# Q1)a)

# 

# Q1)b)

(Error function used = Mean Square Error)

import numpy as np

def forward(W,x):

return np.tanh(np.matmul(W,x))

def backprop(W1, x1, W2, x2, out, targetOut):

one = np.array([1])

delj2 = (out-targetOut)\*(1-out\*out);

c = np.append(one,x2).reshape([3,1]);

gradW2 = np.outer(delj2, c)

print(gradW2.shape)

delj1 = (1-c\*c)\*np.matmul(W2.T, delj2)

delj1 = delj1[1:]

gradW1 = np.outer(delj1, x1)

print(gradW1.shape)

return [gradW1, gradW2];

def calcError(output, targetOutput):

return 0.5\*np.sum(np.square(output - targetOutput));

max\_epochs = 10;

W1 = np.array([[0.3, 0.1, 0.2],[0.2, 0.1, 0.2]]);

x1 = np.array([1, 0.1, 0.2]);

x1 = x1.reshape([3,1]);

W2 = np.array([[0.5, 0.1, 0.2], [0.4, 0.1, 0.2]]);

one = np.array([1]);

targetOut = np.array([[0.4],[0.3]]);

error = 0;

learningRate = 0.1;

for i in range(max\_epochs):

#go over entire training set

x2 = forward(W1, x1);

out = forward(W2, (np.append(one, x2)).reshape([3,1]));

print(x2);

print(out);

error = calcError(out, targetOut);

[gradW1, gradW2] = backprop(W1, x1, W2, x2, out, targetOut)

#finished going over entire training set, now update

W1 = W1 - gradW1;

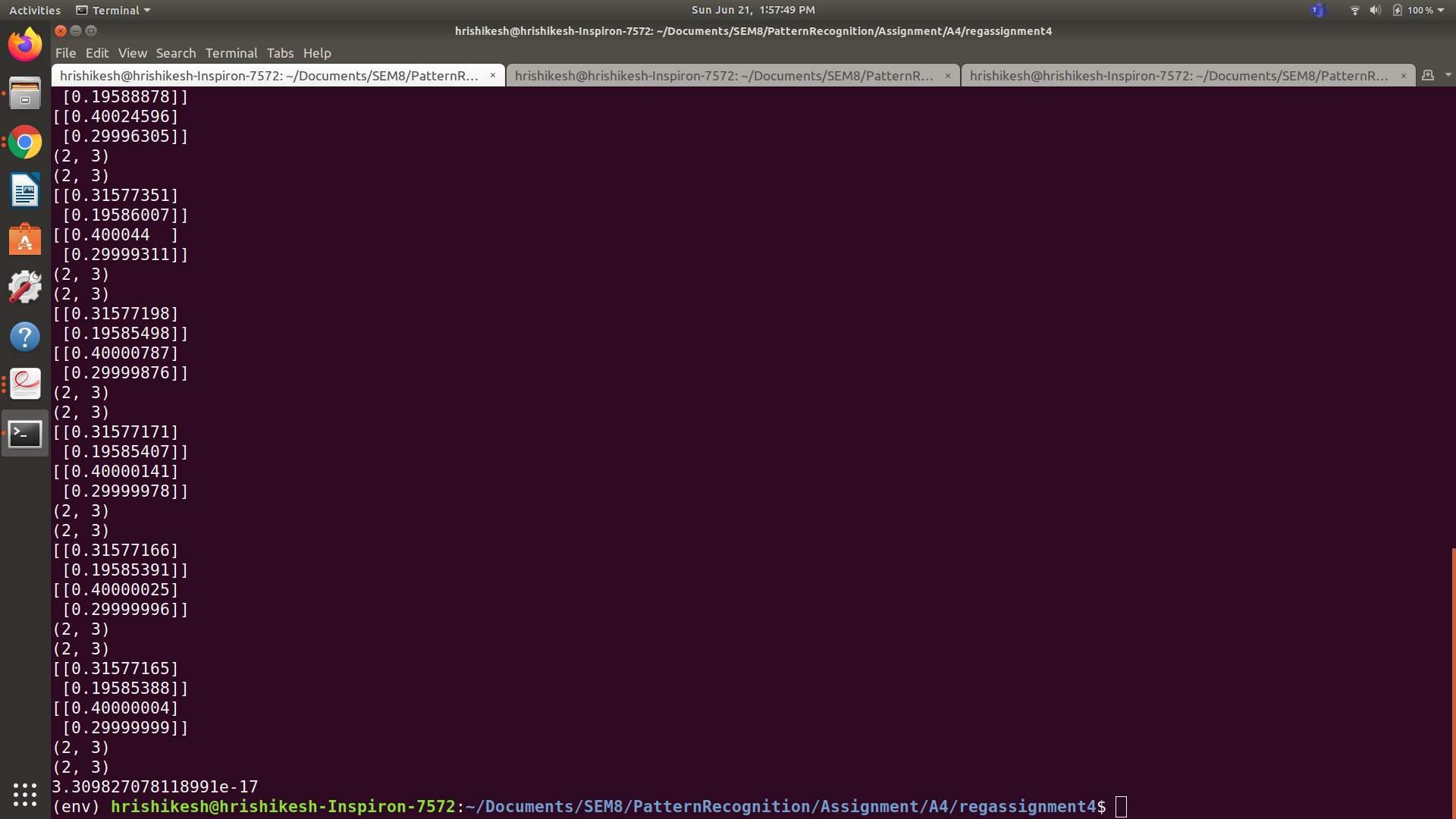
W2 = W2 - gradW2;

x2 = forward(W1, x1);

out = forward(W2, (np.append(one, x2)).reshape([3,1]));

error = calcError(out, targetOut);

print(error);



# Q2)

(Error function used = Mean Square Error)

import numpy as np

import pandas

import random

df = pandas.read\_csv('/home/hrishikesh/Documents/SEM8/PatternRecognition/Assignment/A1/iris.csv')

sepalLength = df.loc[:,"sepal.length"]

sepalWidth = df.loc[:,"sepal.width"]

petalLength = df.loc[:,"petal.length"]

petalWidth = df.loc[:,"petal.width"]

variety = df.loc[:,"variety"]

petalLength = np.array(petalLength)

petalLength = petalLength.astype('float64')

petalLength = petalLength.tolist()

petalWidth = np.array(petalWidth)

petalWidth = petalWidth.astype('float64')

petalWidth = petalWidth.tolist()

sepalLength = np.array(sepalLength)

sepalLength = sepalLength.astype('float64')

sepalLength = sepalLength.tolist()

sepalWidth = np.array(sepalWidth)

sepalWidth = sepalWidth.astype('float64')

sepalWidth = sepalWidth.tolist()

totalIn = [];

totalOut = [];

for i in range(150):

totalIn.append([petalLength[i], petalWidth[i], sepalLength[i], sepalWidth[i]]);

if i<50:

totalOut.append([1,0,0]);

elif i>=50 and i<100:

totalOut.append([0,1,0]);

else:

totalOut.append([0,0,1]);

totalIn = (totalIn - np.mean(totalIn,axis=0))/(np.std(totalIn,axis=0)\*\*2)

print(totalIn)

trainingIn = [];

testIn = [];

trainingOut = [];

testOut = [];

count = 0;

for i in range(150):

if count<40:

count = count + 1;

trainingIn.append(totalIn[i]);

trainingOut.append(totalOut[i]);

elif count>=40 and count < 50:

count = count + 1;

testIn.append(totalIn[i]);

testOut.append(totalOut[i]);

if count == 50:

count = 0;

trainData = np.array(trainingIn);

trainLabel = np.array(trainingOut);

testData = np.array(testIn);

testLabel = np.array(testOut);

def forward(W,x):

return np.tanh(np.matmul(W,x))

def backprop(W1, x1, W2, x2, W3, x3, out, targetOut):

one = np.array([1])

delj3 = (out-targetOut)\*(1-out\*out);

c = np.append(one,x3).reshape([13,1]);

gradW3 = np.outer(delj3, c)

delj2 = (1-c\*c)\*np.matmul(W3.T, delj3)

delj2 = delj2[1:]

d = np.append(one,x2).reshape([17,1]);

gradW2 = np.outer(delj2, d)

delj1 = (1-d\*d)\*np.matmul(W2.T, delj2)

delj1 = delj1[1:];

gradW1 = np.outer(delj1, x1)

return [gradW1, gradW2, gradW3];

def calcError(output, targetOutput):

return 0.5\*np.sum(np.square(output - targetOutput));

max\_epochs = 500;

W1 = np.random.randn(16,5)/np.sqrt(16);

W2 = np.random.randn(12,17)/np.sqrt(12);

W3 = np.random.randn(3,13)/np.sqrt(3);

one = np.array([1]);

error = 0;

learningRate = 0.001;

gradW1 = np.zeros(W1.shape);

gradW2 = np.zeros(W2.shape);

gradW3 = np.zeros(W3.shape);

updateW1 = np.zeros(W1.shape);

updateW2 = np.zeros(W2.shape);

updateW3 = np.zeros(W3.shape);

gamma = 0.005

threshold = 0.0001;

for i in range(max\_epochs):

#go over entire training set

for i in range(120):

x1 = (np.append(one,trainData[i])).reshape([5,1])

x2 = forward(W1, x1);

x3 = forward(W2, (np.append(one, x2)).reshape([17,1]));

out = forward(W3, (np.append(one, x3)).reshape([13,1]));

targetOut = (trainLabel[i].T).reshape([3,1]);

error += calcError(out, targetOut);

[gW1, gW2, gW3] = backprop(W1, x1, W2, x2, W3, x3, out, targetOut)

gradW1 = gradW1 + gW1

gradW2 = gradW2 + gW2

gradW3 = gradW3 + gW3

#finished going over entire training set, now update

updateW1 = gamma\*updateW1 + learningRate\*gradW1;

updateW2 = gamma\*updateW2 + learningRate\*gradW2;

updateW3 = gamma\*updateW3 + learningRate\*gradW3;

W1 = W1 - updateW1;

W2 = W2 - updateW2;

W3 = W3 - updateW3;

print(error)

if error < threshold:

break

error = 0

for i in range(30):

x1 = (np.append(one,testData[i])).reshape([5,1])

x2 = forward(W1, x1);

x3 = forward(W2, (np.append(one, x2)).reshape([17,1]));

out = forward(W3, (np.append(one, x3)).reshape([13,1]));

targetOut = (testLabel[i].T).reshape([3,1]);

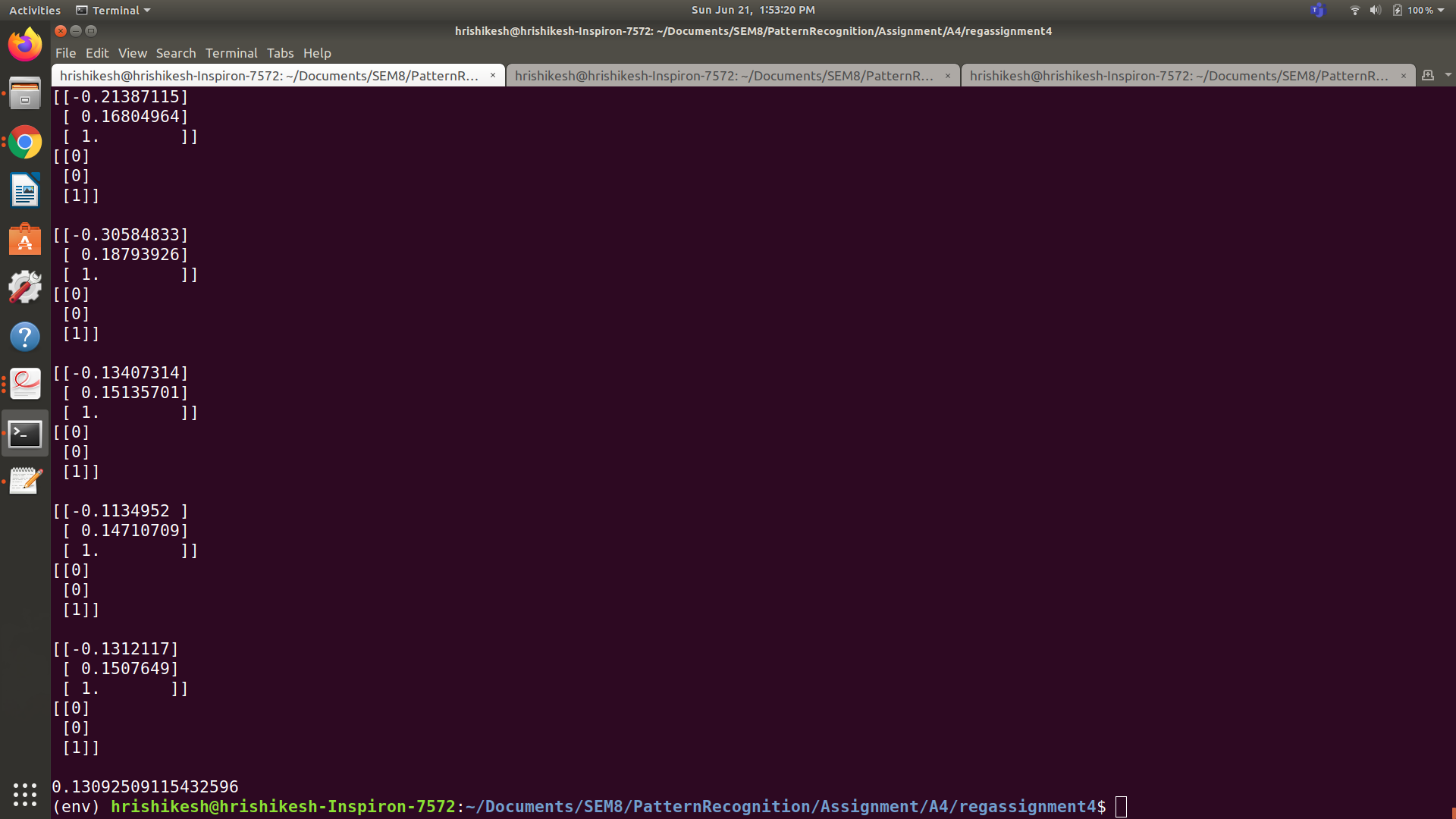
print(out)

print(targetOut)

print()

error += calcError(out, targetOut);

print(error\*2/30)



# Q3)

(Error function used = Mean Square Error)

import sys

import nltk

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer,WordNetLemmatizer

from nltk.tokenize import word\_tokenize

import sklearn.metrics as m

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

np.set\_printoptions(threshold=sys.maxsize)

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

dataset=pd.read\_csv('spam.csv',encoding='latin-1')

sent=dataset.iloc[:,[1]]['v2']

label=dataset.iloc[:,[0]]['v1']

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

label=le.fit\_transform(label)

import re

from nltk.stem import PorterStemmer

stem=PorterStemmer()

sentences=[]

word2count = {}

for sen in sent:

senti=re.sub('[^A-Za-z]',' ',sen)

senti=senti.lower()

words=word\_tokenize(senti)

for i in range(len(words)):

words[i] = words[i].lower()

words[i] = re.sub(r'\W', ' ', words[i])

words[i] = re.sub(r'\s+', ' ', words[i])

for word in words:

if word not in word2count.keys():

word2count[word] = 1

else:

word2count[word] += 1

"""word=[stem.stem(i) for i in words if i not in stopwords.words('english')]

senti=' '.join(word)

senti = senti.join('\n')

sentences.append(senti)"""

sentences.append(words)

import heapq

freq\_words = heapq.nlargest(100, word2count, key=word2count.get)

X = []

for data in sent:

vector = []

for word in freq\_words:

if word in nltk.word\_tokenize(data):

vector.append(1)

else:

vector.append(0)

X.append(vector)

X = np.asarray(X)

label = label.reshape([5572,1])

countP = 0;

countN = 0;

for i in range(5572):

if label[i] == 0:

label[i] = -1;

countN = countN+1;

countP = 5572 - countN;

print(countP," ", countN);

trainDatafull,testData,trainLabelfull,testLabel = train\_test\_split(X,label,test\_size=0.2,random\_state=7)

trainData = np.zeros([2\*countP, 100]);

trainLabel = np.zeros([2\*countP, 1]);

count = 0;

i = 0;

while(count < countP and i < trainDatafull.shape[0]):

if trainLabelfull[i] == 1:

trainData[count] = trainDatafull[i]

trainLabel[count] = trainLabelfull[i]

count = count + 1

i = i+1

i = 0

while(count < 2\*countP and i < trainDatafull.shape[0]):

if trainLabelfull[i] == -1:

trainData[count] = trainDatafull[i]

trainLabel[count] = trainLabelfull[i]

count = count + 1

i = i+1

def forward(W,x):

return np.tanh(np.matmul(W,x))

def backprop(W1, x1, W2, x2, W3, x3, out, targetOut):

one = np.array([1])

delj3 = (out-targetOut)\*(1-out\*out);

c = np.append(one,x3).reshape([151,1]);

gradW3 = np.outer(delj3, c)

delj2 = (1-c\*c)\*np.matmul(W3.T, delj3)

delj2 = delj2[1:]

d = np.append(one,x2).reshape([301,1]);

gradW2 = np.outer(delj2, d)

delj1 = (1-d\*d)\*np.matmul(W2.T, delj2)

delj1 = delj1[1:];

gradW1 = np.outer(delj1, x1)

return [gradW1, gradW2, gradW3];

def calcError(output, targetOutput):

return 0.5\*np.sum(np.square(output - targetOutput));

max\_epochs = 15;

W1 = np.random.randn(300,101)/np.sqrt(300);

W2 = np.random.randn(150,301)/np.sqrt(100);

W3 = np.random.randn(1,151)/np.sqrt(1);

one = np.array([1]);

error = 0;

learningRate = 0.0001;

gradW1 = np.zeros(W1.shape);

gradW2 = np.zeros(W2.shape);

gradW3 = np.zeros(W3.shape);

updateW1 = np.zeros(W1.shape);

updateW2 = np.zeros(W2.shape);

updateW3 = np.zeros(W3.shape);

gamma = 0.002

q = 0;

threshold = -500;

for j in range(max\_epochs):

#go over entire training set

#for i in range(4457):

for i in range(2\*countP):

x1 = (np.append(one,trainData[i])).reshape([101,1])

x2 = forward(W1, x1);

x3 = forward(W2, (np.append(one, x2)).reshape([301,1]));

out = forward(W3, (np.append(one, x3)).reshape([151,1]));

#out = fwd(W3, (np.append(one, x3)).reshape([151,1]));

targetOut = (trainLabel[i].T).reshape([1,1]);

error += calcError(out, targetOut);

[gW1, gW2, gW3] = backprop(W1, x1, W2, x2, W3, x3, out, targetOut)

gradW1 = gradW1 + gW1

gradW2 = gradW2 + gW2

gradW3 = gradW3 + gW3

#finished going over entire training set, now update

updateW1 = gamma\*updateW1 + learningRate\*gradW1;

updateW2 = gamma\*updateW2 + learningRate\*gradW2;

updateW3 = gamma\*updateW3 + learningRate\*gradW3;

W1 = W1 - updateW1;

W2 = W2 - updateW2;

W3 = W3 - updateW3;

print(j, error)

if error < threshold:

q = q+1;

break

error = 0

if q == 1:

q = 0;

break;

error = 0;

for i in range(1115):

x1 = (np.append(one,testData[i])).reshape([101,1])

x2 = forward(W1, x1);

x3 = forward(W2, (np.append(one, x2)).reshape([301,1]));

out = forward(W3, (np.append(one, x3)).reshape([151,1]));

#out = fwd(W3, (np.append(one, x3)).reshape([151,1]));

targetOut = (testLabel[i].T).reshape([1,1]);

print(out, " ", targetOut)

print()

error += calcError(out, targetOut);

print(error\*2/1115)

