ASSIGNMENT - 9

ANIKEIT SETHI (190001003)

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

class All\_Model:

    def activation\_sigmoid(self,x):

        return 1/(1+np.exp(-x))

    def sigmoid\_differentiation(self,x):

        return x\*(1.0 - x)

    def \_\_init\_\_(self,no\_Inputs,no\_hiden,no\_out,learning\_rate=0.001,epochs=1000,t=0.5):

        self.no\_Inputs = no\_Inputs

        self.no\_hiden = no\_hiden

        self.no\_out = no\_out

        self.learning\_rate = learning\_rate

        self.epochs = epochs

        self.threshold = t

        Layer = [self.no\_Inputs] + self.no\_hiden + [self.no\_out]

        self.weights = []

        for i in range(len(Layer)-1):

            w = np.random.randm(Layer[i],Layer[i+1])

            self.weights.append(w)

        self.bias = []

        for i in range(len(Layer)-1):

            b = np.random.randm(1,Layer[i+1])

            self.bias.append(b)

        self.activations = []

        for i in range(len(Layer)):

            self.activations.append(np.random.randm(1,Layer[i]))

        self.wderivatives = []

        for i in range(len(Layer)-1):

            self.wderivatives.append(np.random.randm(Layer[i],Layer[i+1]))

        self.bderivatives = []

        for i in range(len(Layer)-1):

            self.bderivatives.append(np.random.randm(1,Layer[i+1]))

    def forward(self,inputs):

        a = inputs

        self.activations[0] = a.reshape(1,a.shape[0])

        for i,w in enumerate(self.weights):

            z = np.dot(a,w) + self.bias[i]

            a = self.activation\_sigmoid(z)

            self.activations[i+1] = a

        return a

    def backward\_pass(self,error):

        for i in reversed(range(len(self.wderivatives))):

            a\_next = self.activations[i+1]

            delta = error\*self.sigmoid\_differentiation(a\_next)

            delta = delta.reshape(delta.shape[0],-1)

            a\_curr = self.activations[i]

            a\_curr = a\_curr.reshape(a\_curr.shape[1],-1)

            self.wderivatives[i] = np.dot(a\_curr,delta)

            self.bderivatives[i] = delta

            error = np.dot(delta,self.weights[i].T)

        return error

    def updating(self):

        for i in range(len(self.weights)):

            self.weights[i] += self.learning\_rate\*self.wderivatives[i]

            self.bias[i] += self.learning\_rate\*self.bderivatives[i]

    def train(self,inputs,labels):

        for i in range(self.epochs):

            for j,(x,y) in enumerate(zip(inputs,labels)):

                output = self.forward(x)

                error = y - output

                temp = self.backward\_pass(error)

                self.updating()

# Question 1

train\_inputs = np.random.randint(2,size=(20,7))

val\_inputs = np.random.randint(2,size=(10,7))

train\_outputs = []

val\_outputs = []

for x in train\_inputs:

    if np.sum(x,axis=0)<3:

        train\_outputs.append(0)

    else:

        train\_outputs.append(1)

for x in val\_inputs:

    if np.sum(x,axis=0)<3:

        val\_outputs.append(0)

    else:

        val\_outputs.append(1)

train\_outputs = np.array(train\_outputs)

val\_outputs = np.array(val\_outputs)

print("--------------Backpropagation Learning Method for 7 Input Majority Problem----------------")

print("Training from the Random training data....")

inst = Model(7,[15],1)

inst.train(train\_inputs,train\_outputs)

thres = 0.5

tot,cor = 0,0

for (inp,lab) in zip(train\_inputs,train\_outputs):

    out = inst.forward(inp)

    out = (out>thres)\*1

    tot+=1

    if out==lab:

        cor+=1

    accu=cor/tot

    print(accu)

print("The prediction accuracy in Training is: ")

print(accu\*100, "%")

tot,cor = 0,0

for (inp,lab) in zip(val\_inputs,val\_outputs):

    out = inst.forward(inp)

    out = (out>thres)\*1

    tot+=1

    if out==lab:

        cor+=1

    print(accu)

    print(cor/tot)

print("The prediction accuracy in Validaion is: ")

print(accu\*100, "%")

print(inst.weights)

Output: -

--------------Backpropagation Learning Method for 7 Input Majority Problem----------------

Training from the Random training data....

1.0

1.0

1.0

0.75

0.8

0.8333333333333334

0.8571428571428571

0.875

0.8888888888888888

0.9

0.8181818181818182

0.8333333333333334

0.8461538461538461

0.8571428571428571

0.8

0.8125

0.8235294117647058

0.8333333333333334

0.7894736842105263

0.8

The Prediction accuracy in Training is:

80.0 %

0.8

1.0

0.8

0.5

0.8

0.6666666666666666

0.8

0.75

0.8

0.8

0.8

0.8333333333333334

0.8

0.8571428571428571

0.8

0.75

0.8

0.7777777777777778

0.8

0.8

The prediction accuracy in Validation is:

80.0 %

[array([[-0.06660227, 0.72155564, 0.37586231, -0.10741246, -1.65382569,

0.66761229, 0.61488964, -0.91539825, 2.2211288 , 0.46855515,

-0.9924254 , -0.05937517, 0.95093172, -0.02435977, -0.26611339],

[-1.90473897, -1.76578526, -0.52214779, 0.8498715 , 1.79969922,

-0.58866771, 0.41823551, -0.2699563 , 0.04119393, 0.41387243,

1.26056125, -1.59770543, -0.03210174, -0.21123512, -1.29265612],

[ 0.91111313, 0.84204261, -0.06905433, -1.64214804, 0.78951244,

-1.36183371, 0.71314258, -0.44207665, -0.64954111, -1.24881859,

0.6386992 , 0.1099403 , -0.76354102, -0.0633258 , -0.99162939],

[-1.44383373, -0.76480796, 0.52142164, -0.85515827, 0.74708105,

-0.49247987, 1.10841211, 0.32475989, 1.77209451, -0.6966146 ,

-0.23551413, 0.59285101, 0.00282316, -0.17800279, 1.64508227],

[ 2.57453219, -0.70034219, -0.01589378, 0.45038571, 0.05168163,

-1.87891889, 1.60205303, 0.29604735, -0.18557684, 0.66751935,

0.12019122, -1.06343832, 0.40951304, -1.25101041, -1.47741853],

[-0.03249059, -0.23313102, 1.10673468, -0.65099544, 0.01931627,

0.95506936, -1.36133367, 1.72888577, 0.14936569, 0.97763711,

0.03928996, -0.41628484, -0.4712511 , -0.52963986, -2.7769816 ],

[-0.17475631, -2.28847415, -1.32673761, 0.51544357, -0.58200755,

0.16616807, 1.57696166, -0.39750736, -0.70316049, 0.70540364,

0.80265609, 0.4074295 , -1.5220078 , 0.80331609, -0.50037943]]), array([[-0.16615796],

[-0.89001294],

[ 0.80624529],

[-1.77810643],

[-0.05722135],

[ 0.27033676],

[ 1.73266401],

[-0.06863087],

[-0.29580756],

[ 0.43558615],

[ 0.39324364],

[-0.0418154 ],

[ 0.65957816],

[ 0.65975825],

[-0.03343952]])]

# Question 2

parent\_inputs = []

parent\_outputs = []

for i in range(1,100):

    temp = [int(j) for j in bin(i)[2:]]

    repr = []

    for j in range(7-len(temp)):

        repr = [0] + repr

    repr = repr+list(temp)

    parent\_inputs.append(np.array(repr))

    parent\_outputs.append([round(1/(i+1),3)])

parent\_inputs,parent\_outputs = shuffle(parent\_inputs,parent\_outputs,random\_state=0)

train\_inputs = np.array(parent\_inputs[:80])

train\_outputs = np.array(parent\_outputs[:80])

val\_inputs = np.array(parent\_inputs[80:])

val\_outputs = np.array(parent\_outputs[80:])

for hid in range(1,30):

    inst = Model(7,[hid],1)

    inst.train(train\_inputs,train\_outputs)

    tot,cor = 0,0

    for (inp,lab) in zip(train\_inputs,train\_outputs):

        out = inst.forward(inp)

        out[0][0] = round(out[0][0],2)

        lab = round(lab[0],2)

        tot+=1

        if out.item()==lab.item():

            cor+=1

    print(cor/tot)

    tot,cor = 0,0

    for (inp,lab) in zip(val\_inputs,val\_outputs):

        out = inst.forward(inp)

        out[0][0] = round(out[0][0],2)

        lab = round(lab[0],2)

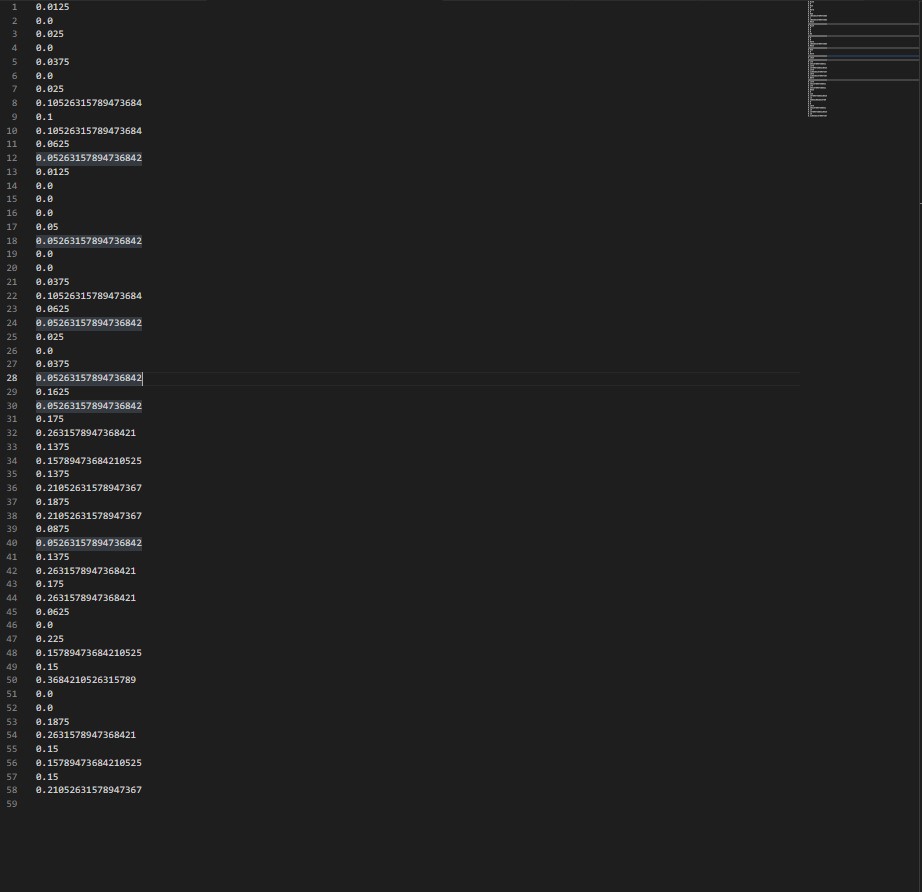
        tot+=1

        if out.item()==lab.item():

            cor+=1

    print(cor/tot)

Output: -



# Question 3

train\_inputs = np.loadtxt(open('train\_data.csv'),delimiter=',',usecols=range(4),skiprows=(1))

train\_outputs = np.loadtxt(open('train\_data.csv'),delimiter=',',usecols=(4),skiprows=(1),dtype=str)

train\_outputs = (train\_outputs == 'versicolor')\*1

train\_inputs,train\_outputs = shuffle(train\_inputs,train\_outputs,random\_state=0)

test\_inputs = np.loadtxt(open('test\_data.csv'),delimiter=',',usecols=range(4),skiprows=(1))

test\_outputs = np.loadtxt(open('test\_data.csv'),delimiter=',',usecols=(4),skiprows=(1),dtype=str)

test\_outputs = (test\_outputs == 'versicolor')\*1

test\_inputs,test\_outputs = shuffle(test\_inputs,test\_outputs,random\_state=0)

inst = Model(4,[5],1)

inst.train(train\_inputs,train\_outputs)

thres = 0.5

tot,cor = 0,0

for (inp,lab) in zip(train\_inputs,train\_outputs):

    out = inst.forward(inp)

    out = (out>thres)\*1

    tot+=1

    if out==lab:

        cor+=1

print(cor/tot)

tot,cor = 0,0

for (inp,lab) in zip(test\_inputs,test\_outputs):

    out = inst.forward(inp)

    out = (out>thres)\*1

    tot+=1

    if out==lab:

        cor+=1

print(cor/tot)

Output: -

