CSE474/574: Introduction to Machine Learning (Fall 2018)
Project 4: Tom and Jerry in Reinforcement Learning

UBIT: sassikes
Person no: 50291558

Introduction:

Our main objective of this project is to implement the reinforcement learning using the deep neural networks to make the agent to learn the shortest path to reach the goal. Here, the task for Tom (an agent) is to find the shortest path to Jerry (a goal).

We then tune the hyperparameters like Epsilon min and max, gamma, lambda etc and observe how the total mean reward varies and how the agent behaves.

Basic Understanding:

Reinforcement learning: Reinforcement Learning is one of the three types of Machine Learning. The other two are Supervised and Unsupervised Learning. Reinforcement learning is a kind of Machine Learning where in the system that is to be trained to do a particular job, learns on its own based on its previous experiences and outcomes while doing a similar kind of a job. The most common application of reinforcement learning are PC games, Robotics and Alpha Go.

The reinforcement learning model consists of:

- A set of environment and agent states S.
- A set of actions A of the agent.
- Policies of transitioning from states to actions.
- Rules that determine the scalar immediate reward of a transition.
- Rules that describe what the agent observes.

Elements of Reinforcement Learning:
Except for the agent and the environment, we have four sub-elements of reinforcement learning system:

- Policy: It defines the learning agent's way of behaving at a given time.
- Reward function: It defines the goal in reinforcement learning problem.
- Value function: It specifies what is good in the long run.

Exploration Vs. Exploitation

Let's imagine a situation where our innocent little agent performs a certain set of actions and finally reaches the state of highest reward. Once it does this, it will keep on repeating these actions. There's a high probability that there also exists another set of actions which is more efficient and also ultimately lead to the state of highest reward but because our agent is obsessed with repeating it's actions, there's a high chance that it will never discover this alternate path!

So in such a scenario, our agent misses out the optimum set of actions just because it didn't explore it's surroundings. And we don't want that to happen!

To incorporate this factor of exploration in our agent, we instruct it to randomly perform actions instead of the optimal one a fraction of the time. This is called the $\epsilon-$ Greedy Approach. The randomness is denoted by the ϵ. Initially, the value of ϵ is high as the agent doesn't really know what to do. A high randomness in the actions chosen initially ensures that the agent explores its available options more often. So it won't miss out anything good. Then as time progresses, the value of ϵ is decayed. This ensures that our agent first explores and ultimately chooses the optimal set of actions and ends up learning the optimal policy!

Implementation:

Neural network:

I have built a three layered neural network with two hidden layers using the keras framework. The activation function of the hidden layers is Relu and the activation function of the output layer is linear. The number of hidden nodes are 128.

Role: Neural network acts as the brain of the agent. When the agent begins to exploit the neural network tell the agent what to do next based on past experience.

The implementation is as follows,

```
model = Sequential()
# Adding the input layer and the first hidden layer
model.add(Dense(units = 128, activation = 'relu', input_dim = 4))
# Adding the second hidden layer
 model.add(Dense(output_dim = 128, activation = 'relu'))
# Adding the output layer
model.add(Dense(units = 4, activation = 'linear'))
```

Exponential-decay formula for epsilon:

Role: Reduce the value of the epsilon during each episode.

Implemented the exponential decay formula for epsilon as,

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

Q-Function:

Role: Used to determine the reward based on the action of agent
Implemented the Q function using the following code,

```
if st_next is None:
    t[act] = rew
else:
    t[act] = rew + self.gamma * np.amax(q_vals_next)
```

Key hyperparameters:

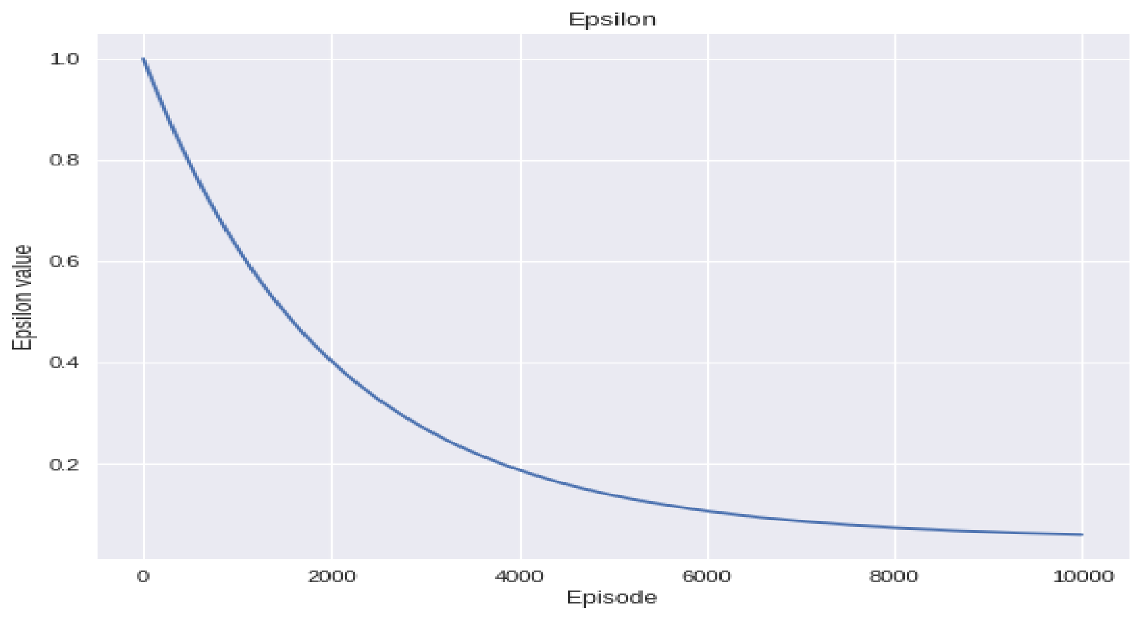
1. Min epsilon
2. Max epsilon
3. Gamma
4. Lamda
5. Number of episodes

Hyperparameter tuning and observed result graph:

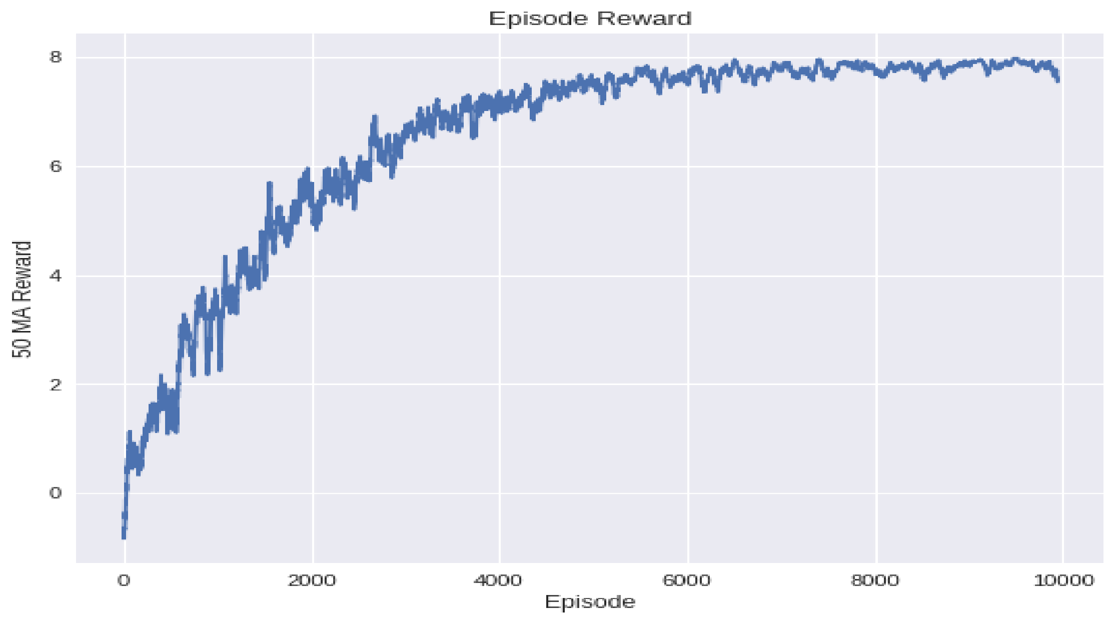
Case1:

gamma = 0.5; MAX_EPSILON = 1; MIN_EPSILON = 0.05; LAMBDA = 0.00005; num_episodes = 10000

```
Time Elapsed: 423.02s
Episode Reward Rolling Mean: 6.4726048362412
Epsilon decay:
```



```
Rewards:
```

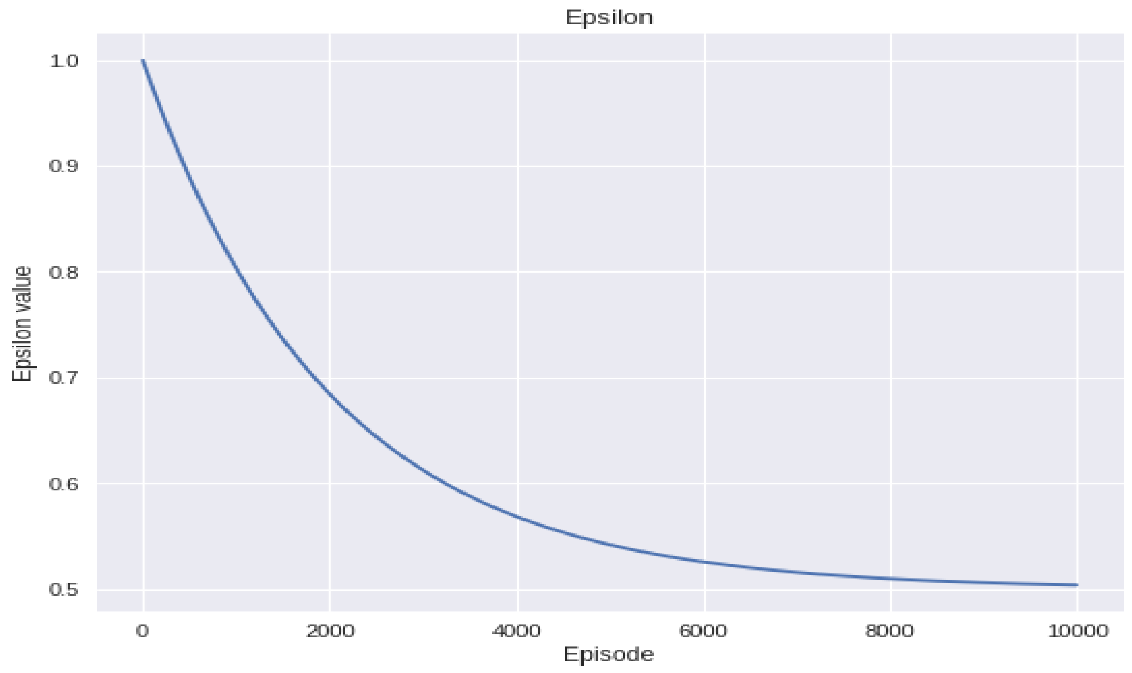


The Rewards becomes similar as from around 4000. This shows the agent took around 3000-4000 episodes to learn.

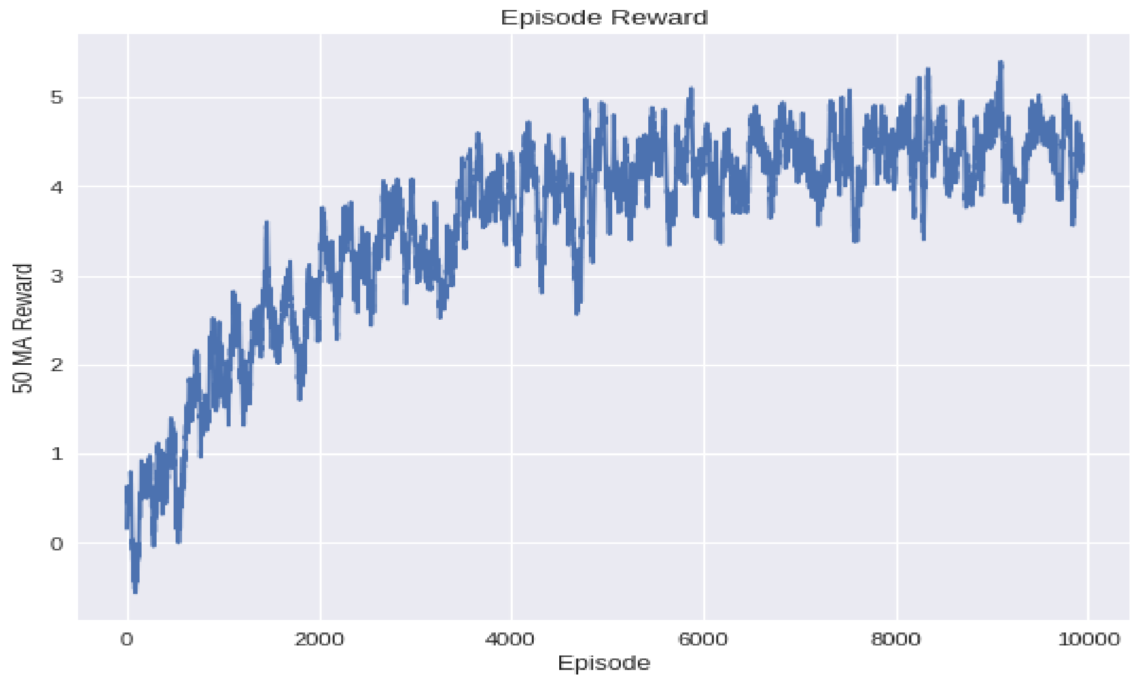
Case2:

gamma = 0.5; MAX_EPSILON = 1; MIN_EPSILON = 0.5; LAMBDA = 0.00005; num_episodes = 10000

```
Time Elapsed: 455.45s
Episode Reward Rolling Mean: 3.5528007346189163
Epsilon decay:
```



```
Rewards:
```

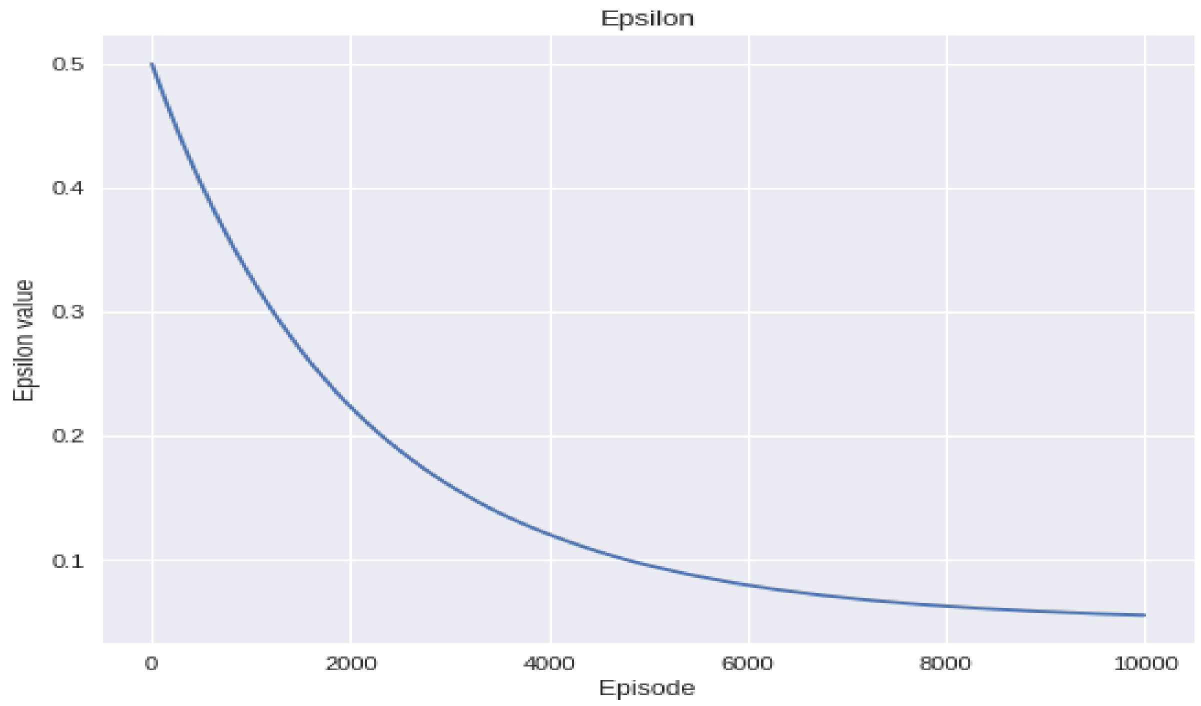


The Rewards for each episode fluctuates more and the max reward is around 5.

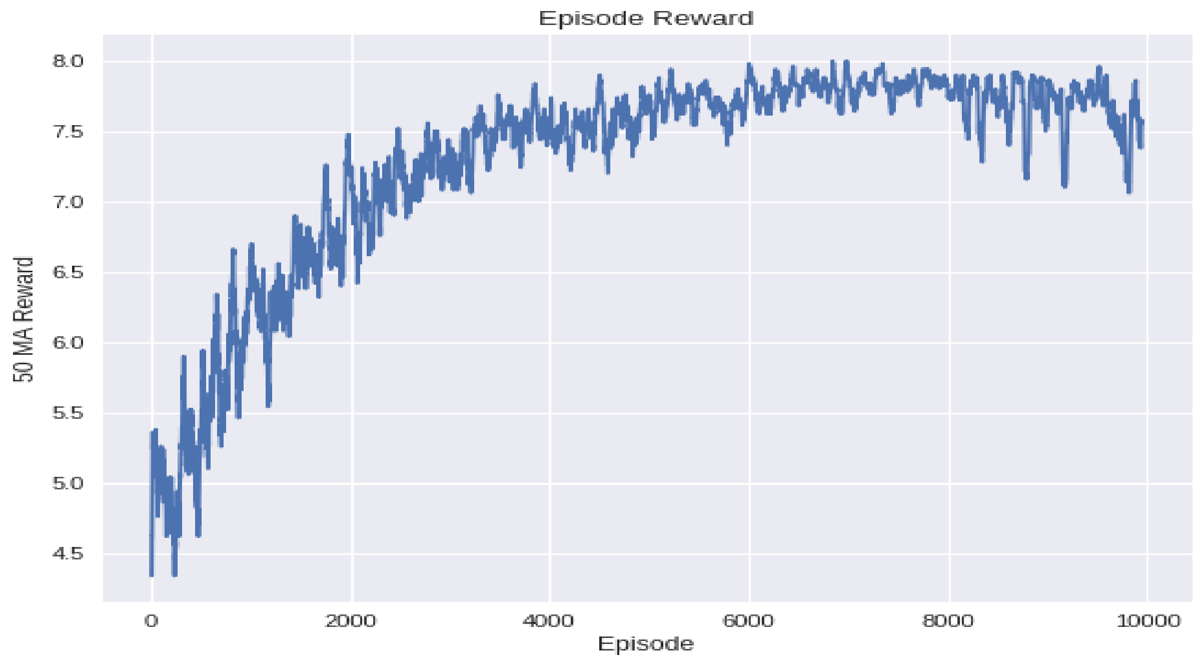
UBIT: sassikes
Person no: 50291558

Case3:
gamma = 0.5; MAX_EPSILON = .55; MIN_EPSILON = 0.05; LAMBDA = 0.00005; num_episodes = 10000

```
Time Elapsed: 446.06s
Episode Reward Rolling Mean: 7.264360779512295
Epsilon decay:
```



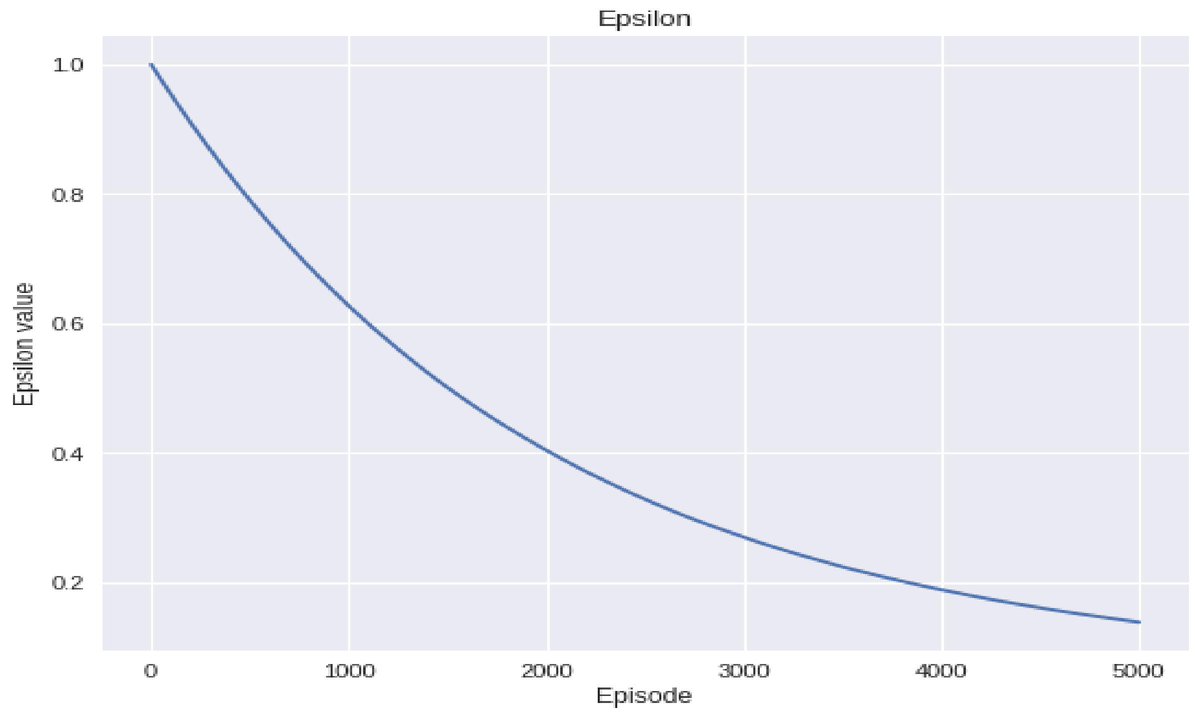
```
Rewards:
```



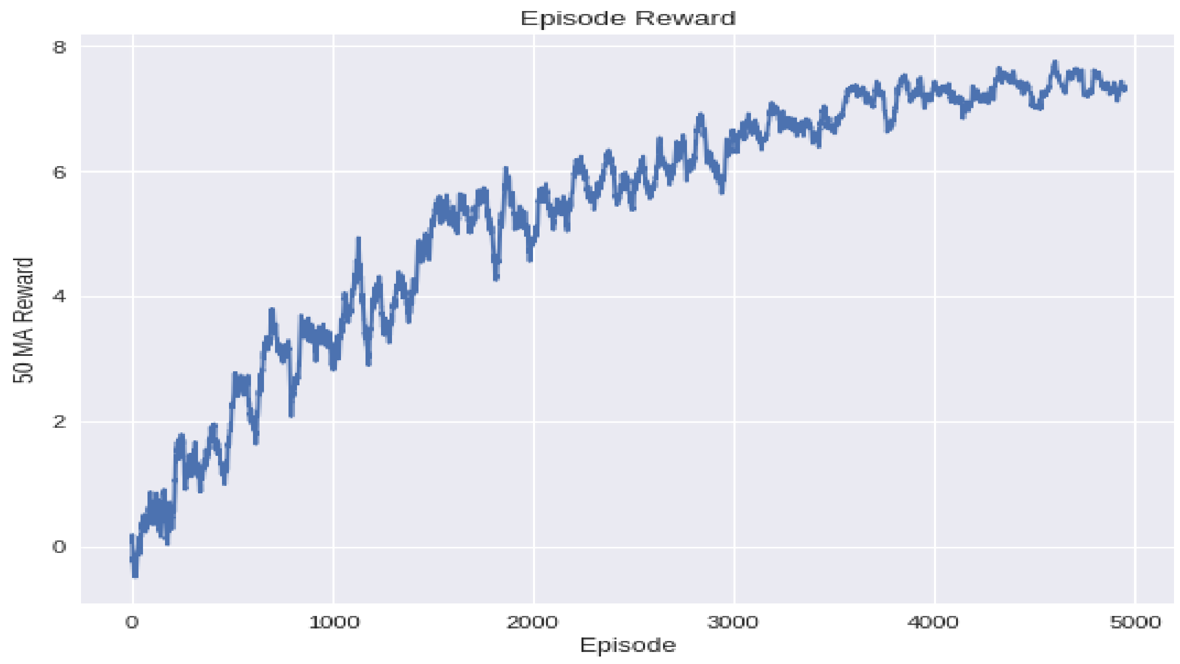
UBIT: sassikes
Person no: 50291558

Case4:
gamma = 0.5; MAX_EPSILON = 1; MIN_EPSILON = 0.05; LAMBDA = 0.00005; num_episodes = 5000

```
Time Elapsed: 241.06s
Episode Reward Rolling Mean: 5.178504478233702
Epsilon decay:
```



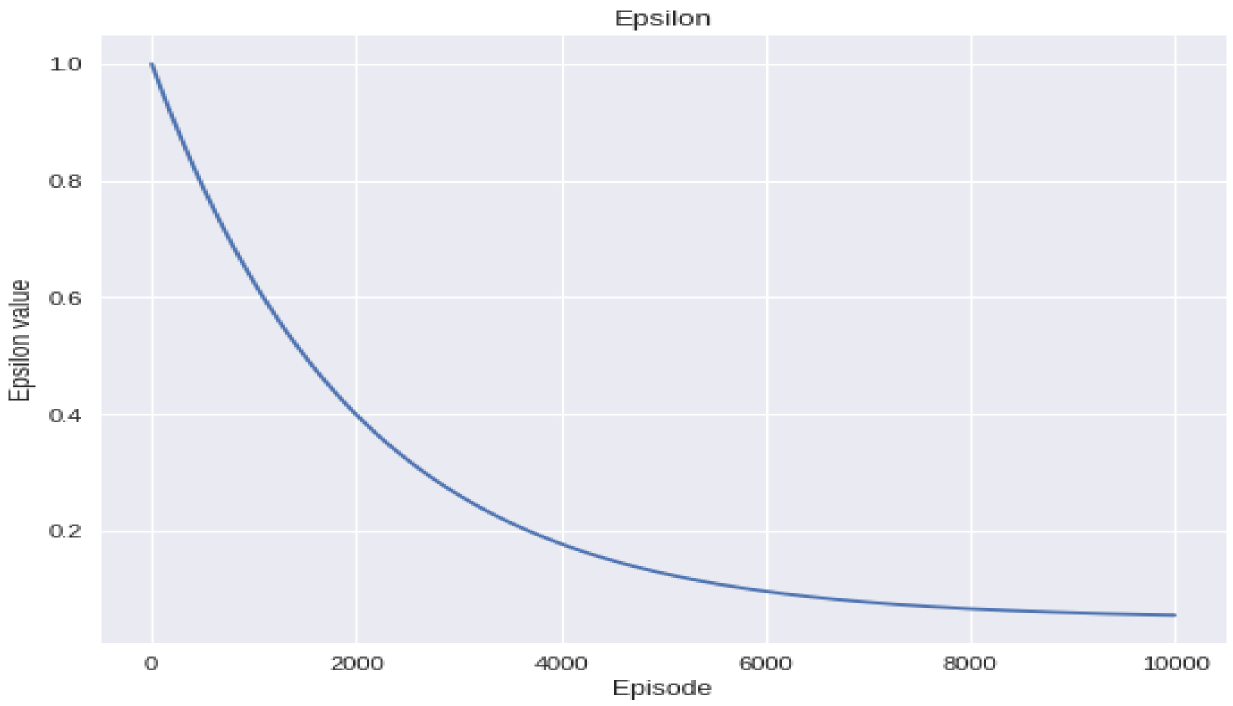
Rewards:



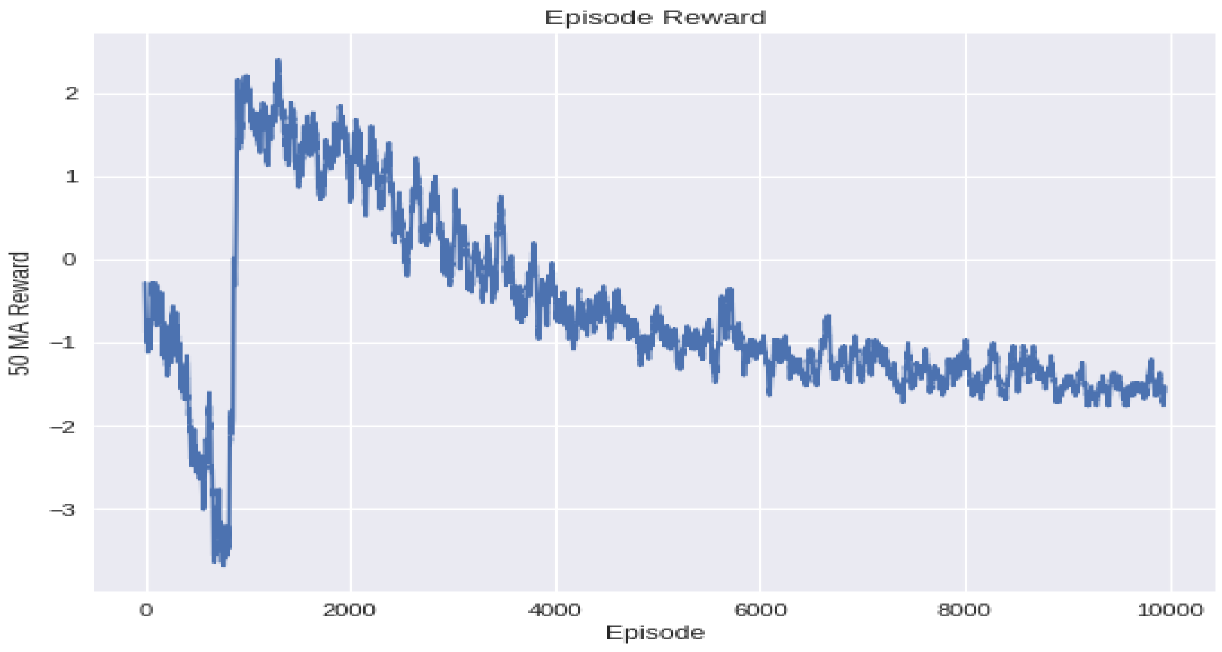
Case5:

gamma = 1; MAX_EPSILON = 1; MIN_EPSILON = 0.05; LAMBDA = 0.00005; num_episodes = 10000

```
Time Elapsed: 454.81s
Episode Reward Rolling Mean: -1.053717901011479
Epsilon decay:
```



```
Rewards:
```



Since, the gamma value is set high, the reward mean has become negative. So, gamma value should be reduced. The agent begun to exploit before gaining enough experience by exploring.

UBIT: sassikes
Person no: 50291558

Obseravtions:

Case	gamma	max epsilon	min epsilon	lambda	#episodes	Reached goal	Mean Reward
case 1	0.5	1	0.05	0.00005	10000	Yes	6.47
case 2	0.5	1	0.5	0.00005	10000	Yes	3.55
case 3	0.5	0.5	0.05	0.00005	10000	Yes	7.26
case 4	0.5	1	0.05	0.00005	5000	Yes	5.17
case 5	1	1	0.05	0.00005	10000	No	-1.05

Role of Neural Network:

Neural network acts as the brain of the agent. When the agent begins to exploit the neural network tell the agent what to do next based on past experience.

Writing Task:

Question 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.

Answer:

If the agent always chooses to maximize the Q-value, then it will not explore enough to find the best possible action from each of the state and begin to exploit using the past experience.

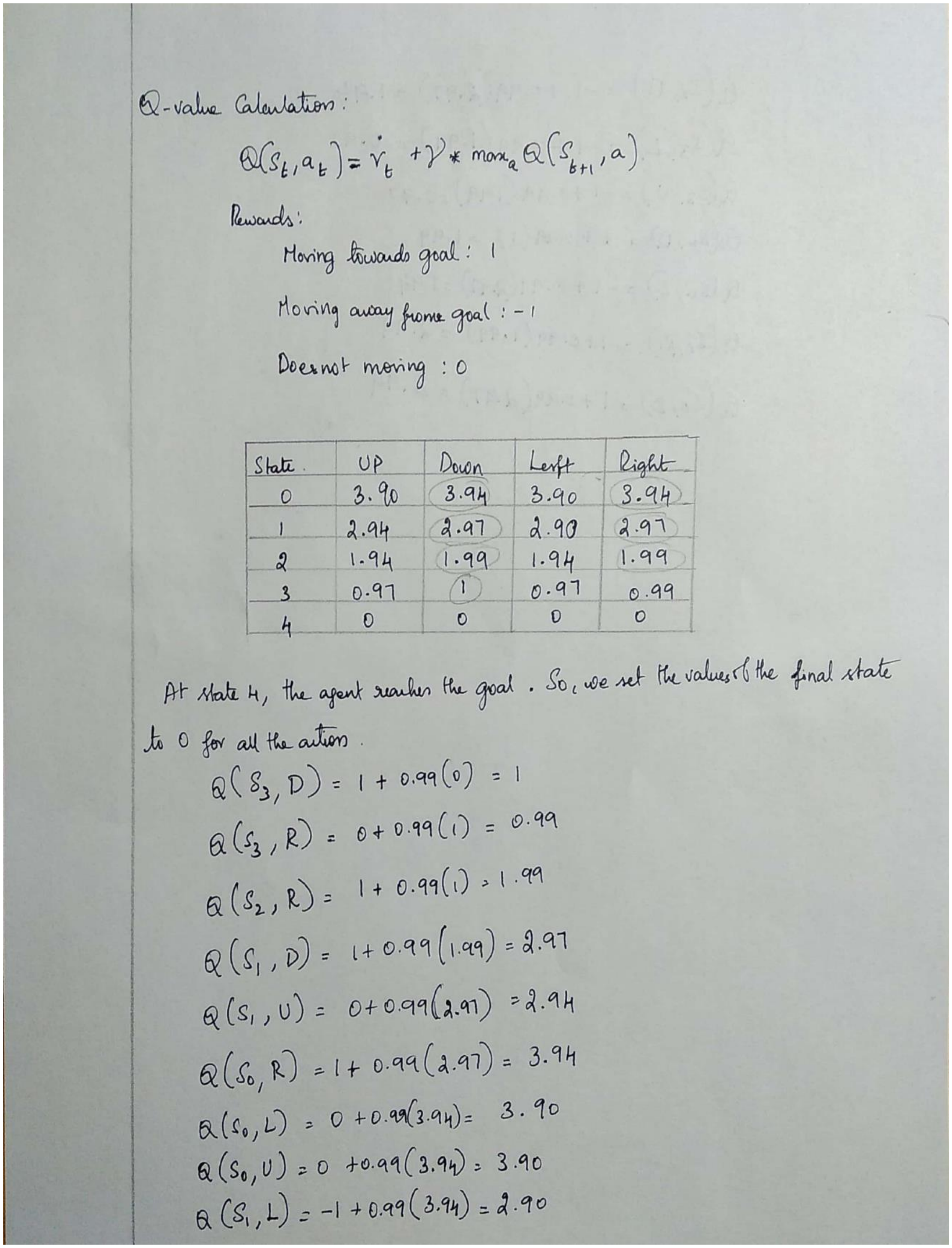
As the agent will explore when the eplison value greater than the random value, to make the agent to explore more we can,

1. Set high initial values of epsilon
2. Decrease the decay rate of the epsilon

Question 2:

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

UBIT: sassikes
Person no: 50291558



Q-value Calculation:

$$Q(S_t, a_t) = r_t + \gamma * \max_a Q(S_{t+1}, a)$$

Rewards:

Moving towards goal : 1

Moving away frome goal : -1

Doesnot moving : 0

State	UP	Down	Left	Right
0	3.90	3.94	3.90	3.94
1	2.94	2.97	2.90	2.97
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

At state 4, the agent reaches the goal. So, we set the values of the final state to 0 for all the actions.

$Q(S_3, D) = 1 + 0.99(0) = 1$

$Q(S_3, R) = 0 + 0.99(1) = 0.99$

$Q(S_2, R) = 1 + 0.99(1) = 1.99$

$Q(S_1, D) = 1 + 0.99(1.99) = 2.97$

$Q(S_1, U) = 0 + 0.99(2.97) = 2.94$

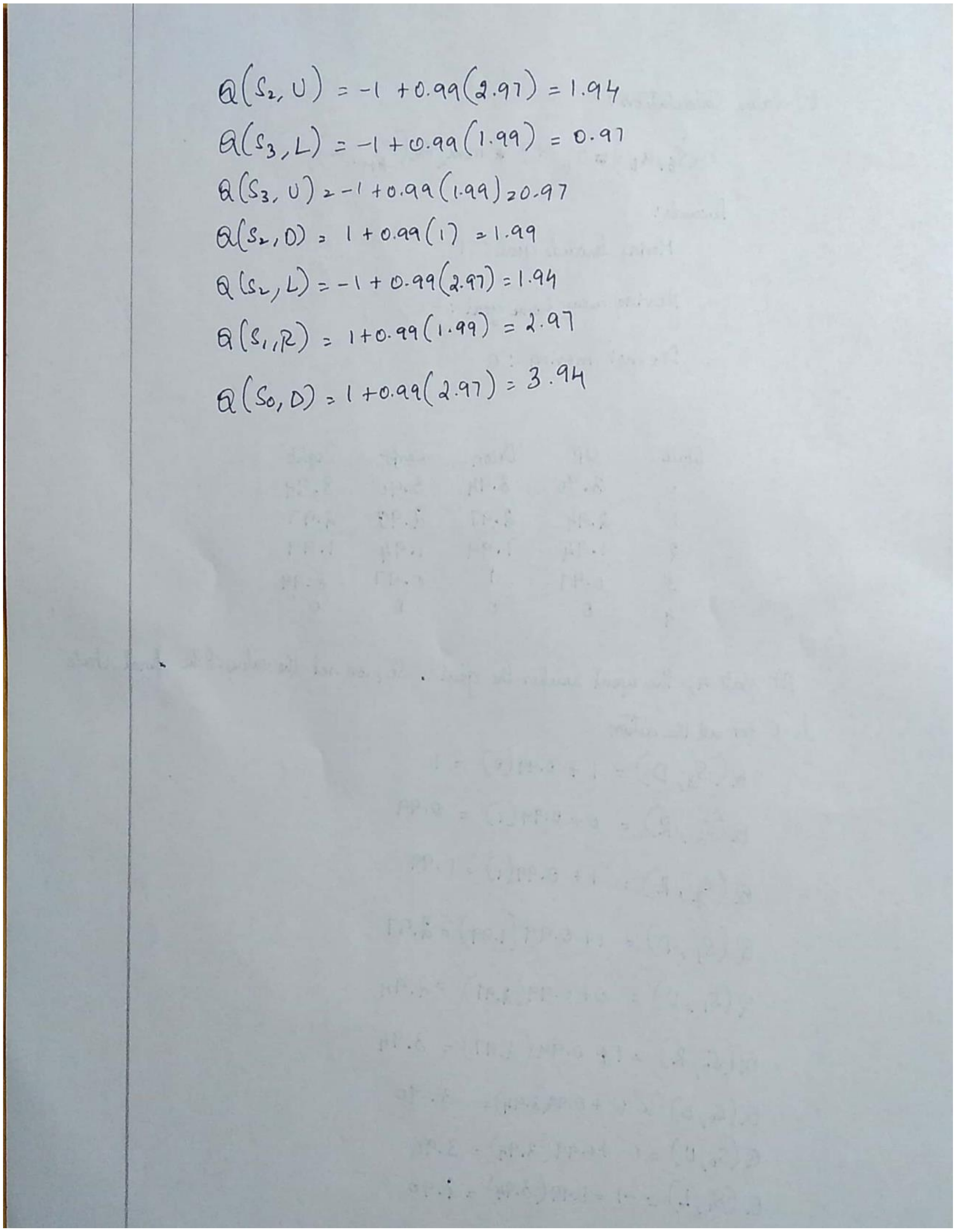
$Q(S_0, R) = 1 + 0.99(2.97) = 3.94$

$Q(S_0, L) = 0 + 0.99(3.94) = 3.90$

$Q(S_0, U) = 0 + 0.99(3.94) = 3.90$

$Q(S_1, L) = -1 + 0.99(3.94) = 2.90$

UBIT: sassikes
Person no: 50291558



$Q(S_2, U) = -1 + 0.99(2.97) = 1.94$

$Q(S_3, L) = -1 + 0.99(1.99) = 0.97$

$Q(S_3, U) = -1 + 0.99(1.99) = 0.97$

$Q(S_2, D) = 1 + 0.99(1) = 1.99$

$Q(S_2, L) = -1 + 0.99(2.97) = 1.94$

$Q(S_1, R) = 1 + 0.99(1.99) = 2.97$

$Q(S_0, D) = 1 + 0.99(2.97) = 3.94$

UBIT: sassikes
Person no: 50291558

References:

https://skymind.ai/wiki/deep-reinforcement-learning

https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

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

https://www.quora.com/