CSL 303 : Artificial Intelligence

TUTORIAL ASSIGNMENTS 8 and 9

Reasoning Under Uncertainty

Date Assigned: 29^{th} October, 2021 **Date Submitted :** 11^{th} November, 2021

Submitted by:

Name: S. Bhavyesh

Roll Number: 191210045

Branch: Computer Science and Engineering

Semester: 5^{th}

Submitted to: Dr. Chandra Prakash

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI
A-7, Institutional Area, near Satyawadi Raja Harish Chandra Hospital, New Delhi, Delhi 110040

CONTRIBUTION

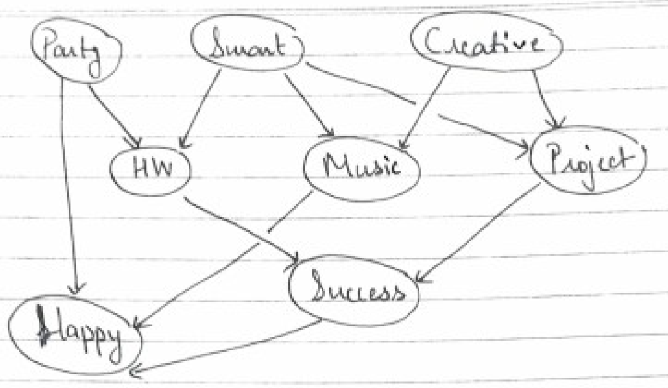
For this tutorial, I partnered up with Udit Kumar(191210051). Actually, we did all the questions independently, but had discussions on some questions. For example, we discussed questions 8.1.3, 8.1.4 and 8.1.5.

PART A: Bayesian Network Representation and Inference (20 Points)

8.1.1 Understanding The Model [4]:

(a) Draw the Bayesian network.
(b) Write joint distribution as a product of conditional probabilities.
(c) What is the number of independent parameters needed for each conditional probability table?
(d) What is the total number of independent parameters?

8.1.1 (a)



(b) P(party, smart, creative, HW, music, project, success, happy)

= P(party) × P(smart) × P(creative) × P(HW | smart, party)
× P(music | creative, smart) × P(project | creative, smart)
× P(success | HW, project) × P(happy | party, music, success)

(c) Number of parameters for each table is given:

→ P(creative = T) = 1 → P(party = T) = 1 → P(smart = T) = 1
→ P(HW = T | smart, party) = 4. → P(music = T | creative, smart) = 4
→ P(project = T | creative, smart) = 4 → P(success = T | HW, proj) = 4
→ P(happy = T | party, music, success) = 8

(d) Total independent parameters —

$1 + 1 + 1 + 4 + 4 + 4 + 4 + 8 = \boxed{27}$

8.1.2 D-Separation [3]:

(a) Using only the Bayesian network structure from part 8.1.1, answer the following True/False questions and provide a brief explanation:

1. Party is independent of Success given HW.
2. Party is independent of Smart given Success.
3. Party is independent of Creative given Happy.

8.1.2 (a) FALSE.

There is an active path from party to success via HW:

Party — HW — Smart — Project — Success.

Two variables are independent iff there exists no active path between the two.

(b) FALSE

There is an independent path from Party to Smart via success:

Party — HW — Smart

↑ linked to success and direct descendant

(c) FALSE.

There is an active path from Party to Creative via Happy:

Party — Happy — Music — Creative.

8.1.3 Confounded Intelligence [2]:

(a) Using only the data in students.csv and Matlab calculate the correlation between success on the homework HW and success on the project Project. You do not need to use the Bayesian network for this question. (Hint: Consider using the numpy.cov() function in Python)

(b) From the model structure, identify a potential common cause variable which may explain the correlation between HW and Project.

CODE and OUTPUT:

In [2]:
```
import pandas as pd
import numpy as np
from prettytable import PrettyTable

# read data from .csv file and create a dataframe
df = pd.read_csv('Tut_8_student.csv', sep = ',', header = None)
```

In [3]:
```
# Question 3 part(a)
hw = df[3].to_numpy()
proj = df[5].to_numpy()

# determine covariance matrix
cov = np.cov(hw,proj)

# calculate correlation
corr = cov[1][0]/np.sqrt(cov[1][1]*cov[0][0])

print(corr)
```

```
0.35574057475116566
```

```
Question 3 part(b)

The variable 'Smart' can be seen as the common link between
'HW' and 'Project'. This can be easily inferred from the Bayesian net
that has been created in 8.1.1 part(a)
```

OBSERVATION/COMMENTS:

1. The formula of correlation is as follows:

$$\boxed{\mathrm{COR}(X,\, Y) = \frac{\mathrm{COV}(X,\, Y)}{\sqrt{\mathrm{VAR}(X)\mathrm{VAR}(Y)}}}$$

2. numpy.cov(), for n variables, returns an n x n matrix *cov*, where *cov[i][j]* represents the covariance between two variables x_i and x_j. In this case, 2 x 2 matrix is returned where the off diagonal elements represent the covariance between HW and Project.

3. Covariance of a variable with itself is the Variance of the variable(Cov(X,X) = Var(X)).

8.1.4 Counting [4] :

(a) Use python and students.csv to calculate the parameters for each conditional probability table by counting with Laplace smoothing. Please consider formatting your conditional probability tables as shown in Table 1.

CODE and OUTPUT:

```
In [3]: # Question 4

table1 = PrettyTable(["Creative","P(Creative = 1)"]) # define table for P(creative)
p_creative = (sum(df[2]==1)+1)/(len(df)+2) # count the instances in dataframe where true,
                                           # then apply Laplace smoothing.
table1.add_row(["1",p_creative])


table2 = PrettyTable(["Smart","P(Smart = 1)"]) # define table for P(smart)
p_smart = (sum(df[1]==1)+1)/(len(df)+2)    # count the instances in dataframe where true,
                                           # then apply Laplace smoothing.
table2.add_row(["1",p_smart])


table3 = PrettyTable(["Party","P(Party = 1)"]) # define table for P(smart)
p_party = (sum(df[0]==1)+1)/(len(df)+2)    # count the instances in dataframe where true,
                                           # then apply Laplace smoothing.
table3.add_row(["1",p_party])
```

```
# define the 2 possible states of each variable(true/false)
state=[1,0]


# define table for P(Project = 1 | Creative, Smart)
table4 = PrettyTable(["Creative","Smart","P(Project = 1 | Creative, Smart)"])
p_project_given_creative_smart = []
for i in state:
    for j in state:
        temp = (df[2] == i) & (df[1] == j) # Creative & smart
        # P(Project = 1 | Creative, Smart) + Laplace Smoothing
        p_project_given_creative_smart.append( ((sum((df[5] == 1) & temp)) + 1) / (sum(temp) + 2))

        # add row to the table
        table4.add_row([i,j,p_project_given_creative_smart[-1]])


# define table for P(Music = 1 | Creative, Smart)
table5 = PrettyTable(["Creative","Smart","P(Music = 1 | Creative, Smart)"])
p_mac_given_creative_smart = []
for i in state:
    for j in state:
        temp = (df[2] == i) & (df[1] == j) # Creative & smart
        # P(Project = 1 | Creative, Smart) + Laplace Smoothing
        p_mac_given_creative_smart.append( ((sum((df[4] == 1) & temp)) + 1) / (sum(temp) + 2))

        # add row to the table
        table5.add_row([i,j,p_mac_given_creative_smart[-1]])
```

```
# define table for P(HW = 1 | Smart, Party)
table6 = PrettyTable(["Smart","Party","P(HW = 1 | Smart, Party)"])
p_hw_given_smart_party = []
for i in state:
    for j in state:
        temp = (df[1] == i) & (df[0] == j) # Smart & Party
        # P(HW = 1 | Smart, Party) + Laplace Smoothing
        p_hw_given_smart_party.append( ((sum((df[3] == 1) & temp)) + 1) / (sum(temp) + 2))

        # add row to the table
        table6.add_row([i,j,p_hw_given_smart_party[-1]])


# define table for P(Success = 1 | Project, HW)
table7 = PrettyTable(["Project","HW","P(Success = 1 | Project, HW)"])
p_success_given_project_hw = []
for i in state:
    for j in state:
        temp = (df[5] == i) & (df[3] == j) # Project & HW
        # P(Success = 1 | Project, HW) + Laplace Smoothing
        p_success_given_project_hw.append( ((sum((df[6] == 1) & temp)) + 1) / (sum(temp) + 2))

        # add row to the table
        table7.add_row([i,j,p_success_given_project_hw[-1]])
```

```
# define table for P(Happy = 1 | Success, Music, Party)
table8 = PrettyTable(["Success","Music","Party","P(Happy = 1 | Success, Music, Party)"])
p_happy_given_success_mac_party = []
for i in state:
    for j in state:
        for k in state:
            temp = (df[6] == i) & (df[4] == j) & (df[0] == k) # Success & Music & Party
            # P(Happy = 1 | Success, Music, Party) + Laplace Smoothing
            p_happy_given_success_mac_party.append( ((sum((df[7] == 1) & temp)) + 1) / (sum(temp) + 2))

            # add row to the table
            table8.add_row([i,j,k,p_happy_given_success_mac_party[-1]])
```

```
In [4]: print(table1,"\n",table2,"\n",table3,"\n",table4,"\n",table5,"\n",table6,"\n",table7,"\n",table8,"\n")
```

Creative	P(Creative = 1)
1	0.6993202718912435

Smart	P(Smart = 1)
1	0.704718112754898

Party	P(Party = 1)
1	0.6021591363454618

Creative	Smart	P(Project = 1 | Creative, Smart)
1	1	0.9048393655957706
1	0	0.4030710172744 7214
0	1	0.7932647333956969
0	0	0.10730593607305935

Creative	Smart	P(Music = 1 | Creative, Smart)
1	1	0.6856445709638064
1	0	0.8963531669865643
0	1	0.41347053320860616
0	0	0.1232876712328767

Smart	Party	P(HW = 1 | Smart, Party)
1	1	0.8025151374010246
1	0	0.8979000724112962
0	1	0.0944700460829493
0	0	0.3055555555555556

Project	HW	P(Success = 1 | Project, HW)
1	1	0.8963323353293413
1	0	0.2073732718894009
0	1	0.30714285714285716
0	0	0.05066079295154185

Success	Music	Party	P(Happy = 1 | Success, Music, Party)
1	1	1	0.9584199584199584
1	1	0	0.3583662714097497
1	0	1	0.7208201892744479
1	0	0	0.3076923076923077
0	1	1	0.4923413566739606
0	1	0	0.20618556701030927
0	0	1	0.4204322200392927
0	0	0	0.09646302250803858

OBSERVATION/COMMENTS:

1. The following rule of probability was used to compute the values in the tables:

$$\mathrm{P}\left(\frac{A}{B}\right) = \frac{P\ (A \cap B)}{P\ (B)}$$

2. Laplace smoothing is used to prevent the case of 0/0 when the test object has not been encountered during training. And, assigning zero probability to a word we haven't encountered yet is not a good option.

Suppose our original probability was X/Y. With Laplace smoothing, the probability changes to (X+1)/(Y+K), where k is the total number of possible values the random variable can assume.

8.1.5 Inference [7] : With your conditional probability table estimates, calculate the following probabilities:

(a) What is the probability of being happy?
(b) What is the probability of being happy given that you party often, are wicked smart, but not very creative?
(c) What is the probability of being happy given that you are wicked smart and very creative?
(d) What is the probability of being happy given you do not party, and do well on all your homework an class project?
(e) What is the probability of being happy given you own a mac?
(f) What is the probability that you party often given you are wicked smart?
(g) What is the probability that you party often given you are wicked smart and happy?

CODE and OUTPUT:

Edit View Insert Cell Kernel Help

Run Code

```
In [21]: # Answers to 8.1.5

# part (a)
part_a = (sum(df[7] == 1) + 1)/(len(df) + 2)
print("part (a): ", part_a)

# part (b)
temp = (df[0]==1) & (df[1]==1) & (df[2]==0)
part_b = (sum((df[7] == 1) & temp)+1)/(sum(temp) + 2)
print("part (b): ", part_b)

# part (c)
temp = (df[1]==1) & (df[2]==1)
part_c = (sum((df[7] == 1) & temp)+1)/(sum(temp) + 2)
print("part (c): ", part_c)

# part (d)
temp = (df[0]==0) & (df[3]==1) & (df[5]==1)
part_d = (sum((df[7] == 1) & temp)+1)/(sum(temp) + 2)
print("part (d): ", part_d)

# part (e)
temp = (df[4]==1)
part_e = (sum((df[7] == 1) & temp)+1)/(sum(temp) + 2)
print("part (e): ", part_e)

# part (f)
# Since party is independent of smart, the answer will be P(party=1)
part_f = p_party
print("part (f): ", part_f)

# part (g)
temp = (df[1]==1) & (df[7]==1)
part_g = (sum((df[0] == 1) & temp)+1)/(sum(temp) + 2)
print("part (g): ", part_g)
```

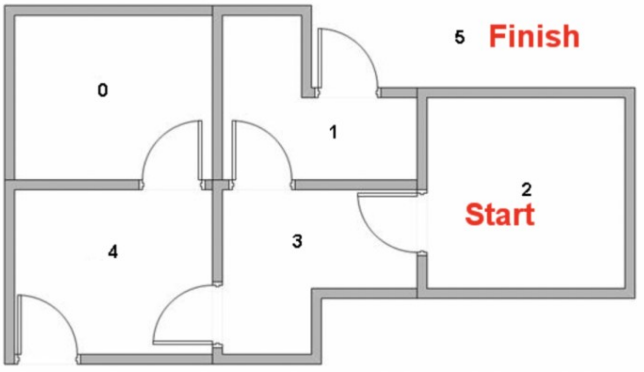
```
part (a):  0.5145941623350659
part (b):  0.703875968992248
part (c):  0.5758438389589264
part (d):  0.3157894736842105
part (e):  0.5590879897238279
part (f):  0.6021591363454618
part (g):  0.7995961635537607
```

OBSERVATION/COMMENTS:

While doing on pen-and-paper, I faced a lot of difficulty in solving because there were too many factors to account for, in each part. So, I simply enumerated cases from the dataframe, and applied Laplace smoothing, instead of using the conditional probability tables. I was searching and learned that the questions can be solved using special Python packages. One such package is pgmPy. But I did not have enough time to explore the package hence I enumerated cases.

PART B : Reinforcement Learning : PathFinder Bot

As discussed in the class suppose we have 5 rooms A to E, in a building connected by certain doors :



We can consider outside of the building as one big room say F to cover the building. There are two doors lead to the building from F, that is through room B and room E.Modeling the environment that can be used for Reinforcement Learning for finding the best possible path. Fill the code in the shared 8 2 RL e xample.ipynb

CODE and OUTPUTS:

File Edit View Insert Runtime Tools Help Last edited on 29 October

```
Rewards = np.matrix([ [-1,-1,-1,-1,0,-1],
                      [-1,-1,-1,0,-1,100],
                      [-1,-1,-1,0,-1,-1],
                      [-1,0,0,-1,0,-1],
                      [0,-1,-1,0,-1,100],
                      [-1,0,-1,-1,0,100]
                     ])

Rewards
```

```
matrix([[ -1,  -1,  -1,  -1,   0,  -1],
        [ -1,  -1,  -1,   0,  -1, 100],
        [ -1,  -1,  -1,   0,  -1,  -1],
        [ -1,   0,   0,  -1,   0,  -1],
        [  0,  -1,  -1,   0,  -1, 100],
        [ -1,   0,  -1,  -1,   0, 100]])
```

```
# Q matrix: zero matrix  of size same as R matrix

Q = np.zeros((Rewards.shape[0],Rewards.shape[1]))
Q
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

```
# Gamma (learning parameter).
gamma = 0.8
```

```
# Initial state. (Usually to be chosen at random)
initial_state = 1

# Write your Code to choose random State
import random
initial_state = random.randint(0,Rewards.shape[0]-1)
initial_state
```

```
2
```

```python
# This function updates the Q matrix according to the path selected and the Q
# learning algorithm
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[0]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]# WRITE YOUR CODE HERE

    # Q learning formula
    Q[current_state, action] = Rewards[current_state, action] + gamma * max_value

# Update Q matrix
update(initial_state,action,gamma)
```

```python
for i in range(10000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)# WRITE YOUR CODE HERE )
    action = sample_next_action(available_act)# WRITE YOUR CODE HERE )
    score= update(current_state,action,gamma)

    # The "trained" Q matrix
print("The Trained Q matrix:")
print(Q)


# Normalize the "trained" Q matrix
print("Trained Normalized Q matrix:")
Q_nor=Q/np.max(Q)*100 # WRITE YOUR CODE HERE
print(Q_nor), i
```

```
The Trained Q matrix:
[[  0.   0.   0.   0. 400.   0.]
 [  0.   0.   0. 320.   0. 500.]
 [  0.   0.   0. 320.   0.   0.]
 [  0. 400. 256.   0. 400.   0.]
 [320.   0.   0. 320.   0. 500.]
 [  0. 400.   0.   0. 400. 500.]]
Trained Normalized Q matrix:
[[  0.    0.    0.    0.   80.    0. ]
 [  0.    0.    0.   64.    0.  100. ]
 [  0.    0.    0.   64.    0.    0. ]
 [  0.   80.   51.2   0.   80.    0. ]
 [ 64.    0.    0.   64.    0.  100. ]
 [  0.   80.    0.    0.   80.  100. ]]
(None, 9999)
```

```python
#-----------------------------------------------------------------------------
# Testing

#STATES = [A,B,C,D,E,F]
#nO_State=[0,1,2,3,4,5]

# Goal state = 5
# Best sequence path starting from 2 -> 2, 3, 1, 5

current_state = 2
current_state = random.randint(0,Rewards.shape[0]-1)
print(current_state)
steps = [current_state]

while current_state != 5:

    next_step_index = np.where(Q[current_state,] == np.max(Q[current_state,]))[0]

    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index
```

```
0
```

```
[ ] # Print selected sequence of steps
    print("Selected path:")
    print(steps)
```

```
Selected path:
[0, 4, 5]
```

OBSERVATION/COMMENTS

1. The code has been made as generalized as possible so that it can be readily used for different scenarios(different reward matrices), without much modification.

2. In a couple of places, I had to change the index of np.where() module from 1 to 0. Earlier, the program wasn't executing.

3. The code can be divided into two parts: training (where we learn the Q matrix by running multple episodes) and testing(where we give an input to the model and determine the optimal sequence of actions based on the learned Q-matrix values).

PART C : Reinforcement Learning in Pacman

Question 8.3.1 (10 points): Value Iteration

CODE:

```
        # Write value iteration code here
        "*** YOUR CODE HERE ***"
        for i in range(self.iterations):

            # store a copy of the current dictionary, so that we can update it
            updated = self.values.copy()

            for state in mdp.getStates():

                # arbitrarily small value
                temp = -100000000

                if not mdp.isTerminal(state): # terminal state has zero reward by convention
                    for action in mdp.getPossibleActions(state):

                        # find max value of all possible actions
                        temp = max(temp,self.computeQValueFromValues(state,action))

                    updated[state] = temp # update value corresponding to the state

            # store updated dictionary
            self.values = updated
```

```
    def computeQValueFromValues(self, state, action):
        """
          Compute the Q-value of action in state from the
          value function stored in self.values.
        """
        "*** YOUR CODE HERE ***"
        Q = 0
        temp = self.mdp.getTransitionStatesAndProbs(state, action)

        # apply the Q-value formula
        for i in temp:
            Q += i[1]*(self.mdp.getReward(state,action,i[0])+self.discount*self.values[i[0]])

        return Q
```

```
    def computeActionFromValues(self, state):
        """
          The policy is the best action in the given state
          according to the values currently stored in self.values.

          You may break ties any way you see fit.  Note that if
          there are no legal actions, which is the case at the
          terminal state, you should return None.
        """
        "*** YOUR CODE HERE ***"
        # arbitrarily small score, and no action, for now
        best_act = [-100000000, None]
        for action in self.mdp.getPossibleActions(state):

            # if an action exists with better Q-value, declare it the best one so far
            if best_act[0] < self.computeQValueFromValues(state, action):
                best_act[0] = self.computeQValueFromValues(state, action)
                best_act[1] = action

        # return action corresponding to best Q-value
        return best_act[1]
```

OUTPUT:

```
Question q1
===========

*** PASS: test_cases/q1/1-tinygrid.test
*** PASS: test_cases/q1/2-tinygrid-noisy.test
*** PASS: test_cases/q1/3-bridge.test
*** PASS: test_cases/q1/4-discountgrid.test

### Question q1: 6/6 ###
```

OBSERVATION/COMMENTS:

Initially, I was making a mistake in the value-iteration method. I was directly operating on the *self.values* dictionary instead of taking its copy and modifying it. It took me some time to debug that problem

Question 8.3.2 (5 point): Bridge Crossing Analysis

CODE:

```
######################
# ANALYSIS QUESTIONS #
######################

# Set the given parameters to obtain the specified policies through
# value iteration.

def question2():
    answerDiscount = 0.9
    answerNoise = -0.2 # updated this value from 0.2 to -0.2
    return answerDiscount, answerNoise
```

OUTPUT:

```
Question q2
===========

*** PASS: test_cases/q2/1-bridge-grid.test

### Question q2: 1/1 ###
```

OBSERVATION/COMMENTS:

The agent was unsuccessful because there was some noise in every step (*answerNoise*). To ensure that the agent reaches safely, I felt that the noise factor should be eliminated or made negative.

Question 8.3.3 (10 points): Policies

CODE:

```
def question3a():
    answerDiscount = 1.0
    answerNoise = 0
    answerLivingReward = -5.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3b():
    answerDiscount = 0.1
    answerNoise = 0.1
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -1.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3d():
    answerDiscount = 0.9
    answerNoise = 0.2
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3e():
    answerDiscount = 10.0
    answerNoise = 0
    answerLivingReward = 100000000
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

OUTPUT:

```
Question q3
===========

*** PASS: test_cases/q3/1-question-3.1.test
*** PASS: test_cases/q3/2-question-3.2.test
*** PASS: test_cases/q3/3-question-3.3.test
*** PASS: test_cases/q3/4-question-3.4.test
*** PASS: test_cases/q3/5-question-3.5.test

### Question q3: 5/5 ###
```

OBSERVATION/COMMENTS:

Below is the intuition for each part(a-e):

1. Risking the cliff means that agent’s life will have a negative value, but not too negative.
2. Avioding the cliff means that the agent’s life will now hold some value.
3. Distant exit means that the agent must take the longer path without the fear of decay(hence decay is 1) and risking the cliff means agent’s life will have a small value.
4. Avoiding the cliff means that agent’s life will now have a considerable value.
5. Infinite episode implies decay > 1 means reward value goes up by each move, and agent’s life has infinite value.

Question 8.3.4 (10 points): Q-Learning

CODE:

```python
    def __init__(self, **args):
        "You can initialize Q-values here..."
        ReinforcementAgent.__init__(self, **args)

        "*** YOUR CODE HERE ***"
        # to keep record of state, action and the value
        # (similar to self.values in valueIterationAgents.py)
        self.QVal = util.Counter()

    def getQValue(self, state, action):
        """
          Returns Q(state,action)
          Should return 0.0 if we have never seen a state
          or the Q node value otherwise
        """
        "*** YOUR CODE HERE ***"
        # simply return Q-value based on the state and action
        return self.QVal[(state, action)]
```

```python
    def computeValueFromQValues(self, state):
        """
          Returns max_action Q(state,action) where the max is over legal actions.  Note that if
          there are no legal actions, which is the case at the terminal state, you should return a v
        """
        "*** YOUR CODE HERE ***"
        temp = -100000000 # arbitrarily small value

        # choose the best action based on Q-value
        for action in  self.getLegalActions(state):
            temp = max(temp, self.getQValue(state, action))

        # If no action exists, return 0 else the value obtained
        if temp == -100000000:
            temp = 0

        return temp

    def computeActionFromQValues(self, state):
        """
          Compute the best action to take in a state.  Note that if there
          are no legal actions, which is the case at the terminal state, you should return None.
        """
        "*** YOUR CODE HERE ***"
        # initially no action 'None' and very small Q-value '-100000000'
        best_act = [-100000000, None]

        # find action with best Q-value
        for action in self.getLegalActions(state):
            if best_act[0] < self.getQValue(state,action):
                best_act[0] = self.getQValue(state,action)
                best_act[1] = action

        # return action corresponding to best Q-value
        return best_act[1]
```

```python
    def update(self, state, action, nextState, reward):
        """
          The parent class calls this to observe a
          state = action => nextState and reward transition.
          You should do your Q-Value update here

          NOTE: You should never call this function,
          it will be called on your behalf
        """
        "*** YOUR CODE HERE ***"
        # simply apply formula and find the
        # expected Q-value based on self.alpha

        temp = reward + self.discount * self.computeValueFromQValues(nextState)
        self.QVal[(state,action)] = self.alpha * temp + (1-self.alpha) * self.QVal[(state,action)]
```

OUTPUT:

```
Question q4
===========

*** PASS: test_cases/q4/1-tinygrid.test
*** PASS: test_cases/q4/2-tinygrid-noisy.test
*** PASS: test_cases/q4/3-bridge.test
*** PASS: test_cases/q4/4-discountgrid.test

### Question q4: 5/5 ###
```

OBSERVATION/COMMENTS:

In the update() method, earlier I was computing the Qvalue for the current state, instead of the next state. The debugging process took a lot of time. Also the coding was a bit easy because it is similar to what I coded in the file *valueIterationAgents.py* .

Question 8.3.5(5 points): Epsilon Greedy

CODE:

```
    def getAction(self, state):
        """
          Compute the action to take in the current state.  With
          probability self.epsilon, we should take a random action and
          take the best policy action otherwise.  Note that if there are
          no legal actions, which is the case at the terminal state, you
          should choose None as the action.

          HINT: You might want to use util.flipCoin(prob)
          HINT: To pick randomly from a list, use random.choice(list)
        """
        # Pick Action
        legalActions = self.getLegalActions(state)
        action = None

        "*** YOUR CODE HERE ***"
        # if legal actions exist
        if legalActions:
            if util.flipCoin(self.epsilon): # if the coin flip declares random choice
                action = random.choice(legalActions) # choose a random action from the valid ones
            else:
                action = self.getPolicy(state) # otherwise, determine action based on policy
        else:
            pass # ignore if no legal actions exist

        return action
```

OUTPUT:

```
Question q5
===========

*** PASS: test_cases/q5/1-tinygrid.test
*** PASS: test_cases/q5/2-tinygrid-noisy.test
*** PASS: test_cases/q5/3-bridge.test
*** PASS: test_cases/q5/4-discountgrid.test

### Question q5: 3/3 ###
```

OBSERVATION/COMMENTS:

If legal actions exist, then the next choice (random, or policy) mostly depends on the epsilon value. Small value of epsilon means that optimal actions(exploitation) will be chosen more often, while large epsilon means random actions(exploration) are preferred.

Question 8.3.6 (5 point): Q-Learning and Pacman

CODE:

Implementation uses the code written in *qlearningAgents.py,* so no extra code is needed.

OUTPUT:

```
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 500.48
        Average Rewards for last 100 episodes: 500.48
        Episode took 1.53 seconds
Average Score: 500.48
Scores:        503.0, 499.0, 503.0, 499.0, 503.0, 499.0, 495.0, 503.0, 503.0, 499.0, 499.0, 495.0, 503.0, 503.0, 499.0, 495.0, 503.0, 495.0, 495.0, 49
5.0, 499.0, 503.0, 495.0, 499.0, 499.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 495.0, 503.0
, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 499.0, 503.0, 495.0, 495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 495.0, 499.0, 503.0, 503.0, 503.0, 503.0, 5
03.0, 503.0, 503.0, 495.0, 503.0, 495.0, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 499.0, 503.0, 495.0, 503.0, 495.0, 495.0, 503.0, 495.0, 503.0, 495.
0, 503.0, 495.0, 499.0, 495.0, 503.0, 499.0, 503.0, 495.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0
Win Rate:      100/100 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q7/grade-agent.test (1 of 1 points)
***     Grading agent using command:  python pacman.py -p PacmanQAgent -x 2000 -n 2100 -l smallGrid -q -f --fixRandomSeed
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q7: 1/1 ###
```

OBSERVATION/COMMENTS:
It took me no further optimizations to arrive at a perfect score of 100 wins out of 100 games. A reason why one may not get perfect performance could be that the corner cases (ex. No legal actions, or unseen actions) have not been properly handled.

AUTOGRADER OUTPUT:

```
Finished at 9:35:35

Provisional grades
==================
Question q1: 6/6
Question q2: 1/1
Question q3: 5/5
Question q4: 5/5
Question q5: 3/3
Question q6: 0/1
Question q7: 1/1
Question q8: 0/3
------------------
Total: 21/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

bridges@bridges-CF-C2AHCCZC7:~/Downloads/TA_8_reinforcement/TA_8_reinforcement$ 
```

NOTE:

1. If we visit the website **http://ai.berkeley.edu/reinforcement.html**, we can see that all the 8 questions have been mentioned. But, for this assignment, only 6 questions were mentioned in the tutorial sheet **https://cprakash86.files.wordpress.com/2021/10/ai_tut_8.pdf**, probably because only those questions were considered whose concepts were taught in class. That is why I did not attempt the remaining 2 questions.

2. Question 7 in the above-mentioned ai.berkeley website corresponds to 6th question given in the tute sheet. That's why for Q6 in tute sheet, the autograder evaluates Q7.