CSE574: Introduction to ML(Fall'18)

Project 4: Tom and Jerry in Reinforcement learning

Vishal Shivaji Gawade
Department of Computer Science
University at Buffalo
Buffalo, NY,14226
vgawade@buffalo.edu

Abstract

The project combines reinforcement learning and deep learning. Our 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.

1. Build a 3-layer neural network using Keras library

First, we build a neural network for our model. We will try to implement it using shown in the project description pdf.

• 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 output layer is 'linear' (that will return real values)
• Input dimensions for the first hidden layer equals to the size of your observation space (state_size) • Number of hidden nodes is 128 for both hidden layers
• Number of the output should be the same as the size of the action space (action_size)

```
        self.model.load_weights( 'brain.h5' )

    def _createModel(self):
        # Creates a Sequential Keras model
        # This acts as the Deep Q-Network (DQN)

        model = Sequential()

        ### START CODE HERE ### (≈ 3 lines of code)
        model.add(Dense(128,input_dim=state_dim))
        model.add(Activation("relu"));
        model.add(Dense(128));
        model.add(Activation("relu"));
        model.add(Dense(action_dim));
        model.add(Activation("linear"));
        ### END CODE HERE ###

        opt = RMSprop(lr=0.00025)
        model.compile(loss='mse', optimizer=opt)

        return model
```

Our built neural network model is going to give us the output as action value (Q val) for all four actions left, right, up and down. After this we take max of these four values and take an action which is giving us the maximum value for next step.

2. Implement exponential-decay formula for epsilon

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

We are going to code this in python for our problem using numpy library. If we see the formula it is pretty straightforward to implement it in python.

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

3. Implement Q-function

We are going to implement Q function using below formula.

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

In our code rew variable contains the value of r_t.We are taking max from array of q_vals_next[i] which is current array of Q values. We get Q values in q_vals_next from our brain i.e. neural network. The other variables gamma (discounting factor) we keep its value 0.99 for all calculations of future discounted rewards.
Now we just implement this logic in python as show in below code snippet.

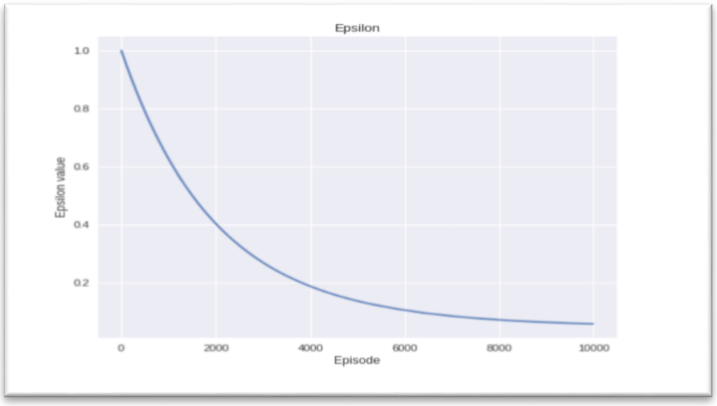
```
### START CODE HERE ### (≈ 4 line of code)
if st_next is None:
  q_vals[i][act]=rew;
else:
  q_vals[i][act]=rew+(self.gamma * np.amax(q_vals_next[i]))

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

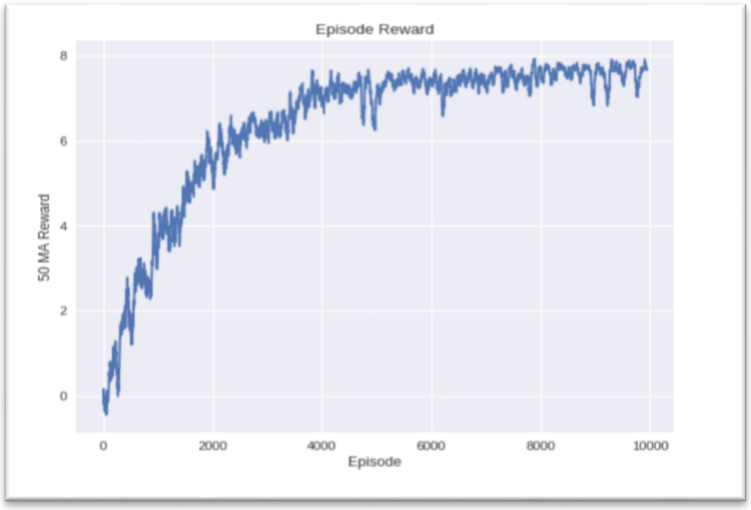
4. Running the code

Now let's run our code and see how agent learns over episodes, find the shortest path and eventually reaches the goal.
First, we will see how over 10k episodes the value of ε changes.

```
Episode 9900
Time Elapsed: 819.87s
Epsilon 0.06026060684412682
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.282114069992858
----------
```



We terminate our code after 10k episodes. Our epsilon value reaches till 0.0602
Now let's see the graph of reward vs episodes.



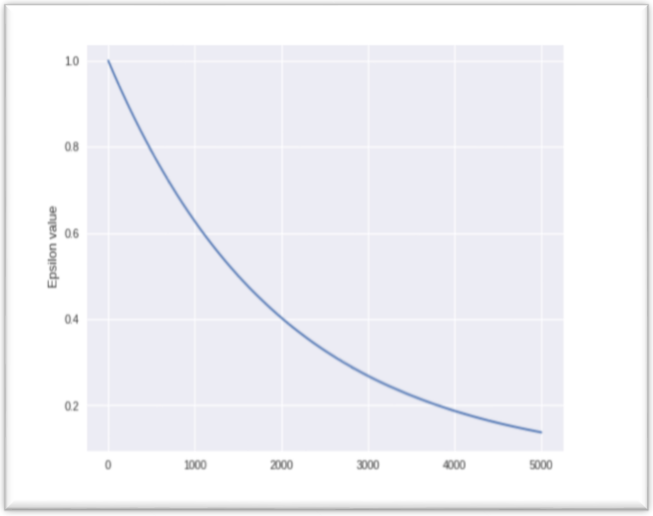
As we can see we are getting max reward of 8 and after 8k episodes we are mostly getting that reward for episodes. So, we can say after 8k rewards slowly our agent starts converging.

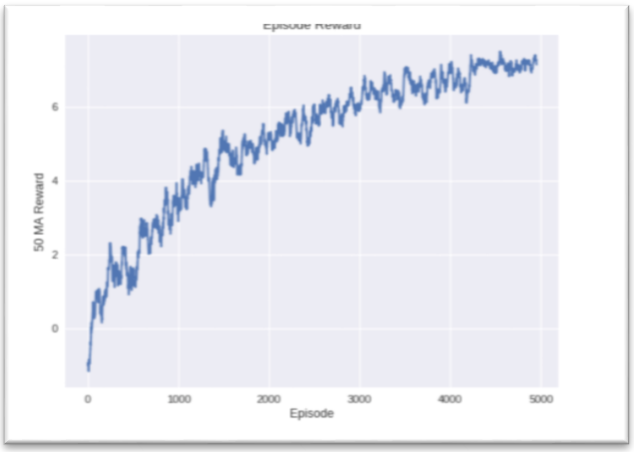
After running our model for 10k episodes we are getting rolling mean reward of 6.33. Now let's try to tune the hyperparameters and see the results.

5. Hyperparameter Tuning

5.1. Network Setting: Changing the value of number of episodes. We are keeping all other parameters same.
For episodes=5000

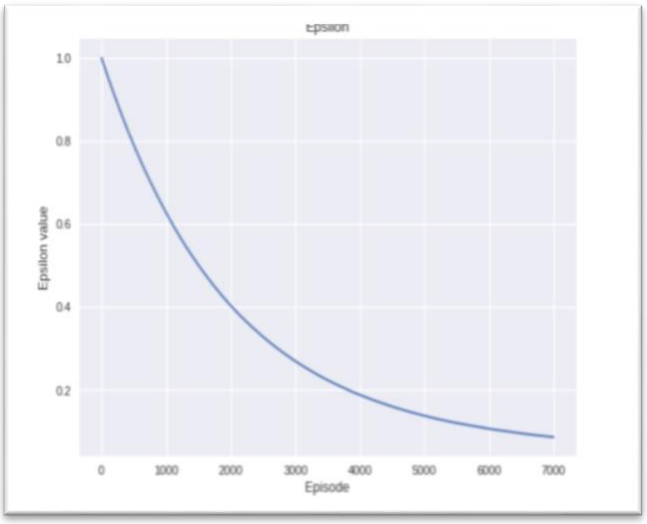
```
----------
Episode 4900
Time Elapsed: 373.34s
Epsilon 0.14156160135355916
Last Episode Reward: 8
Episode Reward Rolling Mean: 4.982920224953134
----------
```

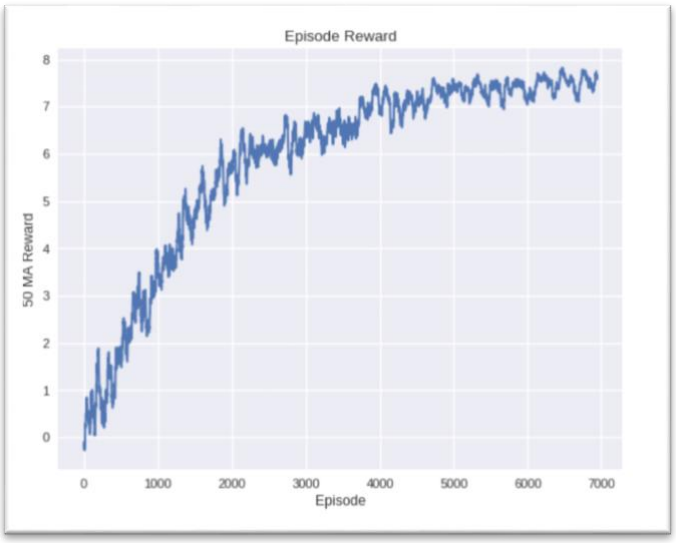




For episodes=7000

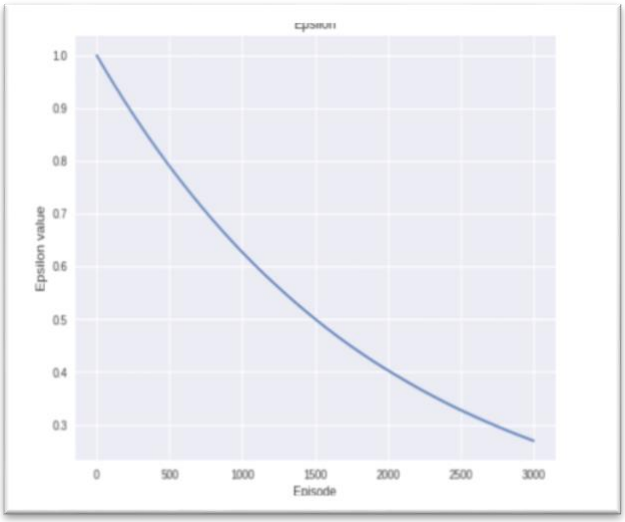
```
----------
Episode 6900
Time Elapsed: 502.18s
Epsilon 0.08803139430630685
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.805763858256139
----------
```

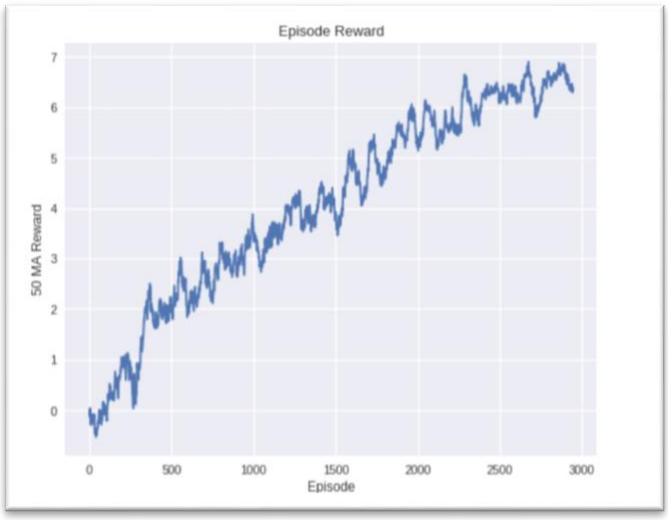




For episodes=3000

```
----------
Episode 2900
Time Elapsed: 231.58s
Epsilon 0.27988104617592646
Last Episode Reward: 8
Episode Reward Rolling Mean: 3.8718314887540166
----------
```





Summing Up

No of episodes	Rolling Mean Reward	Time taken in seconds
3000	3.87	231.58
5000	4.98	373.34
7000	5.80	502.18

As we can see if we increase no of episodes, our rolling mean reward increases. But it will be going to happen till our agent converges and we start getting max reward for every episode. That time our agent is fully learned about the environment. Also, as for less no of episodes we are going to need lest time to train model.

5.2. Network Setting: Changing the value of gamma. We are keeping all other parameters same.

Gamma=0.79

```
----------
Episode 9900
Time Elapsed: 789.84s
Epsilon 0.060792074094771 46
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.457810427507397
----------
```

Gamma=0.49

```
----------
Episode 9900
Time Elapsed: 784.80s
Epsilon 0.06082233424667954
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.46985001530456
----------
```

Gamma=0.19

```
----------
Episode 9900
Time Elapsed: 802.59s
Epsilon 0.060806112914350265
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.49433731251913
----------
```

Summing Up

Gamma Value	Rolling Mean Reward	Time taken in seconds
0.79	6.45	789.84
0.49	6.46	784.80
0.19	6.49	802.59

We observed that a high gamma value is used for this problem space because the player gets a reward every time it moves closer to the mouse/goal. With a problem space where a reward is only given far in the future, like at the end of a game, there is more uncertainty and future rewards have to be discounted and smaller gamma values should be used.Typically, the effect of gamma can be seen on more complex environments, like Atari or partially observed environments, where there is a high level of uncertainty and we would prefer to get reward as soon as possible. For our project that's not the case. Therefore, we don't see lot change in rolling mean reward value.

5.3. Network Setting: Changing the value of epsilon max. We are keeping all other parameters same.

For epsilon max=0.7

```
----------
Episode 9900
Time Elapsed: 816.82s
Epsilon 0.05753058225022626
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.8647076828895015
----------
```

For epsilon max=0.3

```
----------
Episode 9900
Time Elapsed: 817.89s
Epsilon 0.05314373290746619
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.3200693806754416
----------
```

For epsilon max=0

```
Episode 9900
Time Elapsed: 790.28s
Epsilon 0.049218919447746576
Last Episode Reward: 6
Episode Reward Rolling Mean: 7.74359759208244
----------
```

Summing Up

Epsilon max	Rolling Mean Reward	Time taken in seconds
0.7	6.86	816.82
0.3	7.32	817.89
0	7.74	790.28

If the random value is lower than epsilon, then it will pick the action randomly. If it is above epsilon, it will pick the best action. So, as we play more and more episodes, epsilon will get smaller and smaller, thereby decreasing the number of random moves. The agent can perform somewhat random actions and learn to perform well very quickly. When we start the agent, it's DQN is initialized to random values, so that it will start making random choices although it is attempting to make the best choice. So, even though we were running it with `epsilon = 0`, it will still start by taking random actions and will most likely figure out which action to take to maximize reward. The rate of change of epsilon is very slow in the default parameters, and that value can be adjusted accordingly. However, if we ran the environment enough times with no exploration, there is a possibility that Tom converges to a sub-optimal policy. The chances of this, however, is slim for our environment, but in more complex environments, it will be a major factor.

5.4. Network Setting: Changing the value of epsilon min. We are keeping all other parameters same.

For epsilon min=0.1

```
----------
Episode 9900
Time Elapsed: 954.76s
Epsilon 0.10948294349498
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.300071421283542
----------
```

For epsilon min=0.3

```
----------
Episode 9900
Time Elapsed: 966.58s
Epsilon 0.3058686835137126
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.090398938883787
----------
```

Summing Up

Epsilon min	Rolling Mean Reward	Time taken in seconds
0.1	6.30	954.76
0.3	5.09	966.58

As we can see if we increase the epsilon min value from 0.05 to 0.1 or 0.3 the rolling mean reward value is decreasing time taken for agent is also increased.

5.5. Network Setting: Changing the neural networks parameters. We are keeping all other parameters same.

Neural network setup-

```
def _createModel(self):
    # Creates a Sequential Keras model
    # This acts as the Deep Q-Network (DQN)

    model = Sequential()

    ### START CODE HERE ### (= 3 lines of code)
    model.add(Dense(128,input_dim=state_dim))
    model.add(Activation("relu"));
    model.add(Dense(128));
    model.add(Activation("relu"));
    model.add(Dense(128));
    model.add(Activation("relu"));
    model.add(Dense(action_dim));
    model.add(Activation("linear"));
    ### END CODE HERE ###

    opt = RMSprop(lr=0.00025)
    model.compile(loss='mse', optimizer=opt)

    return model
```

Results-

```
----------
Episode 9900
Time Elapsed: 887.99s
Epsilon 0.05958416238521045
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.566472808897052
----------
```

Neural network setup-

```
def _createModel(self):
    # Creates a Sequential Keras model
    # This acts as the Deep Q-Network (DQN)

    model = Sequential()

    ### START CODE HERE ### (= 3 lines of code)
    model.add(Dense(128,input_dim=state_dim))
    model.add(Activation("relu"));
    model.add(Dense(128));
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(128));
    model.add(Activation("relu"));
    model.add(Dense(128));
    model.add(Activation("tanh"));
    model.add(Dense(action_dim));
    model.add(Activation("linear"));
    ### END CODE HERE ###

    opt = RMSprop(lr=0.00025)
    model.compile(loss='mse', optimizer=opt)

    return model
```

Results-

```
----------
Episode 9900
Time Elapsed: 962.01s
Epsilon 0.06013314753885523
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.2260993776145295
----------
```

As we see if we add more layers to model it takes more time to train it. But there is no significant improvement in rolling mean reward value. Therefore, it is best if we stick with combination for two hidden layer neural network with relu and linear activation functions.

6. Q and A

Q. What parts have you implemented?
Ans-We first implemented neural network using two hidden layers. After that we implemented the epsilon formula, and finally we implemented Q value function. All these implementations we can see in report points 1,2,3.

Q. What is their role in training the agent?
Ans-Neural network gives us the q values array. Depending on which we decide which next step to take. We are using epsilon for deciding exploration/exploitation rate.it also helps to decide whether we should explore the environment or exploit the memory. Finally, we are using Q value function, is used to get the highest possible reward for particular action. It is based on Markov decision process. Q val is an indication for how good it is for an agent to pick action a while being in state s. For more explanation please see the report topic 1,2,3.

Q. Can these snippets be improved and how it will influence the training the agent?
Ans-If we see the topic 5 in report, we will see all the improvements and tunings we did to see the influence on the training agent

Q. How quickly your agent was able to learn?
Ans-Our agent first takes random action for exploring the environment. After certain time it start exploiting the memory and learn from these actions. We are getting maximum reward from episode around 8k. If we see the graphs of episode vs reward in report topic 4 we will understand this better.

7.Writing tasks

Q1. 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.
Ans. For first part of question if our agent always chooses the action that maximizes Q value, it will never explore the environment. It will just try to achieve max Q value. The optimal policy is a function that selects an action
for every possible state and actions in different states are independent. So, we can say that agent will be trapped in non-

optimal policies and it will not try to find the best possible action for each state as agent will always choose action which is going to maximize the Q value.
For second part of question the two approaches by which we can force the agent to explore is as follow,
The two ways:
1) If we set the initial values of hyperparameters like epsilon, episodes etc. high, our agent will keep exploring more parts of environment which are generally not explored by agent (called unexplored parts). It will help agent to learn more about its surrounding. If we don't set initial values high, we are not able to explore unexplored parts of environment. Therefore, one way to force agent to explore more is by setting initial values of parameters high.
2) Another way is, we can make agent to pick some random values for action while it is running through episodes so that it starts exploring the environment more. If agent don't take the random action it will not able to explore the unexplored parts of environment. Random action helps it to take random path which leads to exploration of unexplored parts of environment. Therefore, another way to force agent to explore more is by picking random actions sometimes.

Q2. Calculate Q-value for the given states and provide all the calculation steps.

Ans. We are going to show these calculations on paper and then attach a scanned copy of calculations with this report. Please see the calculations on next page.

Vishal Gawade - M2 Proj 4

We are going to start from the last step to calculate the Q-values.

For S4 :- It is a terminal state. When agent reaches this state game ends. Therefore all Q-values will be zero for this state.

$\therefore \boxed{(S_4, UP) = (S_4, DOWN) = (S_4, RIGHT) = (S_4, LEFT) = 0}$

For S3 :-

When agent is in S3 state and ~~the~~ it takes DOWN action it reaches S4 state.

$\therefore (S_3, DOWN) = r_t + \gamma * max(S_4)$

Reward r_t will be 1 for ~~final~~ reaching final state.

and $max(S_4) = 0$; $\gamma = 0.99$

$\therefore \boxed{(S_3, DOWN) = 1 + (0.99 * 0) = 1}$

For other values for S3 we need S2 values.

$\therefore$ lets calculate S2 values 1st.

~~$(S_2, DOWN)$~~

$(S_2, RIGHT) = r_t + (0.99 * max(S_3))$

When we go down from S3 state we get terminal state S4. And right of S2 state gives us S3 state.

$\therefore r_t = 1 \quad \& \quad \max(S_3) = 1$

$$(S_2, \text{RIGHT}) = 1 + (0.99 * 1) = 1.99$$

Similarly we can take DOWN action in S2, and taking RIGHT we get terminal state. Therefore reward will be same as (S_2, RIGHT)

$$(S_2, \text{DOWN}) = 1.99$$

For other S2 states we need S1 values. Therefore lets calculate S1, 1st.

$$(S_1, \text{DOWN}) = r_t + (0.99 * \max(S_2))$$

As we are reaching towards our goal if we take DOWN action, $\therefore r_t = 1$ and $\max(S_2) = 1.99$

$$\therefore (S_1, \text{DOWN}) = 1 + [(0.99) * 1.99]$$

$$(S_1, \text{DOWN}) = 2.97$$

Now lets calculate $(S_0, RIGHT)$

like previous case $r_t = 1$ & $\max(S_1) = 2.97$

$$\therefore (S_0, RIGHT) = 1 + [0.99 * 2.97] = 3.94$$

Now we know,

~~S_T~~ $\max(S_0) = 3.94$

$\max(S_1) = 2.97$

$\max(S_2) = 1.99$

$\max(S_3) = 1$

$\max(S_4) = 0$

$(S_0, LEFT) = r_t + [0.99 * \max(S_0)]$

r_t will be 0, because we are ~~moving away from goal~~ in same state. And $\max(S_0) = 3.94$

$(S_0, LEFT) = 0 + (0.99 * 3.94)$

$$(S_0, LEFT) = 3.90$$

~~Similarly $(S_0, UP) = 3.90$~~

Similarly $(S_0, UP) = 3.90$

Again like $(S_0, \overset{\text{LEFT}}{\cancel{\text{DOWN}}})$, $(S_0, \overset{\text{DOWN}}{\cancel{\text{RIGHT}}})$ have similar reasoning and results.

$\therefore \cancel{(S_0, \text{RIGHT}) = 3.94}$

$\boxed{(S_0, \text{DOWN}) = 3.94}$

Again like (S_1, DOWN), for (S_1, RIGHT) we can give similar reasoning and have similar results.

$\therefore \boxed{(S_1, \text{RIGHT}) = 2.97}$

For (S_1, UP) ; $r_t = 0$ and $\max(S_2) = 2.97$

$\therefore (S_1, \text{UP}) = 0 + (0.99 * 2.97)$

$\boxed{(S_1, \text{UP}) = 2.94}$

For (S_1, LEFT), $r_t = -1$ (moving away from goal) and $\max(S_0) = 3.94$

$\therefore (S_1, \text{LEFT}) = -1 + (0.99 * 3.94)$

$\boxed{\therefore (S_1, \text{LEFT}) = 2.90}$

For (S_2, UP), $r_t = -1$ & $\max(S_1) = 2.97$

$(S_2, UP) = -1 + (0.99 * 2.97)$

$\boxed{(S_2, UP) = 1.94}$

For $(S_2, LEFT)$, $r_t = -1$ & $\max(S_1) = 2.97$

$\boxed{(S_2, LEFT) = 1.94}$

For $(S_3, RIGHT)$, $r_t = 0$ & $\max(S_3) = 1$

$(S_3, RIGHT) = 0 + (0.99 * 1)$

$\boxed{(S_3, RIGHT) = 0.99}$

For $(S_3, LEFT)$, $r_t = -1$, $\max(S_2) = 1.99$

$(S_3, LEFT) = -1 + (0.99 * 1.99)$

$\boxed{(S_3, LEFT) = 0.97}$

~~For (S_3, UP)~~

For (S_3, UP), $r_t = -1$, $\max(S_2) = 1.99$

$$\therefore \boxed{(S_3, UP) = 0.97}$$

Finally lets draw Q-Table.

STATE	ACTIONS			
	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

8.Conclusion

- We successfully applied deep reinforcement learning algorithm - DQN (Deep Q-Network) for problem.
- We also learnt about how to train the agent by exploration and exploitation.
- We also understood how Q values and neural network plays a role in deciding next action of agent to find optimal path to the goal.

References

[1] https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa

[2] https://stats.stackexchange.com/questions/132890/why-is-the-optimal-policy-in-reinforcement-learning-independent-of-the-initial