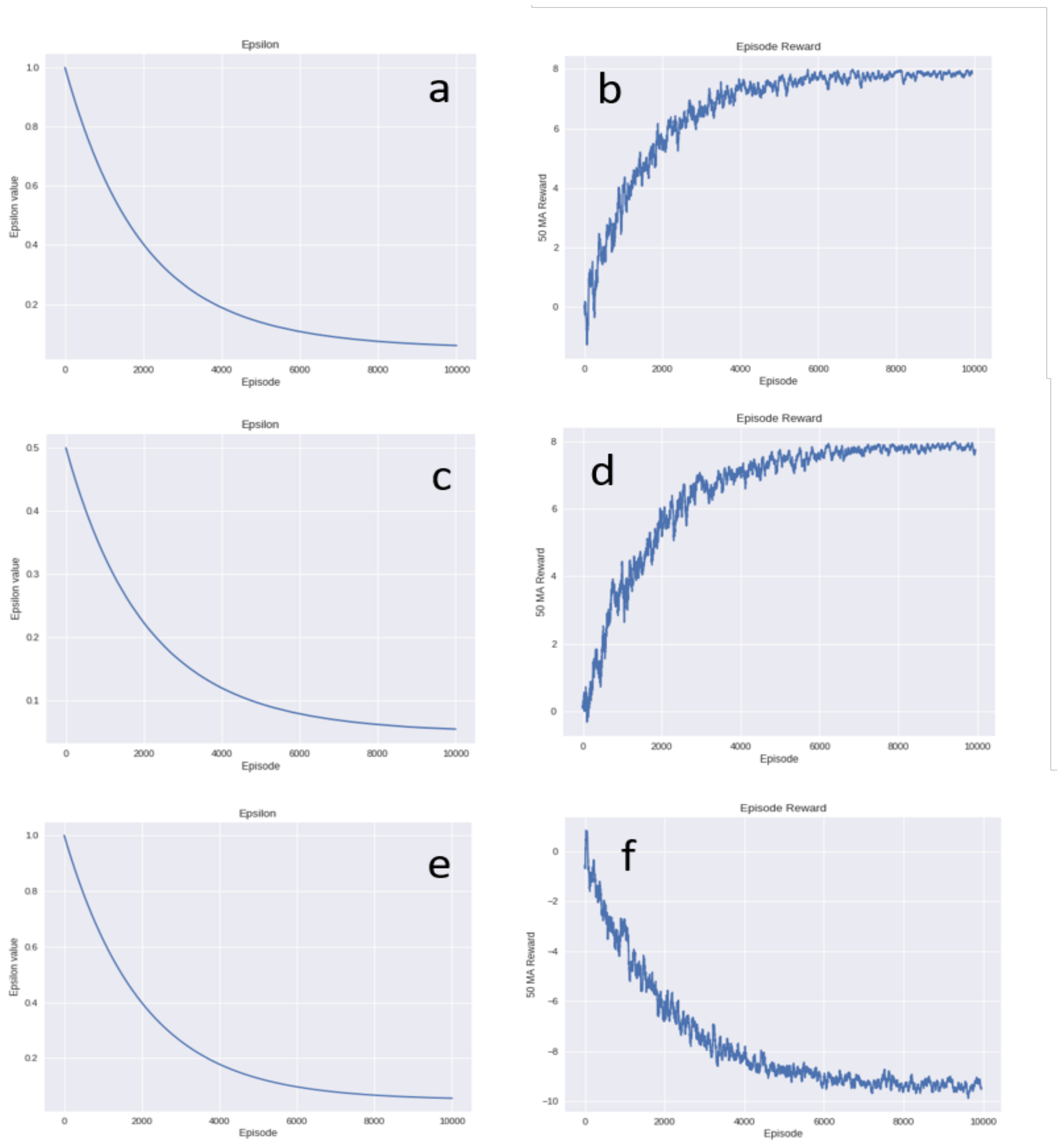


Figure 3. a), c) and e) : Epsilon vs Episode for gamma 0.1, 0.5 and 1.1 respectively. b), d) and f) mean reward vs Episode gamma 0.1, 0.5 and 1.1 respectively

4.2 Hyper-parameter: Epsilon Minimum

Here I tuned epsilon minimum values as shown in the figure 4 below. In figure 4a,4c and 4e, epsilon values vs episode for 0.5, 0.3 and 0.1 epsilon minimum values respectively are shown. The mean reward as a function of episode is shown in figure 4b, 4d, and 4f for epsilon minimum values 0.5, 0.3 and 0.1. The epsilon maximum values were kept constant at 1. For all the values, the model converges and the reward saturates after about 6000 episodes.

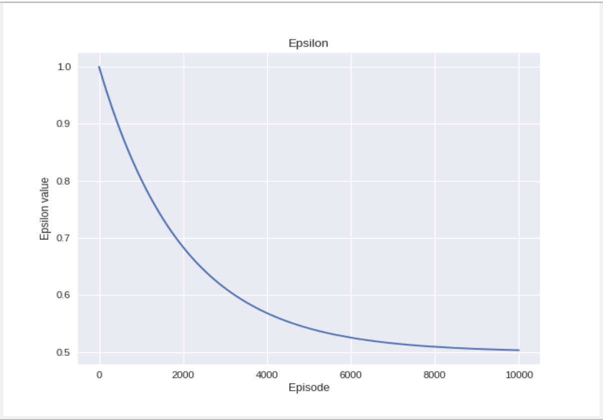
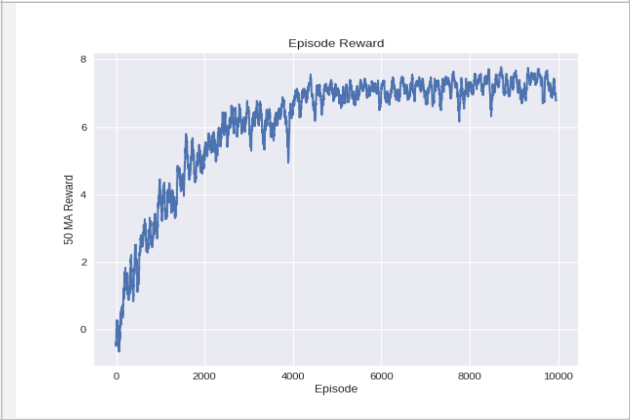
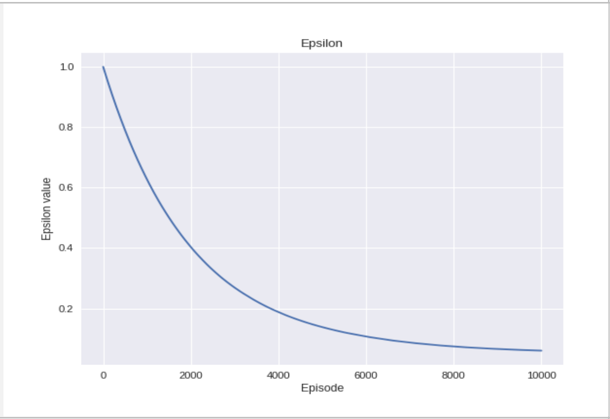
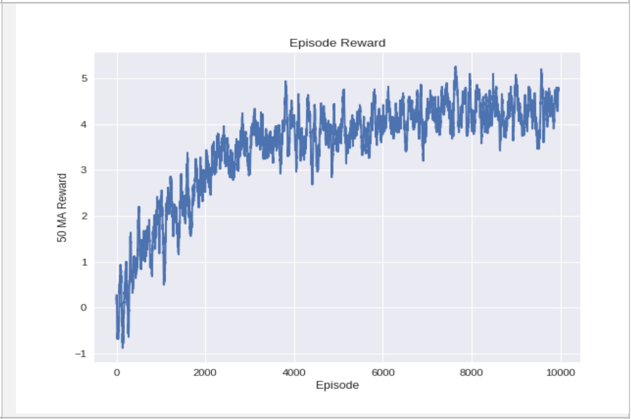
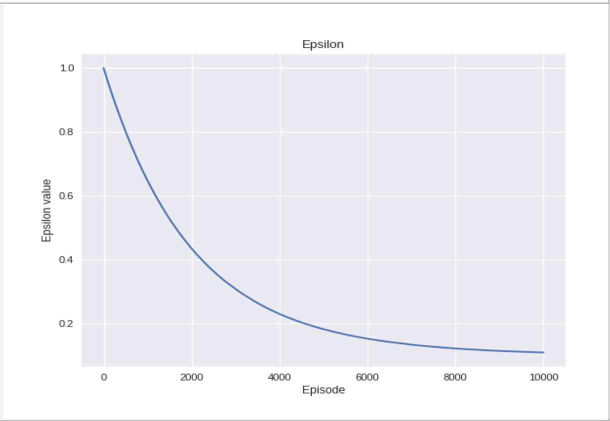
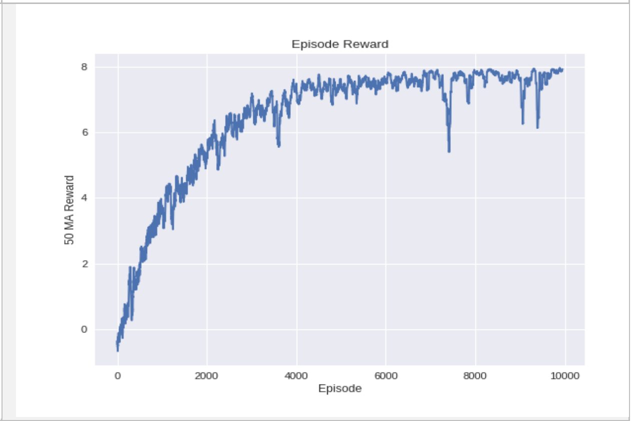
a	b
	
c	**d**
	
e	**f**
	

Figure 4. a), c) and e) : Epsilon vs Episode and b), d) and f): mean reward vs Episode for epsilon minimum 0.5,0.3 and 0.1 respectively.

4.3 Hyper-parameter: Epsilon Maximum

Here I tune epsilon maximum. Figures 5b, 5d and 5f show the mean reward as a function of episodes for epsilon maximum values 0.7, 5 and 10. For epsilon maximum value below 1, 0.7 as shown in figure 5b, the reward saturates to about 7.5 and the model starts learning from the very beginning. However, for epsilon maximum values larger than 1, I observe that the model does not learn up until few thousand episodes has been passed as seen in figures 5d and 5f. This is because for epsilon maximum values larger than 1, we are forcing the agent to follow the policy and to choose the action that maximizes the Q-value. The agent will not explore the environment.

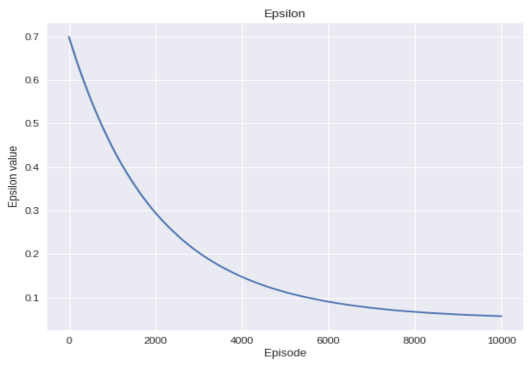
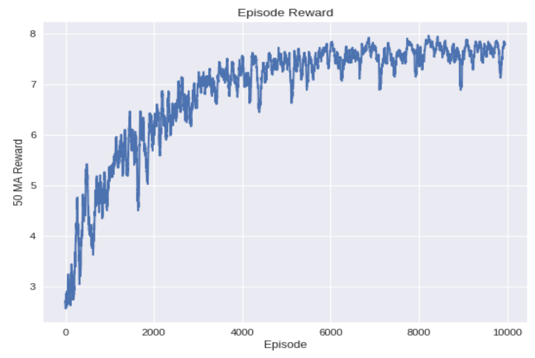
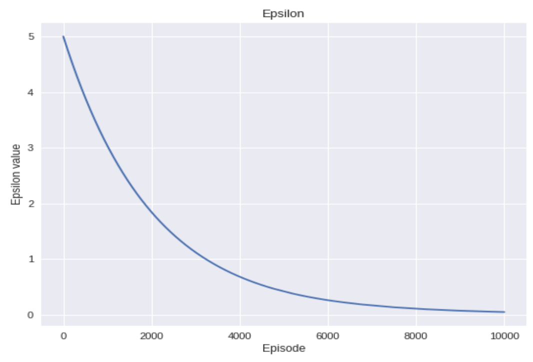
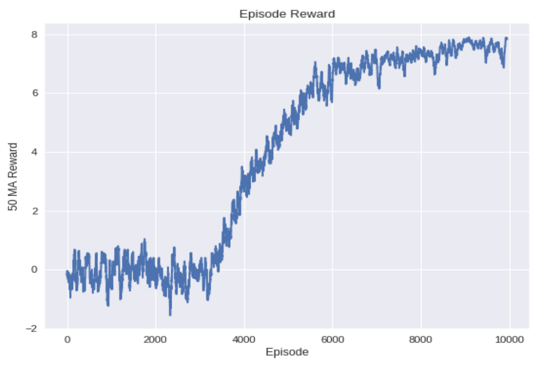
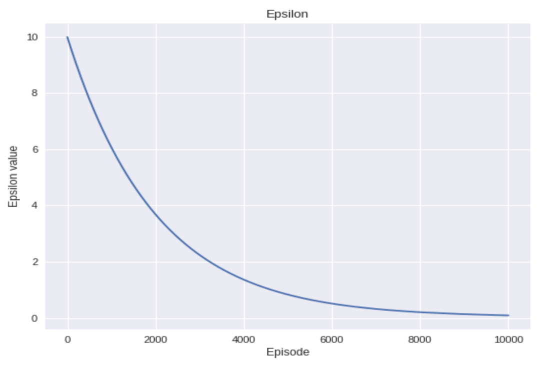
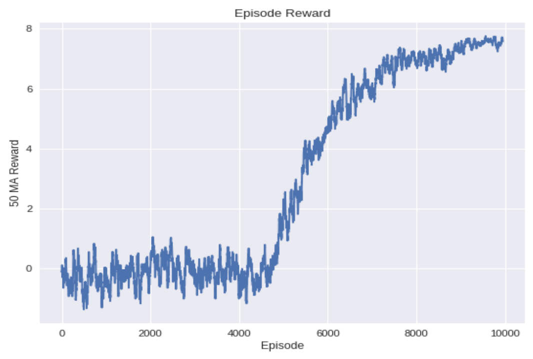
a	**b**
	
c	**d**
	
e	**f**
	

Figure 5. a), c) and e) : Epsilon vs Episode and b), d) and f): mean reward vs Episode for epsilon maximum 0.7, 5 and 10 respectively.

4.3 Hyper-parameter: number of episodes

Here I experimented with the number of episodes and see how fast the agent learns the best path to the goal which provides a highest reward. As can be seen in figure 6b, for 1000 episodes of learning, the agent has not learned yet and the mean reward has not saturated. In figure 6d, the reward almost saturated for 5000 number of episodes. Finally, in figure 6f, we can see that the mean reward has stabilized and saturated for 10,000 episodes. We can understand that 10,000 episodes is enough for the agent to learn the best solution.

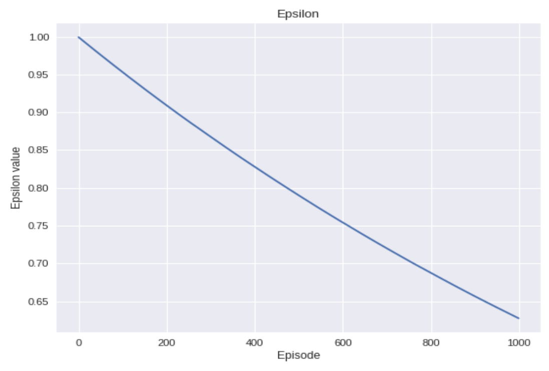
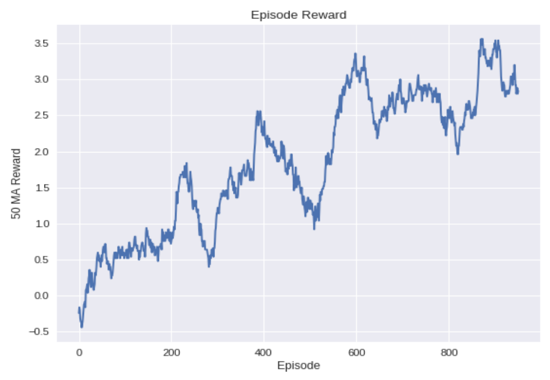
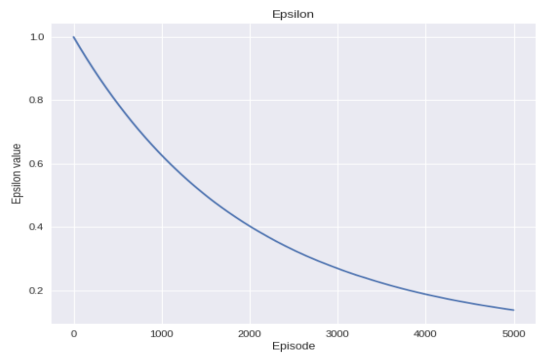
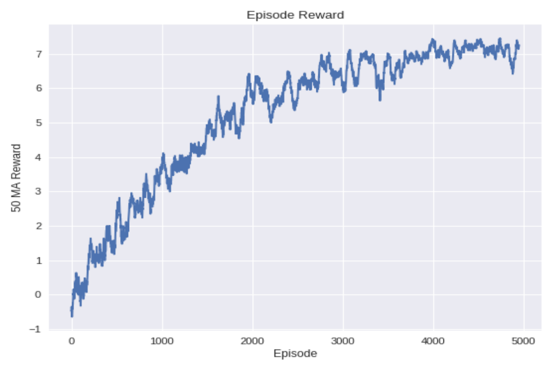
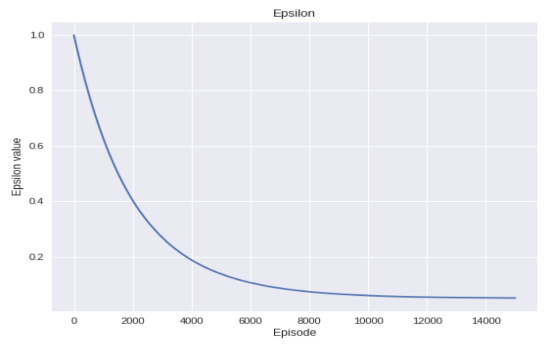
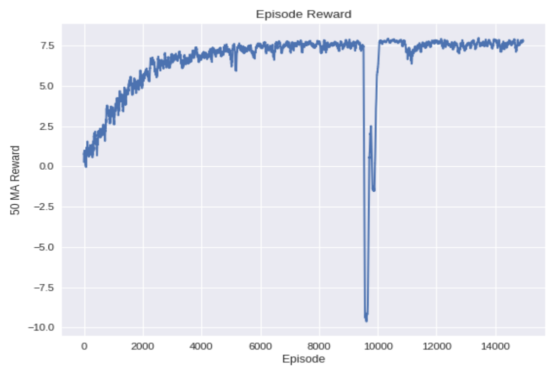
a	**b**
	
c	**d**
	
e	**f**
	

Figure 6. a), c) and e) : Epsilon vs Episode and b), d) and f): mean reward vs Episode for episodes 1000, 5000 and 10000 respectively.

5 Conclusion

A reinforcement learning and Deep-Q learning is described and implemented. I have performed experiment by tuning hyper-parameters to understand their roles and also to obtain higher optimal mean rewards. The agent is allowed to explore and at the same time exploit the environment by using epsilon greedy method. Q-learning is used to train the agent and find the optimal Q-values for the optimal path to the goal. Neural network is used to map state to action paths used as the agent's brain.
Finally, I got highest mean reward of about 8.

Section II: Writing Tasks

1 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.

If the agent always chooses the action that maximizes the Q-value rather than exploring the environment is exploitation. Too much exploitation can lead the agent to being stuck in a local optimum. It is always a good idea to force the agent to explore in a controlled fashion because exploration may waste time exploring irrelevant part of the environment. Therefore, exploitation and exploration must happen simultaneously.

Ways to force agent to explore:

a. **Epsilon-greedy**

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|}$$

Epsilon or exploration rate ($0<\epsilon < 1$) decreasing strategy forces the agent to explore as epsilon decreases exponentially as the agent learns. It is an **undirected exploration**. In this method, the agent will choose whether or not to follow the policy when it chooses its actions or it randomly choose action based on the epsilon. The agent will randomly select it action at first so that it starts to see the patterns. The number of random action of the agent decreases as the epsilon decays with learning. For example, if the epsilon is 0.2 at a time, then the agent has a 0.2 probability of randomly selecting an action and 0.8 probability of choosing the action that the brain says is optimal. The advantage of this strategy is that it is not dependent on specific data. One drawback of this method is that exploration action is selected uniform randomly from the set of possible actions. Therefore, it is as likely to choose the worst appearing action as it is to choose the best appearing action if an exploration action is selected.

b. **Boltzmann (or softmax) exploration**

It uses the Boltzmann distribution function to assign a probability ($\pi(s_t, a)$) to the actions in order to create a graded function of estimated value:

$$\pi(s_t, a) = \frac{e^{Q_t(s_t,a)/T}}{\sum_{i=1}^{m} e^{Q_t(s_t,a)/T}}$$

$\pi(s_t, a)$ denotes the probability the agent selects action a in state s_t and $T \geq 0$ is the temperature parameter used in Boltzmann distribution. When T=0, the agent does not explore at all and when $T \rightarrow \infty$ the agent selects random actions. The agent most likely selects the best action by using intermediate values for T, however, other actions are ranked instead of randomly chosen.
[http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/Exploration_QLearning.pdf]

c. Counter Based exploration:

Another strategy to force the agent to explore is counter-based exploration which is one of the **directed exploration methods.** Counter-based exploration keeps count of how many times a state-action pair has been visited, and evaluates actions based on a linear combination of an exploitation term and an exploration term.

2. Q-table calculation for 3x3 grid

Here, we consider a deterministic environment of 3x3 grid as shown in the figure below. The 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 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 goal
0 : when it does not move at all (or when it move into an edge)

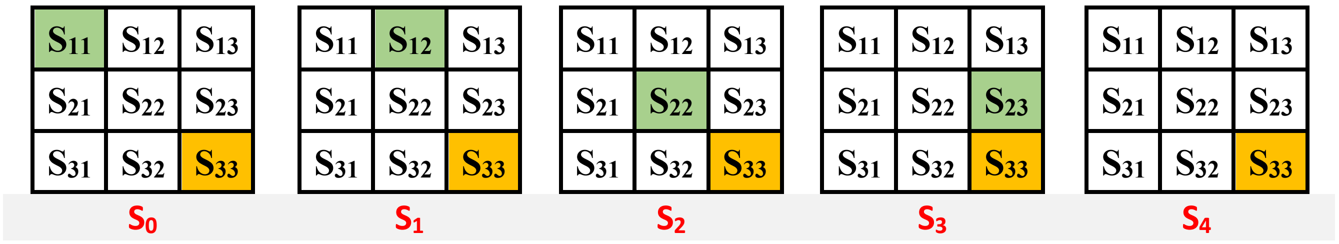


Figure 7: One of the possible optimal actions sequence

The Q-values for the states given in figure 1 (green and yellow grids) are calculated and their Q-table is constructed as shown in table 1.

Q-values are calculated from this state. The Q values are solved with Q-function given as

$$\boldsymbol{Q(s_t, a_t) = reward + \gamma * \max[Q_a(s_{t+1}a_{t+1})]}$$

where $\gamma = 0.99$ and the value of reward is based on the action of the agent. Q-value is a function of state and action.

We first initialize the goal, state S$_4$ or S$_{33}$ to zeros for all actions, where

$$\boldsymbol{Q_{a'}(s_{33}, goal) = 0}$$

Detailed calculation description for state s_{23} for action: up.

Start from present state s_{23} or s_3:
Action 1: up and reach s_{13}, reward = -1 (goes away from the target).
$Q(s_{23}, up) = -1 + 0.99 * \max\,[Q_{a'}(s_{13}, down)]$

To find max $[Q_{a'}(s_{13}, a')]$, we chose the optimal path to the goal. (Down and down)
Action 2: down and reach s_{23} and find max $[Q_{a'}(s_{23}, down)]$
$Q(s_{23}, up) = -1 + 0.99\{1 + 0.99 * \max\,[Q_{a'}(s_{23}, down)]\}$
Action 3: down and reach goal: s_{33} ; $Q_{a'}(s_{33}, goal) = 0$

$Q(s_{23}, up) = -1 + 0.99 * \{1 + 0.99 * [1 + 0.99 * \max(Q_{a'}(s_{33}, goal))\ ]\ \}$
$Q(s_{23}, up) = -1 + 0.99 * \{1 + 0.99 * [1 + 0.99 * (0)\ ]\ \}$
$Q(s_{23}, up) = 0.97$

Calculation for the rest the states and actions are shown.

Q-values for state s_{23}: actions down, left and right
Action: start from s_{23}, down (goal: s_{33})
$Q(s_{23}, down) = 1 + 0.99 * \max [Q_{a'}(s_{33}, down) = 1 + \ 0.99(0) = \ 1$
Action: start from s_{23}, left (s_{22}), right (s_{23}), down (goal: s_{33})
$Q(s_{23}, left) = -1 + 0.99 * \max [Q_{a'}(s_{22}, down)] = \ 0.97$
Action: start from s_{23}, right (s_{23}), down (goal: s_{33})
$Q(s_{23}, right) = 0 + 0.99 * \max [Q_{a'}(s_{23}, down) = 0.99 * (1) = 0.99$

Q-values for State S2 or S22:
Action: start from s_{22}, up (s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{22}, up) = -1 + 0.99 * \max [Q_{a'}(s_{12}, a)]$
$Q(s_{22}, up) = -1 + 0.99 * [1 + 0.99(1 + 0.99)] = 1.94$
Action: start from s_{22}, down (s_{32}), right (goal: s_{33})
$Q(s_{22}, down) = 1 + 0.99 * \max [Q_{a'}(s_{32}, right)]$
$Q(s_{22}, down) = 1 + 0.99 * [1] = 1.99$
Action: start from s_{22}, left (s_{21}), right (s_{22}), right (s_{23}), down (goal: s_{33})
$Q(s_{22}, left) = -1 + 0.99 * \max [Q_{a'}(s_{21}, right)]$
$Q(s_{22}, left) = -1 + 0.99 * \left[1 + 0.99 * \left(1 + 0.99(1)\right)\right] = 1.94$
Action: start from s_{22}, right (s_{23}), down (goal: s_{33})
$Q(s_{22}, right) = Q(s_{22}, down) = 1.99$

Q-values for State S1 or S12:
Action: start from s_{12}; up (no move: s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{12}, up) = 0 + 0.99 * \max [Q_{a'}(s_{12}, right)]$
$Q(s_{12}, up) = 0 + 0.99 * [1 + 0.99(1 + 0.99)] = 2.94$
Action: start from s_{12}; down (s_{22}), down (s_{32}), right (goal: s_{33})
$Q(s_{12}, down) = 1 + 0.99 * \max [Q_{a'}(s_{22}, down)]$
$Q(s_{12}, down) = 1 + 0.99 * [1 + 0.99] = 2.97$
Action: start from s_{12}; left (s_{11}), right (s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{12}, left) = -1 + 0.99 * \max [Q_{a'}(s_{11}, right)]$
$Q(s_{12}, left) = -1 + 0.99 * \left[1 + 0.99 * \left(1 + 0.99 * (1 + 0.99)\right)\right] = 2.9$
Action: start from s_{12}, right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{12}, right) = 1 + 0.99 * [1 + 0.99] = 2.97$

Q-values for State S0 or S11:
Action: start from s_{11};up (no move: s_{11}), right (s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{11}, up) = 0 + 0.99 * \left[1 + 0.99 * \left(1 + 0.99 * (1 + 0.99)\right)\right] = 3.90$
Action: start from s_{11}; down (s_{21}), right (s_{22}), right (s_{23}), down (goal: s_{33})
$(s_{11}, down) = 1 + 0.99 * [1 + 0.99 * (1 + 0.99)] = 3.94$
Action: start from s_{11}; left (no move: s_{11}), right (s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{11}, left) = 0 + 0.99 * \left[1 + 0.99 * \left(1 + 0.99 * (1 + 0.99)\right)\right] = 3.90$
Action: start from s_{11}; right (s_{12}), right (s_{13}), down (s_{23}), down (goal: s_{33})
$Q(s_{11}, up) = 1 + 0.99 * [1 + 0.99 * (1 + 0.99)] = 3.94$

Following is the calculation for Q-values calculation for the states not in figure 1.

$Q(s_{13}, up) = 0 + 0.99 * [1 + 0.99] = 1.97$
$Q(s_{13}, down) = 1 + 0.99 * [1] = 1.99$
$Q(s_{13}, left) = -1 + 0.99 * [1.97] = 0.95$
$Q(s_{13}, right) = 0 + 0.99 * [1 + 0.99] = 1.97$

$Q(s_{21}, up) = -1 + 0.99 * \left[1 + 0.99 * \left(1 + 0.99 * (1 + 0.99)\right)\right] = 2.90$
$Q(s_{21}, down) = 1 + 0.99 * [1 + 0.99] = 2.97$
$Q(s_{21}, up) = 0 + 0.99 * [1 + 0.99 * (1 + 0.99)] = 2.94$
$Q(s_{21}, right) = 1 + 0.99 * [1 + 0.99] = 2.97$

$Q(s_{31}, up) = -1 + 0.99 * [1 + 0.99 * (1 + 0.99)] = 1.94$
$Q(s_{31}, down) = 0 + 0.99 * [1 + 0.99] = 1.97$
$Q(s_{31}, left) = 0 + 0.99 * [1 + 0.99] = 1.97$
$Q(s_{31}, up) = 1 + 0.99 * [1] = 1.99$

$Q(s_{32}, up) = -1 + 0.99 * [1 + 0.99] = 0.97$
$Q(s_{32}, down) = 0 + 0.99 = 0.99$
$Q(s_{32}, left) = -1 + 0.99 * [1 + 0.99] = 0.97$
$Q(s_{32}, right) = 1 + 0 = 1$

Table 1. All states and all actions

State	Up	Down	Left	Right
S_{11}	3.90	3.94	3.90	3.94
S_{12}	2.94	2.97	2.90	2.97
S_{13}	1.97	1.99	0.95	1.97
S_{21}	2.90	2.97	2.94	2.97
S_{22}	1.94	1.99	1.94	1.99
S_{23}	0.97	1	0.97	0.99
S_{31}	1.94	1.97	1.97	1.99
S_{32}	0.97	0.99	0.97	1
S_{33}	0	0	0	0

Table 2: Q-Table (for states in figure 1)

State	Up	Down	Left	Right
S_0 (S_{11})	3.90	3.94	3.90	3.94
S_1(S_{12})	2.94	2.97	2.90	2.97
S_2(S_{22})	1.94	1.99	1.94	1.99
S_3(S_{23})	0.97	1	0.97	0.99
S_4(S_{33})	0	0	0	0