Project 4 Disha Mehra

Instructor: Prof. Sargur N. Srihari Person Number:50288911

Objective:

The objective of this project is to implement machine learning method that is Reinforcement Learning.

The project combines reinforcement learning and deep learning. The task is to teach the agent 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 (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.

Implementation:

- Code Snippet 1:

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

model.add(Dense(128, input_dim=state_dim, activation='relu'))

model.add(Dense(128,activation='relu'))

model.add(Dense(action_dim,activation='linear'))

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

Implementation of neural network to get the values of Q

While it is easy to have a small table for a simple grid world, the number of possible states in any modern game or real-world environment is nearly infinitely larger. For most interesting problems, tables simply don't work. We instead need some way to take a description of our state, and produce Q-values for actions without a table: that is where neural networks come in. By acting as a function approximator, we can take any number of possible states that can be represented as a vector and learn to map them to Q-values.
We have implemented neural network by taking number of nodes as 128 and activation function as 'relu' to get the values of Q.

- Code Snippet 2:

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

Implementation of epsilon

We specify an exploration rate "epsilon," which we set to 1 in the beginning. This is the rate of steps that we'll do randomly. In the beginning, this rate must be at its highest value, because we don't know anything about the values in Q-table. This means we need to do a lot of exploration, by randomly choosing our actions.
We generate a random number. If this random number > epsilon, then we will do "exploitation" (this means we use what we already know to select the best action at each step). Else, we'll do exploration.

The idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively as the agent becomes more confident at estimating Q-values. The implementation of epsilon is the formula to calculate the epsilon for the next time for the comparison with the random number.

- Code Snippet 3:

```
    ### START CODE HERE ### (≈ 4 line of code)
    if (st_next==no_state).all():
        t[act]=rew
    else:
        t[act]=rew+self.gamma*np.amax(q_vals_next[i])
### END CODE HERE ###
```

Implementation for calculating Q

Q-function is implemented to give rewards to the agent based on its actions. This is a math function that takes two arguments: the current state, and a given action. While in state S, we estimate the future reward for each possible action A. We assume that after we have taken action A and moved to the next state S', everything works out perfectly. The expected future reward Q(S,A) for a given a state S and action A is calculated as the immediate reward R, plus the expected future reward thereafter Q(S',A'). We assume the next action A' is optimal. As, there is no surety about the future, we discount Q(S',A') by the factor gamma.

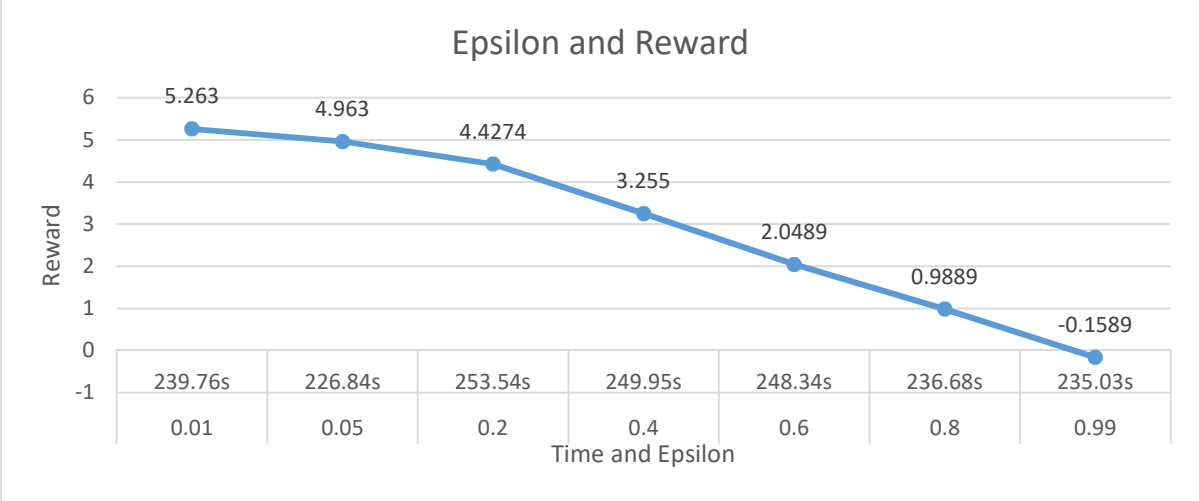
Analysis And Testing:

*Reward in all the cases is referred to as the mean reward.

1. Epsilon:

Epsilon	Time(in sec)	Reward
0.01	239.76s	5.263
0.05	226.84s	4.963
0.2	253.54s	4.4274
0.4	249.95s	3.255
0.6	248.34s	2.0489
0.8	236.68s	0.9889
0.99	235.03s	-0.1589

Table 1 shows the value of reward and time when the epsilon changes.

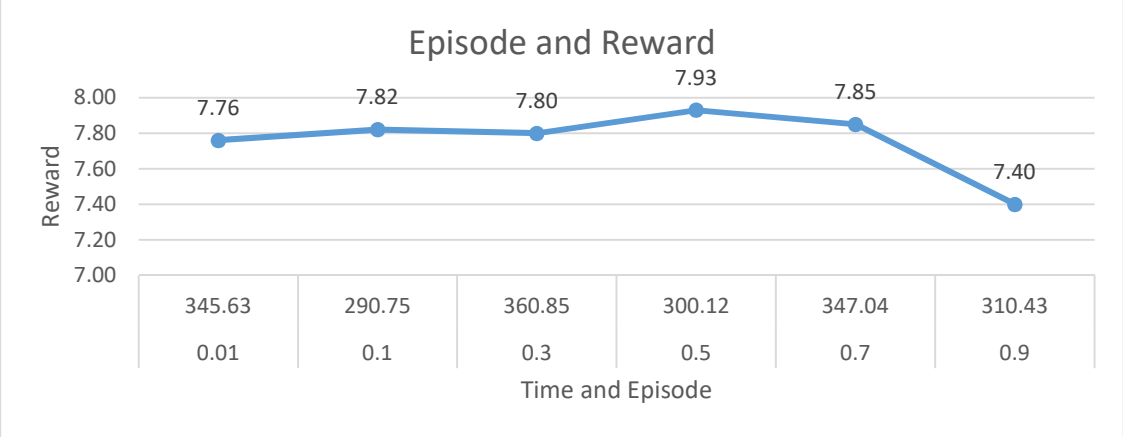


According to the graph we see that as the min_epsilon(epsilon) increases, exploration of the agent increases and as a result the reward tends to decrease. Therefore, higher is the min_epsilon, lower is the reward.

2. Episodes:

Episodes	Time(in sec)	Reward
500	16.26s	0.54152
1000	46.69s	1.02996
2500	118.71s	3.4854
5000	246.42s	5.24474
7500	384.35s	6.0146
10000	502.40s	6.4281

Table 2 shows the value of reward and time when the number of episode changes.

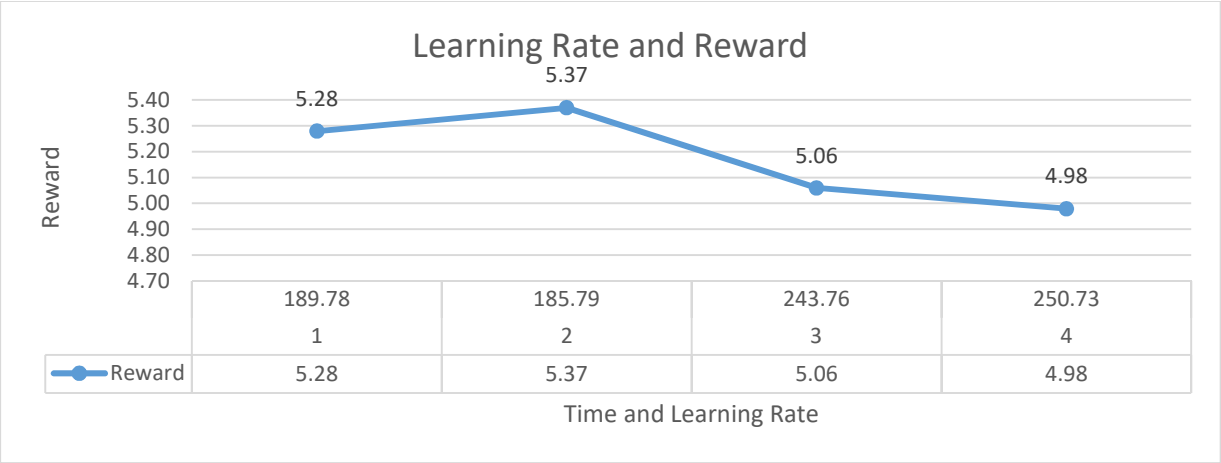


According to the graph, as the number of episode increases the reward tends to increases as the agent will be able to learn the environment properly. Therefore, higher are the number of episodes, higher is the reward.

3. Learning Rate:

Learning Rate	Time(in sec)	Reward
0.001	233.98s	4.7531
0.01	232.52s	3.7584
0.1	263.38s	0.492
0.2	242.40s	0.4438
0.5	254.49s	0.368
0.75	276.99s	-1.097

Table 3 shows the value of reward and time when the learning rate changes.

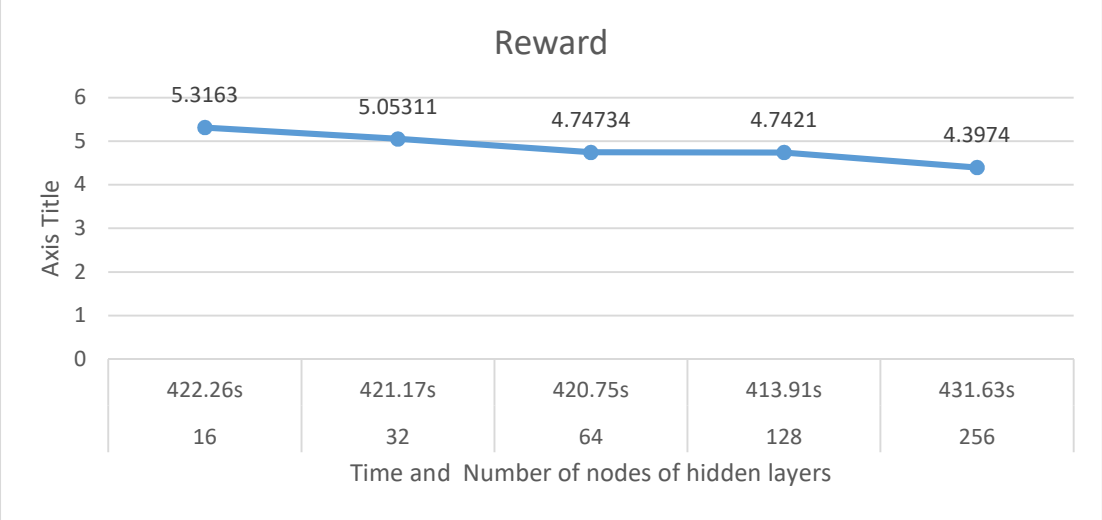


As per the graph as the learning rate is increasing the reward value tends to decrease. Therefore, more is the learning rate lesser is the reward.

4. Number of nodes in hidden layers:

Number of nodes	Time(in sec)	Reward
16	422.26s	5.3163
32	421.17s	5.05311
64	420.75s	4.74734
128	413.91s	4.7421
256	431.63s	4.3974

Table 4 shows the value of reward and time when the number of nodes in hidden layers changes.

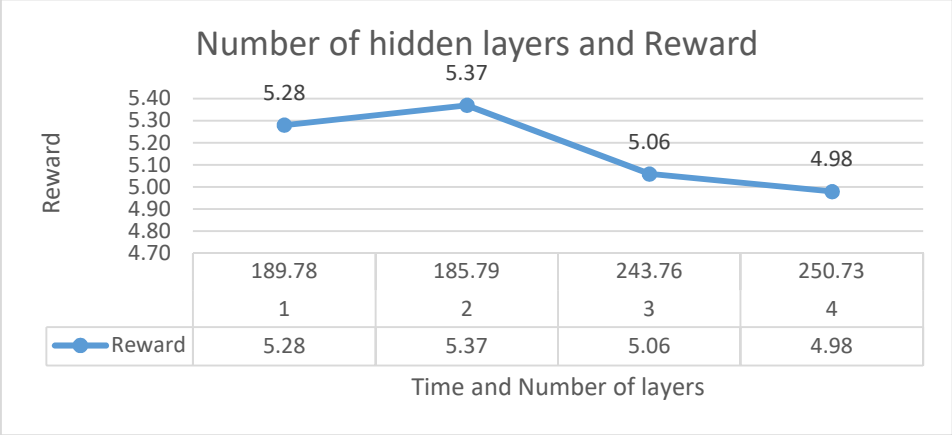


As per the graph above, as the number of hidden nodes in the hidden layers of neural networks increases the reward value is decreasing. Therefore, lesser the nodes more the reward.

5. Number of hidden layers:

No. of Hidden Layers	Time (in sec)	Reward
1	189.78	5.28
2	185.79	5.37
3	243.76	5.06
4	250.73	4.98

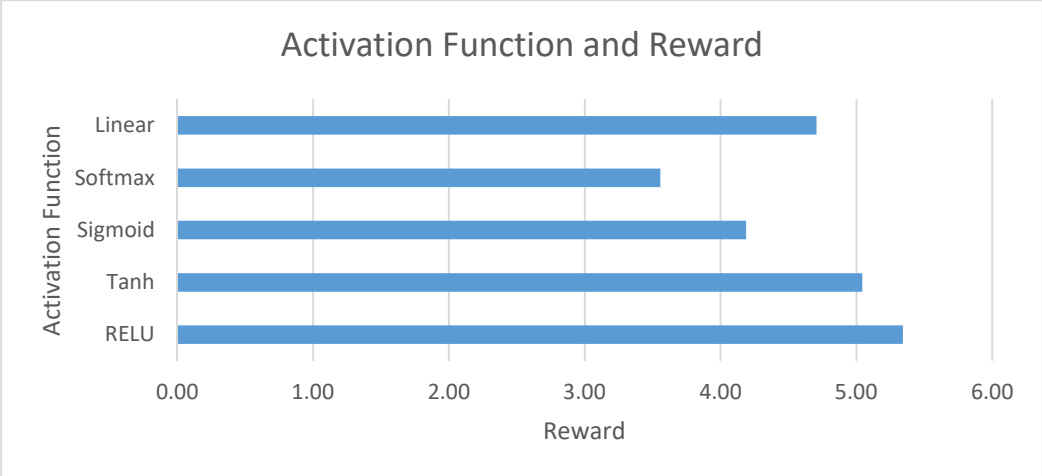
Table 5 shows the value of reward and time when the number of hidden layers changes.



6. Activation Function:

Activation	Time (in sec)	Reward
RELU	180.79	5.34
Tanh	173.98	5.04
Sigmoid	196.4	4.19
Softmax	231.5	3.56
Linear	194.99	4.71

Table 6 shows the value of reward and time when the activation function changes.

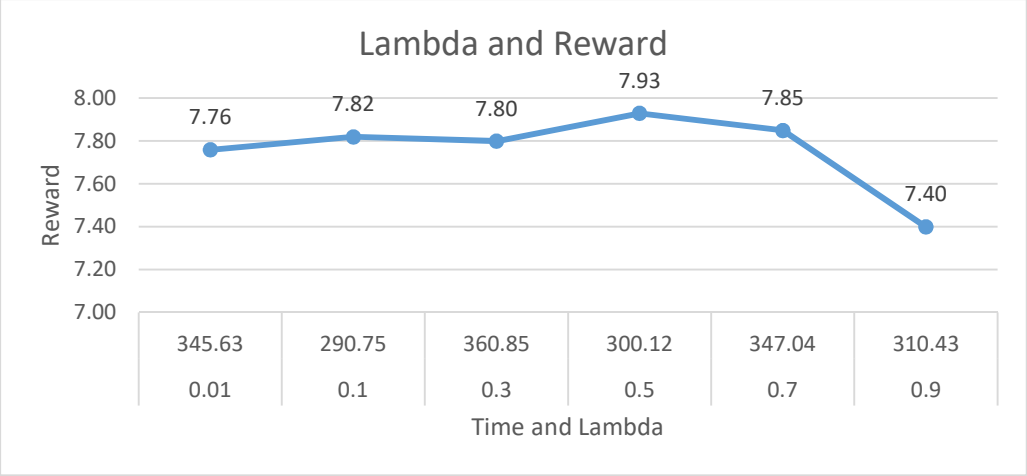


As per the table, RELU activation function gives us the maximum award in minimum amount of time and is the best activation function for this program.

7. Lambda:

Lambda	Time (in sec)	Reward
0.01	345.63	7.76
0.1	290.75	7.82
0.3	360.85	7.80
0.5	300.12	7.93
0.7	347.04	7.85
0.9	310.43	7.40

Table 7 shows the value of reward and time when the lambda changes.

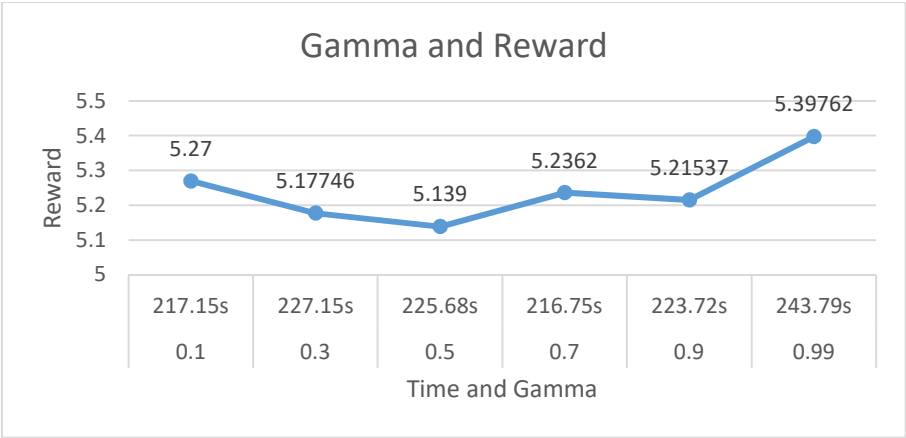


As per the graph, the trend is that as lambda increases the reward decreases.

8. Gamma:

Gamma	Time(in sec)	Reward
0.1	217.15s	5.27
0.3	227.15s	5.17746
0.5	225.68s	5.139
0.7	216.75s	5.2362
0.9	223.72s	5.21537
0.99	243.79s	5.39762

Table 8 shows the value of reward and time when the gamma changes.



As seen from the graph, the value of gamma changes the reward in uncertain way.

Writing Task:

Task 1:

In Reinforcement Learning, the agent has to reach the particular target in an environment specified. Agent tries to maximize it's expected return. If the agent takes some action in a state and gets some rewards and he changes the state. Now after a particular number of steps, either it will reach to the target or it will terminate as the number of episodes gets used up. Next time if the agent chooses the action which maximizes the Q-value, the agent will take same path again and again because as he has not explored other states and tends to exploit the same state only. If the agent always chooses the action that maximizes the Q-value, then the agent will not be able to explore enough to get the best action from each state. This will result in the formation of non-optimal policy. As the agent hasn't explored well enough there might be better rewards for different action taken from different states. There might be another optimal policy which is better than the policy the agent has taken before.

Two ways for the agent to explore are:

1. Pick random actions occasionally.
2. Set the initial values high, so that unexplored regions look good and could be explored.

Task 2:

It is in the zip file with name Task2.

Resources:

1. https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc
2. https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe
3. https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa
4. https://medium.com/@curiousily/solving-an-mdp-with-q-learning-from-scratch-deep-reinforcement-learning-for-hackers-part-1-45d1d360c120
5. https://towardsdatascience.com/my-journey-to-reinforcement-learning-part-1-q-learning-with-table-35540020bcf9