Iowa State University

Department of Electrical and Computer Engineering

Deep Machine Learning: Theory and Practice

EE 526X

Homework 5

Author:
Vishal Deep

Instructor:
Dr. Zhengdao Wang

December 8, 2019

IOWA STATE UNIVERSITY

Problem 1

Shortest Path

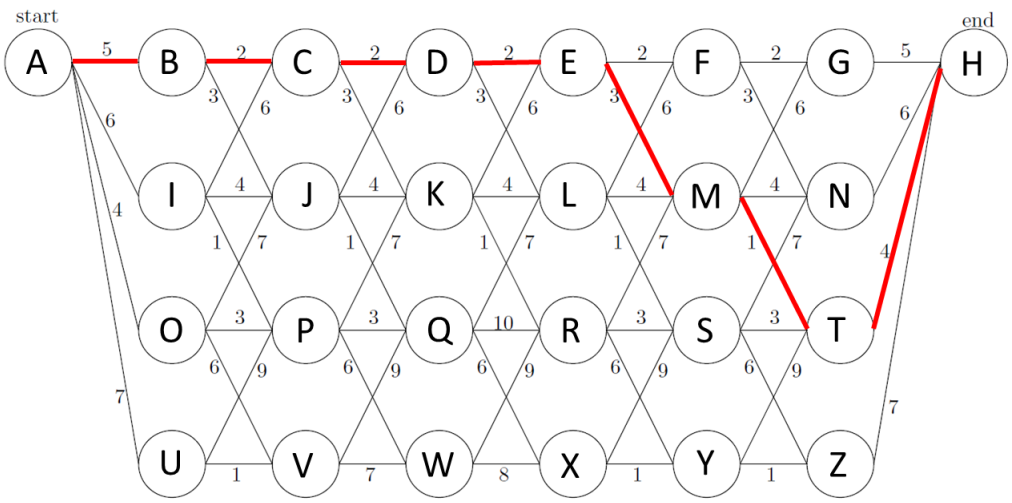

Figure 1: Shortest Path in the Network Graph

Longest Path

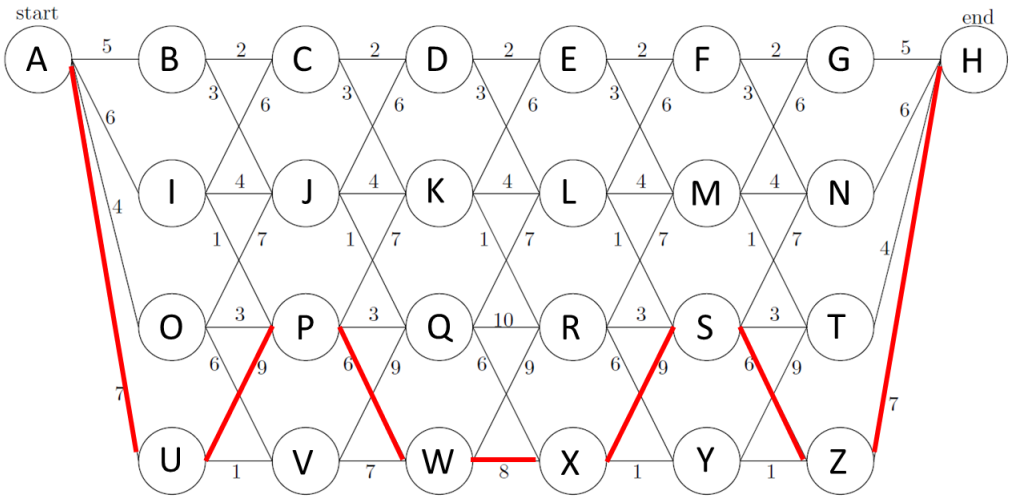

Figure 2: Longest Path in the Network Graph

Problem 2

States: S = 0, 1
Actions: A = 1, 2
Rewards:

$$R_S^{(a)} = \begin{cases} 1 & (s,a) = (0,1) \\ 4 & (s,a) = (0,2) \\ 3 & (s,a) = (1,1) \\ 2 & (s,a) = (1,2) \end{cases}$$

Transition Probabilities:

$$\begin{bmatrix} P_{00}^{(1)} & P_{00}^{(2)} \\ P_{10}^{(1)} & P_{10}^{(2)} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{2} \\ \frac{1}{4} & \frac{2}{3} \end{bmatrix}$$

Discount factor: $\gamma = \frac{3}{4}$
The Bellman's expectation equation is given by

$$V_\pi(S) = R_S + \gamma \sum_{s' \epsilon S} P_{SS'} V_\pi(S')$$

2(a):

choosing action 1 in state 0, and action 2 in state 1,

$$V_\pi(0) = R_0^{(1)} + \gamma(P_{00}^{(1)} V_\pi(0) + P_{01}^{(1)} V_\pi(1))$$
$$V_\pi(1) = R_1^{(2)} + \gamma(P_{10}^{(2)} V_\pi(0) + P_{11}^{(2)} V_\pi(1))$$

Substituting the values,

$$V_\pi(0) = 1 + \frac{3}{4}(\frac{1}{3} V_\pi(0) + \frac{2}{3} V_\pi(1))$$
$$V_\pi(1) = 2 + \frac{3}{4}(\frac{2}{3} V_\pi(0) + \frac{1}{3} V_\pi(1))$$

Solving these 2 equations, we get

$$V_\pi(0) = \frac{28}{5}$$
$$V_\pi(1) = \frac{32}{5}$$

2(b):

```
vpi_0 =0
vpi_1 =0
gamma =3/4

for i in range(0, 5):
    vpi_0_temp =1 +3/4 *( 1/3 *vpi_0 +2/3 *vpi_1)
    vpi_1 =2 +3/4 *( 2/3 *vpi_0 +1/3 *vpi_1)
```

```
    vpi_0 =vpi_0_temp
    print(f"Iteration = {i:.4f}, vpi_0 = {vpi_0:.4f}, vpi_1 = {vpi_1:.4f}")
```

```
Iteration = 0.0000, vpi_0 = 1.0000, vpi_1 = 2.0000
Iteration = 1.0000, vpi_0 = 2.2500, vpi_1 = 3.0000
Iteration = 2.0000, vpi_0 = 3.0625, vpi_1 = 3.8750
Iteration = 3.0000, vpi_0 = 3.7031, vpi_1 = 4.5000
Iteration = 4.0000, vpi_0 = 4.1758, vpi_1 = 4.9766
```

2(c):

The Bellman's expectation equation for $q_\pi(s,a)$ is

$$q_\pi(s,a) = R_S^{(a)} + \gamma \sum_{s' \epsilon S} P_{SS'}^{(a)} V_\pi(S')$$

$$q_\pi(0,1) = R_0^{(1)} + \gamma(P_{00}^{(1)} V_\pi(0) + P_{01}^{(1)} V_\pi(1))$$

$$q_\pi(0,1) = 1 + \frac{3}{4}(\frac{1}{3} V_\pi(0) + \frac{2}{3} V_\pi(1)) = \frac{28}{5}$$

$$q_\pi(0,2) = R_0^{(2)} + \gamma(P_{00}^{(2)} V_\pi(0) + P_{01}^{(2)} V_\pi(1))$$

$$q_\pi(0,2) = 4 + \frac{3}{4}(\frac{1}{2} V_\pi(0) + \frac{1}{2} V_\pi(1)) = \frac{17}{2}$$

$$q_\pi(1,1) = R_1^{(1)} + \gamma(P_{10}^{(1)} V_\pi(0) + P_{11}^{(1)} V_\pi(1))$$

$$q_\pi(1,1) = 3 + \frac{3}{4}(\frac{1}{4} V_\pi(0) + \frac{3}{4} V_\pi(1)) = \frac{153}{20}$$

$$q_\pi(1,2) = R_1^{(2)} + \gamma(P_{10}^{(2)} V_\pi(0) + P_{11}^{(2)} V_\pi(1))$$

$$q_\pi(1,2) = 4 + \frac{3}{4}(\frac{1}{2} V_\pi(0) + \frac{1}{2} V_\pi(1)) = \frac{32}{5}$$

2(d)

From the part c,

$$State0 : q_\pi(0,1) < q_\pi(0,2)$$

$$State1 : q_\pi(1,1) > q_\pi(1,2)$$

Improved policy would be to choose action 2 in state 0 and action 1 in state 1.

2(e)

```python
import numpy as np

def q_pi(s, a, V0, V1):
    gamma =3/4

    if s==0 and a==1:
        R =1
        P0 =1/3
        P1 =1 -P0
    elif s==0 and a==2:
        R =4
        P0 =1/2
        P1 =1- P0
    elif s==1 and a==1:
        R =3
        P0 =1/4
        P1 =1 -P0
    elif s==1 and a==2:
        R =2
        P0 =2/3
        P1 =1- P0

    v_star =R +gamma *(P0*V0 +P1*V1)
    return v_star

# Intialize to zero
V0 =0
V1 =0

for i in range(0, 10):
    V0_temp =np.maximum(q_pi(0, 1, V0, V1), q_pi(0, 2, V0, V1))
    V1 =np.maximum(q_pi(1, 1, V0, V1), q_pi(1, 2, V0, V1))
    V0 =V0_temp
    print(f"Iteration: {i}, V0: {V0:.4f}, V1: {V1:.4f}")

q01 =q_pi(0, 1, V0, V1)
q02 =q_pi(0, 2, V0, V1)
q11 =q_pi(1, 1, V0, V1)
q12 =q_pi(1, 2, V0, V1)

print(f"q(0,1) = {q01:.4f}")
print(f"q(0,2) = {q02:.4f}")
print(f"q(1,1) = {q11:.4f}")
print(f"q(1,2) = {q12:.4f}")
```

```
Iteration: 0, V0: 4.0000, V1: 3.0000
Iteration: 1, V0: 6.6250, V1: 5.4375
Iteration: 2, V0: 8.5234, V1: 7.3008
Iteration: 3, V0: 9.9341, V1: 8.7048
Iteration: 4, V0: 10.9896, V1: 9.7591
Iteration: 5, V0: 11.7808, V1: 10.5500
Iteration: 6, V0: 12.3741, V1: 11.1433
Iteration: 7, V0: 12.8190, V1: 11.5882
Iteration: 8, V0: 13.1527, V1: 11.9219
Iteration: 9, V0: 13.4030, V1: 12.1722
q(0,1) = 10.4369
q(0,2) = 13.5907
q(1,1) = 12.3599
```

```
q(1,2) = 11.7446
```

2(f)

As calculated from the code above, optimal action values are

$$q(0,1) = 10.4369$$
$$q(0,2) = 13.5907$$
$$q(1,1) = 12.3599$$
$$q(1,2) = 11.7446$$

Therefore, optimal policy is action 2 for state 0 and action 1 for state 1.

Problem 3

Model-free Prediction

```python
import numpy as np

def setStates(nStates):
    # Generates 10000 random states between 0 to 1
    states =np.random.randint(0, 2, size=nStates)
    # Make sure 1st state is 0
    states[0] =0
    return states

def setRewards(states):
    # Initialize reward array
    rewards =np.zeros(states.shape)
    # policy  that chooses action 1 in state 0 => Reward = 1,
    # and action 2 in state 1 => Reward = 2
    rewards =updateRewards(states)
    return rewards

def Gt(states, rewards, gamma):
    G_t =0
    # Calculate discounted rewards
    for i in range(1, len(states)):
        G_t +=rewards[i] *np.power(gamma, i-1)
    return G_t

def updateRewards(states):
    for idx, state in enumerate(states):
        if state ==0:
            rewards[idx] =1
        elif state ==1:
            rewards[idx] =2
    return rewards

def getNextValue(iteration, n, states):
```

```
    if states[iteration+n] ==0:
        V_next =V0
    elif states[iteration+n] ==1:
        V_next =V1
    return V_next

def nStepTD(states, rewards, gamma, n, V0, V1, alpha=0.1):
    for i in range(len(states)-n):
        if states[i] ==0:
            # Get discounted rewards
            Gt_n =Gt(states[i:i+n+1], rewards[i:i+n+1], gamma)
            # Get n-step future value
            V_next =getNextValue(i, n, states)
            # Estimate value function
            V0 =V0 +alpha *(Gt_n +np.power(gamma, n)*V_next -V0)
        else:
            # # Get discounted rewards
            Gt_n =Gt(states[i:i+n+1], rewards[i:i+n+1], gamma)
            # Get n-step future value
            V_next =getNextValue(i, n, states)
            # Estimate value function
            V1 =V1 +alpha *(Gt_n +np.power(gamma, n)*V_next -V1)
    return V0, V1

nStates =10000
gamma =3/4
alpha =0.1
V0, V1, N0, N1 =0, 0, 0, 0

states =setStates(nStates)
rewards =setRewards(states)

print("==============Monte Carlo Policy==============")
# Monte carlo policy
# Iterate through each episode
for i in range(len(states)):
    if states[i] ==0:
        N0 +=1
        # Get discounted rewards
        G_t =Gt(states[i:], rewards[i:], gamma)
        # Calculate value function with incremental mean
        V0 =V0 +1/N0 *( G_t -V0 )
    elif states[i] ==1:
        N1 +=1
        # Get discounted rewards
        G_t =Gt(states[i:], rewards[i:], gamma)
        # Calculate value function with incremental mean
        V1 =V1 +1/N1 *( G_t -V1 )

    if i%999 ==0:
        print(f"Iteration: {i}, V0: {V0:.4f}, V1: {V1:.4f}")

# Initialize again for TD Policy
V0, V1, N0, N1 =0, 0, 0, 0

print("================TD Policy====================")
```

```
# n-step TD Policy where n is 1 to 10
for n in range(1,11):
    (V0, V1) =nStepTD(states, rewards, gamma, n, V0, V1)
    print(f"n-step: {n}, V0: {V0:.4f}, V1: {V1:.4f}")
```

```
==========Monte Carlo Policy==========
Iteration: 0, V0: 7.1590, V1: 0.0000
Iteration: 999, V0: 5.9375, V1: 5.9782
Iteration: 1998, V0: 5.9807, V1: 5.9846
Iteration: 2997, V0: 5.9595, V1: 5.9642
Iteration: 3996, V0: 5.9710, V1: 5.9713
Iteration: 4995, V0: 5.9741, V1: 5.9770
Iteration: 5994, V0: 5.9706, V1: 5.9804
Iteration: 6993, V0: 5.9818, V1: 6.0127
Iteration: 7992, V0: 5.9775, V1: 5.9968
Iteration: 8991, V0: 5.9769, V1: 5.9932
Iteration: 9990, V0: 5.9635, V1: 5.9792
==========TD Policy==========
n-step: 1, V0: 1.4996, V1: 1.4205
n-step: 2, V0: 3.4743, V1: 3.3494
n-step: 3, V0: 5.0082, V1: 4.6691
n-step: 4, V0: 5.7060, V1: 5.3093
n-step: 5, V0: 5.8365, V1: 5.5579
n-step: 6, V0: 5.8742, V1: 5.5994
n-step: 7, V0: 5.9390, V1: 5.5891
n-step: 8, V0: 5.9456, V1: 5.6378
n-step: 9, V0: 5.9471, V1: 5.6487
n-step: 10, V0: 6.0019, V1: 5.6536
```

Model-free Control

```
import numpy as np

def epsilon_greedy(q, epsilon, state):
    if np.random.rand() <epsilon:
        action =np.random.randint(2)
    else:
        action =np.argmax(q[state, :])
    return action

def logQ(iteration):
    print(f"Iteration: {iteration}")
    print(f"q(0,0): {q[0,0]:.4f}, q(0,1): {q[0,1]:.4f}, q(1,0): {q[1,0]:.4f}, q(1,1): {q[1,1]:.4f}"
                                                        )

def init_params():
    q =np.random.rand(2, 2)
    state =np.random.randint(2)
    action =epsilon_greedy(q, epsilon, state)
    rewards =np.array([[1., 4.],
                       [3., 2.]])
    return q, state, action, rewards

epsilon =0.1
alpha =0.1
gamma =3/4

# intialize Q with random numbers
(q, state, action, rewards) =init_params()

print("\n==========SARSA Policy==========")
for i in range(100000):
```

```
    # get next state
    state_next =np.random.randint(2)
    # get next action
    action_next =epsilon_greedy(q, epsilon, state_next)
    # get reward
    R =rewards[state, action]

    # update q
    q[state, action] =q[state, action] +alpha *(R +gamma*q[state_next, action_next] -q[state,
                                                    action])

    state =state_next
    action =action_next

    if i%10000 ==0:
        logQ(i)



(q, state, action, rewards) =init_params()
print("\n========================Q-Learning Policy========================")
for i in range(100000):
    # get next state
    state_next =np.random.randint(2)
    # get next action
    action_next =epsilon_greedy(q, epsilon, state_next)
    # get reward
    R =rewards[state, action]

    # update q
    q[state, action] =q[state, action] +alpha *(R +gamma*np.max(q[state_next, :]) -q[state, action]
                                                    )

    state =state_next
    action =action_next

    if i%10000 ==0:
        logQ(i)
```

```
========================SARSA Policy========================
Iteration: 0
q(0,0): 0.0486, q(0,1): 0.2211, q(1,0): 0.8352, q(1,1): 0.4369
Iteration: 10000
q(0,0): 11.2595, q(0,1): 14.2353, q(1,0): 13.4361, q(1,1): 12.2050
Iteration: 20000
q(0,0): 11.0295, q(0,1): 14.1369, q(1,0): 13.1112, q(1,1): 12.2691
Iteration: 30000
q(0,0): 11.2686, q(0,1): 14.6300, q(1,0): 13.4772, q(1,1): 12.1915
Iteration: 40000
q(0,0): 11.1310, q(0,1): 14.4624, q(1,0): 13.3777, q(1,1): 12.0822
Iteration: 50000
q(0,0): 11.0851, q(0,1): 14.1555, q(1,0): 13.3284, q(1,1): 12.1937
Iteration: 60000
q(0,0): 11.0201, q(0,1): 14.4014, q(1,0): 13.5121, q(1,1): 12.2782
Iteration: 70000
q(0,0): 11.2019, q(0,1): 14.2638, q(1,0): 13.1307, q(1,1): 12.1635
Iteration: 80000
q(0,0): 11.1117, q(0,1): 14.3455, q(1,0): 13.4174, q(1,1): 11.9247
Iteration: 90000
q(0,0): 11.1230, q(0,1): 14.0838, q(1,0): 13.0826, q(1,1): 11.9977

========================Q-Learning Policy========================
Iteration: 0
q(0,0): 0.1245, q(0,1): 0.5790, q(1,0): 0.9468, q(1,1): 0.1210
Iteration: 10000
```

```
q(0,0): 11.3922, q(0,1): 14.3925, q(1,0): 13.3032, q(1,1): 12.3208
Iteration: 20000
q(0,0): 11.5667, q(0,1): 14.6827, q(1,0): 13.5772, q(1,1): 12.5422
Iteration: 30000
q(0,0): 11.5650, q(0,1): 14.5763, q(1,0): 13.5902, q(1,1): 12.5269
Iteration: 40000
q(0,0): 11.4970, q(0,1): 14.7709, q(1,0): 13.6710, q(1,1): 12.4856
Iteration: 50000
q(0,0): 11.4699, q(0,1): 14.5548, q(1,0): 13.6559, q(1,1): 12.5678
Iteration: 60000
q(0,0): 11.2410, q(0,1): 14.5413, q(1,0): 13.3637, q(1,1): 12.5054
Iteration: 70000
q(0,0): 11.4190, q(0,1): 14.3120, q(1,0): 13.4151, q(1,1): 12.2987
Iteration: 80000
q(0,0): 11.4765, q(0,1): 14.5599, q(1,0): 13.5022, q(1,1): 12.5060
Iteration: 90000
q(0,0): 11.4824, q(0,1): 14.7722, q(1,0): 13.5385, q(1,1): 12.5165
```

With SARSA policy:

$$q(0,1) = 11.1230$$

$$q(0,2) = 14.0838$$

$$q(1,1) = 13.0826$$

$$q(1,2) = 11.9977$$

With Q-learning policy:

$$q(0,1) = 11.4824$$

$$q(0,2) = 14.7722$$

$$q(1,1) = 13.5385$$

$$q(1,2) = 12.5165$$