Project 4: Tom and Jerry in Reinforcement Learning

Srivenkata Krishnan Sesharamanujam
CSE574 Introduction to Machine Learning
UB Person no: 50288730
srivenka@buffalo.edu

1 Introduction

The following report talks about the process of teaching an agent to navigate in a grid based environment. In this project,the task for Tom (an agent) is to find the shortest path to Jerry (a goal). Most of the code has been given as a head start, now the job is to fill in the remaining parts. The rest of the report is structured as follows: Section 2 answers the two questions as a part of the writing task in the project. Section 3 explains the parts of the code which has been implemented, their role in determining the agent, how it influences in the training process and the time taken for the agent to learn. Section 4 talks about how change of epsilon max/min, number of episodes, gamma, etc. influences the total time of training and the mean reward and we finally conclude in section 5.

2 Writing Tasks

2.1 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

If the agent chooses the action that maximizes the Q-value it gets stuck in non-optimal policies because it does not explore enough to find the best action from each state.

Two ways to force the agent to explore are

- Pick random actions occasionally.
- Setting the initial values high, so that unexplored regions look good.

2.2 Question 2: Calculate Q-value for the given states and provide all the calculation steps

The values are calculated from the last state.

The Q-table after first iteration is given by,

State	Up	down	left	Right
0	3.9	?	3.9	3.94
1	2.94	2.97	?	?
2	?	?	?	1.99
3	?	1	?	0.99
4	0	0	0	0

The final Q- table is given by,

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

3 Implementation

3.1 Parts to be implemented

3.1.1 Build a 3-layer neural network using Keras library

How the neural network has to be designed as given in the question ?

- Build a three-layer neural network with two hidden layers
- 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'
- Input dimensions 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)

This is implemented using the following code by using the keras library as shown in 1

Figure 1: Implementation of Neural Network

```
### START CODE HERE ### (≈ 3 lines of code)

model.add(Dense(128,input_dim = self.state_dim, activation='relu'))
model.add(Dense(128,activation = 'relu'))
model.add(Dense(4,input_dim = self.action_dim,activation = 'linear'))

#print("Model Intiated")
```

The first part of the implementation is complete with respect to building a 3 layer neural network.

3.1.2 Implement exponential-decay formula for epsilon

The exponential decay formula is given by,

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda S} \tag{1}$$

where,

$\epsilon_{max}, \epsilon_{min} \in [0, 1]$
λ Hyper parameter for epsilon
S , total no of steps

The following is implemented in the given program using the parameters used in the equation and code snippet is shown in figure 2

3.1.3 Implement Q - Function

The Q function is defined as follows,

Figure 2: Implementation of Exponential decay formula for epsilon

```
def observe(self, sample):  # in (s, a, r, s_) format
  """The agent observes an event.
  We pass a sample (state, action, reward, next state) to be stored in memory.
  We then increment the step count and adjust epsilon accordingly.
  """
  self.memory.add(sample)

  # slowly decrease Epsilon based on our eperience
  self.steps += 1

  ### START CODE HERE ### (≈ 1 line of code)
  self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * math.exp(-self.lamb * self.steps)
  ### END CODE HERE ###

  self.epsilons.append(self.epsilon)
```

$$Q_t = \begin{cases} r_t & If episode terminates at t+1 \\ r_t + \gamma * max(Q(s_t, a_t; \theta)) & otherwise \end{cases}$$

The following formula is adapted to the variables in the program and implemented. The snippet of the code is given by 3

Figure 3: Implementation of Q - Function

```
### START CODE HERE ### (≈ 4 line of code)
if st_next is None:
  t[act] = rew
else:
  t[act] = rew + self.gamma * np.amax(q_vals_next[i])


### END CODE HERE ###
```

3.2 How does these implementations affect the agent ?

3.2.1 Neural Network

Neural networks are the agent that learns to map state-action pairs to rewards. Like all neural networks, they use coefficients to approximate the function relating inputs to outputs, and their learning consists to finding the right coefficients, or weights, by iteratively adjusting those weights along gradients that promise less error.

In reinforcement learning, convolutional networks can be used to recognize an agents state; e.g. the screen that Mario is on, or the terrain before a drone. That is, they perform their typical task of image recognition.

But convolutional networks derive different interpretations from images in reinforcement learning than in supervised learning. In supervised learning, the network applies a label to an image; that is, it matches names to pixels.

At the beginning of reinforcement learning, the neural network coefficients may be initialized stochastically, or randomly. Using feedback from the environment, the neural net can use the difference between its expected reward and the ground-truth reward to adjust its weights and improve its interpretation of state-action pairs.

This feedback loop is analogous to the back propagation of error in supervised learning. However, supervised learning begins with knowledge of the ground-truth labels the neural network is trying to predict. Its goal is to create a model that maps different images to their respective names.

Reinforcement learning relies on the environment to send it a scalar number in response to each new action. The rewards returned by the environment can be varied, delayed or affected by unknown variables, introducing noise to the feedback loop.

3.2.2 Exponential Decay Formula

One of the most important issues for the temporal difference learning algorithms is maintaining a balance between exploration and exploitation. That is, the agent must sometimes choose actions that it believes to be sub optimal in order to find out whether they might actually be good. This is particularly true in problems which change over time (which is true of most behavioral experiments), since actions that used to be good might become bad, and vice-versa. The theorems proving that temporal difference methods work usually require much experimentation: all actions must be repeatedly tried in all states. In practice, it is common to choose policies that always embody exploration (such as choosing a random action some small fraction of the time, but otherwise the action currently believed

3.2.3 Q -Formula

Q-learning is a value-based Reinforcement Learning algorithm that is used to find the optimal action-selection policy using a q function.It evaluates which action to take based on an action-value function that determines the value of being in a certain state and taking a certain action at that state. The Goal is to maximize the value function Q (expected future reward given a state and action). Maximizing the expected reward by selecting the best of all possible actions.

4 Results for different Epsilons, no of episodes, gamma

The following section list the different values of epsilons, no of episodes , gamma affect the performance of the agent

4.1 Ideal Values - $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$

The graphs are given by 4, 5

Figure 4: Epsilon vs Epsilon Value Ideal Values - $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$

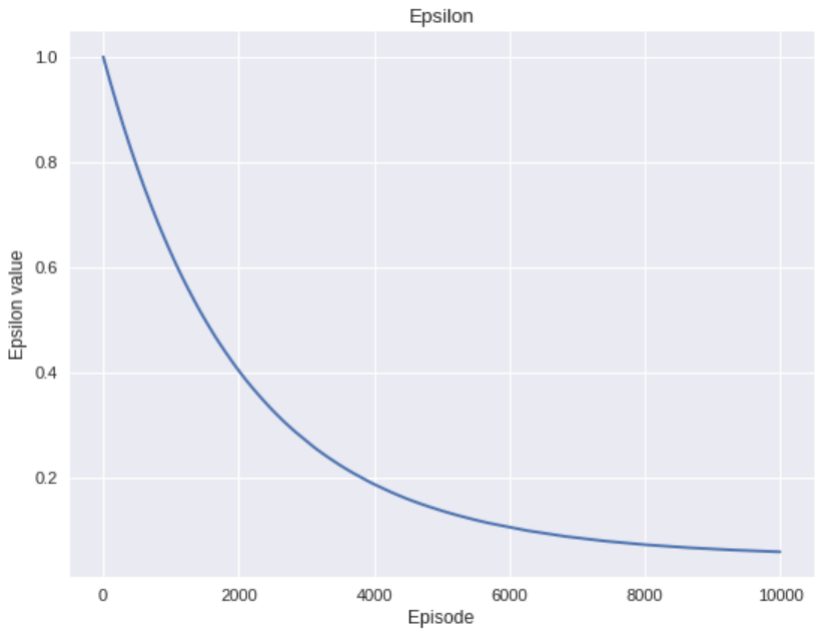
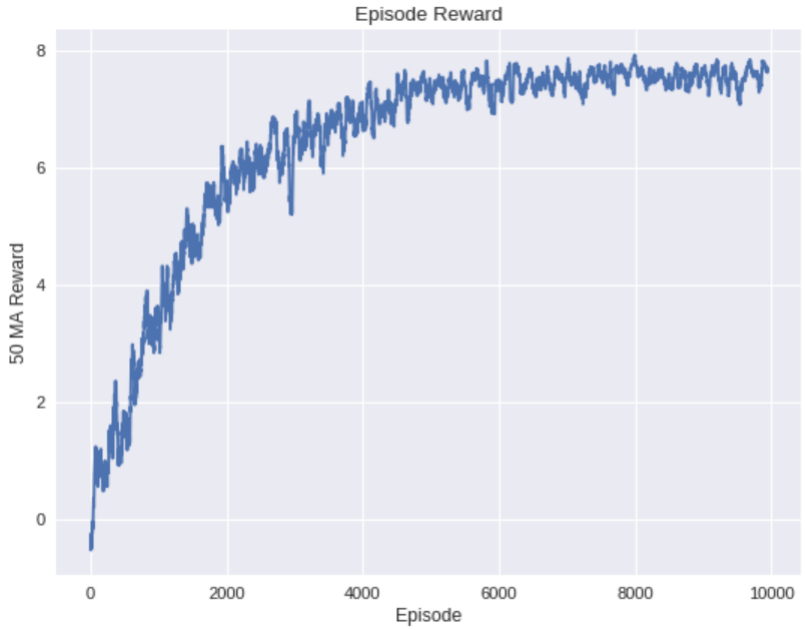


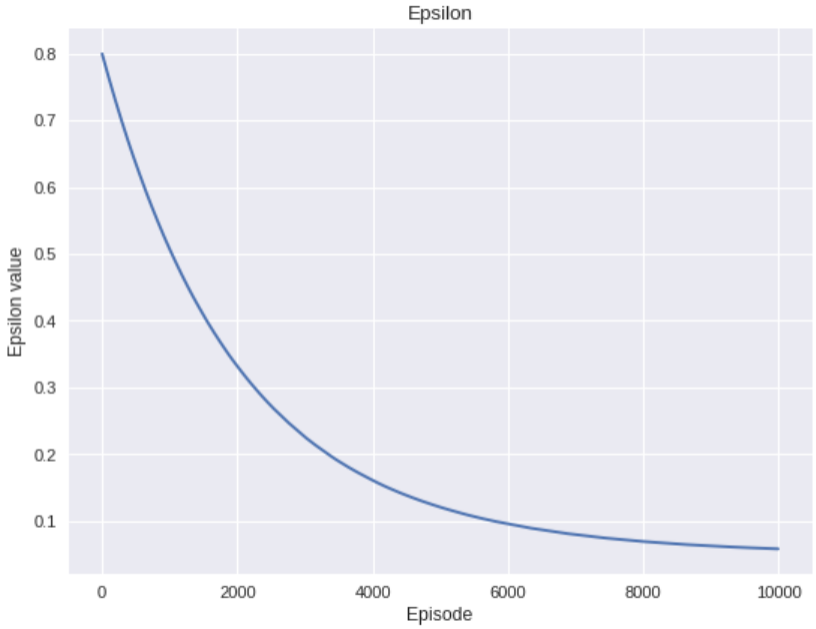
Figure 5: Episode vs Epsilon Reward Ideal Values - $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$



4.2 $\epsilon_{max} = 0.8, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$

The graphs are given by 6, 7

Figure 6: Epsilon vs Epsilon Value $\epsilon_{max} = 0.8, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$



4.3 $\epsilon_{max} = 1, \epsilon_{min} = 0.2, \gamma = 0.99, \lambda = 0.001$

The graphs are given by 8, 9

Figure 7: Episode vs Epsilon Reward $\epsilon_{max} = 0.8, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.001$

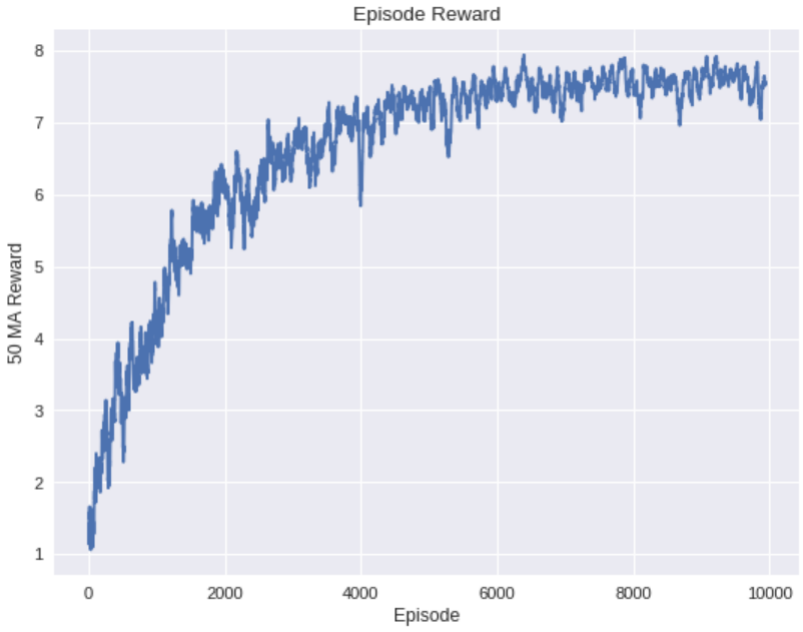
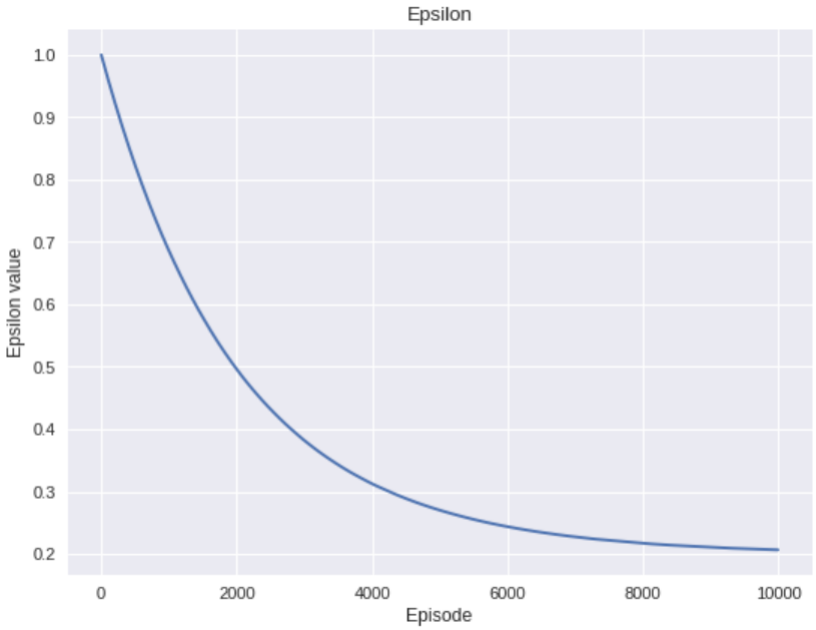


Figure 8: Epsilon vs Epsilon Value $\epsilon_{max} = 1, \epsilon_{min} = 0.2, \gamma = 0.99, \lambda = 0.001$



4.4 $\epsilon_{max} = 0.6, \epsilon_{min} = 0.4, \gamma = 0.99, \lambda = 0.001$

The graphs are given by 10, 11

4.5 $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.8, \lambda = 0.001$

The graphs are given by 12, 13

Figure 9: Episode vs Epsilon Reward $\epsilon_{max} = 1, \epsilon_{min} = 0.2, \gamma = 0.99, \lambda = 0.001$

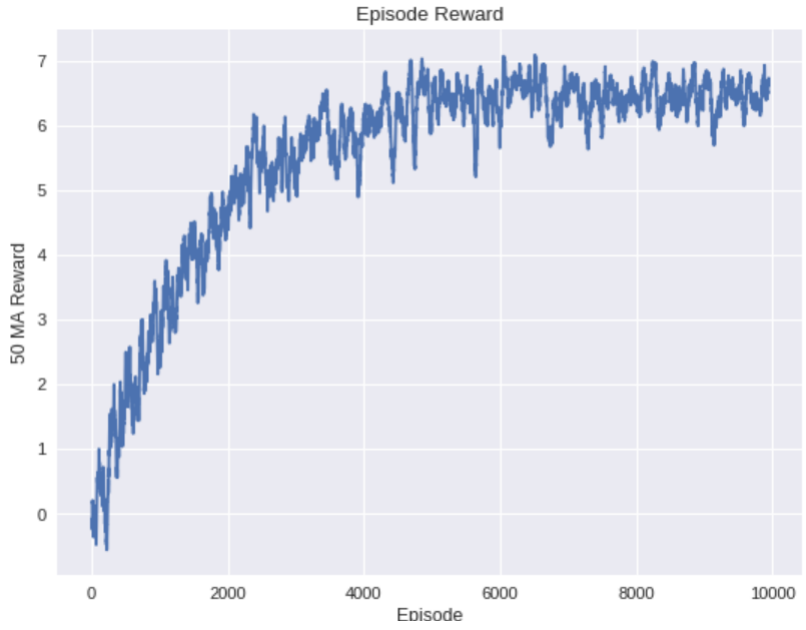
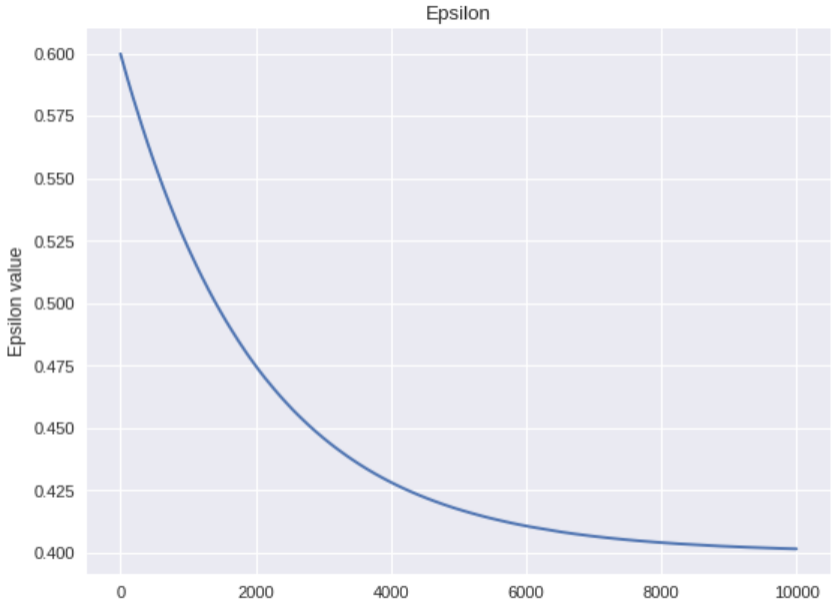


Figure 10: Epsilon vs Epsilon Value $\epsilon_{max} = 0.6, \epsilon_{min} = 0.4, \gamma = 0.99, \lambda = 0.001$



4.6 $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.2, \lambda = 0.001$

The graphs are given by 14, 15

4.7 $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.1$

The graphs are given by 16, 17

Figure 11: Episode vs Epsilon Reward $\epsilon_{max} = 0.6, \epsilon_{min} = 0.4, \gamma = 0.99, \lambda = 0.001$

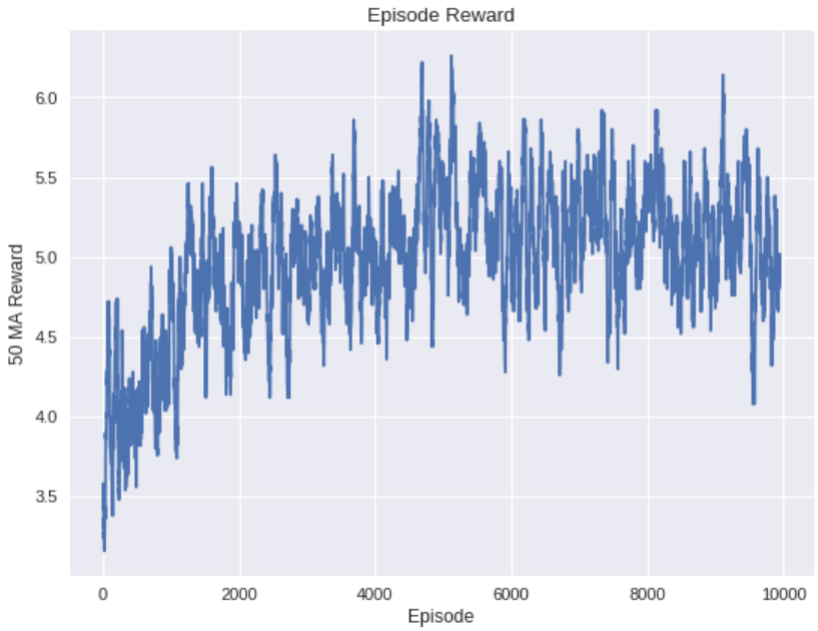
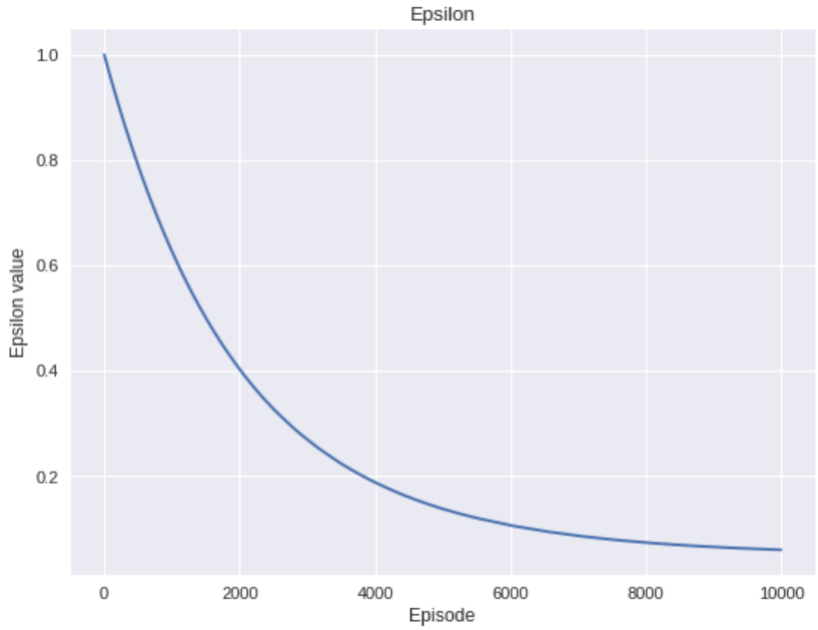


Figure 12: Epsilon vs Epsilon Value $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.8, \lambda = 0.001$



4.8 $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.9$

The graphs are given by 18, 19

5 Conclusion

From the following project, we came to understand how reinforcement learning works and factors which influence the agent for the output.

Figure 13: Episode vs Epsilon Reward $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.8, \lambda = 0.001$

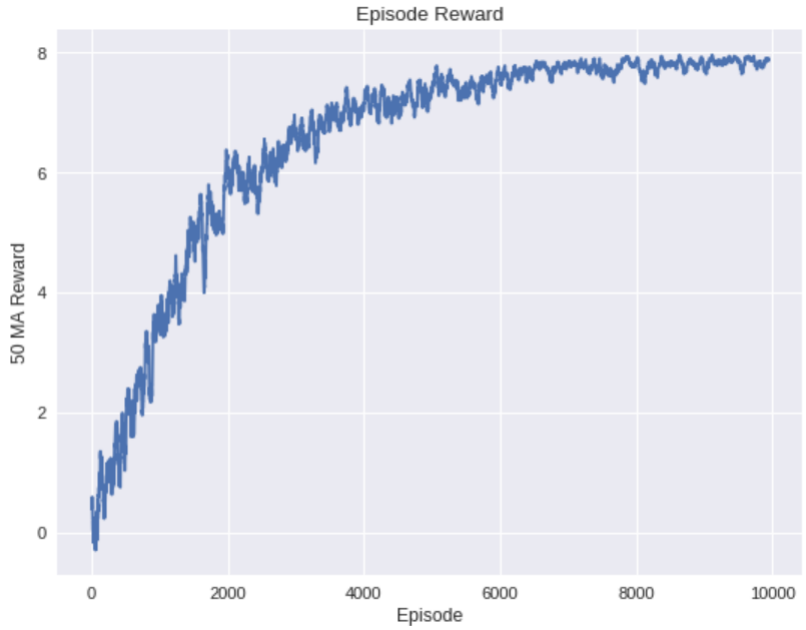


Figure 14: Epsilon vs Epsilon Value $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.2, \lambda = 0.001$

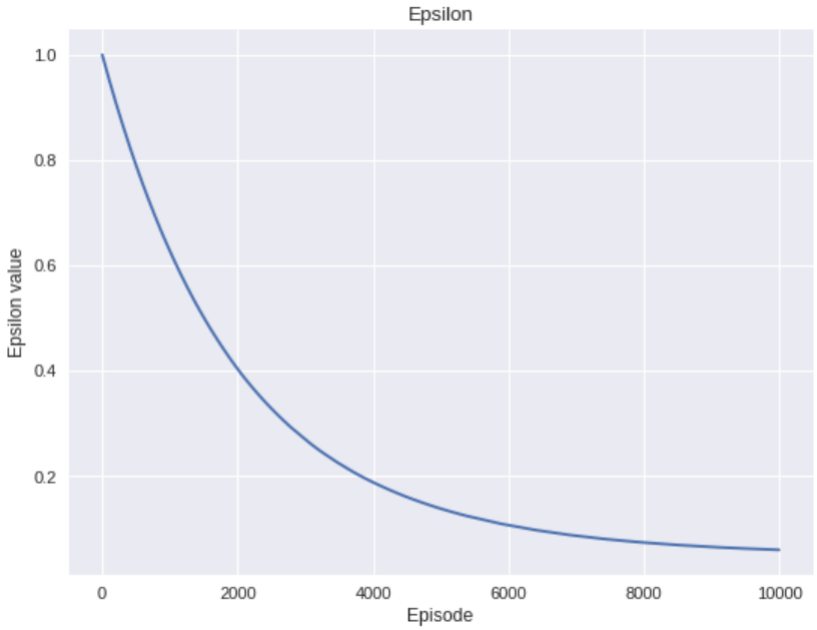


_Figure 15: Episode vs Epsilon Reward $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.2, \lambda = 0.001$

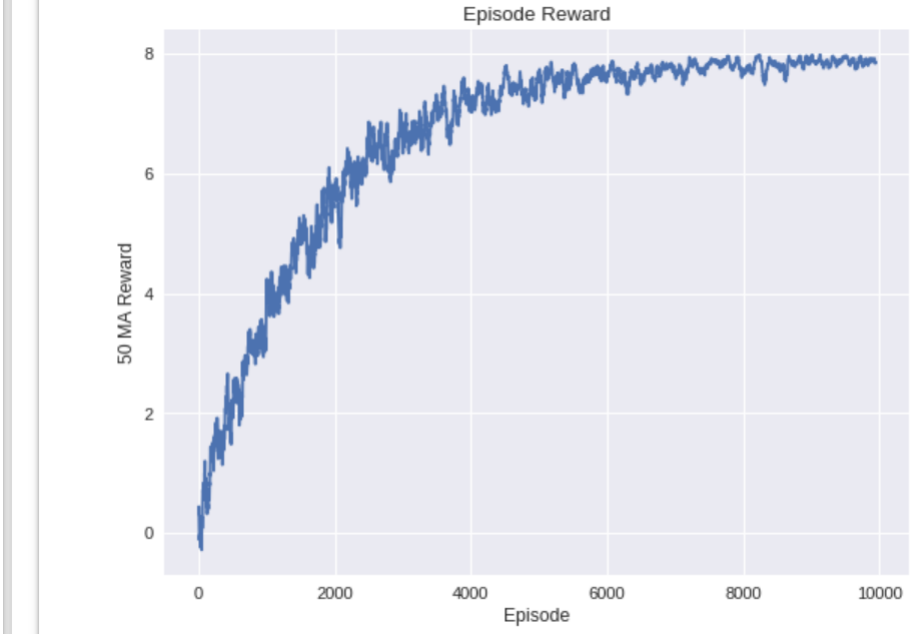


Figure 16: Epsilon vs Epsilon Value $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.1$

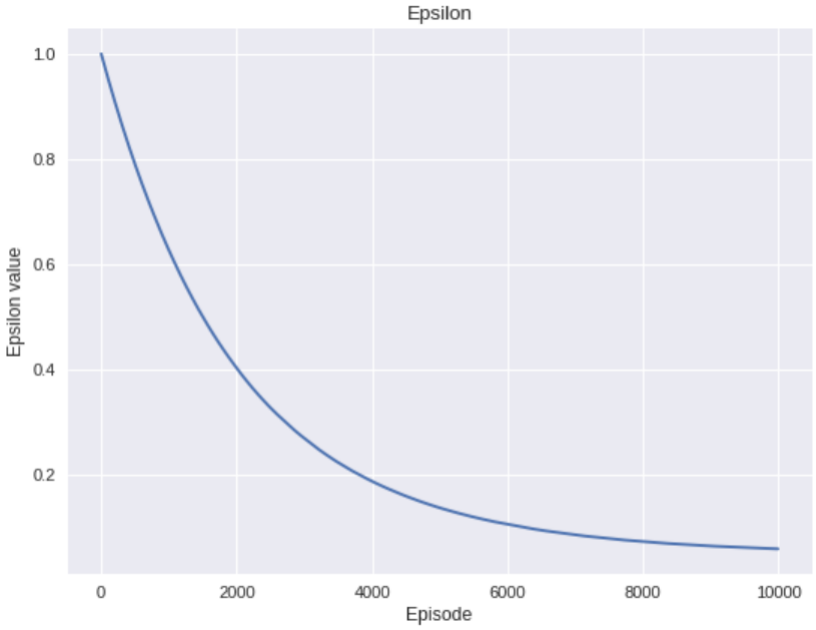


Figure 17: Episode vs Epsilon Reward $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.1$

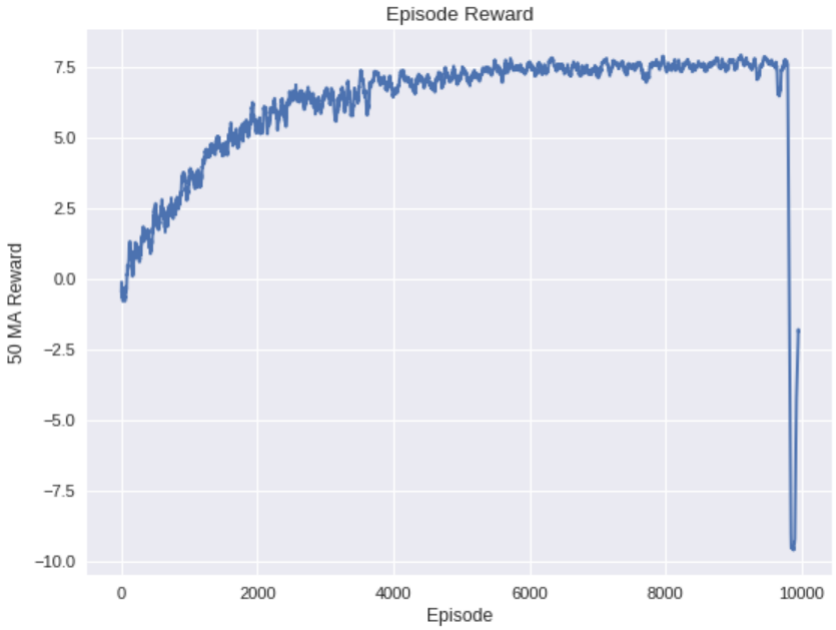


Figure 18: Epsilon vs Epsilon Value $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.9$

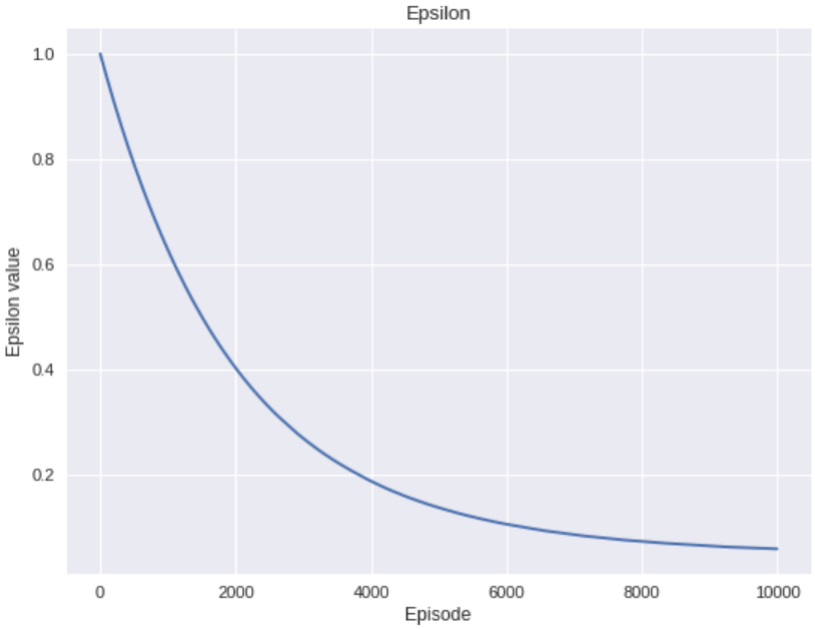


Figure 19: Episode vs Epsilon Reward $\epsilon_{max} = 1, \epsilon_{min} = 0.01, \gamma = 0.99, \lambda = 0.9$