**Tom and Jerry in Reinforcement Learning**

### Shraddha Dwivedi

UB Person Number: 50290266

Computer Science and Engineering

University at Buffalo

[sdwivedi@buffalo.edu](mailto:sdwivedi@buffalo.edu)

**Reinforcement Learning:**

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.

Reinforcement learning can be understood using the concepts of agents, environments, states, actions and rewards.

Agent: An **agent** takes actions;

Action (A): A is the set of all possible moves the agent can make. An **action** is almost self-explanatory, but it should be noted that agents choose among a list of possible actions.

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.

State (S): A **state** is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes.

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

**Description of code-snippets implementation:**

**Implementation of Neural network:**

I have implemented the 3-layer neural network with the two hidden layers. The activation function for first and second layer is relu and for output layer, it’s linear. We get 4 output values in output layer.

For reinforcement learning, we need incremental neural networks since every time the agent receives feedback, we obtain a new piece of data that must be used to update some neural network. When we have a very large number of state-action pairs, it is not feasible to store every Q-factor separately. Then, it makes sense to store the Q-factors for a given action within one neural network. When a Q-factor is needed, it is fetched from its neural network. When a Q-factor is to be updated, the new Q-factor is used to update the neural network itself.



**Implementation of epsilon using exponential-decay formula**

I have implemented the epsilon using the exponential decay formula given below:

****

The probability of taking random actions is known as epsilon. Implementing the exponential-decay formula for epsilon will allow the agent to explore the environment. Epsilon value changes with changing the hyperparameters and with the total number of steps.

Epsilon value can be improved by tuning the hyperparameter and setting the total number of steps, so that agent will explore the environment more and select the action from explored environment, which in turn increase the reinforcement learning.



**Implementation of Q-function**

I have implemented the Q-function which uses the Bellman equation and take two inputs:

State and Action. This is an iterative process of updating the values. As we start to explore the environment, the Q-function give us the better approximation by updating the Q-values in table. Our goal will be to maximize the Q-function



**Hyperparameters:**

I have changed the following hyperparameters to observe the influence in the total time of training and the mean award:

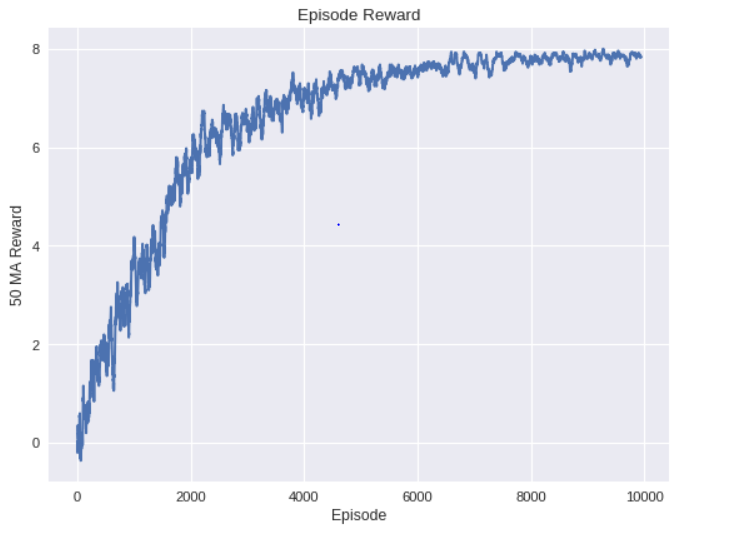
1. Maximum epsilon: It is the maximum rate in which agent randomly decides the action.

2. Minimum epsilon: It is the maximum rate in which agent randomly decides the action.

3. Lamba: It can be defined as learning rate of the model or speed of decay for epsilon.

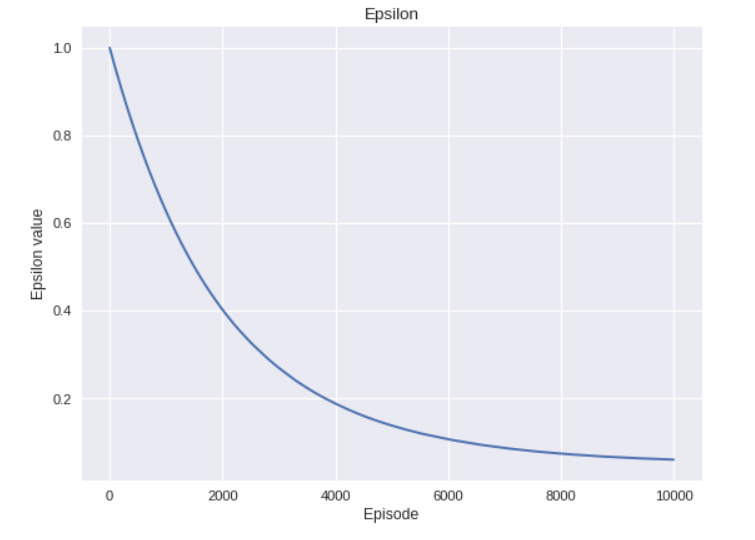
4. Number of episodes: Number of games agents play.

1) MAX\_EPSILON = 1 , MIN\_EPSILON = 0.05 , LAMBDA = 0.00005, num\_episodes = 10000

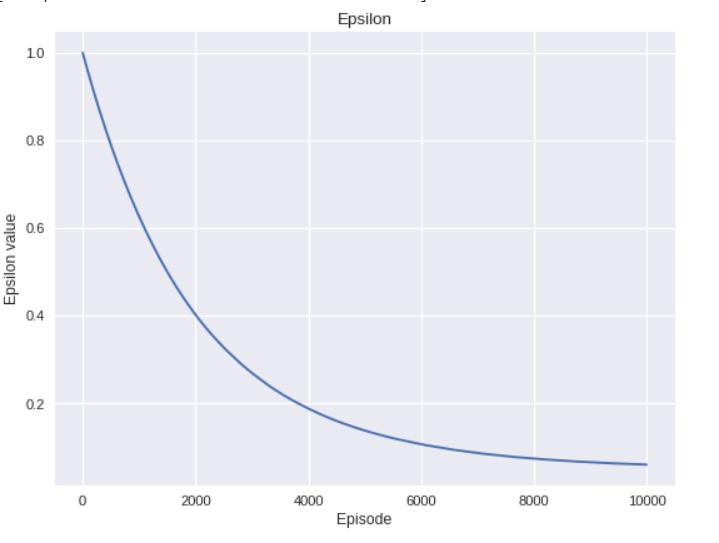
When the number of episodes set to 10000 and lambda value set to 0.00005, then the reward value get stable 

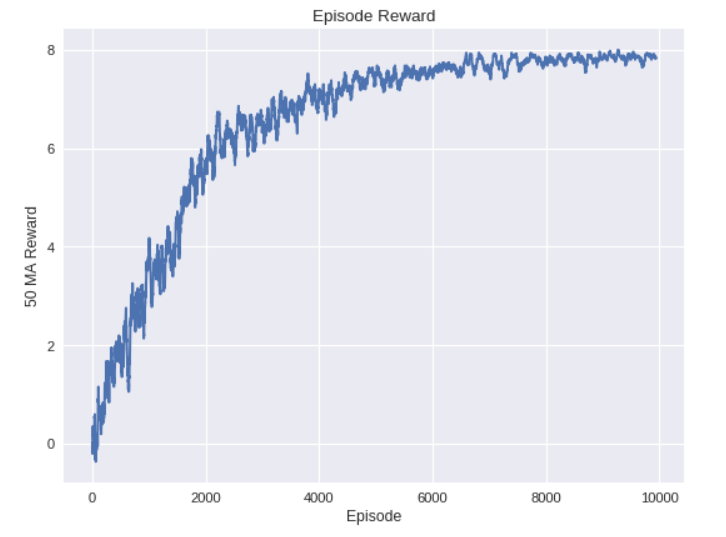
When the number of episodes set to 10000 and lambda value set to 0.00005, then the reward value get stable

After 7000 episodes and reaches to 8 rewards.



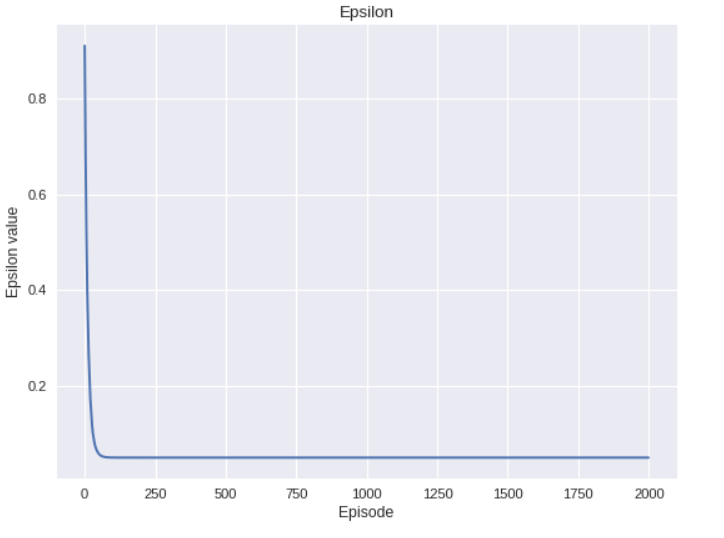
2) MAX\_EPSILON = 1 , MIN\_EPSILON = 0.05, LAMBDA = 0.00005 , num\_episodes = 2000

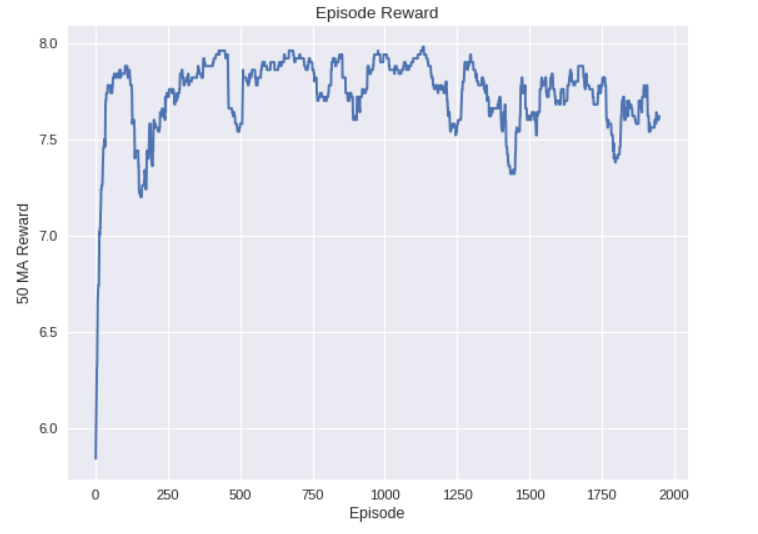


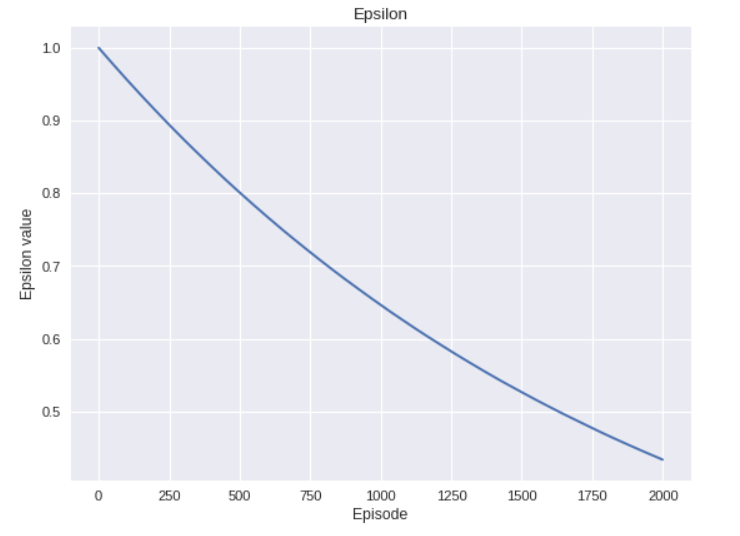


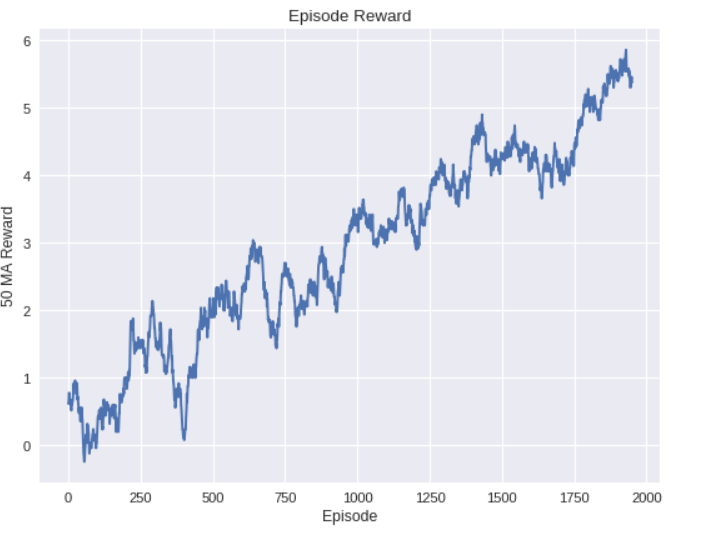
When the number of episodes set to 2000, maximum epsilon to 1, minimum epsilon to 0.05 and lambda value set to 0.00005, then the reward value get stable after 6500 episodes and reaches to 8 rewards.

3) MAX\_EPSILON = 1 , MIN\_EPSILON = 0.05 , LAMBDA = 0.01, num\_episodes = 2000



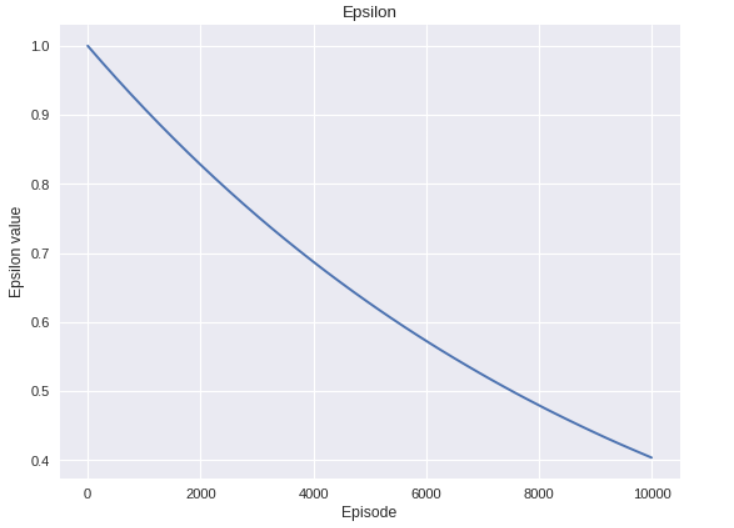


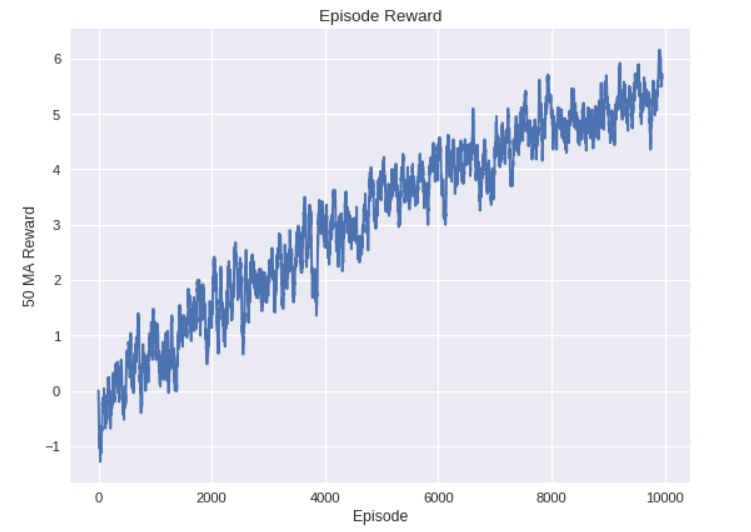
4) MAX\_EPSILON = 1, MIN\_EPSILON = 0.1 , LAMBDA = 0.00005 , num\_episodes = 2000 



Here rewards value is not getting stable for the given hyper parameters.

5) MAX\_EPSILON = 1, MIN\_EPSILON = 0.05, LAMBDA = 0.00001 , num\_episodes = 10000





When the number of episodes set to 2000, maximum epsilon to 1, minimum epsilon to 0.05 and lambda value set to 0.00001, then the reward value get stable after 10000 episodes and reaches to 6 rewards.

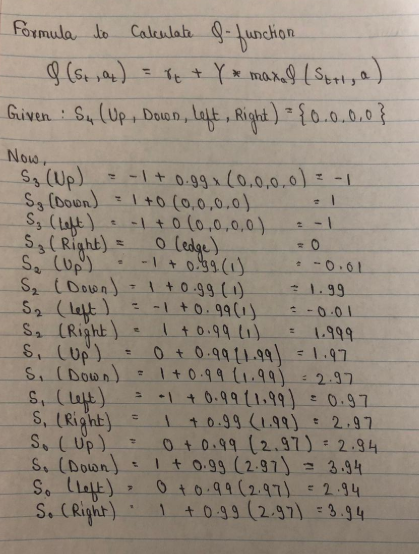
**Task 1.2**

**1)** If the agent always chooses the action which maximizes the Q-value, then the environment will not be fully explored to find the best action for each and every state and it get stuck in non-optimal policies. We can use the following two ways to force the agent to explore the environment:

First, we can set the initial values high.

Secondly, we can occasionally pick the random actions

**2)**



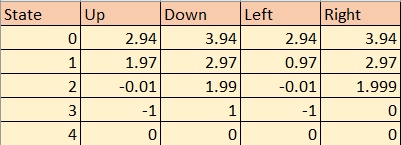


Fig: Q-table for states

**References**

[1] https://[www.geeksforgeeks.org/](http://www.geeksforgeeks.org/)

[2] https://medium.com/the-theory-of-everything/

[3] https://www.cs.ubc.ca/~conati/422/422-2010World/Final/MoreQuestionsSolutions.pdf