ECE/CS 559 - Fall 2020 - Final
Due: 12/11/2020, 10:00am Chicago time.

NAME: Zisheng **ID:** 655677562

WARNING: Any plagiarism regarding any part of the exam will result in a full negative grade (-100). Write all computer codes yourself and refrain from using online resources as well as codes from your peers.

Write the answers on the provided spaces only.

Answers outside the boxes will not be read!!!

Q0. (5 pts) In the space below, include a copy of the email saying that you completed the course evaluations.

Completion for Online Course Evaluation: ECE-559 with Professor Koyuncu Inbox ×

UIC Teaching Evaluations <UICTEACHINGEVAL@uic.edu> Wed, Nov 25, 8:37 PM
to me

Dear participant zliu231@uic.edu,

This email confirms that you completed the online course evaluation survey for ECE-559 with Professor Koyuncu on 11/25/2020 06:37:56 am

The survey was carried out in period Fall 2020.

We thank you for participating.

Sincerely,

Office for Faculty Affairs

Q1. (45 pts) Consider an $n \times n$ environment with locations (i, j), $i = 1, \ldots, n$, $j = 1, \ldots, n$. A miner named Sue begins at location $(1, 1)$ and can travel up, left, down, or right, provided that she remains within the environment at all times. There is a gold mine at location $(n, 1)$ and miner's home at location $(1, n)$. Sue begins with 0 coins in her backpack. Each time she visits the mine, she can get one gold. The maximum amount of gold that Sue can carry is given by some constant integer $G > 0$. Sue and the gold mine cannot be on the same location $(n, 1)$; instead Sue "visits" the gold mine by first traveling to a location that is adjacent to the gold mine (either $(n - 1, 1)$ or $(n, 2)$) and then taking an action that would place herself on the mine. This results in Sue acquiring one more gold provided that she is not carrying G golds already, and meanwhile, Sue's location remains the same. Sue can visit her home whenever she wants, similarly by first traveling to a location that is adjacent to her home, and taking an action that would place herself on top of her home. This results in Sue unloading all the gold she carries to her home. She receives the amount of gold she carries as the reward. Similarly, her location remains the same in this process.

An example scenario for $n = 3$ and $G = 2$ is as follows. Given the initial location $(1, 1)$, consider the sequence of actions RDRRRUULDDRUUL$\cdots$ The first action R will move Sue from (1,1) to (2,1). She receives no reward. Next action D attempts to place Sue outside the environment. Her location will remain the same and she will again receive no rewards. Next action, R, will acquire one gold from the mine, Sue will again stay at (2,1), no reward is received. Next action, R, will acquire another gold from the mine, Sue will again stay at (2,1), no reward is received. Next action, R, will not acquire another gold from the mine because Sue already has the maximum amount of golds she can carry, she again stays at (2,1), and no reward is received. Next two actions UU will move Sue to location (2,3). Subsequent action L will unload all golds to home. Sue's location will again be (2,3), but she will receive a reward of 2. Note that events where gold is unloaded to home are the only events for which the environment gives rewards. The next sequence of actions DDRUUL will go back to the square adjacent to the mine, pick up one gold, and drop it to home, receiving a reward of 1 only.

The cumulative reward is calculated in the standard manner. Let γ be the discount factor. Then, we define the cumulative reward as $\sum_{t=0}^{\infty} \gamma^t R_t$, where R_t is the reward at time t. The cumulative reward for the example above is then $2\gamma^7 + \gamma^{13} + \cdots$.

In the following, let $n = 5$ and $G = 3$. Find the optimal policies using Q learning. Initialize your Q table entries to be independent and identically distributed Gaussian random variables with zero mean and variance 1.

(a) How many states and actions does the problem have? Justify your answer. A rough answer (ignoring a few inadmissable states) is sufficient.

Scenario = n*n , G
Number of states is related to the size of scenario and how many gold Sue already carry
Number of States = (n*n - 2) * (G+1)

4 kind of action in every state : { up, down, left, right}

(b) Let $\gamma = 0.9$. Consider a pure-exploitation strategy that always chooses the best action during training.

i. Describe the policy learned by your algorithm, i.e. what does the miner do under this policy? Also, write the first 40 actions with the policy.

Begin with a start point:[1,1], Sue go aroun to approach the [2,n] and [1,n-1] near the mine ,and take right action, then she got 3 gold. Then go up and left several time to home receive the reward. And go down and right again,appoch the mine, start a new loop.
But the policy is very heavily influenced by the initial value. In many cases, It can't convergent to a good policy, and it will have some redundant steps,like Sue will turn around a little bit in the middle of map, which is not necessarily to appoch the mine
The action list is below:

['U', 'U', 'R', 'R', 'U', 'R', 'D', 'D', 'R':mining, 'L', 'D':mining, 'R':plus mining, 'U', 'R', 'U', 'U', 'U', 'L', 'L', 'L', 'L':home, 'R', 'D', 'R', 'D', 'D', 'R':mining, 'L', 'D':mining, 'R':plus mining, 'U', 'R', 'U', 'U', 'U', 'L', 'L', 'L', 'L':home, 'R']

ii. What is the cumulative reward of the policy? Write down the value you obtained through your simulations.

Cumulative reward : 0.4194739746659103

iii. Do you think this is the optimal policy that maximizes the cumulative reward for the given γ? Justify your answer.

Not, the policy is very heavily influenced by the initial value.And in many cases, It can't convergent to a good policy, and it will have some redundant steps, like after receiving the reward at home, Sue will go back to the start point, which is not necessarily to appoach the mine.

(c) Let $\gamma = 0.9$. Consider instead an exploration/exploitation strategy that chooses random actions with some probability, instead of the best actions.

i. Describe the policy learned by your algorithm, i.e. what does the miner do under this policy? Also, write the first 40 actions with the policy.

Random probability = 0.15. Begin with a start point:[1,1], Sue go to approach [1,n-1] near the mine ,and take right action, then she got 3 gold. Then go up and left several time to home to receive the reward. And go down and right again,appoch the mine, start a new loop.

['R', 'R', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L', 'L', 'L', 'D', 'D', 'D', 'D', 'R', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L', 'L', 'L', 'D', 'D', 'D', 'D', 'R', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L']

ii. What is the cumulative reward of the policy? Write down the value you obtained through your simulations.

Cumulative reward = 1.0217378204521206

iii. Do you think this is the optimal policy that maximizes the cumulative reward for the given γ? Justify your answer.

Almost is the max cumulative reward
The fast way to go to mine then go home from start point costs 12 step, and the fast way to go to mine the go back from home costs 15 step, discount rate is 0.9.
So the optimal policy will convergent the cumulative reward to the value: (3*0.9^12)/(1-(0.9)^15) = 1.06.
And within 40 steps, optimal cumulative reward will be about 1.05

(d) Let $\gamma = 0.6$. Consider exploration/exploitation training.

i. Describe the policy learned by your algorithm, i.e. what does the miner do under this policy? Also, write the first 40 actions with the policy. Attach the code that generates the policy to the end of this exam file. Also, upload the code to the Box as FQ1_LastName_IDNumber.ext, where ext is either m or py. Upload one and only one file only. Without a functioning code for this part, answers to above questions will also not be given credit.

Random probability = 0.15. Begin with a start point:[1,1], Sue go around to appoach the [1,n-1] near the mine ,and take just one more right action, then she just got 2 gold. Then go up and left several time to home receive the reward. And go down and right again,appoch the mine, start a new loop.

['R', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L', 'L', 'L', 'D', 'D', 'D', 'D', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L', 'L', 'L', 'D', 'D', 'D', 'D', 'R', 'R', 'R', 'U', 'U', 'U', 'U', 'L', 'L', 'L', 'D']

ii. What is the cumulative reward of the policy? Write down the value you obtained through your simulations.

Cumulative reward = 0.0072616316334374 96

iii. Compare and contrast the policy to the policy obtained in (c). If they are the same, explain why they should be the same. If they are different, explain why they should be different.

Compared with policy in (c) which make Sue get 3 gold every time while approaching the mine , the policy in d(0.6) just make it 2.
The difference is because policy(c,0.9) emphasizes more about the future destination. But policy(d,0.6) emphasizes nearer future, and how fast it can get a certain reward.

.

(e) Do you think a deep Q network would be more appropriate for this problem, e.g. in terms of providing faster convergence and better policies?

No, because the state space of this problem is small and the optimal policy is simple and straightforward.

Q2. (50 pts) Download dataset.txt from the Piazza website. The dataset contains 50,000 lines, where each line has 101 columns. Each line is a training entry. The first column corresponds to the class label, and the next 100 columns is the 100-dimensional data vector. There are a total of 8 classes, and 6250 samples per class in the training set. Let $y_{i,j}$, $i \in \{1, \ldots, 50000\}, j \in \{1, \ldots, 100\}$ be the entry in the ith line, $(j+1)$st column of the dataset. For different values of i, plot the graph formed by the points $(j, y_{i,j})$, $j = 1, \ldots, 100$ to understand the nature of the dataset, and a rough idea of members of different classes. Design a neural network based classifier to classify the dataset. You can use any method we have learned in class.

(a) Describe your classifier architecture and the rationale behind why you chose the architecture.

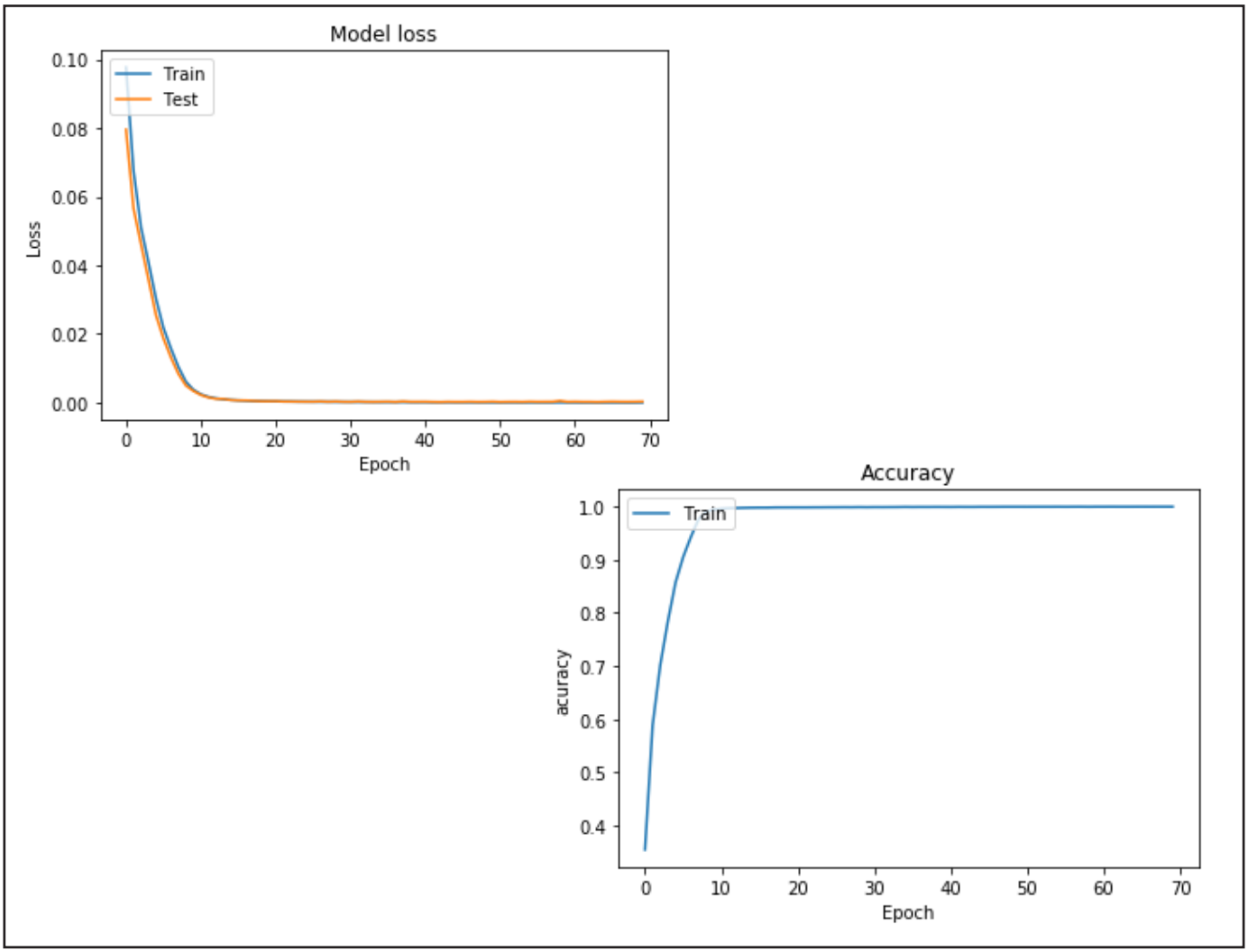
An neural network has been designed, because this problem is a typical classification problem with training data and training data label.
The classifier consists of three fully connected layers with sigmoid activation function. The hidden layers should larger than the output layer and input layer. The first layer has 100 nodes. The second layer has 150 nodes.The third layer is the output, and has 8 nodes. The index of the max value among these 8 output nodes represented the predicted class of the data.

(b) Describe your training procedure, cost function, optimizer, weight initialization, hyperparameters you utilized, and the rationale behind your choices, etc. You can split part of the training data for validation purposes. Indicate if you do so.

We split date set. we let 1~48000 row of data as a training set, and the 48001~ 50000 as a validation set.

The loss function is mean_squared_error used as the feedback of the neural network.
optimizer is adam
Then the batch size while training is 100. Training go through 70 epochs

(c) Provide below a graph showing the epoch number vs your per cent training accuracy. At the end of the exam document, attach the code that reads the dataset and trains the network. Also, upload the code to Box as FQ2_LastName_IDNumber.ext, where ext is either m or py.



(d) Download test.txt from Piazza website. The test set also contains 50,000 lines, but each line has 100 columns (not 101!!). The true class labels are not included in this file. Each line is a test data vector. Evaluate your trained model on this file and obtain the predicted outputs for each line. The labels should be chosen among $\{0, \ldots, 7\}$, and should be saved to file FQ2Predicts_LastName_IDNumber.txt without any new lines, spaces etc in ASCII format. For example, if the first three predictions are 4,7, and 0, for the first three lines of test.txt, respectively, then your file should look like $470\cdots$. Your file size should be exactly 50000 bytes at the end. Failure to follow this guideline will result in losing most of the grade for this question because I will use an automatic script to evaluate how well your model is performing according to your predicted model outputs. I cannot check by hand if the script fails to process your file.

```
# -*- coding: utf-8 -*-
import numpy as np
import random
import itertools
class Qlean():
    def __init__(self,n = 3,G = 2,rf = 0.9, arate = 0.6, epoch = 10,step = 40):
        self.n = n
        self.G = G
        self.initEnv()
        self.rf = rf
        self.arate = arate
        self.epoch = epoch
        self.step = step
    def initloc(self):
        self.state = np.array([[1,1],0])
    def initEnv(self):
        self.initloc()
        self.q = np.random.randn((self.n*self.n - 2)*(self.G+1),7)
        g = range(0,self.G+1)
        row = range(1,self.n+1)
        col = range(1,self.n+1)
        label = itertools.product(row,col,g)
        qrow = 0
        for i in label:
            if (i[0:2] == (1,self.n)) or (i[0:2] == (self.n,1)):
                continue
            self.q[qrow][0] = i[0]
            self.q[qrow][1] = i[1]
            self.q[qrow][2] = i[2]
            qrow = qrow + 1
    def faction(self):
        for i in range(len(self.q)):
            if (self.q[i][0] == self.state[0][0]) and (self.q[i][1] == self.state[0][1]) and (self.q[i][2] == self.state[1]):
                action_list = [self.q[i][3],self.q[i][4],self.q[i][5],self.q[i][6]]
                return i, action_list.index(max(action_list))
    def qaction(self):
        for i in range(len(self.q)):
#           print(i)
            if (self.q[i][0] == self.state[0][0]) and (self.q[i][1] == self.state[0][1]) and (self.q[i][2] == self.state[1]):
                action_list = [self.q[i][3],self.q[i][4],self.q[i][5],self.q[i][6]]
                # random action
                if np.random.rand(1) > 0.85:
                    return i, np.random.randint(0,4)
                return i, action_list.index(max(action_list))

    def updateState(self,action1):
        # action up=[1,0], down = [-1,0], left = [0,-1], right = [0,1]
        if action1 == 0:
            action = [1,0]
        elif action1 == 1:
            action = [-1,0]
        elif action1 == 2:
            action = [0,-1]
        else:
            action = [0,1]
        reward = 0
        nexloc = self.state[0] + np.array(action)
        # boundary
        if (nexloc[0] == 0) or (nexloc[0] == self.n+1) or (nexloc[1] == self.n+1) or (nexloc[1] == 0):
            reward = 0
        elif (nexloc[0] == self.n) and (nexloc[1] == 1) :#  nexloc == np.array([self.n,1]):
            if self.state[1] != 0:
                reward = self.state[1]
            self.state[1] = 0
        elif ((nexloc[0] == 1) and (nexloc[1] == self.n-1)) or ((nexloc[0] == 2) and (nexloc[1] == self.n)):
            if self.state[1] < self.G:
                self.state[1] =  self.state[1] + 1
            self.state[0] = nexloc
        # plus mining
        elif (nexloc[0] == 1) and (nexloc[1] == self.n):
            if self.state[1] < self.G:
                self.state[1] =  self.state[1] + 1
        else:
            self.state[0] = nexloc
        return reward

    def getQvalue(self,action):
        for i in range(len(self.q)):
            if (self.q[i][0] == self.state[0][0]) and (self.q[i][1] == self.state[0][1]) and (self.q[i][2] == self.state[1]):
                if action == 'max':
                    return max(self.q[i][3],self.q[i][4],self.q[i][5],self.q[i][6])
                else:
                    return self.q[i][action + 3]
                break

    def updateQ(self,i,action,r,Qn):
        self.q[i][action + 3] = (1-self.arate)*self.q[i][action + 3] + self.arate*(r + self.rf*Qn)

    def training(self):
        for ep in range(self.epoch):
            self.initloc()
            rc = 0
            for i in range(self.step):
                old_state, action = self.qaction()
                r = self.updateState(action) # reward while motion
                Qn = self.getQvalue('max') # return next Qmax
                self.updateQ(old_state,action,r,Qn)

    def testing(self):
        self.initloc()
        rc = 0
        steps = []
        for i in range(self.step):
            # pick an action
            old_state, action = self.faction()

            print(self.state)
            steps.append(actiondic[action])

            r = self.updateState(action) # reward while motion
            print(i,r)
            rc = rc + r*pow(self.rf,i+1)#self.rf
        print(steps)
        print(rc)
```

Q1 code

```
actiondic = {0:'U',1:'D',2:'L',3:'R'}
# self,n = 3,G = 2,rf = 0.9, arate = 0.6, epoch = 10,step = 40
test = Qlean(5,3,0.6,1,1000,40)
test.training()
test.testing()
```

Q2 code

```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec  9 20:53:08 2020

@author: 10596
"""
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
import tensorflow.keras
from tensorflow.python.keras.layers.core import Dense, Flatten

try:
    print(epochssssssss)# if you don't want to run it, delete it
except:
    datas = np.loadtxt('dataset.txt')
    #plt.plot(datas[45500])

    y_train = datas[0:48000,0]
    x_train = datas[0:48000,1:101]

#    y_train = datas[:,0]
#    x_train = datas[:,1:101]

    y_test = datas[48001:50000,0]
    x_test = datas[48001:50000,1:101]

    y_train = tensorflow.keras.utils.to_categorical(y_train, 8)
    y_test = tensorflow.keras.utils.to_categorical(y_test, 8)

    batch_size_x = 100
    epochs = 70

    model = Sequential()
    model.add(Dense(100, activation='sigmoid'))
    model.add(Dense(150, activation='sigmoid'))
    model.add(Dense(8, activation='sigmoid'))
    model.compile(loss=tensorflow.keras.losses.mean_squared_error,
                  optimizer='adam',
                  metrics=['accuracy'])

    history = model.fit(x_train, y_train, batch_size=batch_size_x, epochs=epochs, validation_data=(x_test,y_test))

    # Plot
    plt.figure(1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])

    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')

    plt.figure(2)
    plt.plot(history.history['accuracy'])
    plt.title('Accuracy')
    plt.ylabel('acuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

testset = np.loadtxt('test.txt')

pre = model.predict(testset)

f = open('FQ2Predicts_Zisheng_655677562.txt', 'w')
for i in range(len(pre)):
    temp = pre[i].tolist()
    f.write(str(temp.index(max(temp))))

f.close()
```