CSE574 Introducion to Machine Learning

By Prof: Sargur Srihari

Project 4 Report.

Submitted by :
Sai Varun Alapati
Ubit : saivarun
Personal Number :50290571

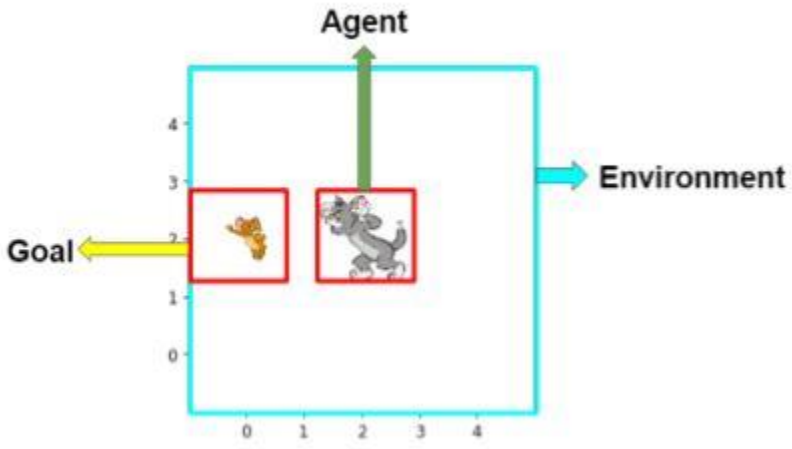
Table of Contents

1. Introduction and Problem Definition :

This project combines reinforcement learning and deep learning. Main task is to teach the agent to navigate in the grid-world environment. Problem contains modeled game Tom and Jerry cartoon, where Tom, a cat, is chasing Jerry, a mouse. In this modeled game the task for Tom (an agent) is to find the shortest path to Jerry (a goal), given that the initial positions of Tom and Jerry are deterministic. To solve the problem, we would apply deep reinforcement learning algorithm - DQN (Deep Q-Network), that was one of the first breakthrough successes in applying deep learning to reinforcement learning.

2.Environment Description:

The environment is designed as a grid-world 5x5:



• States - 25 possible states (0, 0), (0, 1), (0, 2), ... ,(4, 3), (4, 4)
• Actions - left, right, up, down
• The goal (yellow square) and the agent (green square) are dynamically changing the initial position on every reset.

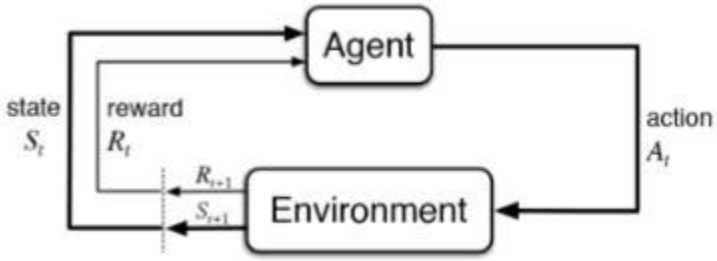
3. Objective/Goal:

Our main goal is to let our agent learn the shortest path to the goal. In the environment the agent controls a green square, and the goal is to navigate to the yellow square (reward +1), using the shortest path. At the start of each episode all squares are randomly placed within a 5x5 grid-world. The agent has 100 steps to achieve as large a reward as possible. They have the same position over each reset, thus the agent needs to learn a fixed optimal path.

4. Reinfocement Learning :

Reinforcement learning is a direction in Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results. In reinforecement learning, an agent learns from trial-and-error feedback rewards from its environment, and results in a policy that maps states to actions to maximize the long-term total reward as a delayed supervision signal. Reinforcement learning combining with the neural networks has made great progress recently, including playing Atari games and beating world champions at the game of Go. It is also widely used in robotics.

4.1 Markov Decision Process:



Basic reinforcement is modeled as a Markov decision process a 5-tuple (S,A,Pa,Ra,γ), where
• S is a finite set of states
• A is a finite set of actions
• Pa(s,s0) = Pr(st+1 = s0 | st = s,at = a) is the probability that action a in state s at time t will lead to state s0 at time t + 1
• Ra(s,s0) is the immediate reward (or expected immediate reward) received after transitioning from state s to state s0 due to action a
• γ ∈ [0,1) is the discount factor, which represents the difference in importance between future rewards and present rewards

In application, we typically have an environment, which handles state and reward, and an agent, which decides which action to take given a particular state. An environment starts with some initial state s0, which is passed to the agent. The agent passes an action a0, based on the state s0, back to the environment. The environment reacts to the action, then passes the next state, s1, along with the resulting reward for taking the action, r0, back to the agent. This process continues until we reach a terminal state, indicating the end of an episode, at which point the process may start over again. In reinforcement learning, we attempt to choose actions which yields the best results given some predefined criteria. This involves learning a mapping from state to action which attempts to maximize discounted accumulative reward. In deep reinforcement learning, a neural network is used approximate this mapping. Currently, there are two frequently used approaches to doing this: off-policy learning and on-policy learning.

4.2 Discounting factor (γ) :

The discounting factor (γ ∈ [0,1]) penalize the rewards in the future. Reward at time k worth only γ^{k-1}
• The future rewards may have higher uncertainty like in stock market
• The future rewards do not provide immediate benefits (as human beings, we might prefer to have fun today rather than 5 years later)
• Discounting provides mathematical convenience (we do need to track future steps infinitely to compute return)
• It is sometimes possible to use undiscounted Markov reward processes (e.g. γ = 1), if all sequences terminate.

5. Deep Q-Learning

Deep Q-Learning Algorithm

5.1 Experience Replay:

Experience replay will help us to handle three main things:

- Avoid forgetting previous experiences.
- Reduce correlations between experiences.
- Increases learning speed with mini-batches.

The main idea behind the experience replay is that by storing an agent experiences, and then randomly drawing batches of them to train the network, we can more robustly learn to perform well in the task. By keeping the experiences we draw random, we prevent the network from only learning about what it is immediately doing in the environment, and allow it to learn from a more varied array of past experiences. Each of these experiences are stored as a tuple of <state, action, reward, next state>. The Experience Replay buffer stores a fixed number of recent memories (memory capacity), and as new ones come in, old ones are removed. When the time comes to train, we simply draw a uniform batch of random memories from the buffer, and train our network with them.

The Algorithm is:

Algorithm 1 Deep Q-learning with Experience Replay

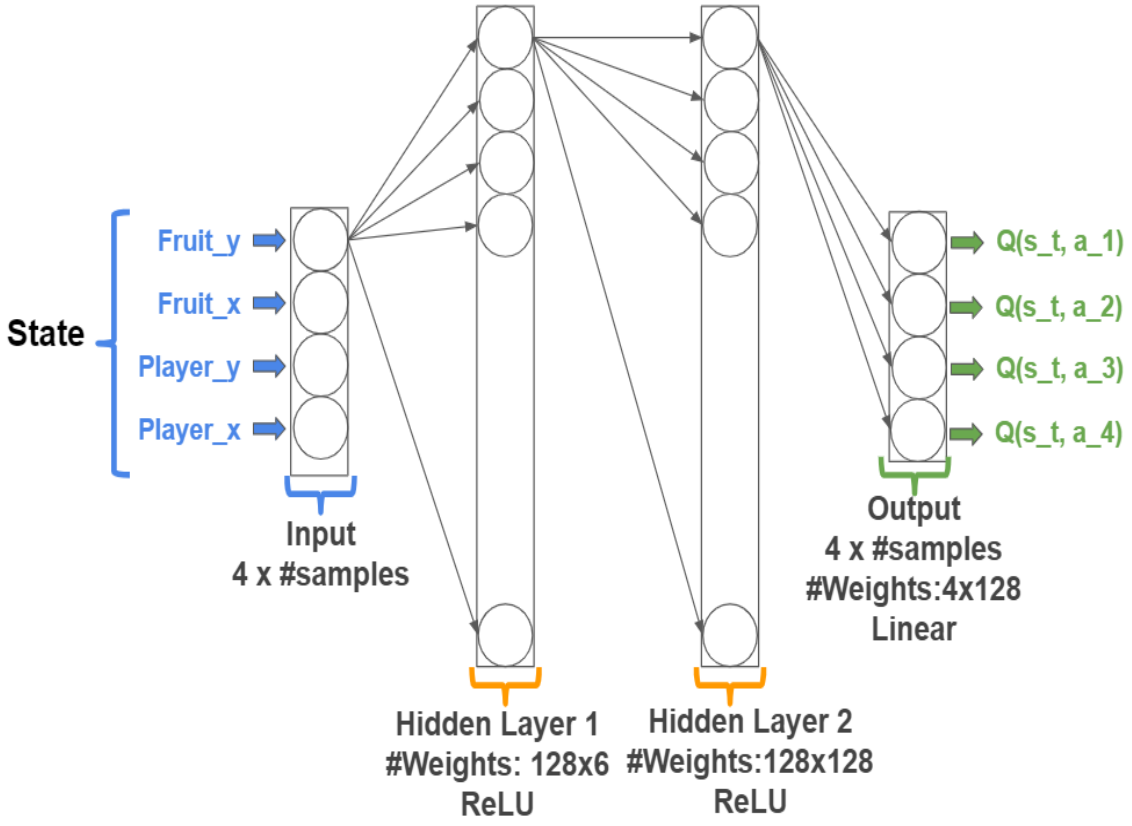
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

6. Coding Tasks :

6.1 Building a 3-layer neural network using Keras library

The 'brain' of the agent is where the model is created and held.

Neural Network structure for our task



Our DQN takes a stack of six-tuple as an input. It is passed through two hidden networks, and output a vector of Q-values for each action possible in the given state. **Example:** Q(st,a1) - q-value for a given state s, if we choose action a1 We need to choose such an action, that will return the higest Q-value.

In the beginning, the agent does really badly. But over time, it begins to associate states with best actions to do.

The Neural network model:

- The model's structure is: LINEAR -> RELU -> LINEAR -> RELU -> LINEAR.
- Activation function for the first and second hidden layers is 'relu'
- Activation function for the final layer is 'linear' (that returns real values)
- Input dimentions for the first hidden layer equals to the size of your observation space (state_dim)
- Number of hidden nodes is 128
- Number of the output should be the same as the size of the action space (action_dim

6.1.a Code written :

```
model.add(Dense(output_dim=128, activation='relu', input_dim=self.state_dim))
model.add(Dense(output_dim=128, activation='relu'))
model.add(Dense(output_dim=self.action_dim, activation='linear'))
```

6.2. Implementing exponential-decay formula for epsilon

Epsilon :

Our agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon'. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it 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. We want our agent to decrease the number of random action, as it goes, so we indroduce an exponential-decay epsilon, that eventually will allow our agent to explore the evironment.

Exponential-decay formula for epsilon:

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

where $\epsilon_{min},\epsilon_{max} \in [0,1]$

λ - hyperparameter for epsilon

$|S|$ - total number of steps

6.2.a Code Written:

```
self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon)\* math.exp(-self.lamb * self.steps)
```

6.3 Implementing Q-function :

Q-function maps state-action pairs to the highest combination of immediate reward with all future rewards that might be harvested by later actions in the trajectory. This step is crutial for training the agent.

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

6.3.a Code Written:

```
if s_ is None:
      # If terminal state means no 'next state' ,  Q-Value is reward
      t[act] = rew
    else:
      # If its not, we calculate the  expected discounted rewards
      t[act] = rew + self.gamma * np.amax(q_vals_next[i])
```

7. Experimental Results :

7.1 Hyperparameters used :

- **Gamma(γ) :** The discount rate, to calculate the future discounted reward
- **Epsilon :** Our agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon'.
- **Max Epsilon :** Maximum rate in which an agent randomly decides its action
- **Min Epsilon:** Minimum rate in which an agent randomly decides its action
- **Episodes :** The agent have a starting point and an ending point (a terminal state). This creates an episode: a list of States, Actions, Rewards, and New States. Simply saying it is number of games we want the agent to play

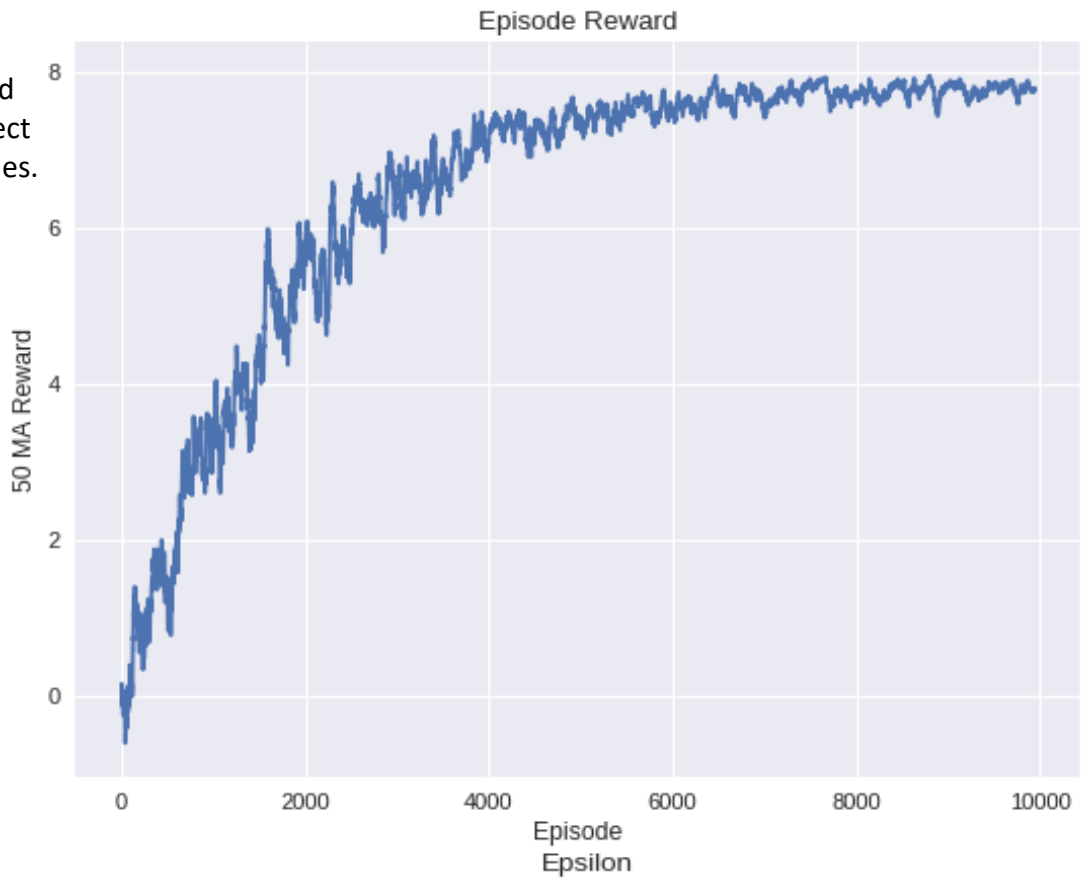
7.2 Varying Gamma (γ) :

We observed the behavior of Episode rewards and Epsilon values with respect to number of Episodes for different values of γ keeping number of episodes as 10k and Min Epsilon as 0 and Max Epsilon as 1

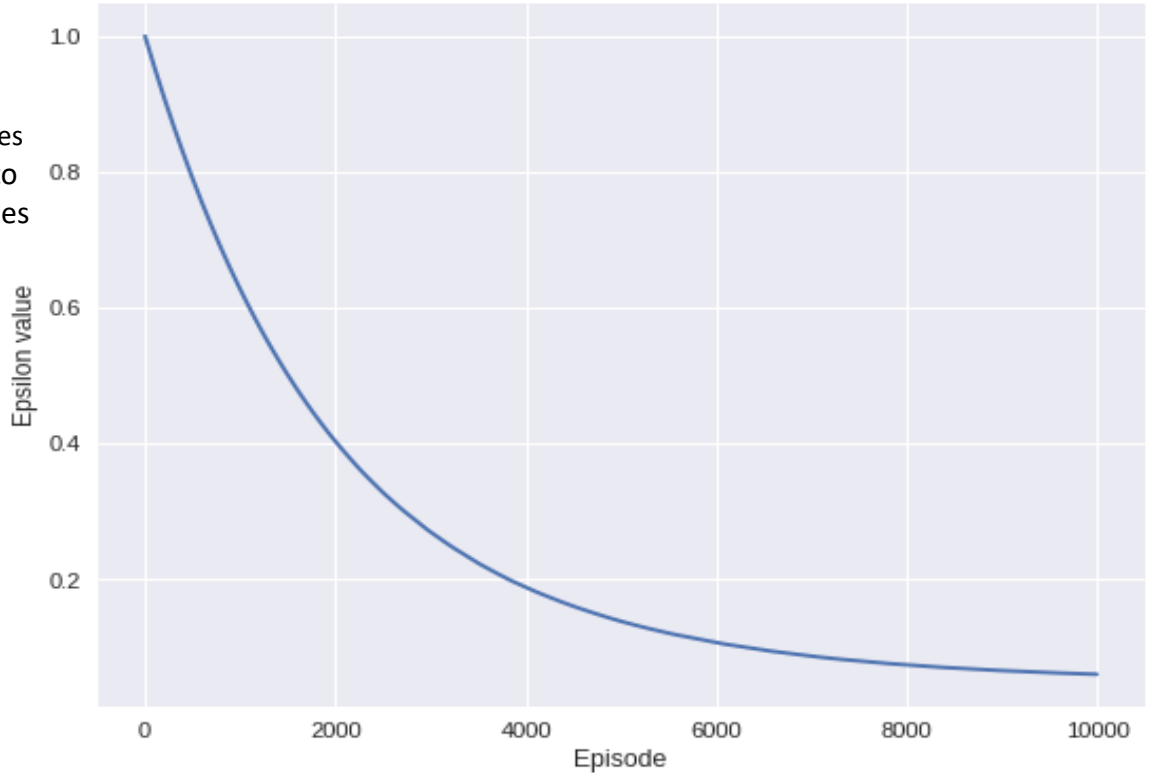
Values of γ	Total Time Elapsed	Epsilon Value at Last Episode	Last Episode Reward	Episode Reward Rolling Mean
0.5	922.37s	0.05573	6	6.12
0.8	844.58s	0.0607	7	6.38
0.99	777.22s	0.06079	8	6.4067

7.2.1 For γ = 0.5

The Episode reward is plotted with respect to number of Episodes.

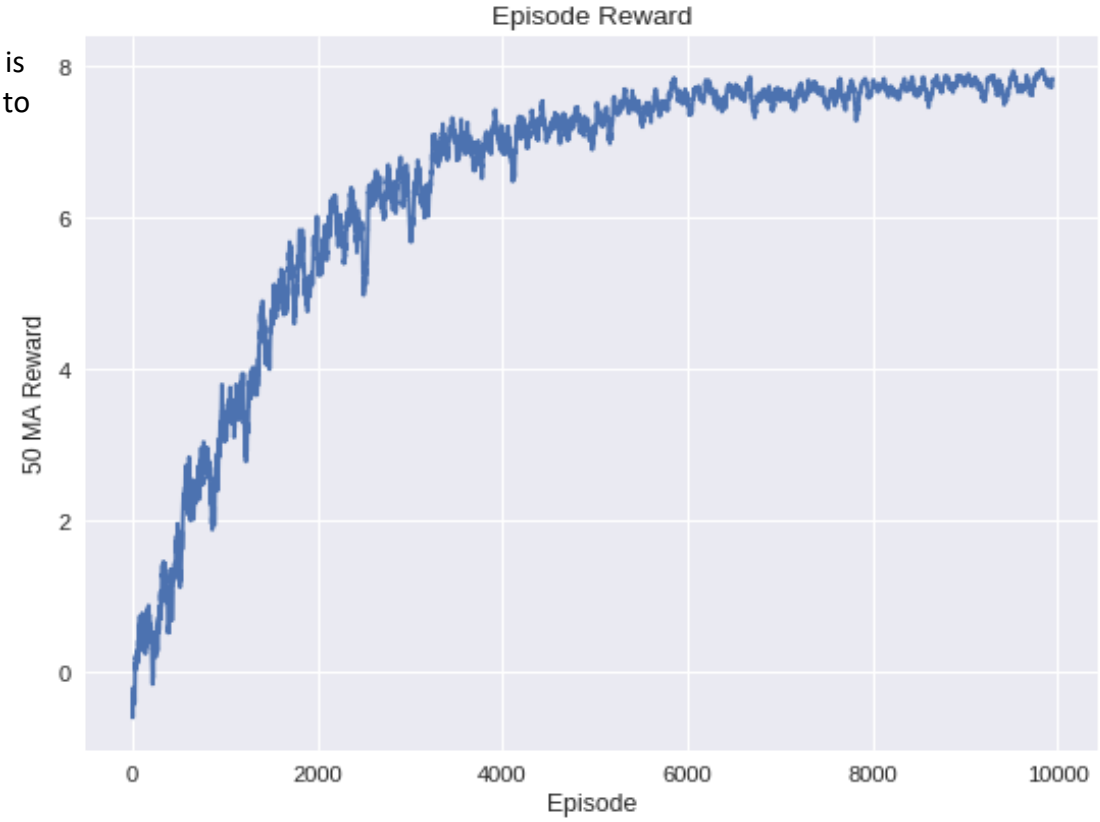


The Epsilon values with respect to number of Episodes

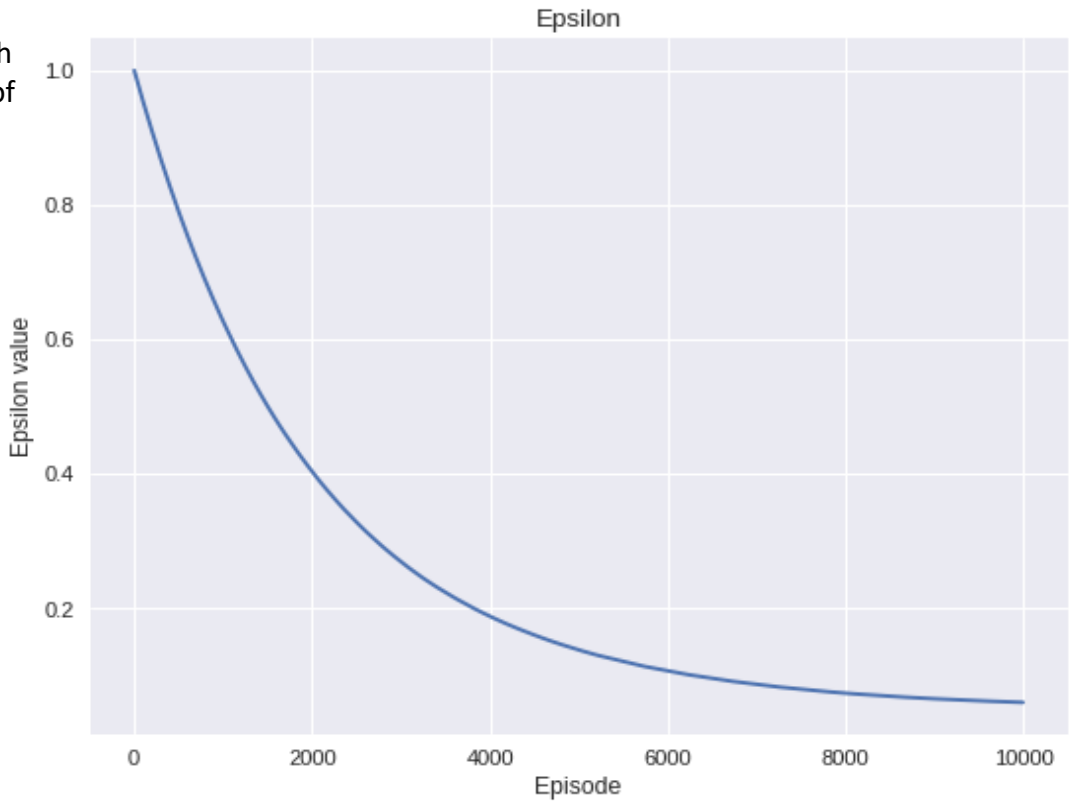


7.2.2 For γ = 0.8

The Episode reward is plotted with respect to number of Episodes.

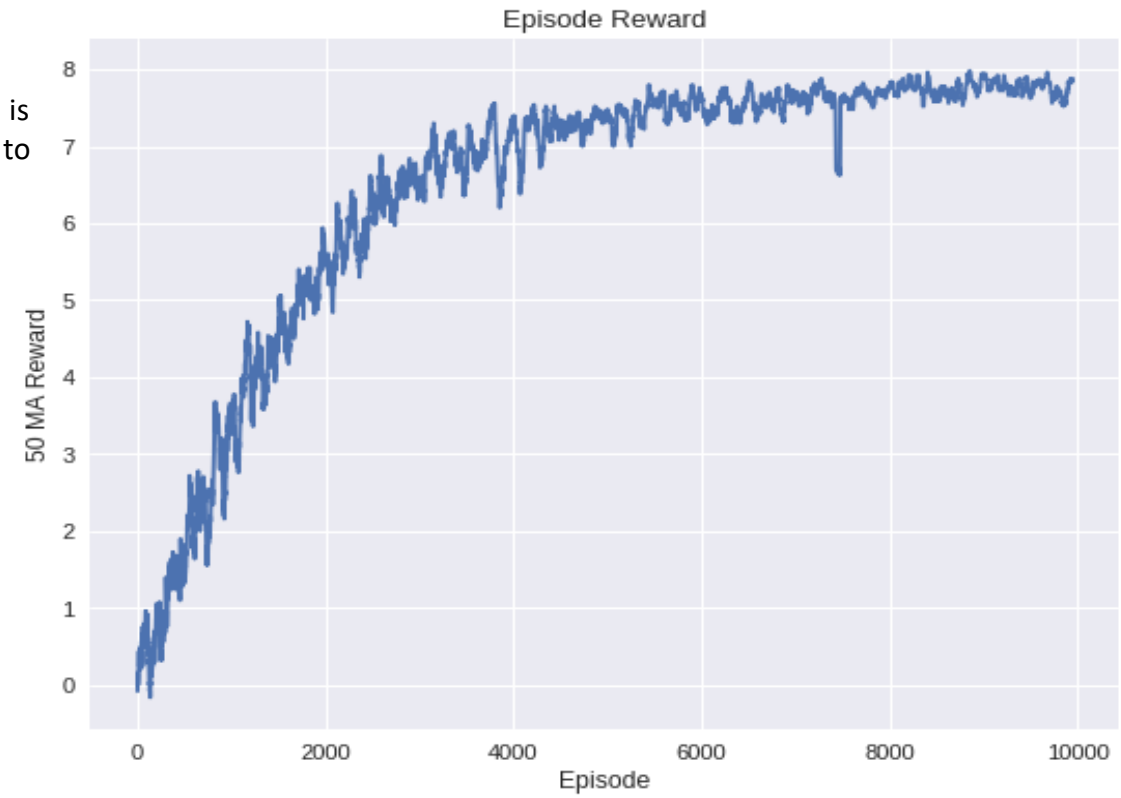


The Epsilon values with respect to number of Episodes

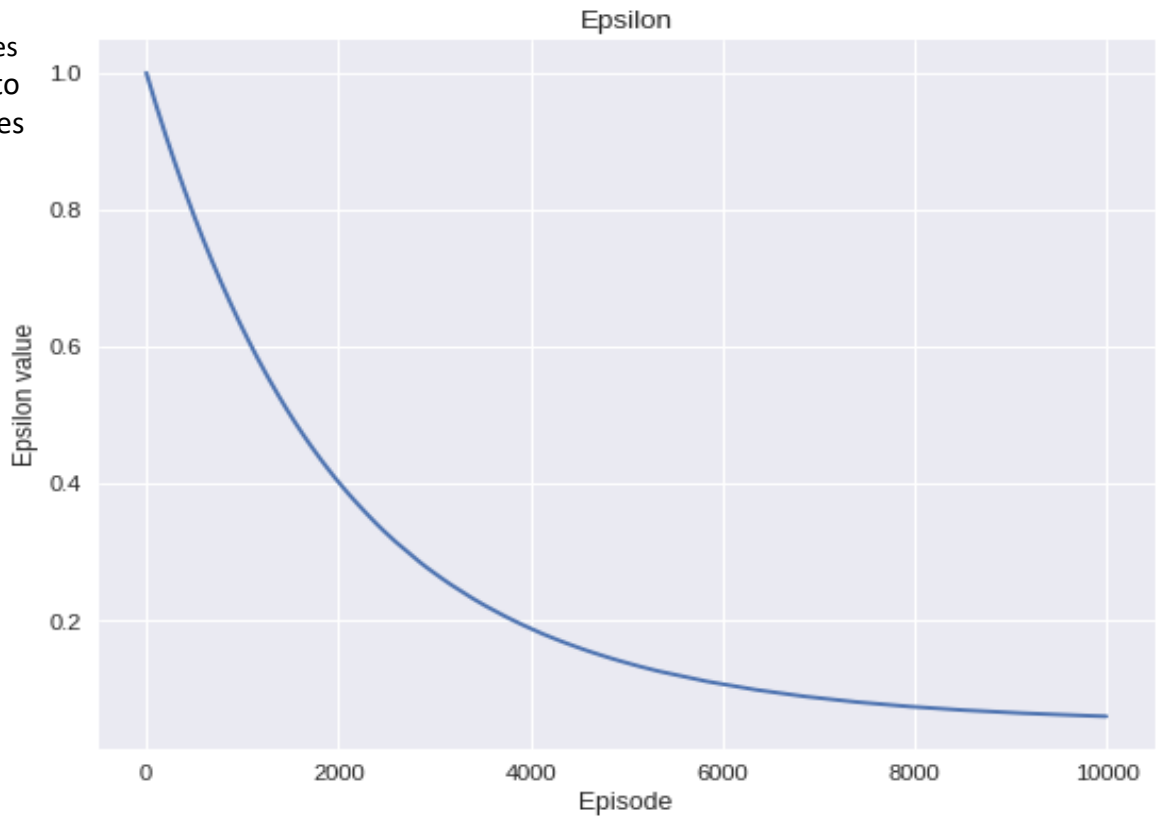


7.2.3 For γ = 0.99

The Episode reward is plotted with respect to number of Episodes.



The Epsilon values with respect to number of Episodes



7.2.4 Observations :

As described before γ is a discounting factor where it belongs to (γ ∈ [0,1]) which penalize the rewards in the future. We observed that the value of γ is crucial for calculating the Q-function and as its value is increasing the total time elapsed got decreased and the mean reward value are increased over the number of episodes.

7.3 Varying Epsilon :

Epsilon :

Our agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon'. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it 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. We want our agent to decrease the number of random action, as it goes, so we indroduce an exponential-decay epsilon, that eventually will allow our agent to explore the evironment.

Exponential-decay formula for epsilon:

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

where $\epsilon_{min},\epsilon_{max} \in [0,1]$
λ - hyperparameter for epsilon
$|S|$ - total number of steps

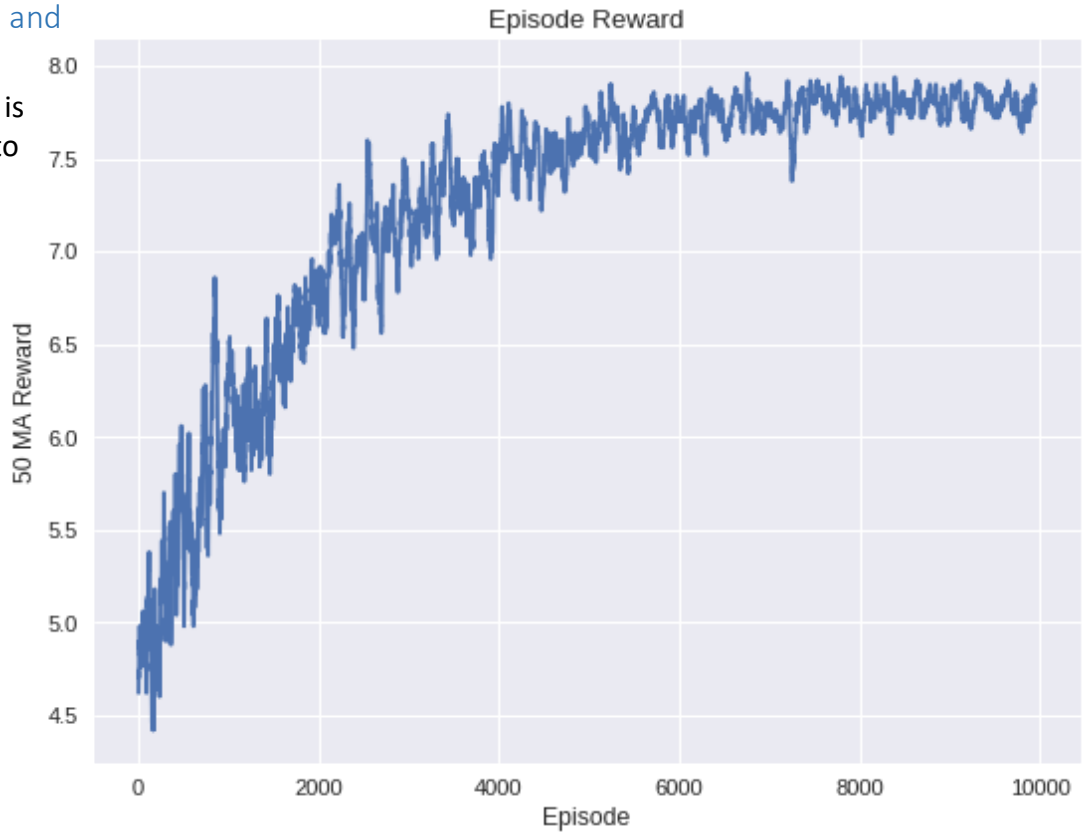
- **Gamma(γ) :** The discount rate, to calculate the future discounted reward
- **Max Epsilon :** Maximum rate in which an agent randomly decides its action
- **Min Epsilon:** Minimum rate in which an agent randomly decides its action
- **Episodes :** The agent have a starting point and an ending point (a terminal state). This creates an episode: a list of States, Actions, Rewards, and New States. Simply saying it is number of games we want the agent to play

We observed the behavior of Episode rewards and Epsilon values with respect to number of Episodes for different values of Min and Max Epsilon Values keeping number of Episodes as 10k and γ = 0.99 .

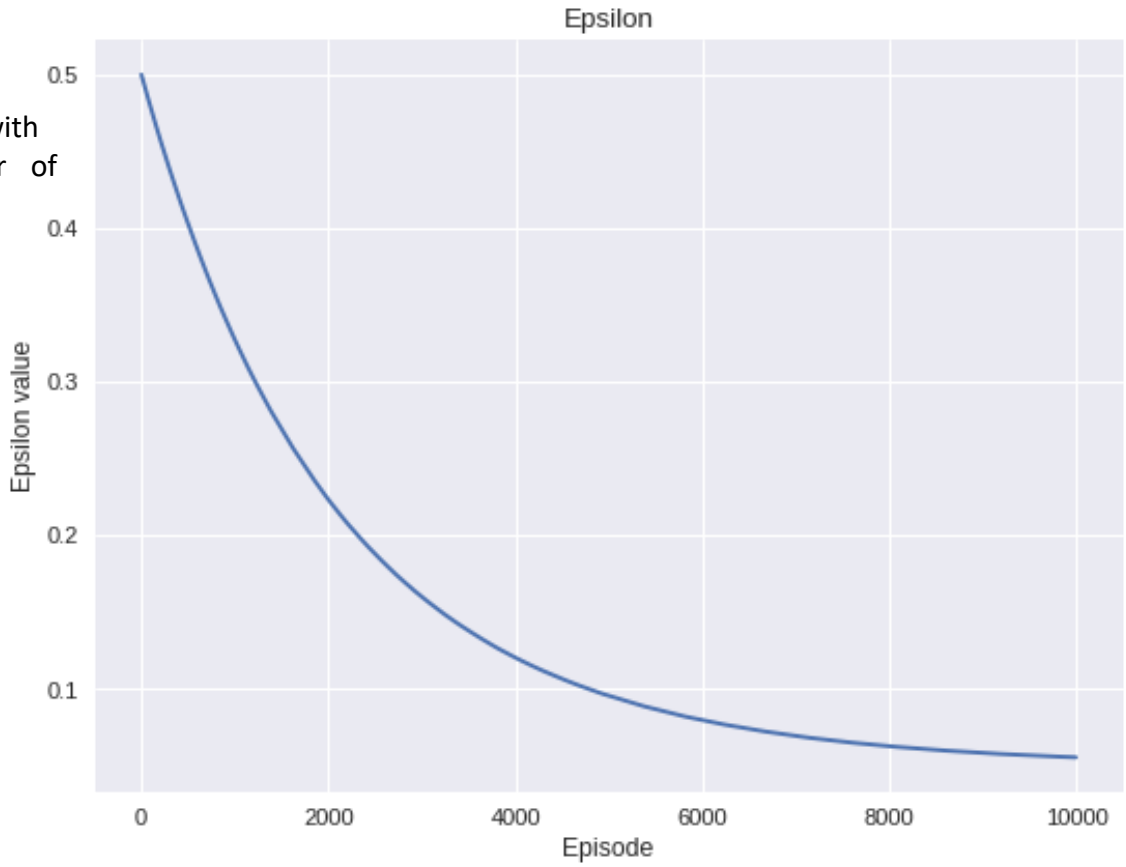
Epsilon Min	Epsilon Max	Total Time Elapsed	Epsilon Value at Last Episode	Last Episode Reward	Episode Reward Rolling Mean
0	0.5	902.37s	0.05573	8	7.225
0.25	1	933.29s	0.256	8	5.397
0	1	777.22s	0.06079	8	6.4067

7.3.1 For Epsilon min = 0 and Epsilon max =0.5 :

The Episode reward is plotted with respect to number of Episodes.

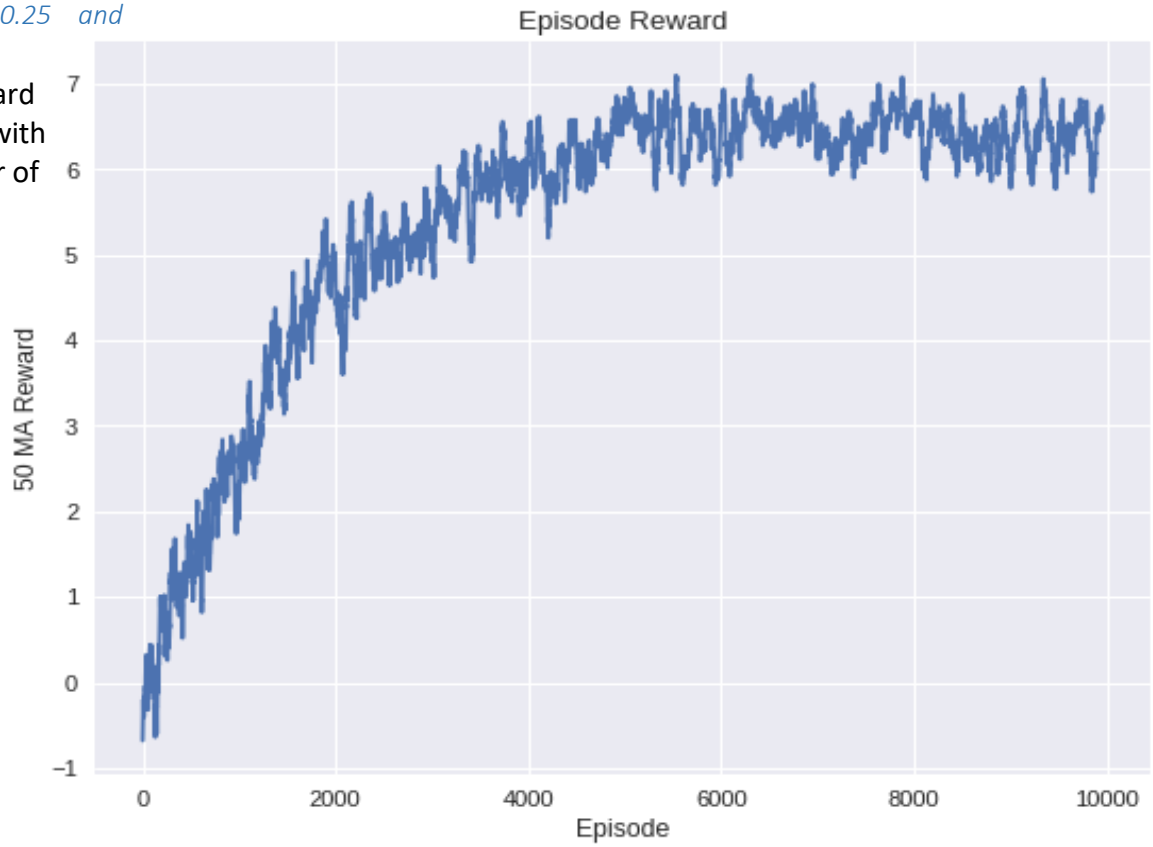


The Epsilon values with respect to number of Episodes

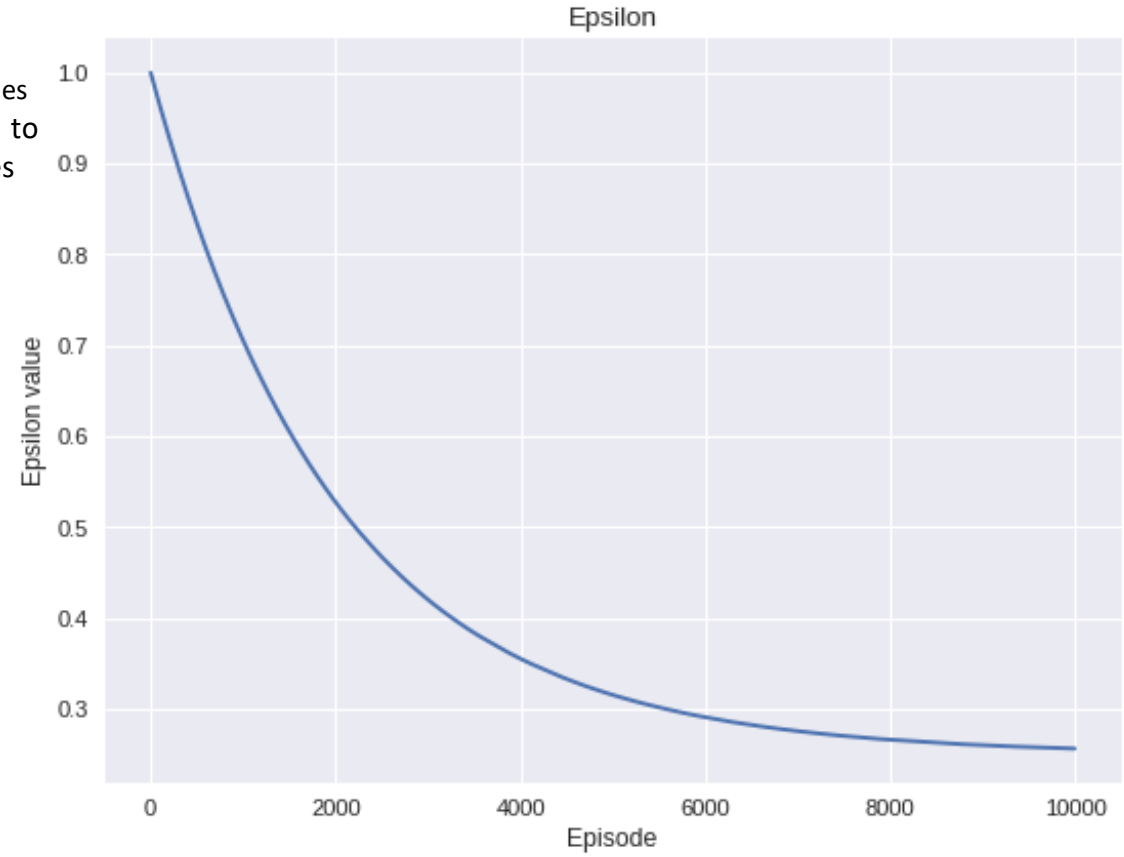


For Epsilon min = 0.25 and Epsilon max =1 :

The Episode reward is plotted with respect to number of Episodes.

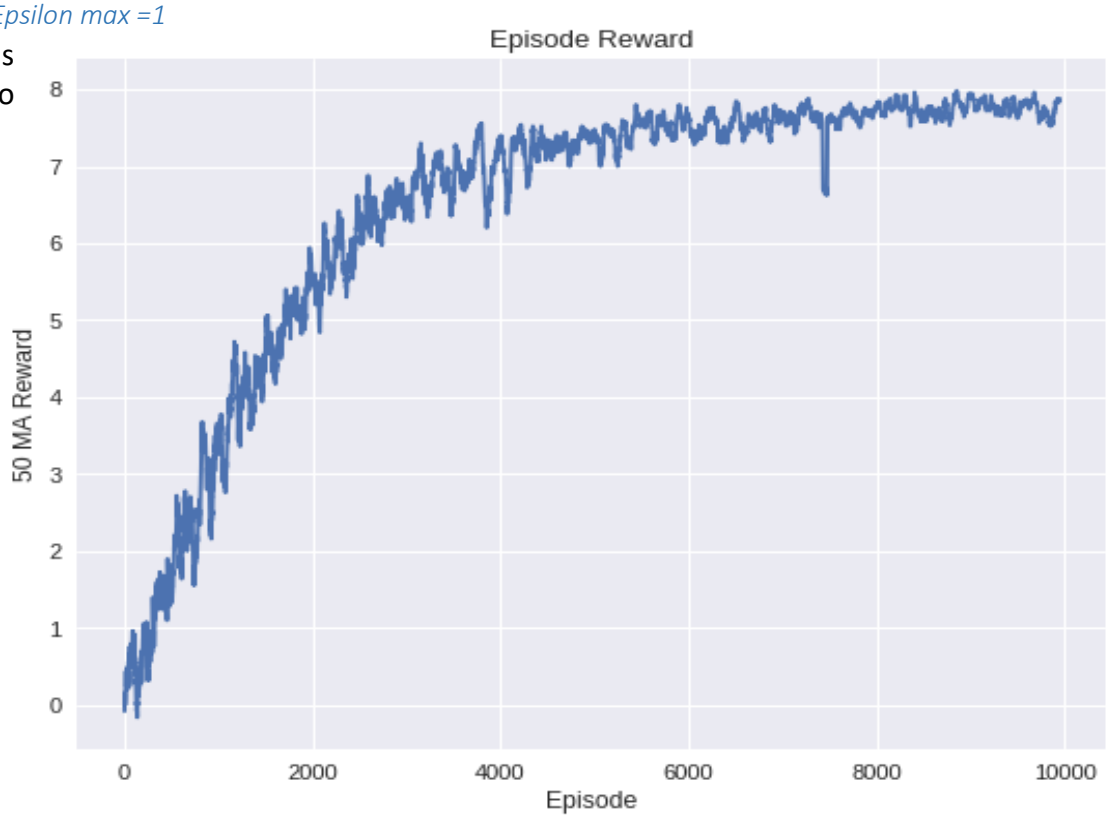


The Epsilon values with respect to number of Episodes

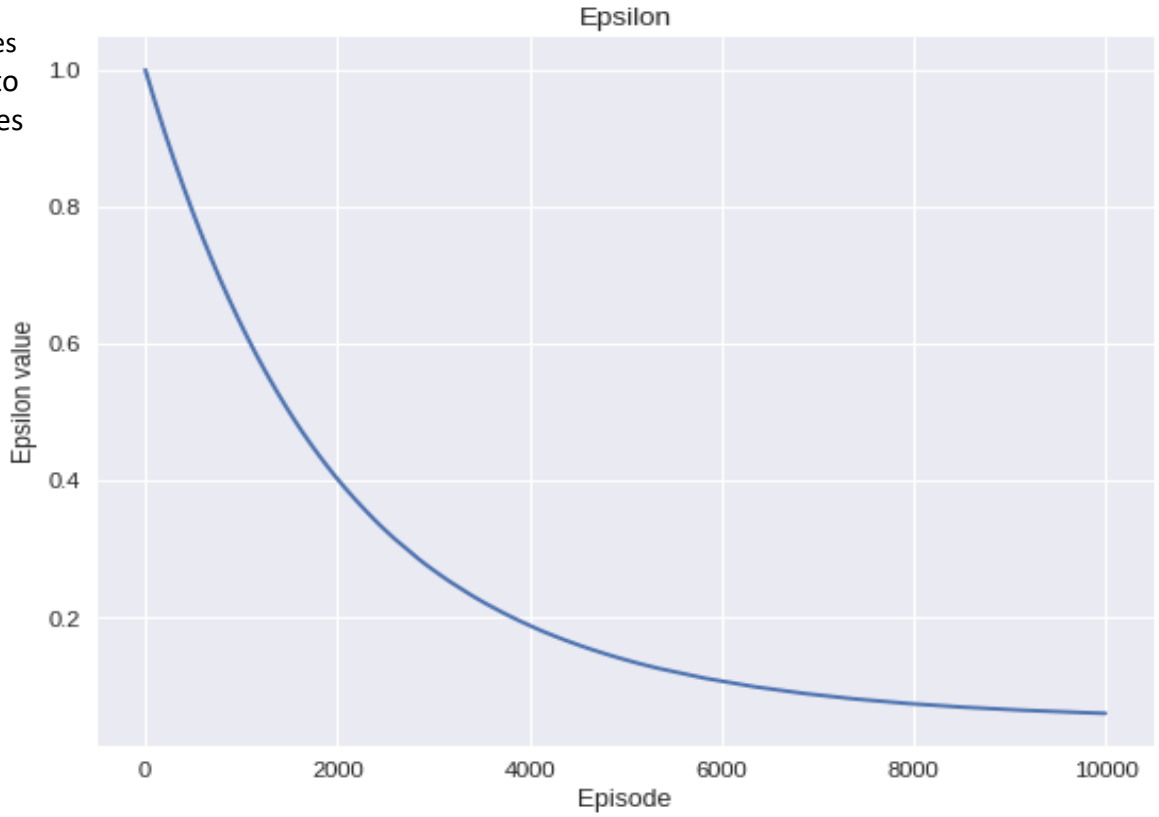


For Epsilon min = 0 and Epsilon max =1

The Episode reward is plotted with respect to number of Episodes.



The Epsilon values with respect to number of Episodes



7.3.4 Observations :

As described before, 'exploration rate' or 'epsilon' is a method which randomly selects its action at first by a certain percentage. We observed that the value of 'epsilon' is important for determining initial actions and as its value is increasing the total time elapsed got decreased and the mean reward value are increased over the number of episodes.

7.4 Varying Episodes :

Episodes : The agent have a starting point and an ending point (a terminal state). This creates an episode: a list of States, Actions, Rewards, and New States. Simply saying it is number of games we want the agent to play

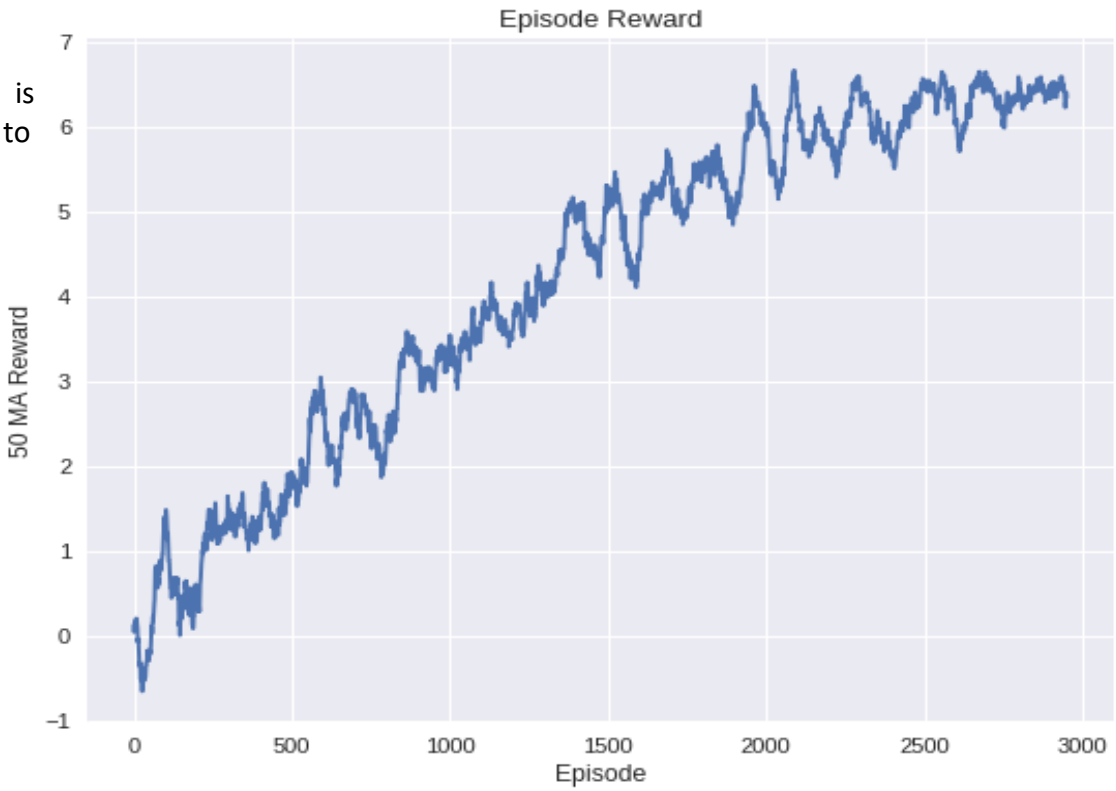
- **Gamma(γ) :** The discount rate, to calculate the future discounted reward
- **Epsilon :** Our agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon'.
- **Max Epsilon :** Maximum rate in which an agent randomly decides its action
- **Min Epsilon:** Minimum rate in which an agent randomly decides its action

We observed the behavior of Episode rewards and Epsilon values with respect to number of Episodes for different Episodes values Min Epsilon as 0 and Max Epsilon as 1 and γ = 0.99 .

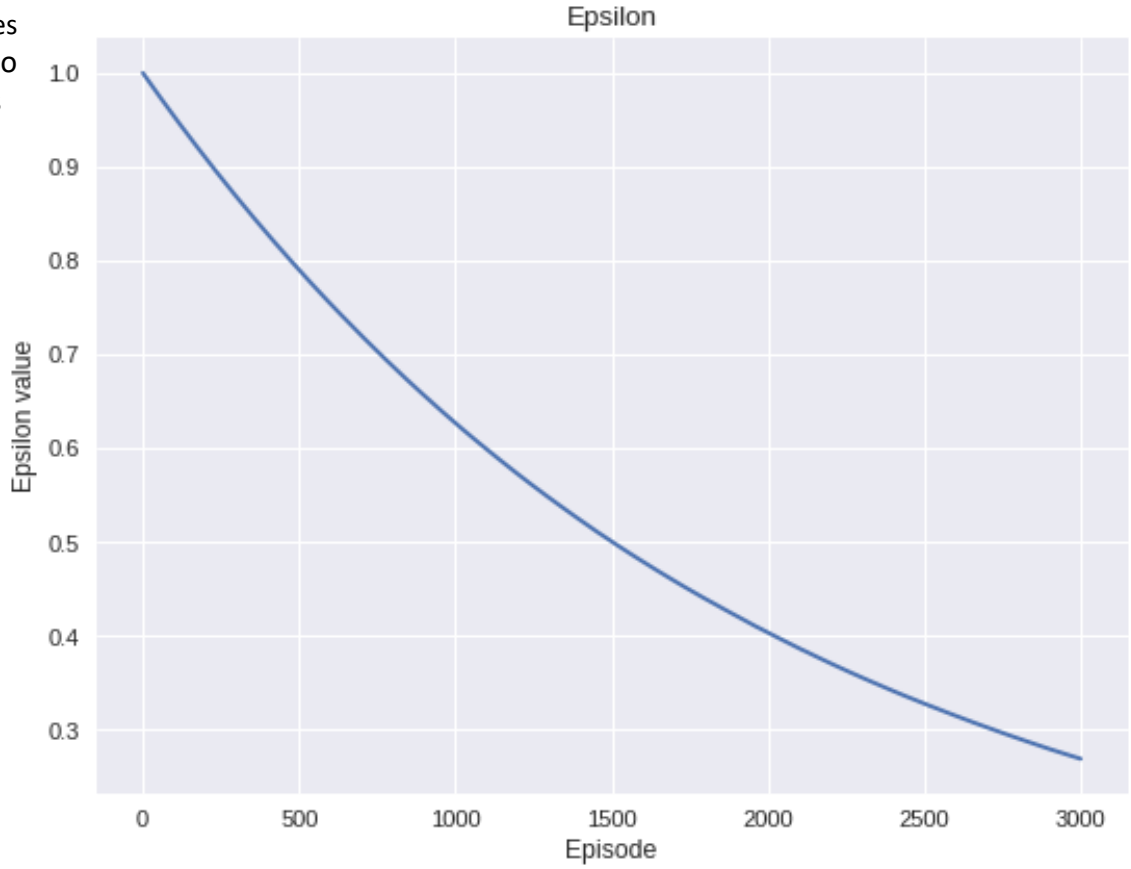
Episodes	Total Time Elapsed	Epsilon Value at Last Episode	Last Episode Reward	Episode Reward Rolling Mean
3k	277.66s	0.279	6	3.99
6k	538.41s	0.109	8	5.484
10k	777.22s	0.06079	8	6.4067

7.4.1 For Episodes=3k

The Episode reward is plotted with respect to number of Episodes.

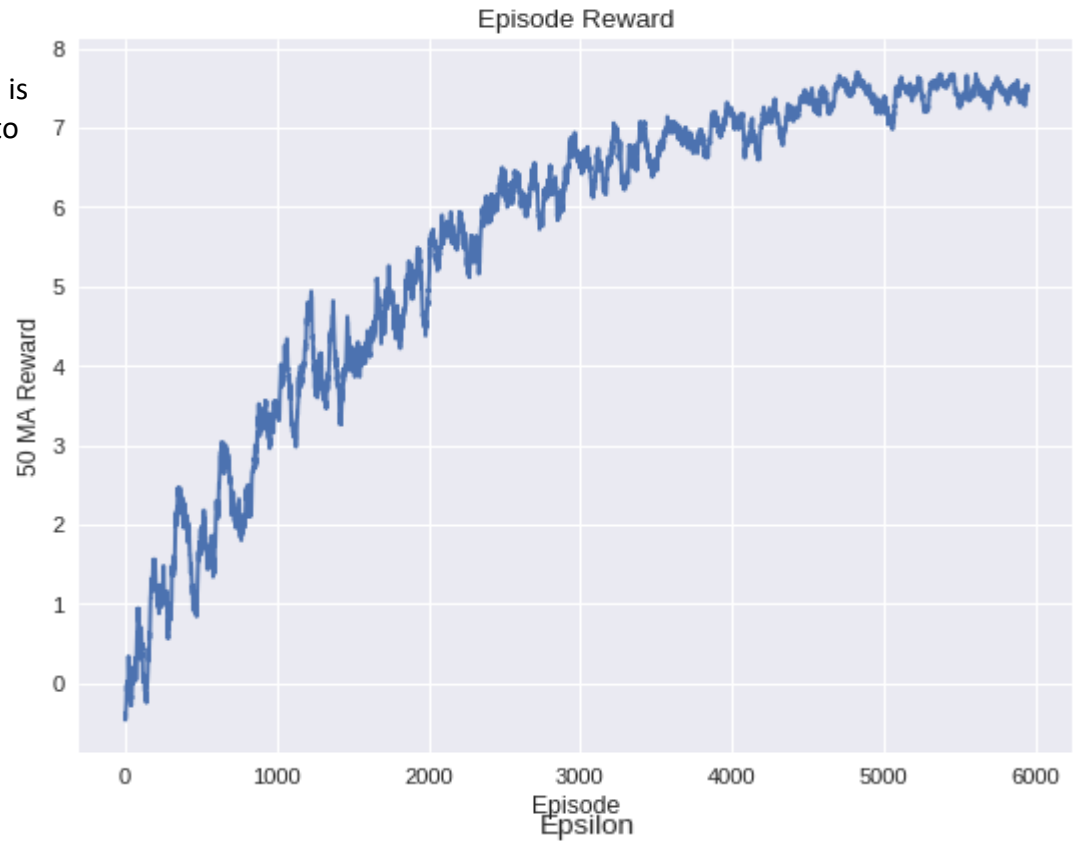


The Epsilon values with respect to number of Episodes

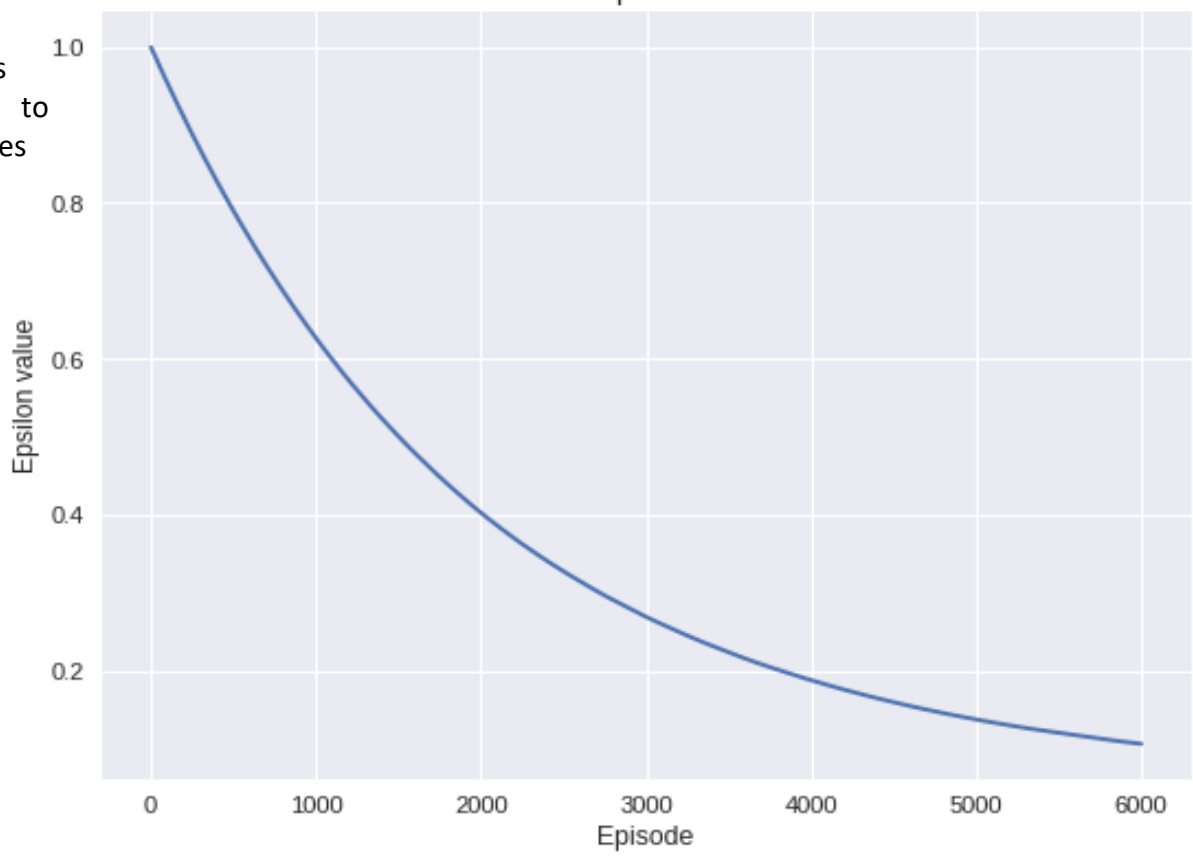


7.4.2 For Episodes =6k

The Episode reward is plotted with respect to number of Episodes.

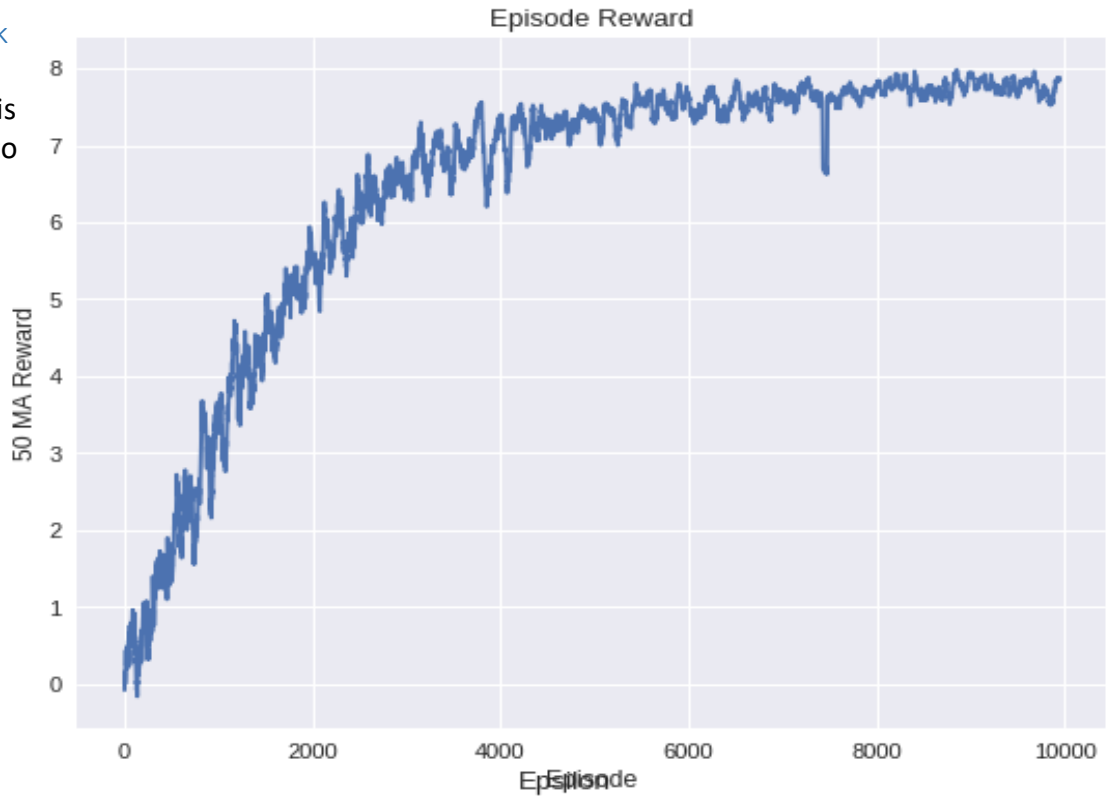


The Epsilon values with respect to number of Episodes

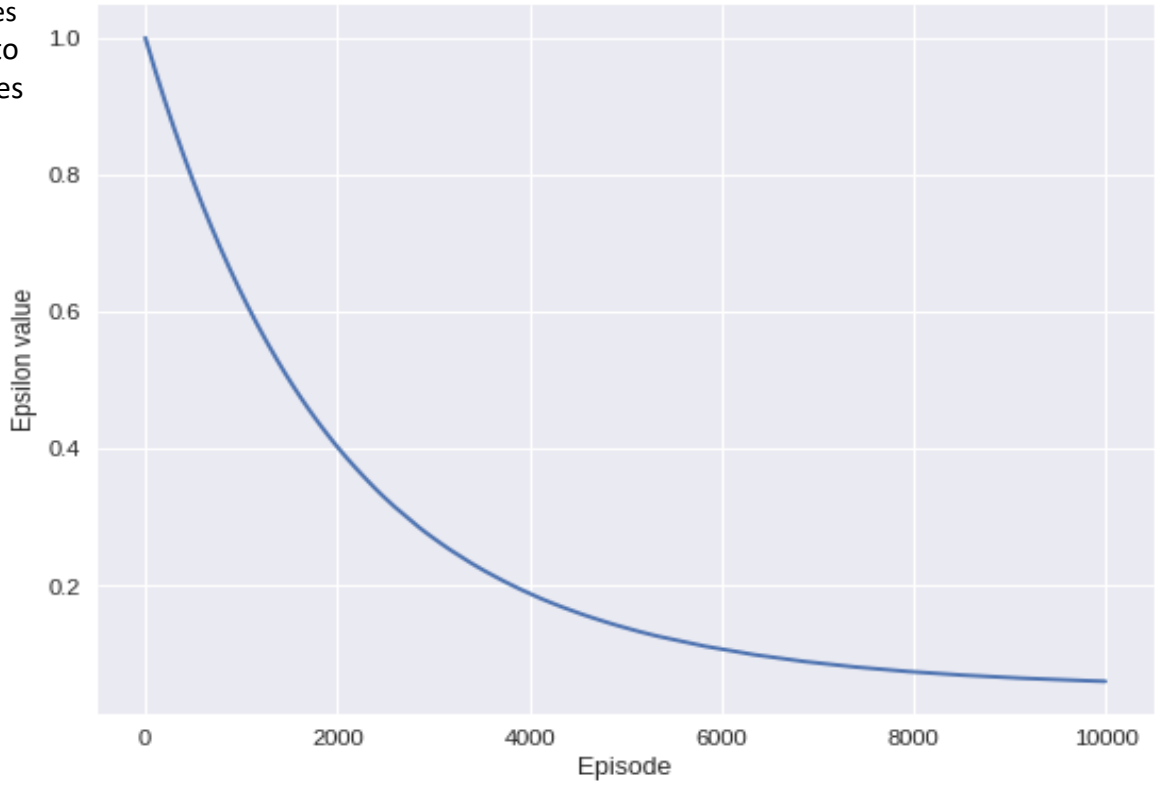


7.4.3 For Episodes =10k

The Episode reward is plotted with respect to number of Episodes.



The Epsilon values with respect to number of Episodes



7.4.4 Observations :

As described before, episodes are the number of games we want the agent to play. We observed that the value of 'epsiode' is important for determining the stable path from the starting state to terminating state and we identified it needs min of 6k for finding stable shortest path and get maximum rewards for every next episode.

8 Conclusion and Summary

- The best model hyper parameters needed to give the maximum rewards for each episode and to find the shortest stable path for Tom to catch the jerry are :

Gamma(γ)	0.99
Max Epsilon	1
Min Epsilon	0
Epsiodes	10k

- This model needs minimum of 6k episodes for finding stable shortest path and get maximum rewards for every next episode.

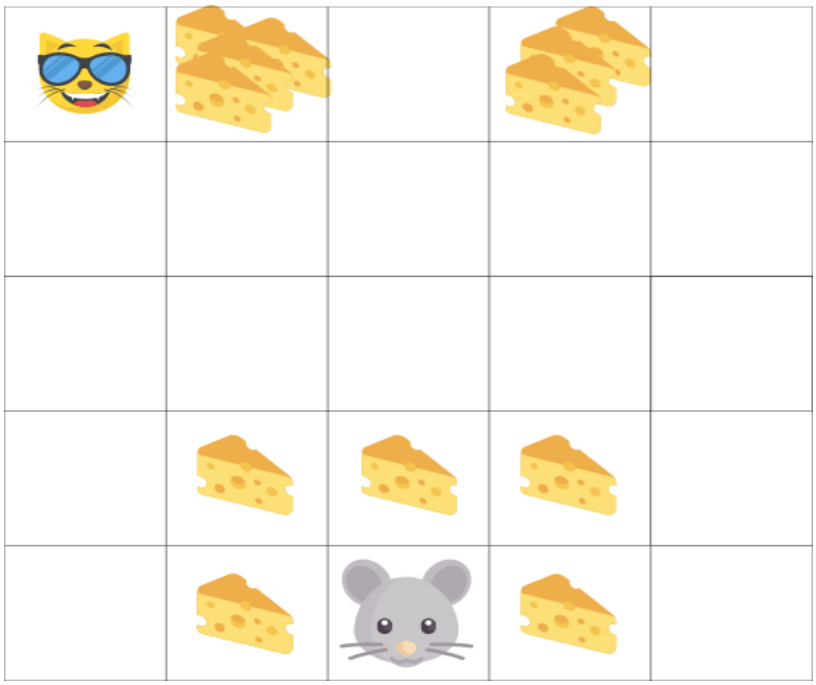
9 Writing Part :

9.1 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: In this case, the agent will get stuck in non-optimal policies as the agent will not explore enough to find the best possible action from each of the state as the agent will always choose the action to maximize Q.

Let us illustrate this with a sample example :

From beside figure, Let's say our agent is this small mouse and its opponent is the cat. Our agent's goal is to eat the maximum amount of cheese before being eaten by the cat.
As we can see in the diagram, if the agent always chooses the action that maximizes the Q-value it's more probable to eat the cheese near it than the cheese close to the cat (the closer we are to the cat, the more dangerous it is).
As a consequence, the reward near the cat, even if it is bigger (more cheese), will be discounted. We're not really sure our agent will be able to eat it.



The two ways by which agent we can force the agent to explore is :
1) Set the initial values high. If the initial values are high the unexplored region will look good.
2) Make it pick random values occasionally so that it starts exploring.

9.2 Calculate Q-value for the given states and provide all the calculation steps.

Given,

$Q(s_t; a_t) = r_t + \gamma * max_a Q(s_t+1; a)$

$\gamma = 0.99$

The Calculated Q table is :

State	Actions			
	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

In a given deterministic 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)

Now let's see each state individually in the 3x3 grid,

S0	S1-b	N-b
S1-a	S2	S3-b
N-a	S3-a	Goal

There are some same states due to symmetry which are :

- S1-a and S1-b is same due to symmetry and therefore considered as S1 only.
- NS-aand NS-b are same due to symmetry and therefore considered as N only.
- S3a and S3b is same due to symmetry and therefore considered as S3 only.

Calculations :

Now for the state S4, going to any direction will yield zero reward. Therefore, we fill the above table with zeros.

Now for state S3-b (S3 in given diagram),

Action UP

$Q_t3 = -1 + 0.99 * \max Q_tN = -1 + 0.99*1.99 = -1 + 1.97 = 0.97$

Action DOWN

$Q_t3 = 1 + 0.99 * \max Q_tG = 1$

Action LEFT
$Q_t3 = -1 + 0.99 * \max Q_t2 = - 1 + 0.99*1.99 = 0.97$

Action RIGHT
$Q_t3 = 0 + 0.99 * \max Q_t3 = 0.99$

Now for state S2,
Action UP
$Q_t2 = -1 + 0.99 * \max Q_t1 = -1 + 0.99 *2.97 = 1.94$

Action DOWN
$Q_t2 = 1 + 0.99 * \max Q_t3 = 1.99$

Action RIGHT
$Q_t2 = 1 + 0.99 * \max Q_t3 = 1.99$

Action LEFT
$Q_t2 = -1 + 0.99 * \max Q_t1 = -1 + 0.99 *2.97 = 1.94$

Now for state S1-b (S1 in given diagram) ,
Action UP
$Q_t1 = 0 + 0.99 * \max Q_t1 = 0.99 *2.97 = 2.94$

Action DOWN
$Q_t1 = 1 + 0.99 * \max Q_t2 = 1+ 0.99*1.99 = 1+1.97 = 2.97$

Action LEFT
$Q_t1 = -1 + 0.99 * \max Q_t0 = -1 + .99*3.94 = 2.90$

Action RIGHT
$Q_t1 = 1 + 0.99 * \max Q_tP = 1+ 0.99*1.99 = 1+1.97 = 2.97$

Now for state N-b,
Action UP
$Q_tN = 0 + 0.99 * \max Q_tN = 1.97$

Action DOWN
$Q_tN = 1 + 0.99 * \max Q_t3 = 1.99$

Action RIGHT

$Q_tN = 0 + 0.99 * \max Q_tN = 1.97$

Action LEFT
$Q_tN = -1 + 0.99 * \max Q_t1 = -1 + 0.99*2.97 = 1.94$

Now for state S0,
Action UP
$Q_t0 = 0 + 0.99 * \max Q_t0 = 3.90$

Action DOWN
$Q_t0 = 1 + 0.99 * \max Q_t1 = 1+ 0.99*2.97 = 3.94$

Action RIGHT
$Q_t0 = 1 + 0.99 * \max Q_t1 =1+ 0.99*2.97 = 3.94$

Action LEFT
$Q_t0 = 0 + 0.99 * \max Q_t0 = 3.90$

References :
https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419