CSE 574 Introduction to Machine Learning

Project 4 Report

The task was to teach the agent (Tom) to navigate in the grid-world environment. We have modeled Tom and Jerry cartoon, where Tom, a cat, is chasing Jerry, a mouse. In our modeled game the task for Tom is to find the shortest path to Jerry (a goal), given that the initial positions of Tom and Jerry are deterministic.

What is Reinforcement Learning ?

Reinforcement Learning is an aspect of Machine learning where an agent learns to behave in an environment, by performing certain actions and observing the rewards/results which it get from those actions. [1]

In reinforcement 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.

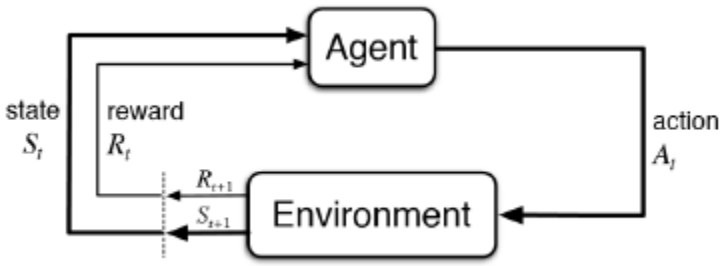


Figure 1: Markov Decision process

The above figure illustrates the Markov Decision process which works as the basis of reinforcement learning. 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.

The pipeline for our task of reinforcement learning is as follows:

1. Define Environment
2. Neural Network structure
3. Memory – used to store experiences of agent
4. Defining the agent who learns to navigate the environment
5. Running the game
   1) Initializing the environment
   2) Initialize the agent
   3) Algorithm

Questions related to Coding Part

1) ## What parts have you implemented?

Task 1: Build a 3-layer neural network.

- We implemented the brain of the agent using a 3 layer neural network using the Keras library. The 'brain' of the agent is where the model is created and held.
- Our three-layer neural network consists of two hidden layers.
- The activation function for the first and second hidden layers is 'relu', while we use ‘linear’ activation function for the final layer to produce real values
- Input dimensions for the first hidden layer are kept as size of the observation space (state_dim) and we keep the number of hidden nodes as 128.
- We keep the number of the output same as the size of the action space (action_dim).

- The DQN takes the input as a stack of six-tuple. It is passed through two hidden networks, and output a vector of Q-values for each action possible in the given state. We observe that in the beginning, the agent does really badly. But over time, it begins to associate states with best actions to do. It generally converges around 6000 episodes but can change with changes in the value of lambda.

```
### START CODE HERE ### (≈ 3 lines of code)
model.add(Dense(128, activation='relu', input_dim=(self.state_dim)))
model.add(Dense(128, activation='relu'))
model.add(Dense(self.action_dim, activation='linear'))
```

Task 2: Implement exponential-decay formula for epsilon.

We implemented the following formula for epsilon:

$$\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) * e^{-\lambda|S|},$$

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

The following code was used for implementing exponential decay formula:

```
### 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 ###
```

Task 3: Implement Q-function
We implemented the Q function using a simple if else logic.

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

The Q function was implemented as follows:

```
    ### 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])
```

2) What is their role in training the agent ?

We use the neural network i.e. the brain of the agent to reduce the randomness in getting the next state so that by reducing the randomness we can predict better mean reward. So we use neural network to train the agent so as to reduce the randomness and to predict a better mean reward.

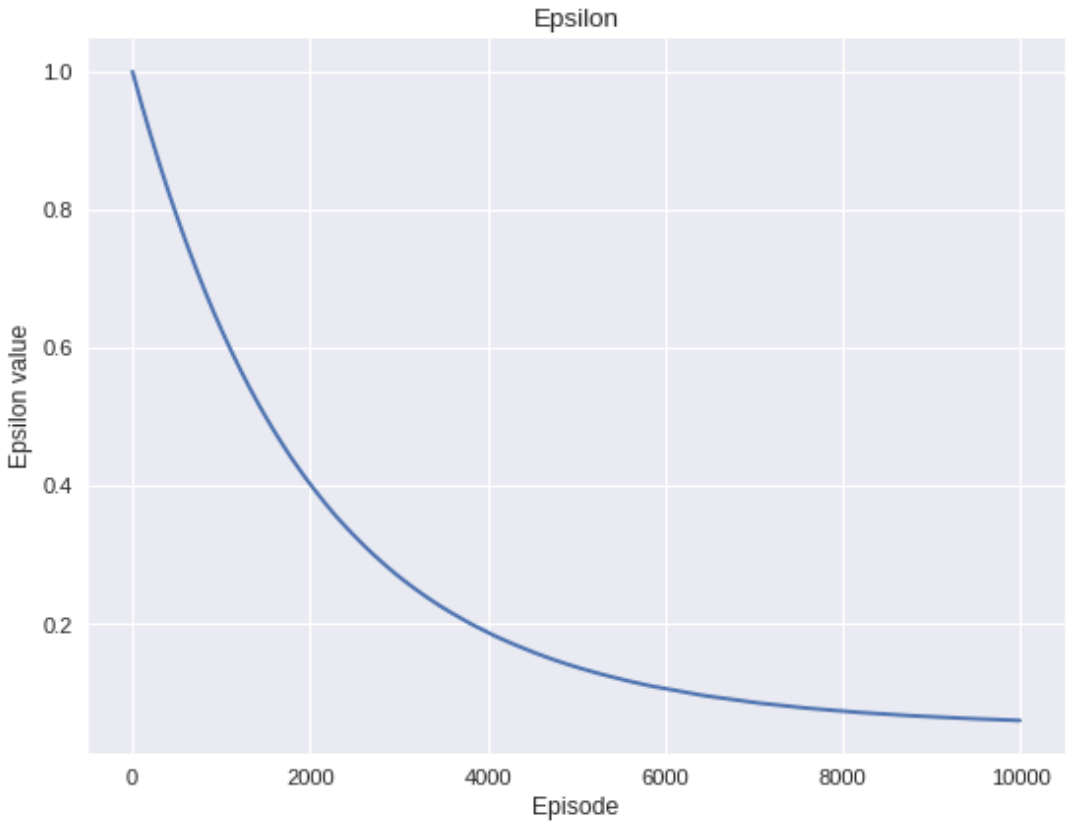
3) Can these snippets be improved and how it will influence the training the agent?

These snippets can be improved by tweaking hyperparameters such as min epsilon, max epsilon , gamma and lambda as well as number of episodes.
The training of the agent will be better if we tune the hyperparameters which is depicted in the results table below. f

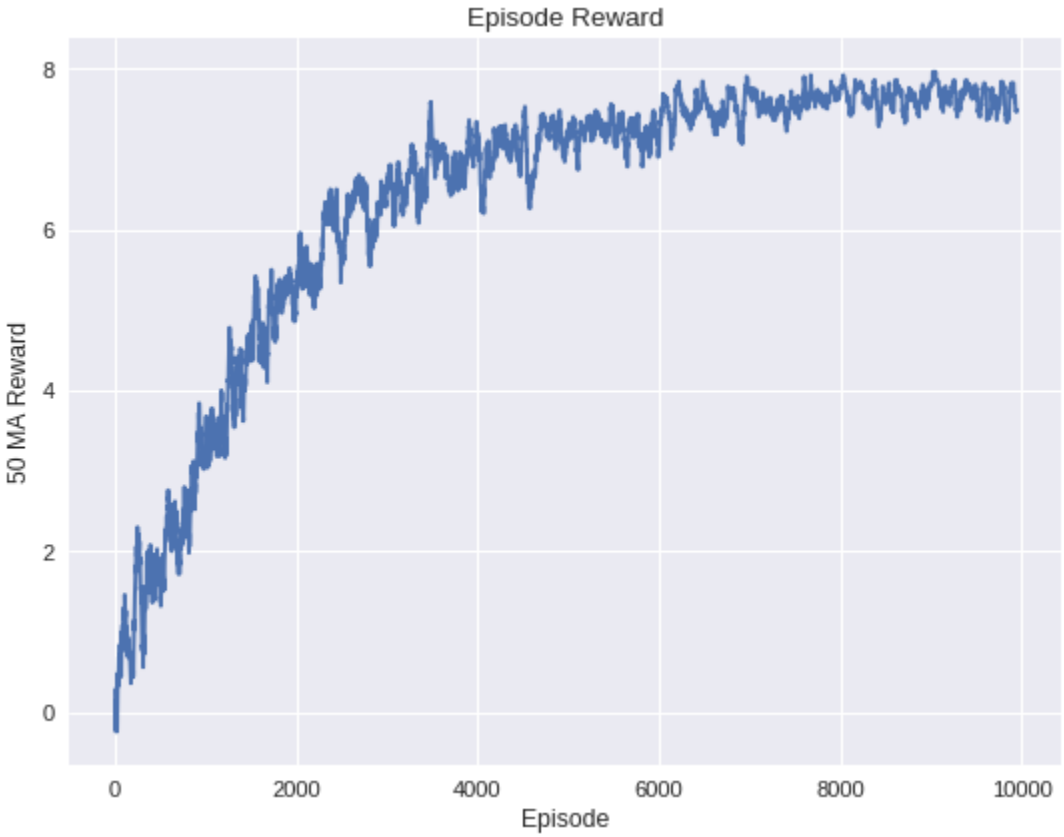
4) How quickly your agent were able to learn?

The agent was able to learn at about 6000 episodes but by tweaking the hyperparameters we can reduce the number of episodes to about 2000 episodes by tweaking the hyperparameters like gamma and lambda.
This is shown below in the results table. We have proved that the model can converge quicker by tweaking the model hyperparameters.

Results :



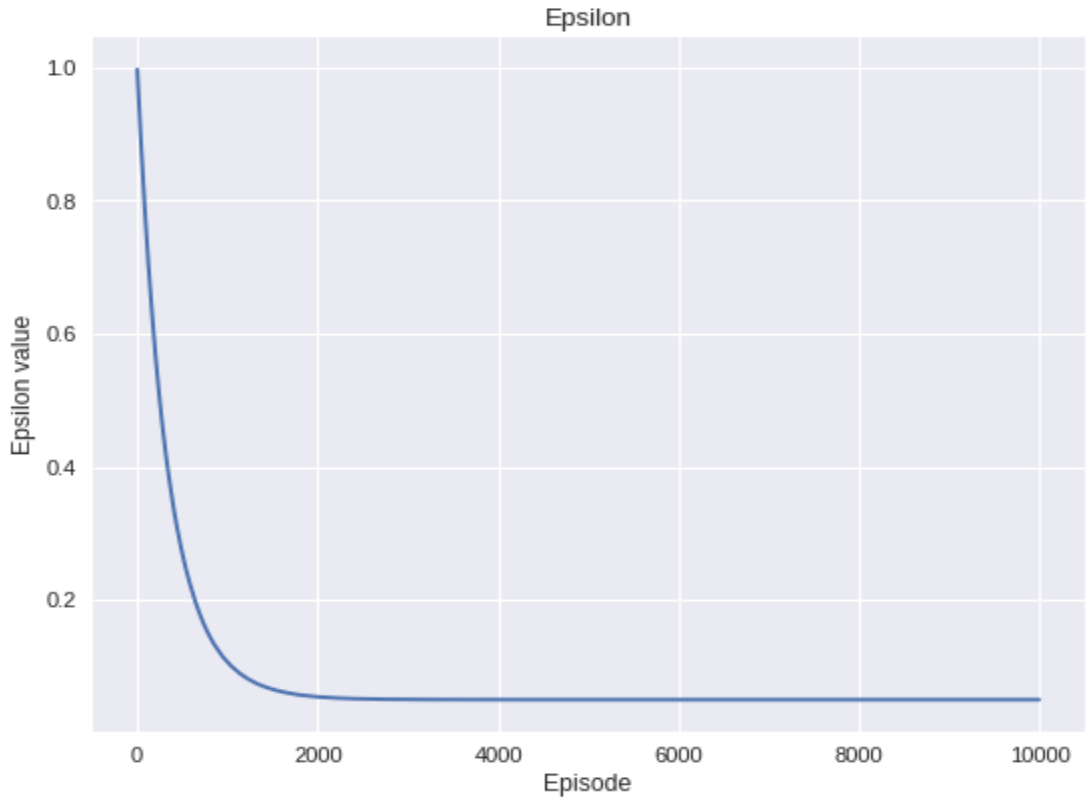
In the above and below graphs, we have kept the values as default as were provided by the teaching assistants. We see that the model converges at more than 6000 episodes and we can see that when we tweak the hyperparameters we see that the mode converges quickly at less than 2000 episodes and gives better mean reward.



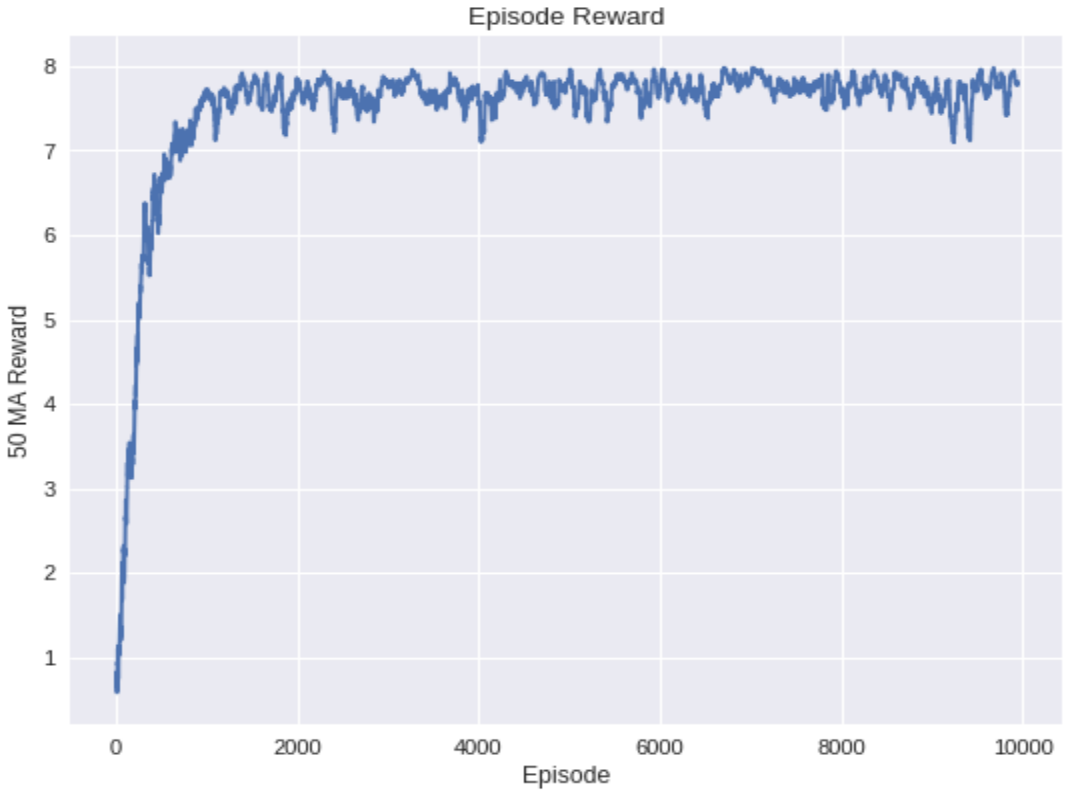
In the below table, we vary the value of lambda value while keeping all other values as same. In the below table we see that as we increase the value of lambda we find that the mean reward increases and the model converges quickly.

lambda	epsilon	Total time taken	Mean Reward
0.00001	0.407	866	3.14
0.00005	0.060	834	6.35
0.0001	0.050	814	6.72
0.0003	0.05	809	7.48

Gamma	Epsilon	Total time taken	Mean Reward
0.39	0.06081	884	6.51
0.59	0.0604	872	6.47
0.79	0.0605	875	6.42
0.89	0.0610	851	6.41



UBitName: abhave personNumber:50289049



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.

If the agent always chooses the action that maximizes the Q-value in reinforcement learning, then it may happen that the agent may not explore a lot. The agent might be stuck performing non-optimal actions. The agent gets stuck in non-optimal policies because it does not explore enough to find the best action from each state. [2]

2 ways to force agent to explore

1. We use a greedy strategy so that the all the maximum values are selected. We could also set the initial values high. We keep the initial values high so that the unexplored regions or unexplored areas look better to the agent.

2. To explore more, we can also pick random actions occasionally. By doing the so we force the agent to explore more and not just exploit the given actions but also explore more.

Question 2

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

Question 2 Q - Table

	ACTIONS			
STATE	UP	DOWN	LEFT	RIGHT
0	2.940	3.940	2.940	3.940
1	1.970	2.970	0.970	2.970
2	-0.01	1.99	-0.01	1.99
3	-1	1	-1	0
4	0	0	0	0

Calculation :

$Q^*(s_t, a_t) = r_t + \gamma * \max_a Q(s_{t+1}, a)$, Gamma = 0.99

For state 4, initialize all values to zero.

For state 3, max state 4 = 0

$Q(up) = -1 + \gamma * 0 = -1$

$Q(down) = 1 + \gamma * 0 = 1$

$Q(left) = -1 + \gamma * 0 = -1$

$Q(right) = 0 + \gamma * 0 = 0$

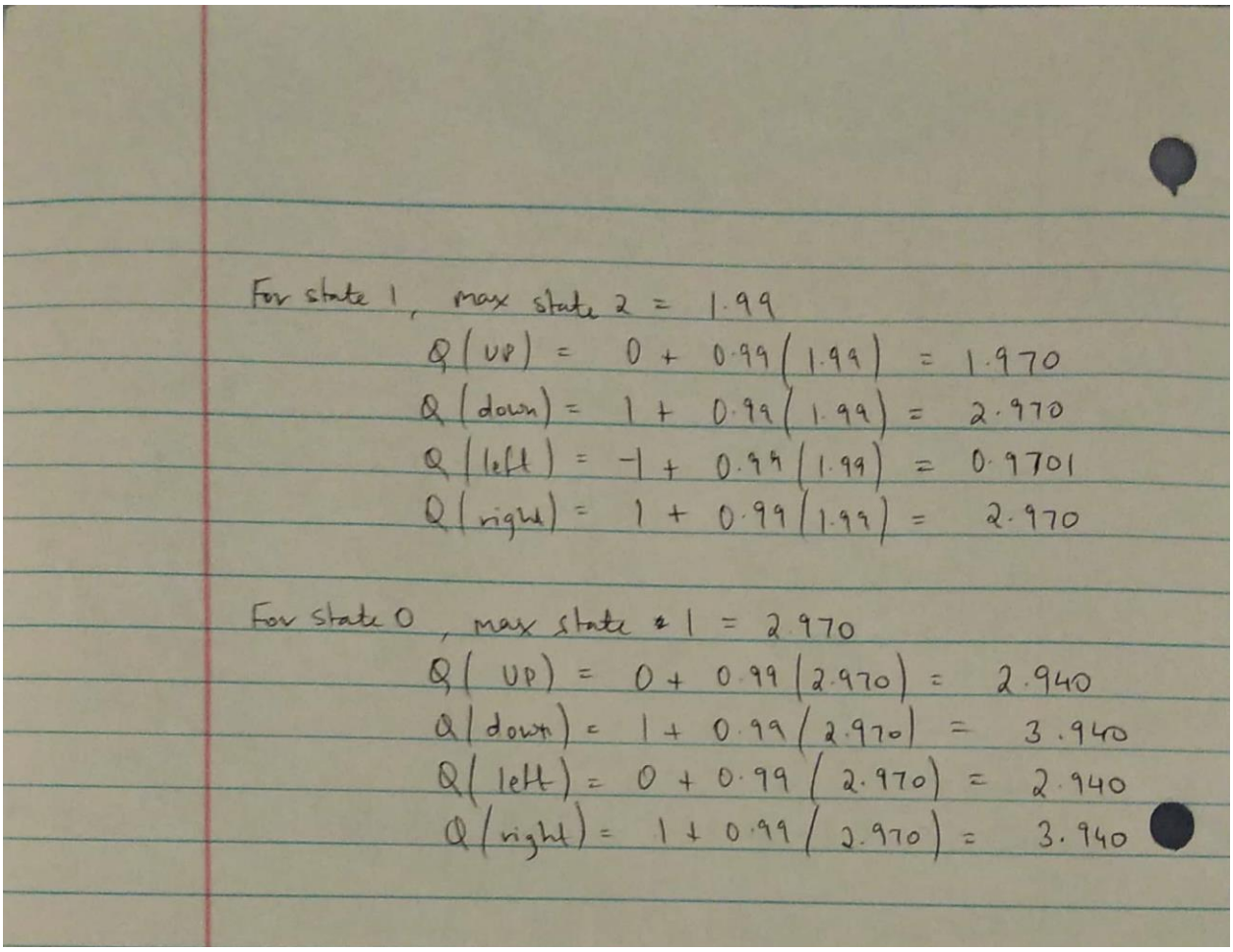
For state 2, max state 3 = 1

$Q(up) = -1 + 0.99(1) = -0.01$

$Q(down) = 1 + 0.99(1) = 1.99$

$Q(left) = -1 + 0.99(1) = -0.01$

$Q(right) = 1 + 0.99(1) = 1.99$



References:

1- https://medium.freecodecamp.org/a-brief-introduction-to-reinforcement-learning-7799af5840db

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

3- http://www.aispace.org/exercises/solutions/11a/4.shtml

4- https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc

5- https://en.wikipedia.org/wiki/Reinforcement_learning