Project 4 Reinforcement Learning and Deep Learning

Student Name: **Varsha Ganesh |** UB-ID: **50288433 |** UB-Email: **vganesh2@buffalo.edu**

1.Introduction

The main objective of our project is incorporate both Reinforcement Learning and Deep Learning in-order to teach our agent Tom (from Tom and Jerry Cartoon series) to navigate in a 05 X 05 grid- world to catch Jerry (his goal). In this project, we have been given two sets of Tasks. They are listed below:

1. The Coding Tasks
2. The Writing Tasks

2. Project Model

The project is packaged as four different classes, which are:

1. The Environment,
2. The agent,
3. The Memory, and
4. The Brain

These classes work together to help our agent reach his goal by incorporating Reinforcement and Deep -Q - Learning. Each of these classes are explained in detail below.

2.1 The Environment

We have created an environment which is almost similar to Open-AI Gym Environments. It has the following functionalities:

- **The __init__() Method**

  This method is called to initialize the environment and create 05 X 05 grid world environment for Tom and Jerry.

- **The _get_reward() Method**

  Whenever Tom takes an action, this method will be called to update his reward based on his action.

- **The _is_over() Method**

  This method returns true on two conditions, i.e, when a episode's time limit is exhausted or when episode is complete after Tom catching Jerry. If neither of these conditions are met, the method will return false.

- **The step() Method**

  The step() method takes as input the direction of Tom's action, i.e, { '0'for LEFT, '1' for RIGHT, '2' for DOWN, '3' for UP} and returns the new state along with the reward for performing the action and further indicates whether the run is over or not.

- **The render() Method**

  This method is used for rendering the image of the environment with the diagnostic data. This method does not promise any efficiency and is solely for the purpose of visualization and debugging. This method can be omitted to speed up training time.

- **The reset() Method**

  The reset() method is invoked to initialize the the environment with a fresh episode, allowing us to effectively run new episodes.

2.2 The Agent

In this project, our agent is Tom. The agent class has all the functionalities that will help Tom in learning to navigate our environment. Our agent class has the following functionalities:

- **The __init__() Method**

  The __init__() method initializes the agent and makes the agent aware of the input and output dimensions of the brain as well.

- **The act() Method**

  The **act()** method takes as input the current state and returns the agent's selected action. Based on the epsilon value , the act method decides whether it should return a random state or predict a state to maximise the chance of Tom reaching Jerry.

- **The observe() Method**

  The observe() method takes four values as input (s, a, r, s_) and saves them in the Memory. These four values represent the current state, action , reward and future state of the agent, respectively.

- **The replay() Method**

  The replay() method is where the actual learning occurs. The agent trains the brain based on his previous experiences.

2.3 Memory

The Memory class provides the necessary functionality to store and manage the agent's past experiences. The memory class has the following functionalities:

- **The __init__ () Method**

  The __init__() method initializes the memory with certain capacity.

- **The add() Method**

  The add() method appends the agent's experience to the memory.

- **The sample() Method**

  The sample() method takes 'n' as input and returns a sample of 'n' experiences from the memory.

2.4 The Brain

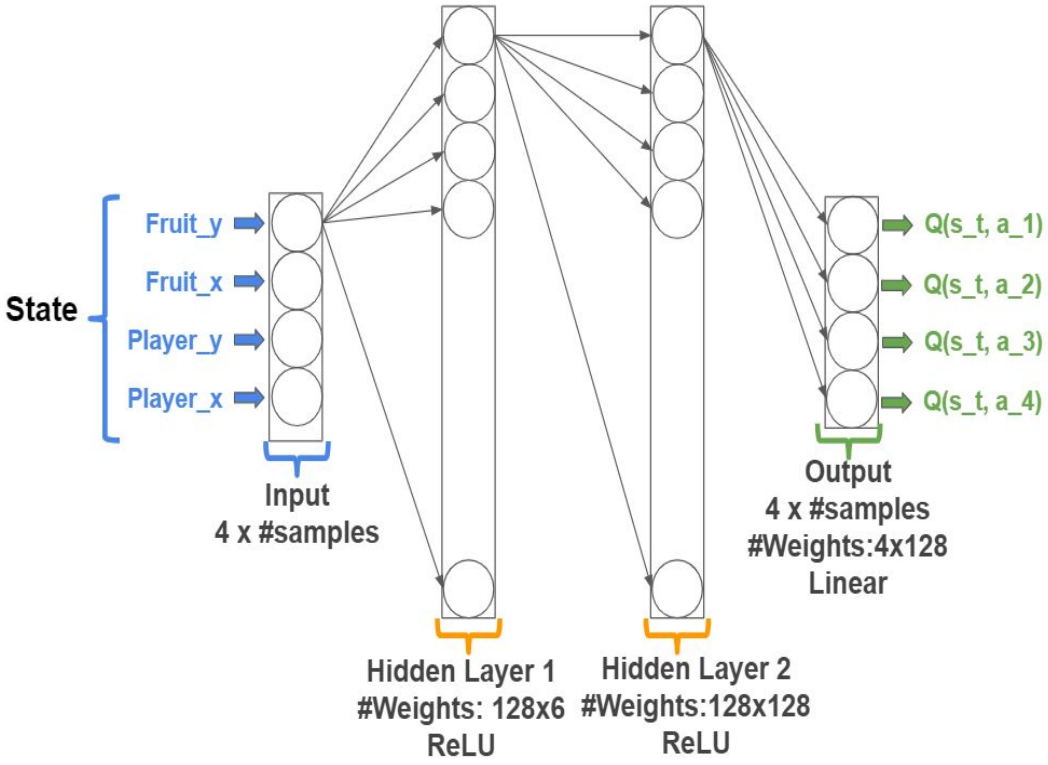
The brain class encompasses the necessary functions to model and Train the agent using a Multi-layer Neural network model. The functionalities of the brain class includes:

- **The __init__() Method()**

  The __init__() method initializes the Multi-Layer Neural Network Model.

- **The _createModel() Method**

  The _createModel() method is where the Multi-Layer Neural Network Model is fully constructed to result in the following model as represented below.



- **The train() Method**

  The train() method is invoked to train the model on the agent's past experiences.

- **The predict() Method**

  The predict() method is invoked to predict the future state of the agent.

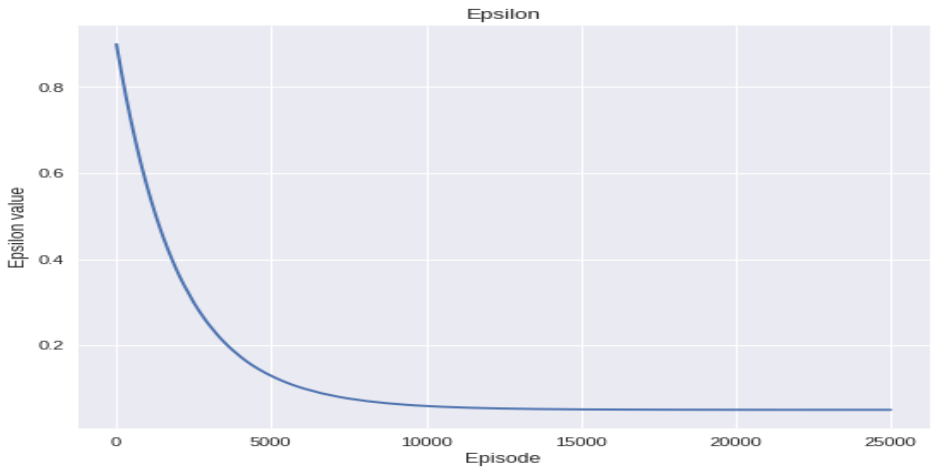
3. Analysing the Model

We can analyse and try to improve the Model's performance by tuning the hyper-parameters. For this problem, the main hyper-parameter we can consider is the number of episodes.

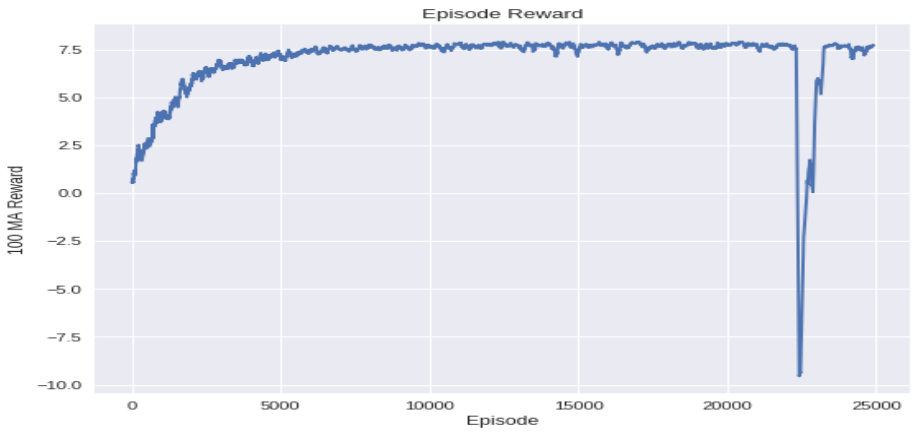
Tuning Number of Episodes

Number of Episodes	Mean Reward
9000	6.651
8000	5.77
10000	6.55
12000	7.01
13000	7.12
14000	7.17
15000	7.2
18000	7.31
19000	7.304
20000	7.37
25000	7.44

From the table we can see that increasing the number of episodes improves the model's efficiency.



From the above graph, we can infer how our agent's exploring rate (epsilon) decreases with increase in number of episodes, which is exactly what we exactly desired.



The above graph shows how the agent's rewards increase with increase in number of episodes.

4. Training the Agent

Inorder for the agent to choose an action that maximizes the Q-value, the agent should have enough past-experience that would help in making a wise decision. This experience can be gained only choosing random actions initially. If prior experience was already available, and then if the agent always chooses the action that maximizes the Q-value, then Optimal policy will be achieved and our agent will be able to follow optimal path to reach goal. Without prior experience data we should make sure our agent explores random actions. Two ways to make sure the agent explores could be:

1. In this project, we have enforced this by using the epsilon rate. Our agent at first will take random actions depending on the epsilon rate. Once, it gains enough experience we gradually reduce the number of random action by introducing an exponential-decay function of epsilon.
2. Adding adaptive noise to the dataset is also one exploration strategy that frequently boosts performance.

Conclusion

Through this project , we learn the application of Reinforcement learning and Deep-Q- Learning using the Tom and Jerry example. I was also exposed to several libraries that had in built DQN networks such as the gym and the stable baseline libraries.