PROJECT 4

Tom and Jerry in Reinforcement Learning

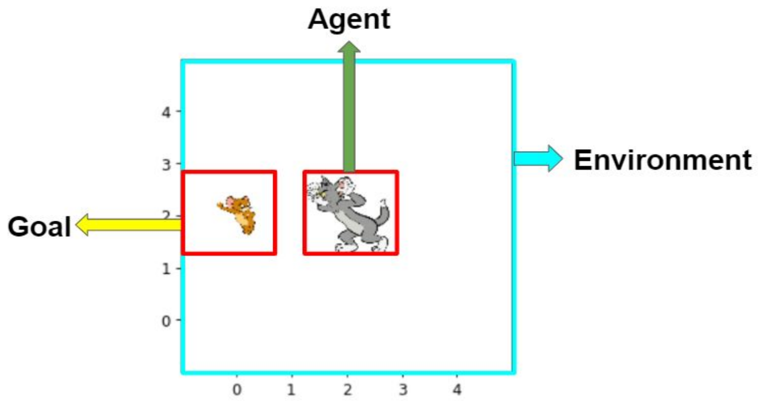
Venktesh Kaviarasan
Department of Computer Science
University at Buffalo
UBIT No: 50289400

Abstract

★ The task given is to teach the agent to navigate in the grid-world environment, where the agent here is (Tom) and the goal is (Jerry) such that it finds the shortest path to the goal considering the initial positions of both the agent and goal are deterministic. For solving this problem, we would apply deep reinforcement learning algorithms -DQN (Deep Q-Network).

1 ABOUT THE TASK :

- Our main goal is to let our agent learn the shortest path to the goal. In the environment the agent controls a green square, and the goal is to navigate to the yellow square (reward +1), using the shortest path. At the start of each episode all squares are randomly placed within a 5x5 grid-world. The agent has 100 steps to achieve as large a reward as possible. They have the same position over each reset, thus the agent needs to learn a fixed optimal path.



- States - 25 possible states (0, 0), (0, 1), (0, 2), ... ,(4, 3), (4, 4). The goal (yellow square) and the agent (green square) are dynamically changing the initial position on every reset.

2 MODEL ADOPTED AND SOLUTION TO THE PROBLEM:

2.1 *DATA SETS :*

- Unlike supervised and unsupervised learning, reinforcement learning does not just experience a fixed data set
- Reinforcement learning algorithms interact with an environment Q-learning generates data exclusively from experience, without incorporation of the prior knowledge. If we put all our history data into a table with state, action, reward, next state and then sample from it, it should be possible to train our agent that way, without the dataset.

TERMINOLOGIES USED:

- Action (A): possible moves that agent can take
  State (S) : Current situation returned by environment
  Reward (R) : Immediate return sent back from the environment to evaluate the last action
  Policy (π) : Strategy that agent employs to determine next action based on current state
  Value (V) : Expected long-term return with discount, as opposed to short-term reward

2.2 **IMPLEMENTATION:**

- The initial step involves setting up the environment

Environment :

- ***def _init_ function :*** Here we set up a 5*5 grid environment and in the _init_ function definition we read the cat, mouse and confetti images for reproducing them in the gui.
- ***def_update_state function:*** In this function we get the values of the state say left, right, down and up in variables like fy,fx,py and px respectively. Using this we calculate the old and new distance values. There is also a overall timer which we keep decrementing so that the code does not go into a infinite loop
- We also define functions for rewards, checking of the game if it is over , function for steps and for rendering the output and resetting it as well.

Random Actions:

- This is used for running the environment using random actions from the set of four random actions for the game to get started initially before the learning process even starts

Brain:

- The brian of the agent is where the model is created and held.. Here the DQN takes five tuples as an input and it then passed through two hidden layers which outputs a **Q-value vector.** *Q(*st*,a1)* - q-value for a given state s, if we choose action a1 We need to choose such an action, that will return the highest Q-value

Memory:

- In this block we are defining the main functions that will be used to store the experiences of our agent. In the sample function in class memory we append the values of variables such as state,action, reward and next state

Agent:

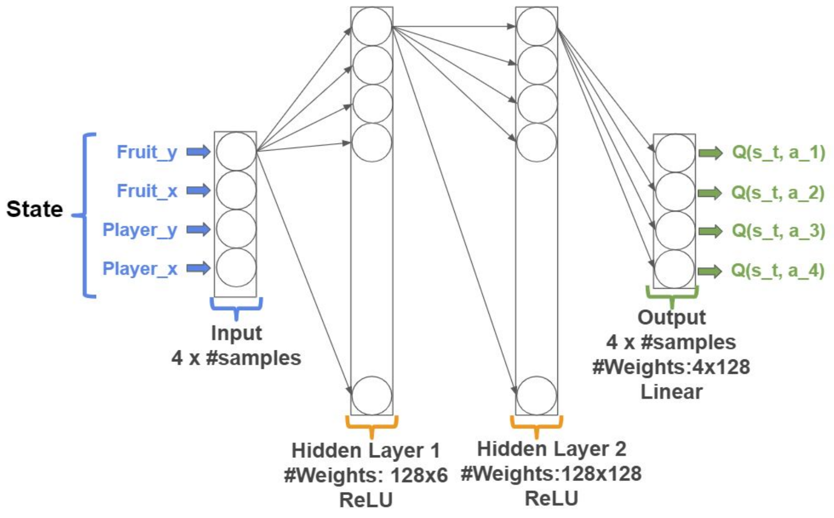
- The agent, which learns to navigate the environment. At first our agent hasn't explored our environment. Hence it takes random actions to start exploring the environment. This random action is taken by a certain rate called epsilon or exploration rate.
- When the agent doesn't select an action by random, it will pick the action that will give the highest reward with the help of our Deep Q-Network. At

<u>Algorithm</u>:

- Here we use an algorithm which is a biologically inspired mechanism termed ***experience replay*** that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over the changes in the data distribution.

2.3 <u>EXPLANATION OF CODING TASK IMPLEMENTATION :</u>

TASK 1 : <u>Building a 3- Layer Neural Network using Keras Library</u>



gm

- For the given task we have implemented a sequential neural network, the sequential model is basically a linear stacking up of layers.
- This sequential neural network acts as the **Deep Q-Network** for our model, in the code we create **2 hidden layers with 128 nodes in each.**
- Using the keras library we do :
  - model.add(Dense(output_dim=128, activation='relu', input_dim=self.state_dim))

  This above line of code **creates a hidden layer with 128 nodes** and uses **ReLu as the activation function** and we map input dimensions equivalent to the state dimensions that is four.
  - model.add(Dense(output_dim=128, activation='relu'))

  The above code is used for creating another hidden layer that is our second hidden layer, this layer also contains 128 nodes and uses activation function as ReLu.
  - model.add(Dense(output_dim=self.action_dim, activation='linear'))

  This is used for creating the output layer with the dimensions same as the actions, which is up,down,left,right therefore creates four nodes. The activation function used here is linear ***Linear Activation Function:*** Linear Activation Function is Neural Network basically means no activation function, but in keras we have it to just specify explicitly. **Linear activation function is similar to an Identity Function.**

TASK 2 : **Implementing Exponential Decay Formula for Epsilon:**

- In exponential decay formula we specify min and max epsilon. When epsilon value is maximum this means that the **agent will continue to explore** and we set a minimum epsilon value as well.
- We reduce the epsilon iteratively and when the max epsilon value reaches the minimum value that we have set, the agent will stop exploring and it will start exploiting to gain the rewards. **We generate random values as long as random values are less than the epsilon that is we will let the agent to explore but at the same time we will be decrementing epsilon simultaneously when random value become greater than epsilon value we stop taking random steps we start making predictions.** The exponential decay formula can be expressed as,

Exponential-decay formula for epsilon:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|}, \text{ where } \epsilon_{min}, \epsilon_{max} \in [0,1]$$

λ - speed of decay for epsilon ; $|S|$ - total number of steps

- self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * math.exp(-self.lamb * self.steps). Implementation in code is shown as above.

TASK 3 : **Implementation of Q-Function:**

- We specify two conditions here:
  - When there is no next state the Q-value is just the reward, otherwise
  - We calculate the expected discounted rewards.
  - $\mathbf{Q(s_t, a_t) = r_t + \gamma * \max_a Q(s_{t+1}, a)}$
- Learning the Q function corresponds to learning the optimal policy and finding the reliable way to estimate training value for Q given only a sequence of immediate rewards r spread out over time is accomplished through iterative approximation. We draw a table basically between the states and the actions.

2.4 *HYPER-PARAMETER TUNING:*

- There are various hyper -parameters which can be tuned for the above task, some of the hyper-parameters which are tuned are given the tabular column below,

Model No.	*No. of Episodes*	*Lambda*	*Min Epsilon*	*Max Epsilon*	*Gamma*	*Activation Function in Neural Network*	*No. of Hidden Layers*
1.	10000	0.00005	0.05	1	0.99	ReLu	2
2.	5000	0.005	0.05	1	0.99	ReLu	2
3.	15000	0.00005	0.4	0.8	0.50	ReLu	2
4.	10000	0.00005	0.05	1	0.99	Softmax	3
5.	10000	0.000001	0.01	1	0.99	ReLu	2
6.	10000	0.00005	0.05	1	0.99	ReLu	4

2.5 GRAPHS BASED ON ABOVE MODELS:

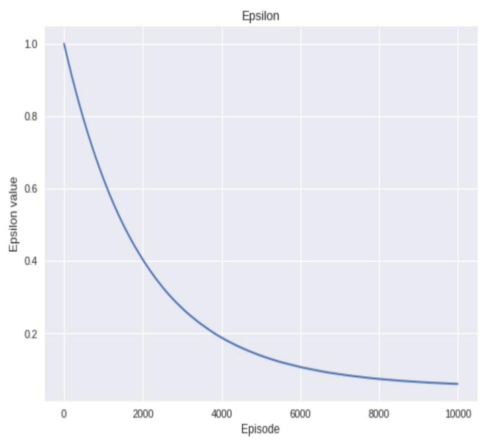


Fig 1.1 : Epsilon Value vs Episodes Model 1

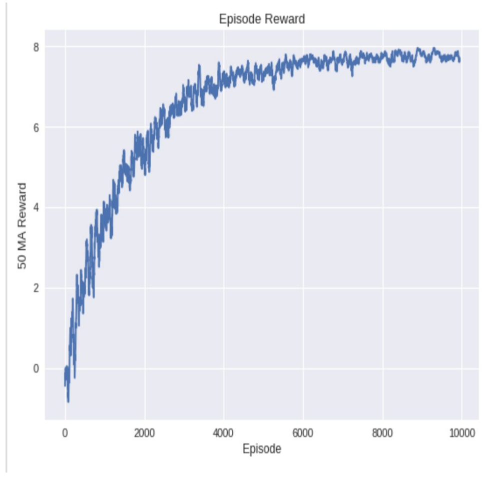


Fig 1.2 : Episode vs Rewards Model 1

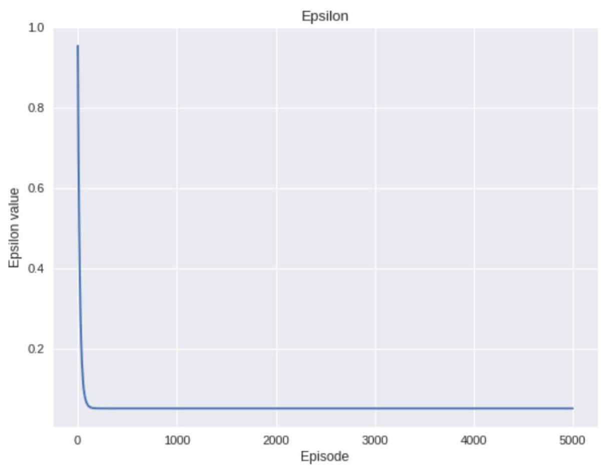


Fig 1.3 : Epsilon Value vs Episodes Model 2

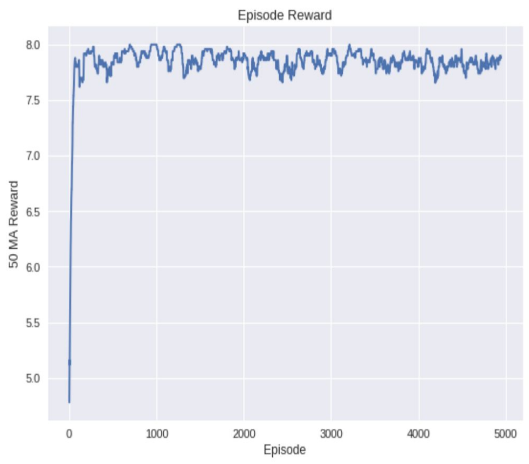


Fig 1.4 : Episode vs Rewards Model 2

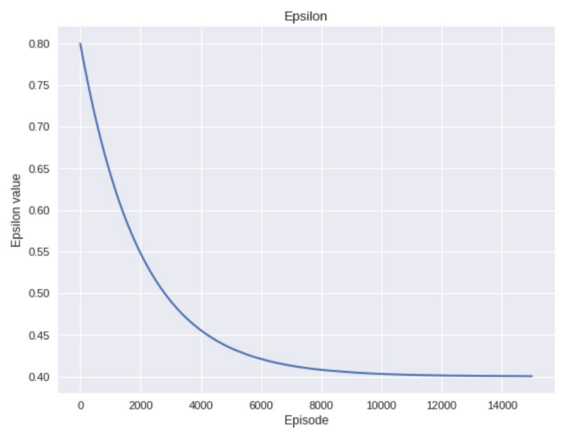
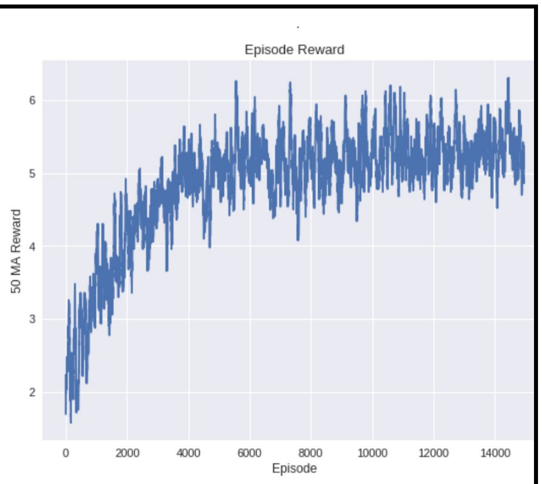


Fig 1.5 : Epsilon Value vs Episodes Model 3



Fif 1.6 : Episode vs Rewards Model 3

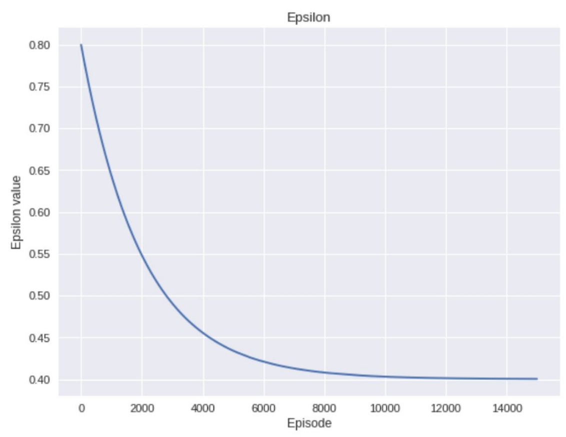


Fig 1.7 : Epsilon Value vs Episodes Model 4

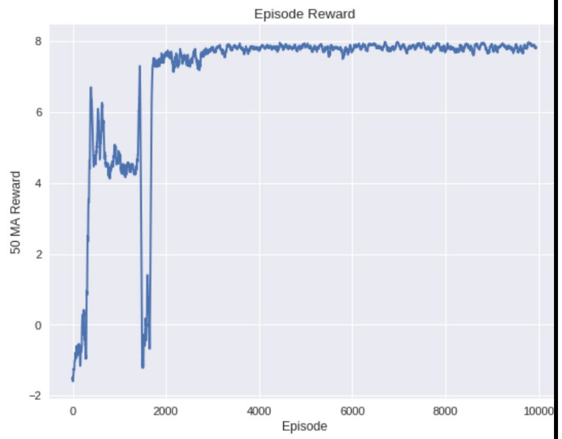


Fig 1.8 : Episode vs Rewards Model 4

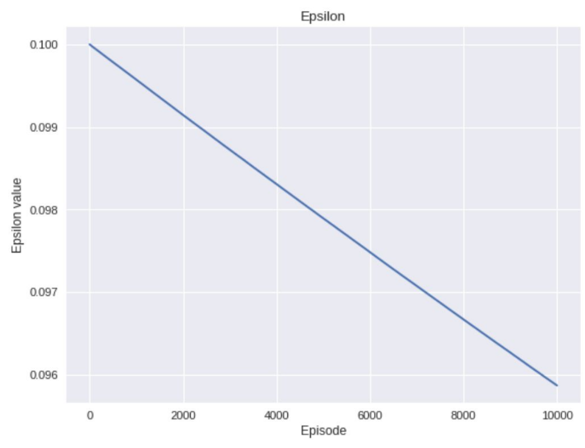


Fig 1.9 : Epsilon Value vs Episodes Model 5

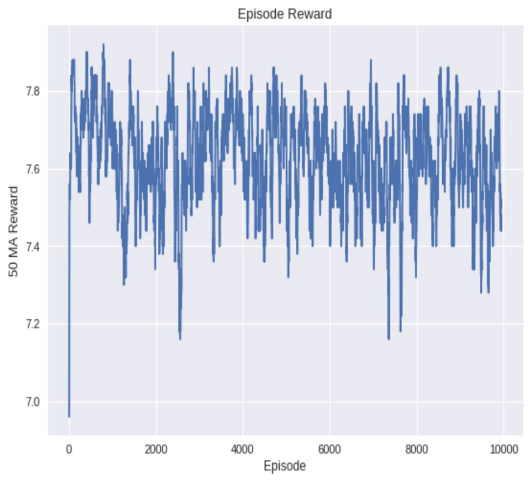


Fig 1.10 : Episode vs Rewards Model 5

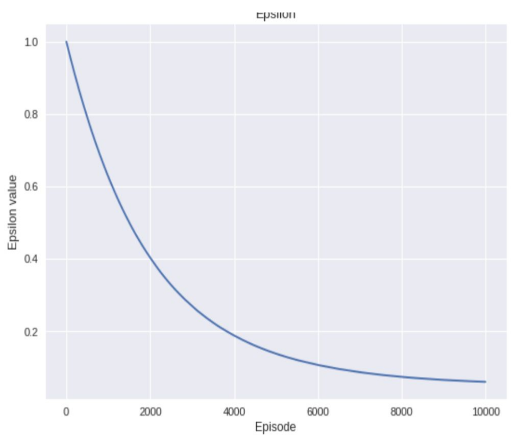


Fig 1.11 : Epsilon Value vs Episodes Model 6

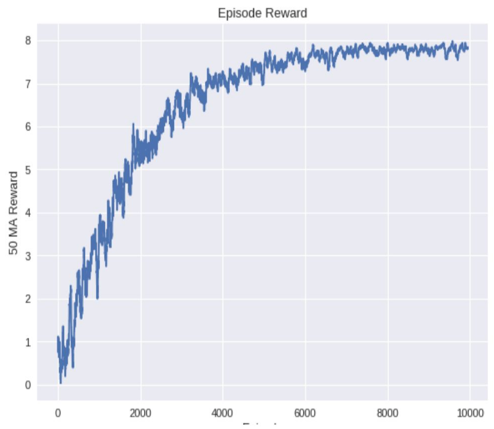


Fig 1.12 : Episode vs Rewards Model 6

1. **How Quickly was the model able to learn?**

- **At around the 6000 episode the model started exploiting the environment after exploring the environment till 6000 episodes, which is evident from the graph as the rewards become constant after 6000 episodes.**

2.6 ***<u>MODEL DESCRIPTION :</u>***

- ***<u>MODEL 1:</u>*** Model 1 is the normal implementation which was the default model given to us. The agent seems to explore the environment properly and then it starts to exploit properly.

- ***<u>MODEL 2 : (Completely Exploited Model)</u> :***
  Tuned Hyper-parameter -
  - ***Lambda - 0.05***
  - ***Episodes - 5000***

  Here in Model 2 we decrease the Lambda value from 0.00005 to 0.05 which means we increase the speed of the decay rate of epsilon which means that the agent will reach the minimum value of epsilon very quickly, ***therefore the exploration stops at a very early stage and the agent starts to exploit the environment i.e.., learning is poor in this model.***
  which is visible from the above graph

- ***<u>MODEL 3</u> : <u>(Model where exploration and exploitation is poor)</u>***
  Tuned Hyper-Parameters -
  - ***Episodes - 15000***
  - ***Min Epsilon - 0.4***
  - ***Max Epsilon - 0.8***
  - ***Gamma - 0.50***

  Here in Model 3 we increase the number of episodes but we increase the minimum epsilon and decrease the maximum epsilon value, and the discounting factor is also lowered from 0.99 to 0.50. Decreasing the maximum value and increasing the minimum value results in a very small range for the random actions to be taken. From 0.8 to 0.4 the agent will do the exploration used to randomly generated values but since the range is low there is not much exploration happening here hence here both exploration and exploitation tend to be poor.

- ***MODEL 4, Model 5 and 6:***
  Tuned Hyper-Parameters - Model 4
  - ***Activation Function - Softmax***
  - ***Number of Hidden Layers - 3***

  Tuned Hyper-Parameters - Model 5
  - ***Lambda - 0.000001***

  Lambda is reduced to a very low value so the agent tends to spend more time on exploration by generating random value before it goes into exploitation and starts predicting values.
  Tuned Hyper-Parameters - Model 6
  - ***Number of Hidden Layers - 4***

3 ***WRITING TASK:*** References for writing task:

]Simple Reinforcement Learning with tensor flow

http://home.deib.polimi.it/restelli/MyWebSite/pdf/rl5.pdf

QUESTION 1 :

1. **What happens when an agent always chooses actions that maximizes the Q-value?.**

- Learning the Q function corresponds to learning the optimal policy and finding the reliable way to estimate training value for Q given only a sequence of immediate rewards r spread out over time is accomplished through iterative approximation.
- **When an agent always chooses the action that maximizes the Q-value the agents gets stranded in non-optimal policies** and it does not explore enough to find the optimal path or optimal action from the each state.
- Say when an environment is not completely explored and if the agent always takes the action which maximizes the Q-value then the action taken may not correspond to the optimal policy. This will lead considerable poor performance in the long run.

QUESTION 2:

1. **Suggest two ways to force the agent to explore ?.**

- ***METHOD 1 FOR EXPLORATION:***

GREEDY EPSILON

In reinforcement learning and in the model we implemented we set maximum epsilon and minimum epsilon values.

Exponential-decay formula for epsilon:

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

where $\epsilon min, \epsilon max \in [0,1]$

λ - speed of decay for epsilon ; $|S|$ - total number of steps

- There are two scenarios in the model:

```
if random.random() < self.epsilon:
        if verbose:
                print("Random Action.")
                return random.randint(0, self.action_dim-1)
        else:
                actions = self.brain.predictOne(s)
                if verbose:
                        print("Actions:", actions)
                        return np.argmax(actions)
```

- From the above code it is visible that as long as the random value is less than the epsilon value the if statement is executed and the agent continues taking random actions that is agent continues its exploration.
- But based on the above formula we perform epsilon decay so that it reaches the minimum epsilon value and agent starts to predict and exploits the environment so that it reaches its goal.
- ***Therefore when we set the minimum epsilon value very low or maximize the maximum value such that the range between them is very high then the agent will continue exploring the environment till the random.random becomes greater than the epsilon value.***
- ***Another method is to reduce the epsilon decay rate (Lambda) by setting lambda as a very low value the epsilon value decays very slowly and the random.random will take a much longer time to become greater than epsilon, hence this means that eh agent will continue to explore.***

Shortcomings of Greedy Epsilon:

Despite the prevalence of greedy epsilon being used widely, this method is far from optimal, since it takes into account only whether actions are most rewarding or not.

- ***METHOD 2 FOR EXPLORATION:***

BOLTZMANN APPROACH :

- The Boltzmann approach ensures that it utilizes all information present in the Q-values produced by our network.
- In this approach instead of always taking optimal action or taking a random action, this approach involves choosing an action with weighted probabilities.
- Here we bias the explorations towards the promising actions. The softmax action selection methods grade action probabilities by estimated values.
- The softmas uses the ***Gibbs or Boltzmann distribution*** which can be expressed as,
- $\pi(a|s) = e^{Q(s,a)/t} / e^{\Sigma Q(s,a)/t}$ τ is a "computational" temperature : $\tau \rightarrow \infty$: P = 1/ |A| $\tau \rightarrow 0$: greedy
- ***Depending on the computational temperature, which acts as threshold , lower the temperature value the more unbiased the model will be and when we set the temperature as high the model will exploit more.***
- ***So here we mean to say that when the temperature threshold is set low the agent will continue to explore environment extensively.***

Name: VENKTESH KAVIARASA
50289400

QUESTION-2 WRITING TASK

Q-Table Calculation:

STEP1: The initial step is to fill the table in a bottom-up apway. The states here are s_0, s_1, s_2, s_3 and s_4.

Actions

states	UP	DOWN	LEFT	RIGHT
s_0	0		0	
s_1	0.			
s_2				
s_3		1		0
s_4	0	0	0	0

State s_4: In s_4 the goal is already reached, so there is no action required hence all the actions here are updated as 0.

State s_3: In s_3 if we take a down action we will get the rewards as 1 as we have reached the goal. as we do not know the s_3 right action is 0 (reward). next state.

State s_2: All actions are possible

State s_1: s_1 down left and right are possible. s_1(up) action is 0 as the next state is unknown

State s_0: s_0 up and ~~dofo~~ actions are of zero reward.

Q- TABLE CALCULATION :- STEP 2 (INTERMEDIATE)

ACTIONS

STATE	UP	DOWN	LEFT	RIGHT
S_0	0	3.9403	0	3.9403
S_1	0	2.9701	2.9008	2.9701
S_2	1.9403	1.99	1.9403	1.99
S_3	0.9701	1	0.9701	0
S_4	0	0	0	0

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

Starting from S4, we fill the values in the Q-table.

Initially for state S4 : since it is our goal we need not move/make any action hence it is all 0's

Rewards:

+ve Reward = 1

-ve Reward = -1

No Reward = 0.

For State 3:

for s_3 Actions possible are:

(i) $Q(s_3, a_{up})$ (ii) $Q(s_3, a_{left})$

$Q(s_3, a_{up}) = -1 + 0.99 \times \max_a Q(s_2, a) \longrightarrow ①$

It is $\max_a Q(s_2, a)$ because $Q(s_3, a_{up})$ is symmetric w.r.t $Q(s_2, a)_{up}$.

$Q(s_3, a_{left}) = -1 + 0.99 \times \max_a Q(s_2, a) \longrightarrow ②$.

for state 2:

for s_2 actions possible are:

(i) $Q(s_2, a_{up})$, $Q(s_2, a_{down})$, $Q(s_2, a_{left})$, $Q(s_2, a_{right})$.

$Q(s_2, a_{up}) = -1 + 0.99 \times \underbrace{\max_a Q(s_1, a)}_{\text{By symmetry}} \longrightarrow ③$

$Q(s_2, a_{down}) = 1 + 0.99 \times \max_a Q(s_3, a) \longrightarrow ④$

$Q(s_2, a_{left}) = -1 + 0.99 \times \max_a Q(s_1, a) \longrightarrow ⑤$

$Q(s_2, a_{right}) = 1 + 0.99 \times \max_a Q(s_3, a) \longrightarrow ⑥$

For state 1:

For s_1 actions possible are:

(i). $Q(s_1, a_{down})$ (ii). $Q(s_1, a_{left})$ (iii). $Q(s_1, a_{right})$

$Q(s_1, a_{down}) = 1 + 0.99 \times \max_a Q(s_2, a) \longrightarrow ⑦$

$Q(s_1, a_{left}) = -1 + 0.99 \times \max_a Q(s_0, a). \longrightarrow ⑧$

$Q(s_1, a_{right}) = 1 + 0.99 \times \max_a Q(s_2, a) \longrightarrow ⑨$

for state 0:

for So Actions possible are :

(i). $Q(S_0, a_{down})$ (ii). $Q(S_0, a_{right})$

$Q(S_0, a_{down}) = 1 + 0.99 \times \max_a Q(S_1, a) \longrightarrow \textcircled{10}$

$Q(S_0, a_{right}) = 1 + 0.99 \times \max_a Q(S_1, a) \longrightarrow \textcircled{11}$

CalcuLATIONS:

We know the value of $\max_a (S_3, 0)$

so using equations $\textcircled{4}$ & $\textcircled{6}$,

$Q(S_2, a_{down}) = 1 + 0.99 \times 1$

$Q(S_2, a_{down}) \Rightarrow 1.99$

$Q(S_2, a_{right}) = 1.99$

NOW,

$Q(S_3, a_{up}) = -1 + 0.99 \times (1.99) = 0.9701$

$Q(S_3, a_{left}) = 0.9701$

$Q(S_1, a_{down}) = 1 + 0.99 \times 1.99 = 2.9701$

$Q(S_1, a_{right}) = 1 + 0.99 \times 1.99 = 2.9701$

Now we got S_1 max values,

$Q(S_2, a_{up}) = -1 + 0.99 \times 2.9701 \Rightarrow 1.9403$

$Q.(S_2, a_{left}) = -1 + 0.99 \times 2.9701 \Rightarrow 1.9403$

$Q(S_0, a_{down}) = 1 + 0.99 \times 2.9701 = 3.9403$

$Q(S_0, a_{right}) = 1 + 0.99 \times 2.9701 = 3.9403$

finally using somax value,

$Q(s_1, a_{left}) = -1 + 0.99 \times 3.9403$

$= 2.9008$

Q-Table (FINAL STEP):

Actions

States	up	down	left	right
S_0		3.9403		3.9403
S_1		2.9701	2.9008	2.9701
S_2	1.9403	1.99	1.9403	1.99
S_3	0.9701	1	0.9701	
S_4	0	0	0	0

STATE S3:

Action right :- $Q(S_3, a_{right}) = 0 + 0.99 \times max_a Q(S_3, a)$.

$= 0.99 \times (1) \Rightarrow \underline{\underline{0.99}}$

STATE S1:

Action up :- $Q(S_1, a_{up}) = 0 + 0.99 \times max_a Q(S_1, a)$

$= 0.99 \times 2.9701 \Rightarrow 2.9404$

STATE S0:

Action up :- $Q(S_0, a_{up}) = 0 + 0.99 \times Q(S_0, a)$

$= 0.99 \times 3.9403 = 3.9008$.

Action left :- $Q(S_0, a_{left}) = 0 + 0.99 \times Q(S_0, a)$

$= 0.99 \times 3.9403 = 3.9008$

FINAL Q-TABLE :

Actions.

States	UP	DOWN	LEFT	RIGHT
S0	3.9008	3.9403	3.9008	3.9403
S1	2.9404	2.9701	2.9008	2.9701
S2	1.9403	1.99	1.9403	1.99.
S3	0.9701	1	0.9701	0.99
S4	0	0	0	0