Abdul Basit Anees
21600659
EEE-443

Neural Networks

Assignment 2

Question 2

A

For single layer, the gradients are calculated as follows:

Using MSE loss:

$$\frac{\partial E}{\partial W_2} = grad\_w2$$

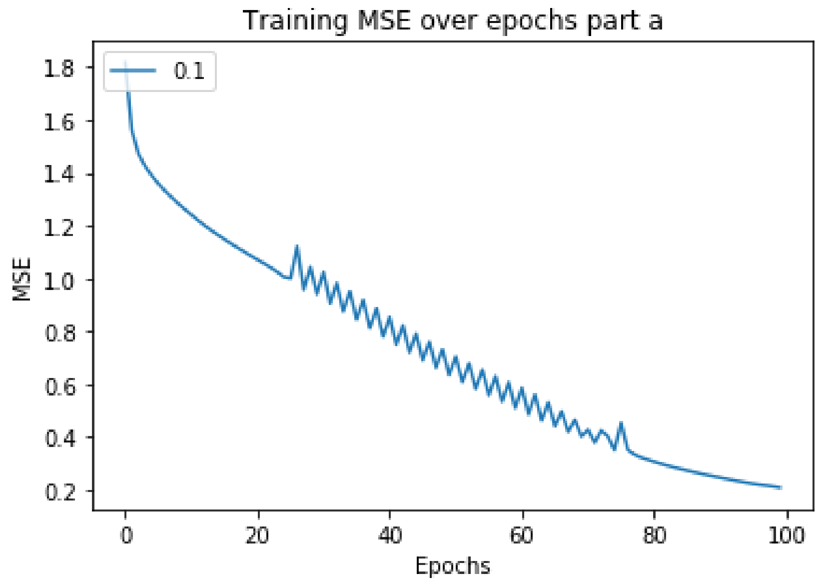
$$\frac{\partial E}{\partial W_1} = grad\_w1$$

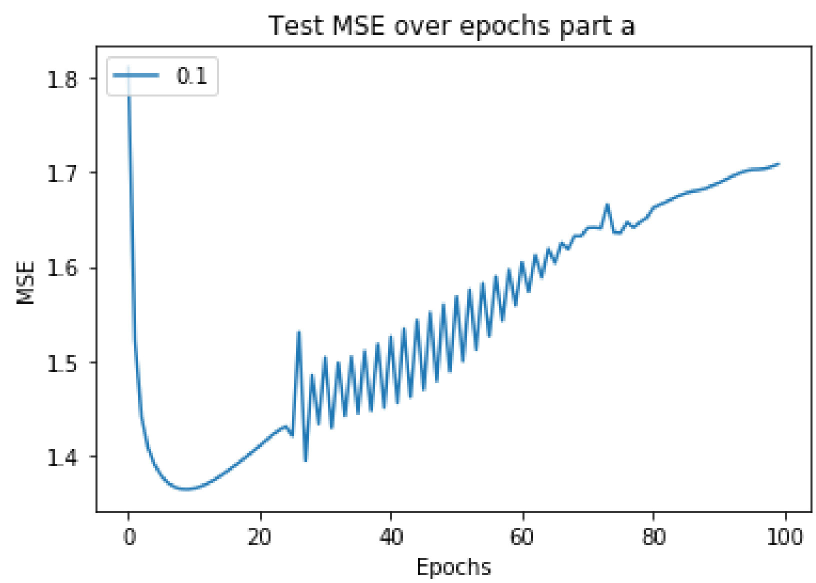
```python
def train(w_1, w_2, x, y, batch_size, rate, nBatches):
    for i in range(nBatches):
        #Load batch
        batch_x = x[i*batch_size:(i+1)*batch_size]
        batch_y = y[i*batch_size:(i+1)*batch_size]
        #Forward pass
        o_1  = tanh(batch_x @ w_1)
        o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
        y_p  = tanh(o_11 @ w_2)
        #Backward pass
        delta_2 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
        grad_w2 = o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
        grad_w1 = batch_x.T @ delta_1
        #Gradient descent
        w_1 = w_1 - rate * grad_w1
        w_2 = w_2 - rate * grad_w2
    return w_1, w_2
```

Following parameters for the model:

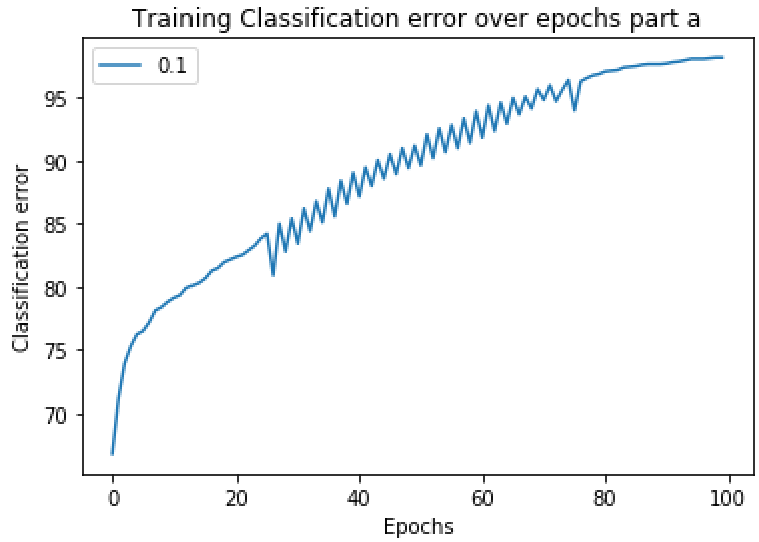
```python
#Initialize network parameters
N          = 25
epochs     = 100
batch_size = 256
std        = 0.003
L          = [0.1]
nBatches   = int(np.ceil(x_tr.shape[0]/batch_size))
w1         = np.random.normal(0, std, ((32 * 32) + 1, N))
w2         = np.random.normal(0, std, (N + 1, 1))
```

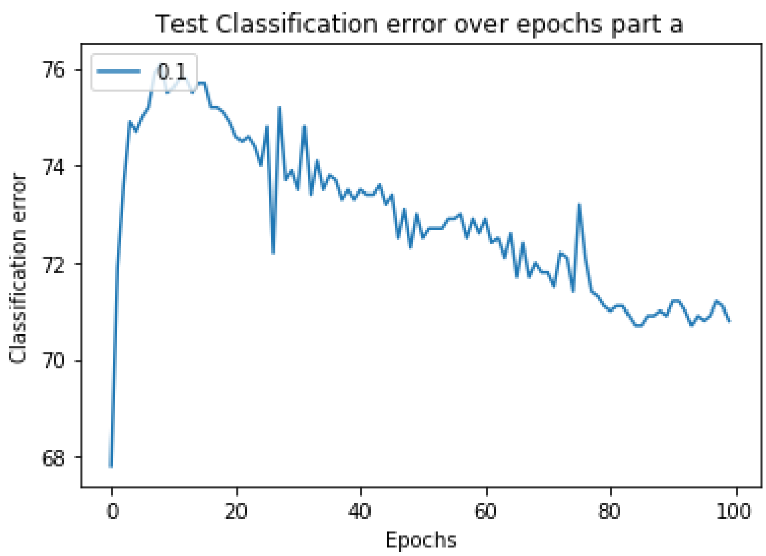
Abdul Basit Anees
21600659
EEE-443





Abdul Basit Anees
21600659
EEE-443





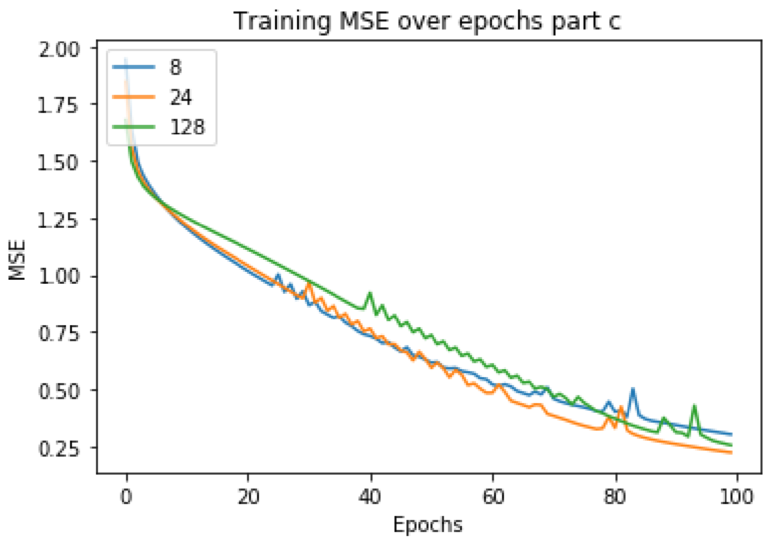
B

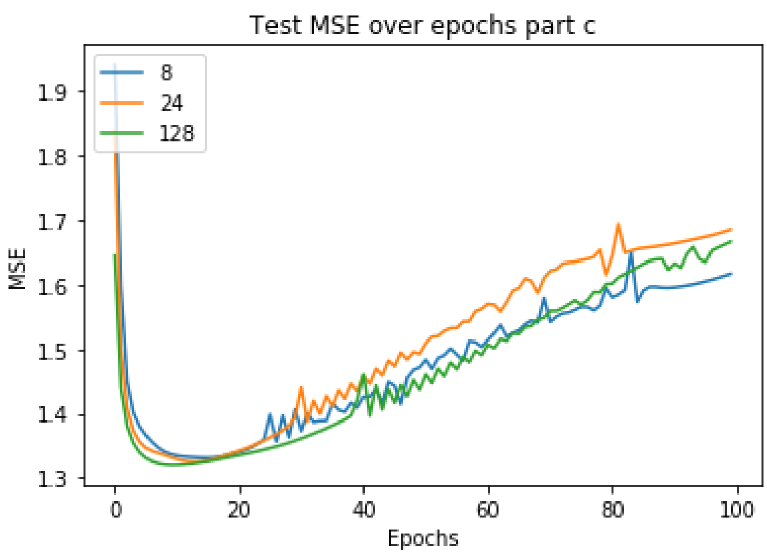
Training MSE decreases and accuracy increases over epochs, whereas same happens for test but until a few epochs after which the model overfits and we see an increase in MSE and decrease in accuracy for test.

C

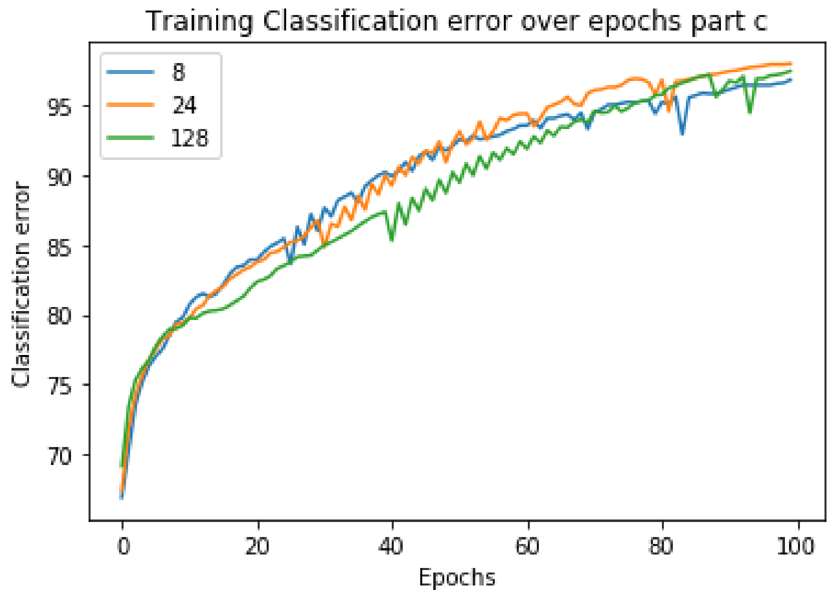
Abdul Basit Anees
21600659
EEE-443

```
#Initialize network parameters
N          = [8, 24, 128]
epochs     = 100
batch_size = 256
L          = 0.1
```





Abdul Basit Anees
21600659
EEE-443



More layers make the convergence faster.

D

For two layers, the gradient is calculated as follows as in the code:

Abdul Basit Anees
21600659
EEE-443

$$\frac{\partial E}{\partial w_3} = grad\_w3$$

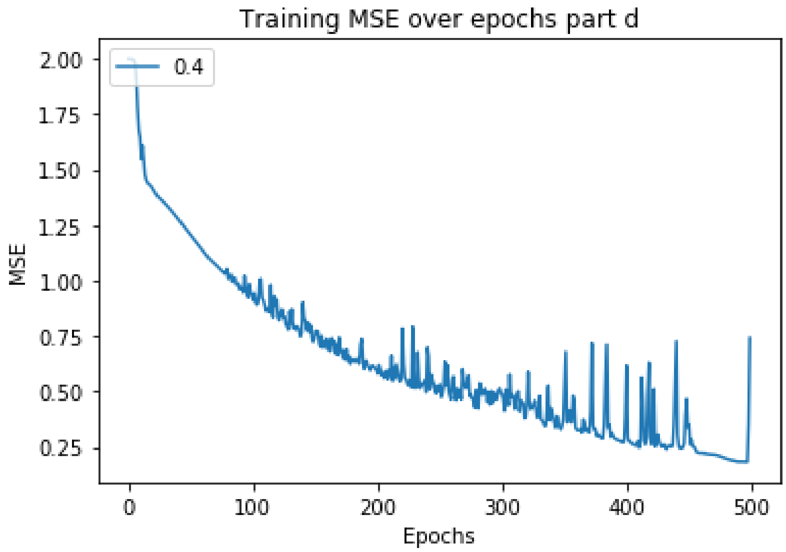
$$\frac{\partial E}{\partial w_2} = grad\_w2$$

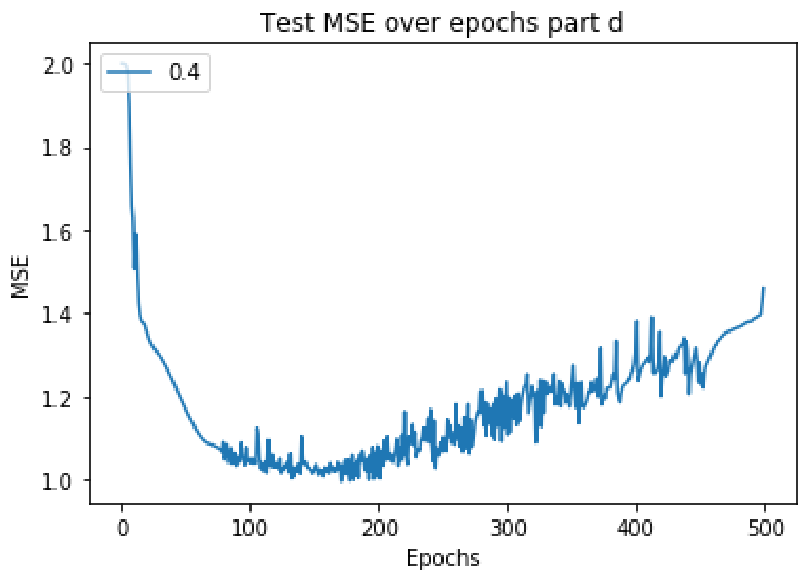
$$\frac{\partial E}{\partial w_1} = grad\_w1$$

```
#Load batch
batch_x = x[i*batch_size:(i+1)*batch_size]
batch_y = y[i*batch_size:(i+1)*batch_size]
#Forward pass
o_1  = tanh(batch_x @ w_1)
o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
o_2  = tanh(o_11 @ w_2)
o_22 = np.hstack((np.ones((o_2.shape[0],1)), o_2))
y_p  = tanh(o_22 @ w_3)
#Backward pass
delta_3 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
grad_w3 = o_22.T @ delta_3
delta_2 = (delta_3 @ w_3[1:].T) * (1 - o_2 ** 2)
grad_w2 = o_11.T @ delta_2
delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
grad_w1 = batch_x.T @ delta_1
#Gradient descent
w_1 = w_1 - rate * grad_w1
w_2 = w_2 - rate * grad_w2
w_3 = w_3 - rate * grad_w3
```

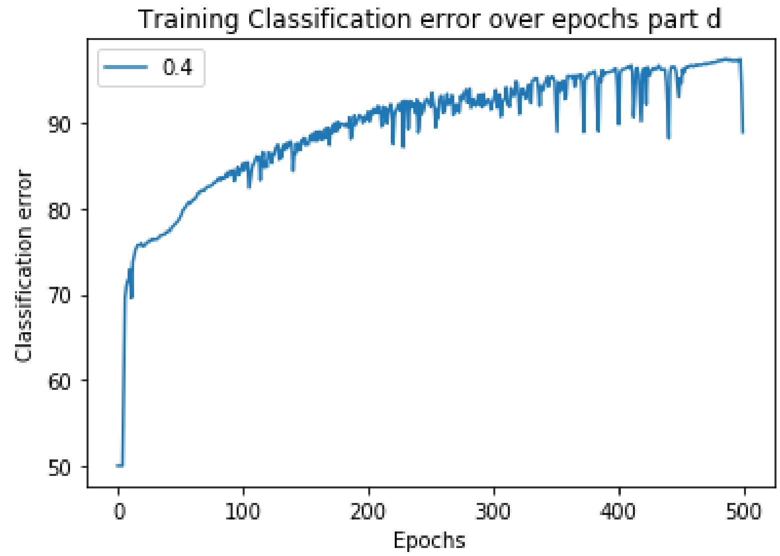
```
#Initialize network parameters
N1         = 16
N2         = 32
epochs     = 500
batch_size = 256
std        = 0.01
L          = [0.4]
```

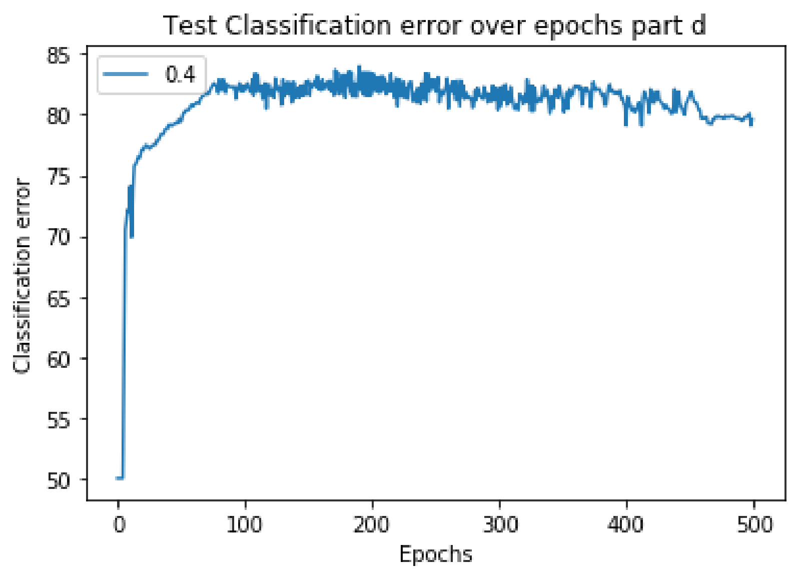
Abdul Basit Anees
21600659
EEE-443





Abdul Basit Anees
21600659
EEE-443





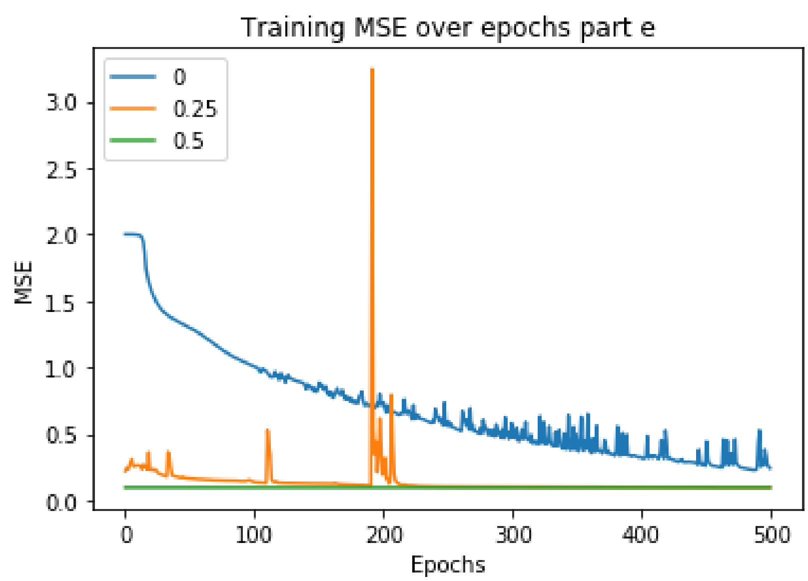
E

```
#Initialize network parameters
N1         = 16
N2         = 32
epochs     = 500
batch_size = 256
std        = 0.01
L          = 0.2
B          = [0, 0.25, 0.5]
```

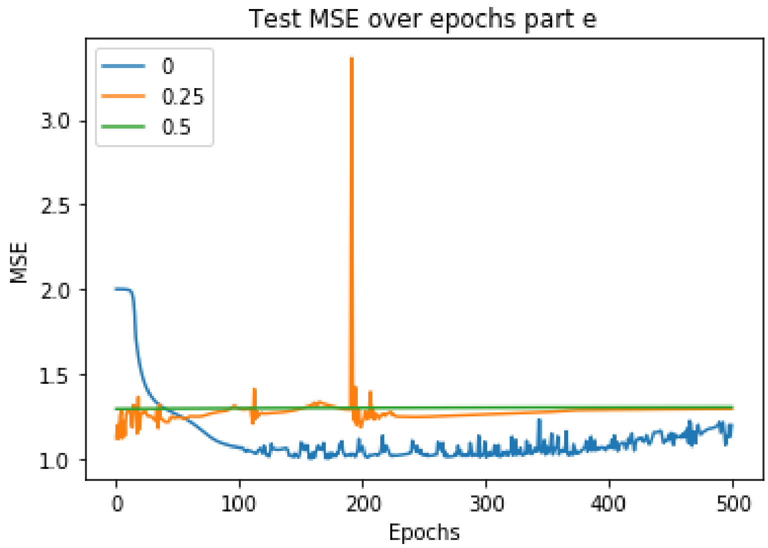
Abdul Basit Anees
21600659
EEE-443

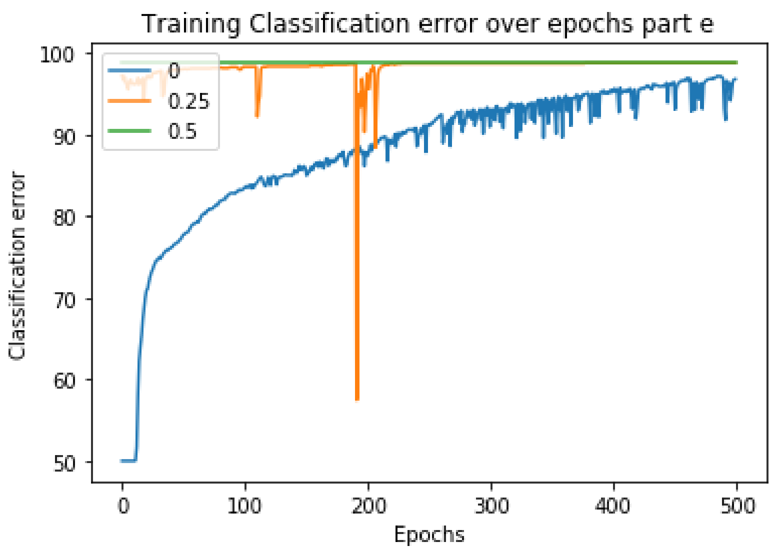
Momentum is implemented as

```
#Backward pass
delta_3 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
grad_w3 = o_22.T @ delta_3
delta_2 = (delta_3 @ w_3[1:].T) * (1 - o_2 ** 2)
grad_w2 = o_11.T @ delta_2
delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
grad_w1 = batch_x.T @ delta_1
#Momentum
v1 = B * v1 - rate * grad_w1
v2 = B * v2 - rate * grad_w2
v3 = B * v3 - rate * grad_w3
#Gradient descent
w_1 += v1
w_2 += v2
w_3 += v3
```

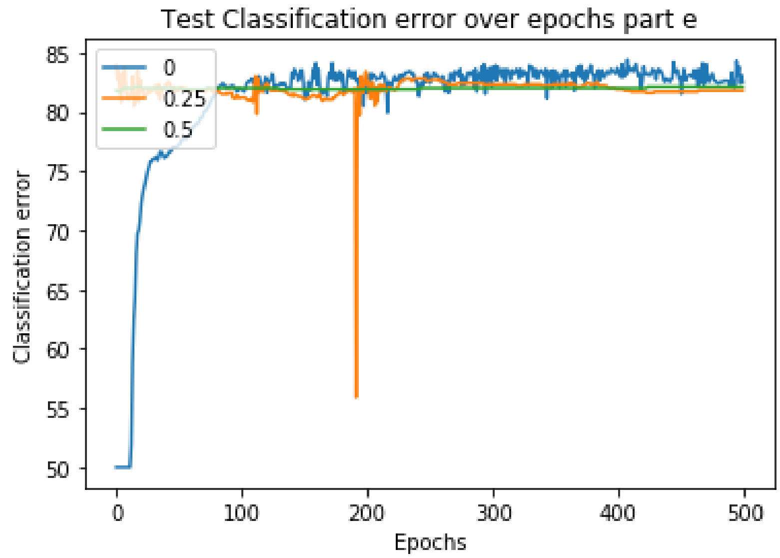


Abdul Basit Anees
21600659
EEE-443





Abdul Basit Anees
21600659
EEE-443



Question 3

Softmax(z_i) = $\frac{e^{zi}}{\Sigma_j e^{zj}}$

Forward propagation is as follows:

$y_1 = \varphi(w_1 x)$ where $\varphi(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$

$y_2 =$ softmax $(w_2 y_1)$

Back Propagation is as follows:

Using cross entropy loss function E, label t and prediction y

E(t,y) = $-\sum_{i=c}^{C} t_c \log(y_c)$

Derivation of y = softmax (z) wrt z is

Assuming i=j,

$$\frac{\partial y_i}{\partial z_i} = \frac{e^{z_i} \sum_{d=1}^{c} e^{z_d} - e^{z_i} e^{z_i}}{\left(\sum_{d=1}^{c} z_d\right)^2}$$

Abdul Basit Anees
21600659
EEE-443

$$= \frac{e^{z_i}}{\left(\sum_{d=1}^{c} z_d\right)}\left(1 - \frac{e^{z_i}}{\sum_{d=1}^{c} z_d}\right)$$

$$= y_i(1 - y_i)$$

Assuming i not equal to j,

$$\frac{\partial y_i}{\partial z_i} = -y_i y_j$$

We find derivative of error with respect to softmax input which is a function of $\frac{\partial y_i}{\partial z_i}$

$$\frac{\partial E}{\partial z_i} = -\sum_{j=1}^{c} \frac{\partial t_j \log(y_i)}{\partial z_i}$$

$$= y_i - t_i$$

Gradients:

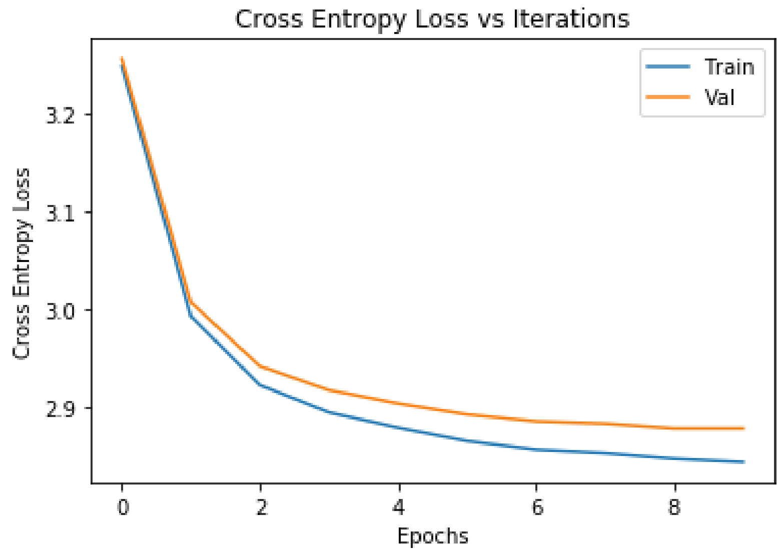
$$\frac{\partial E}{\partial w_2} = grad\_w2$$

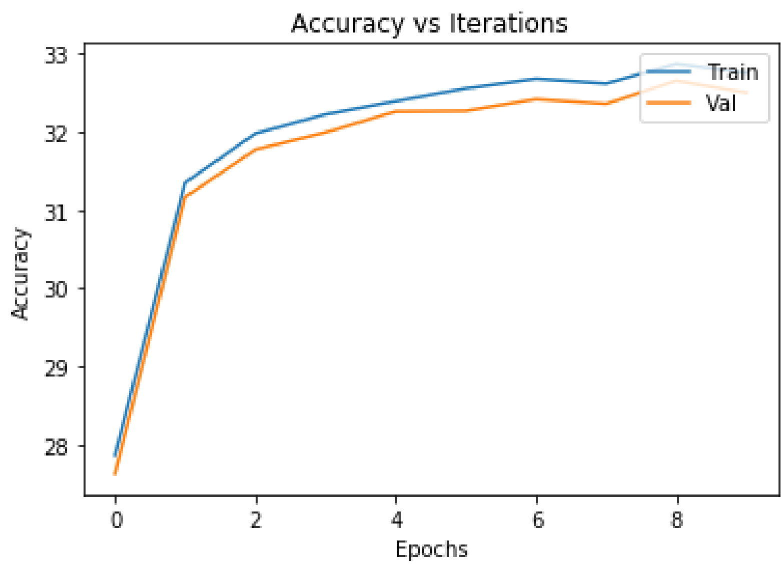
$$\frac{\partial E}{\partial w_1} = grad\_w1$$

$$\frac{\partial E}{\partial w_C} = grad\_c$$

```
#Forward
batch_x = x[i*batch_size:(i+1)*batch_size]
batch_y = y[i*batch_size:(i+1)*batch_size]
batch_size = len(batch_y)
 unique, count = np.unique(batch_x, return_counts = True)
 counts = dict(zip(unique, count))
embed_layer = np.hstack((np.ones((batch_size,1)), C[batch_x-1].reshape((batch_size, 3*D)) ))
o_1  = sigmoid(embed_layer @ w_1)
o_11 = np.hstack((np.ones((batch_size, 1)), o_1))
y_p  = softmax(o_11 @ w_2)
#Backward
delta_2 = (y_p - batch_y)
grad_w2 = (1 / batch_size) * o_11.T @ delta_2
delta_1 = (delta_2 @ w_2[1:].T) * (o_1*(1 - o_1))
grad_w1 = (1 / batch_size) * embed_layer.T @ delta_1
grad_c  = (delta_1 @ w_1[1:].T).reshape((batch_size, 3, D))
#Momentum
v_1 = B * v_1 - L * grad_w1
v_2 = B * v_2 - L * grad_w2
 vc = B * vc - L * grad_c
#Gradient descent
w_1 += v_1
w_2 += v_2
C[batch_x-1] = C[batch_x-1] - L * (grad_c)
```

Abdul Basit Anees
21600659
EEE-443

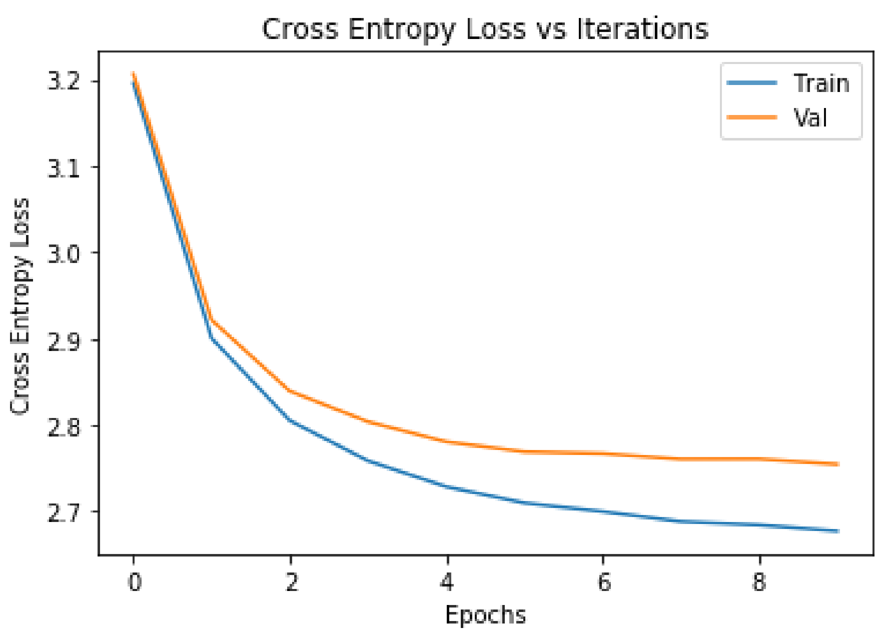




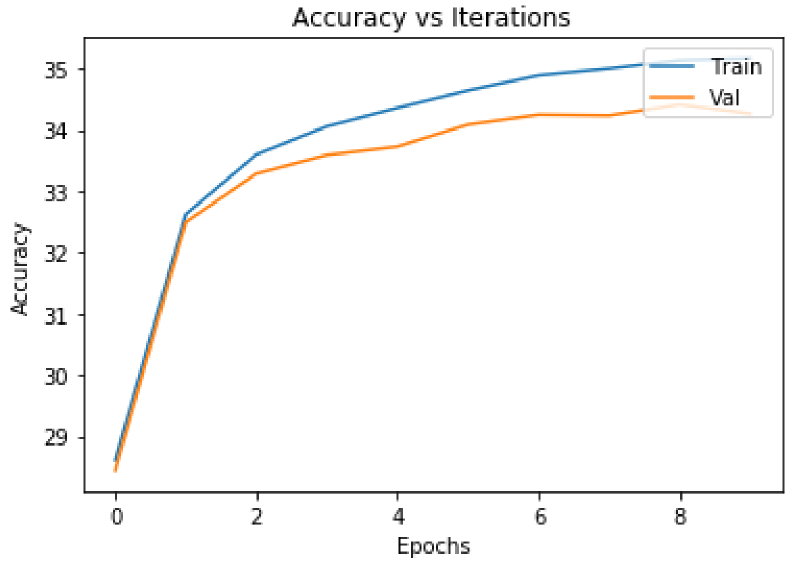
For D,P= 8, 32

```
can be called       | .
The top 10 candidates:
['for' 'up' 'me' 'her' 'him' ',' 'it' 'the' '?' '.']

they are well       | ,
The top 10 candidates:
['here' 'with' 'in' 'at' 'now' '?' 'for' 'there' ',' '.']

are well ,       | she
The top 10 candidates:
['what' 'you' 'it' 'they' 'and' 'i' 'but' 'too' 'she' 'he']

well , she       | said
The top 10 candidates:
['had' 'did' 'does' 'has' 'would' 'is' 'was' "'s" 'says' 'said']

, she said       | .
The top 10 candidates:
['they' 'to' 'today' ':' 'that' 'he' '?' 'it' ',' '.']

so she 's       | made
The top 10 candidates:
['here' 'back' 'all' 'like' '.' 'on' 'right' 'good' 'going' 'not']
```

D,P = 16,128



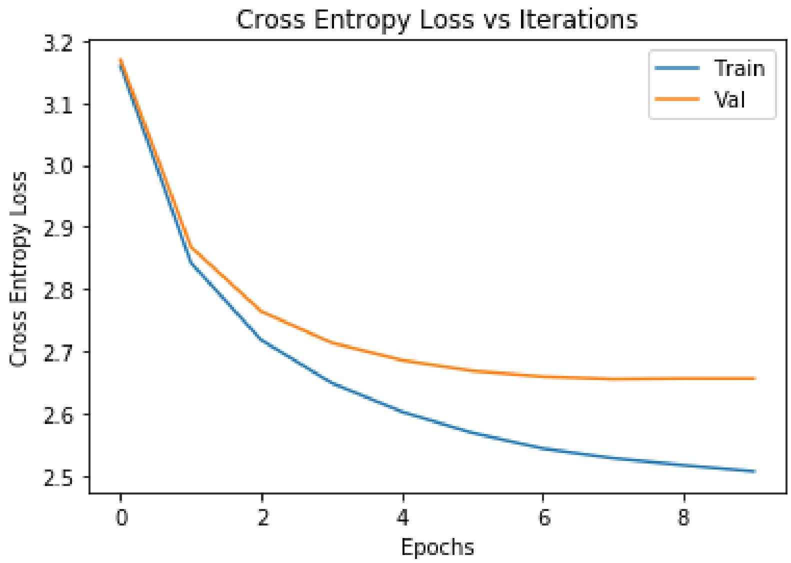
Abdul Basit Anees
21600659
EEE-443

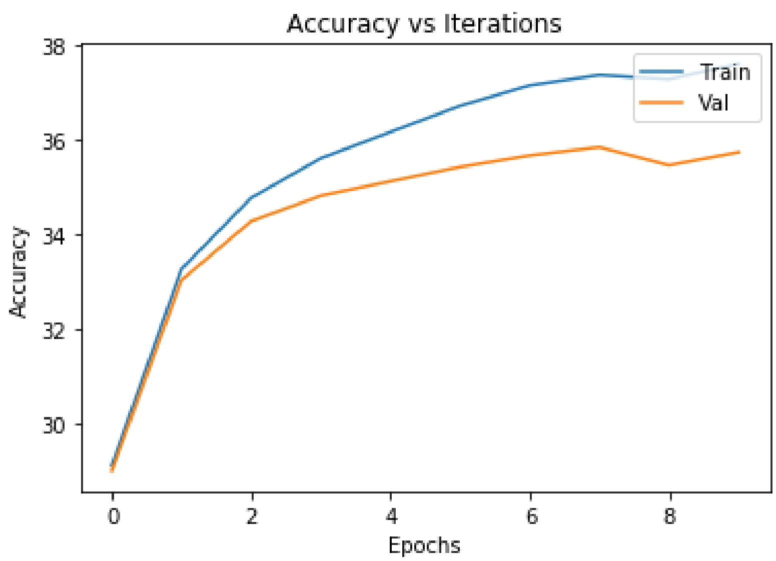



```
can be called        | .
The top 10 candidates:
['to' 'her' 'in' 'them' 'him' 'it' '.' 'up' 'the' '?']

they are well        | ,
The top 10 candidates:
['very' 'down' 'up' 'off' 'to' 'at' 'in' 'and' ',' '.']

are well ,        | she
The top 10 candidates:
['what' 'there' 'that' 'and' 'it' 'too' 'i' 'but' 'she' 'he']

well , she        | said
The top 10 candidates:
['would' 'had' 'is' 'did' 'does' 'has' "'s" 'says' 'was' 'said']

, she said        | .
The top 10 candidates:
['at' 'and' 'he' 'i' 'to' ':' '?' 'it' ',' '.']

so she 's        | made
The top 10 candidates:
['to' 'with' 'what' 'like' 'good' 'here' '.' 'not' 'going' 'all']
```

For D,P = 32,256

Abdul Basit Anees
21600659
EEE-443





```
can be called       | .
The top 10 candidates:
['and' ',' 'us' 'her' 'the' 'him' 'for' 'me' '.' '?']

they are well       | ,
The top 10 candidates:
['going' 'off' 'and' 'at' 'for' 'in' '?' 'on' ',' '.']

are well ,        | she
The top 10 candidates:
['it' 'and' 'they' 'too' 'what' 'i' 'you' 'but' 'she' 'he']

well , she        | said
The top 10 candidates:
['would' 'is' 'did' 'has' 'had' 'does' 'says' "'s" 'was' 'said']

, she said        | .
The top 10 candidates:
['yesterday' '?' 'and' 'it' 'then' 'to' ':' 'today' ',' '.']

so she 's        | made
The top 10 candidates:
['a' 'one' 'all' 'here' 'right' 'out' 'not' 'the' 'going' 'been']
```

Appendix

```
# -*- coding: utf-8 -*-

"""

Created on Sun Nov 17 21:56:22 2019


@author: basit

"""


import sys

import numpy as np

import matplotlib.pyplot as plt

import scipy.io

import pylab as py

import h5py


question = '3'
```

Abdul Basit Anees
21600659
EEE-443

```
def AbdulBasit_Anees_21600659_hw1(question):
    if question == '2' :
        print(question)
        return Q2()
    elif question == '3' :
        print(question)
        return Q3()


def Q2():
    print("Question 2 start")
    print("Q2 part a start")
    Q2a()
    print("Q2 part a end")
    print("Q2 part c start")
    Q2c()
    print("Q2 part c end")
    print("Q2 part d start")
    Q2d()
    print("Q2 part d end")
    print("Q2 part e start")
    Q2e()
    print("Q2 part e end")
    print("Question 2 end")


def Q3():
    print("Question 3 start")
    print("D = 8, P = 64")
    Q3a(8,64)
    print("D = 16, P = 128")
```

```
    Q3a(16,128)
    print("D = 32, P = 256")
    Q3a(32,256)
    print("Question 3 end")

def Q2a():
    #Load data
    data   = scipy.io.loadmat("assign2_data1.mat")
    x_tr   = data["trainims"].transpose((2,0,1))
    x_test = data["testims"].transpose((2,0,1))
    y_tr   = np.copy(data["trainlbls"].T).astype(np.int8)
    y_test = np.copy(data["testlbls"].T).astype(np.int8)

    #Manipulate data for Neural Network
    x_tr   = x_tr.reshape((x_tr.shape[0], 32 * 32))
    x_tr = (x_tr - np.mean(x_tr)) / np.std(x_tr)
    x_test = x_test.reshape((x_test.shape[0], 32 * 32))
    x_test = (x_test - np.mean(x_test)) / np.std(x_test)
    x_tr   = np.hstack((np.ones((x_tr.shape[0],1)), x_tr))
    x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))
    y_tr[y_tr == 0]     = -1
    y_test[y_test == 0] = -1

    #Shuffle data
    random = np.arange(x_tr.shape[0])
    np.random.shuffle(random)
    x_tr = x_tr[random]
    y_tr = y_tr[random]
```

Abdul Basit Anees
21600659
EEE-443

```
#Initialize network parameters
N          = 25
epochs     = 100
batch_size = 256
std        = 0.003
L          = [0.1]
nBatches   = int(np.ceil(x_tr.shape[0]/batch_size))
w1         = np.random.normal(0, std, ((32 * 32) + 1, N))
w2         = np.random.normal(0, std, (N + 1, 1))
J_train    = np.zeros((epochs, 1))
J_test     = np.zeros((epochs, 1))
acc_train  = np.zeros((epochs, 1))
acc_test   = np.zeros((epochs, 1))


#Activation funciton
def tanh(x):
    return (np.exp(2*x) - 1) / (np.exp(2*x) + 1)


#Forwrard pass
def forward(w_1, w_2, x_tr):
    o_1  = tanh(x_tr @ w_1)
    o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
    y_p  = tanh(o_11 @ w_2)
    return y_p


#Training including forward and backward pass for one epoch
def train(w_1, w_2, x, y, batch_size, rate, nBatches):
    for i in range(nBatches):
        #Load batch
```

```
        batch_x = x[i*batch_size:(i+1)*batch_size]
        batch_y = y[i*batch_size:(i+1)*batch_size]
        #Forward pass
        o_1  = tanh(batch_x @ w_1)
        o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
        y_p  = tanh(o_11 @ w_2)
        #Backward pass
        delta_2 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
        grad_w2 = o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
        grad_w1 = batch_x.T @ delta_1
        #Gradient descent
        w_1 = w_1 - rate * grad_w1
        w_2 = w_2 - rate * grad_w2
    return w_1, w_2

#Calculate accuracy
def accuracy(w_1, w_2, x_tr, y_tr, samples):
    y_p   = forward(w_1, w_2, x_tr)
    y_pr  = (2 * (y_p > 0)) - 1
    trues = np.sum(y_pr == y_tr)
    acc   = 100 * (trues / samples)
    return acc

#Plot data with labels
def plot(quantity, lists, epochs, ylabel, title):
    plt.figure()
    for rate in lists:
        plt.plot(np.arange(epochs),quantity[str(rate)], label = str(rate))
```

```
    py.legend(loc = 'upper left')
    plt.xlabel('Epochs')
    plt.ylabel(ylabel)
    plt.title(title)


mse_tr      = {}
mse_test    = {}
accur_train = {}
accur_test  = {}
#Train for different rates
for rate in L:
    w_1 = w1
    w_2 = w2
    for nEpoch in range(epochs):
        w_1, w_2         = train(w_1, w_2, x_tr, y_tr, batch_size, rate, nBatches)
        yp_tr            = forward(w_1, w_2, x_tr)
        yp_test          = forward(w_1, w_2, x_test)
        J_train[nEpoch]   = (1 / (0.5 * x_tr.shape[0])) * (y_tr - yp_tr).T @ (y_tr - yp_tr)
        J_test[nEpoch]    = (1 / (0.5 * x_test.shape[0])) * (y_test - yp_test).T @ (y_test - yp_test)
        acc_train[nEpoch] = accuracy(w_1, w_2, x_tr, y_tr, 1900)
        acc_test[nEpoch]  = accuracy(w_1, w_2, x_test, y_test, 1000)
    mse_tr[str(rate)]      = np.copy(J_train)
    mse_test[str(rate)]    = np.copy(J_test)
    accur_train[str(rate)] = np.copy(acc_train)
    accur_test[str(rate)]  = np.copy(acc_test)


#Plot observations
plot(mse_tr, L, epochs, 'MSE', 'Training MSE over epochs part a')
plot(mse_test, L, epochs, 'MSE', 'Test MSE over epochs part a')
```

```
    plot(accur_train, L, epochs, 'Classification error', 'Training Classification error over epochs part a')

    plot(accur_test, L, epochs, 'Classification error', 'Test Classification error over epochs part a')

    return mse_tr, mse_test, accur_train, accur_test


def Q2c():

    #Load data

    data   = scipy.io.loadmat("assign2_data1.mat")

    x_tr   = data["trainims"].transpose((2,0,1))

    x_test = data["testims"].transpose((2,0,1))

    y_tr   = np.copy(data["trainlbls"].T).astype(np.int8)

    y_test = np.copy(data["testlbls"].T).astype(np.int8)


    #Manipulate data for Neural Network

    x_tr   = x_tr.reshape((x_tr.shape[0], 32 * 32))

    x_tr = (x_tr - np.mean(x_tr)) / np.std(x_tr)

    x_test = x_test.reshape((x_test.shape[0], 32 * 32))

    x_test = (x_test - np.mean(x_test)) / np.std(x_test)

    x_tr   = np.hstack((np.ones((x_tr.shape[0],1)), x_tr))

    x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))

    y_tr[y_tr == 0]     = -1

    y_test[y_test == 0] = -1


    #Shuffle data

    random = np.arange(x_tr.shape[0])

    np.random.shuffle(random)

    x_tr = x_tr[random]

    y_tr = y_tr[random]


    #Initialize network parameters
```

Abdul Basit Anees
21600659
EEE-443

```
N         = [8, 24, 128]

epochs     = 100

batch_size = 256

L         = 0.1

nBatches   = int(np.ceil(x_tr.shape[0]/batch_size))

#w1        = np.random.normal(0, std, ((32 * 32) + 1, N))

#w2        = np.random.normal(0, std, (N + 1, 1))

J_train    = np.zeros((epochs, 1))

J_test     = np.zeros((epochs, 1))

acc_train  = np.zeros((epochs, 1))

acc_test   = np.zeros((epochs, 1))


#Activation funciton

def tanh(x):

    return (np.exp(2*x) - 1) / (np.exp(2*x) + 1)


#Forwrard pass

def forward(w_1, w_2, x_tr):

    o_1  = tanh(x_tr @ w_1)

    o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))

    y_p  = tanh(o_11 @ w_2)

    return y_p


#Training including forward and backward pass for one epoch

def train(w_1, w_2, x, y, batch_size, rate, nBatches):

    for i in range(nBatches):

        #Load batch

        batch_x = x[i*batch_size:(i+1)*batch_size]

        batch_y = y[i*batch_size:(i+1)*batch_size]
```

```
        #Forward pass
        o_1  = tanh(batch_x @ w_1)
        o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
        y_p  = tanh(o_11 @ w_2)
        #Backward pass
        delta_2 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
        grad_w2 = o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
        grad_w1 = batch_x.T @ delta_1
        #Gradient descent
        w_1 = w_1 - rate * grad_w1
        w_2 = w_2 - rate * grad_w2
    return w_1, w_2

#Calculate accuracy
def accuracy(w_1, w_2, x_tr, y_tr, samples):
    y_p   = forward(w_1, w_2, x_tr)
    y_pr  = (2 * (y_p > 0)) - 1
    trues = np.sum(y_pr == y_tr)
    acc   = 100 * (trues / samples)
    return acc

#Plot data with labels
def plot(quantity, lists, epochs, ylabel, title):
    plt.figure()
    for num in lists:
        plt.plot(np.arange(epochs),quantity[str(num)], label = str(num))
    py.legend(loc = 'upper left')
    plt.xlabel('Epochs')
```

```
    plt.ylabel(ylabel)

    plt.title(title)


mse_tr      = {}

mse_test    = {}

accur_train = {}

accur_test  = {}

#Train for different neuron numbers

std  = 0.003

rate = L

for neurons in N:

    w_1 = np.random.normal(0, std, ((32 * 32) + 1, neurons))

    w_2 = np.random.normal(0, std, (neurons + 1, 1))

    for nEpoch in range(epochs):

        w_1, w_2          = train(w_1, w_2, x_tr, y_tr, batch_size, rate, nBatches)

        yp_tr             = forward(w_1, w_2, x_tr)

        yp_test           = forward(w_1, w_2, x_test)

        J_train[nEpoch]   = (1 / (0.5 * x_tr.shape[0])) * (y_tr - yp_tr).T @ (y_tr - yp_tr)

        J_test[nEpoch]    = (1 / (0.5 * x_test.shape[0])) * (y_test - yp_test).T @ (y_test - yp_test)

        acc_train[nEpoch] = accuracy(w_1, w_2, x_tr, y_tr, 1900)

        acc_test[nEpoch]  = accuracy(w_1, w_2, x_test, y_test, 1000)

    mse_tr[str(neurons)]      = np.copy(J_train)

    mse_test[str(neurons)]    = np.copy(J_test)

    accur_train[str(neurons)] = np.copy(acc_train)

    accur_test[str(neurons)]  = np.copy(acc_test)


#Plot observations

plot(mse_tr, N, epochs, 'MSE', 'Training MSE over epochs part c')

plot(mse_test, N, epochs, 'MSE', 'Test MSE over epochs part c')
```

```
    plot(accur_train, N, epochs, 'Classification error', 'Training Classification error over epochs part c')

    plot(accur_test, N, epochs, 'Classification error', 'Test Classification error over epochs part c')


def Q2d():

    #Load data

    data   = scipy.io.loadmat("assign2_data1.mat")

    x_tr   = data["trainims"].transpose((2,0,1))

    x_test = data["testims"].transpose((2,0,1))

    y_tr   = np.copy(data["trainlbls"].T).astype(np.int8)

    y_test = np.copy(data["testlbls"].T).astype(np.int8)


    #Manipulate data for Neural Network

    x_tr   = x_tr.reshape((x_tr.shape[0], 32 * 32))/ 255

    x_tr   = np.hstack((np.ones((x_tr.shape[0],1)), x_tr))

    x_test = x_test.reshape((x_test.shape[0], 32 * 32))/ 255

    x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))

    y_tr[y_tr == 0]     = -1

    y_test[y_test == 0] = -1


    #Shuffle data

    random = np.arange(x_tr.shape[0])

    np.random.shuffle(random)

    x_tr = x_tr[random]

    y_tr = y_tr[random]


    #Initialize network parameters

    N1       = 16

    N2       = 32

    epochs     = 500
```

```
batch_size = 256

std       = 0.01

L        = [0.4]

nBatches   = int(np.ceil(x_tr.shape[0]/batch_size))

w1        = np.random.normal(0, std, ((32 * 32) + 1, N1))

w2        = np.random.normal(0, std, (N1 + 1, N2))

w3        = np.random.normal(0, std, (N2 + 1, 1))

J_train    = np.zeros((epochs, 1))

J_test     = np.zeros((epochs, 1))

acc_train  = np.zeros((epochs, 1))

acc_test   = np.zeros((epochs, 1))


#Activation function

def tanh(x):

   return (np.exp(2*x) - 1) / (np.exp(2*x) + 1)


#Forwrard pass

def forward(w_1, w_2, w_3, x_tr):

   o_1  = tanh(x_tr @ w_1)

   o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))

   o_2  = tanh(o_11 @ w_2)

   o_22 = np.hstack((np.ones((o_2.shape[0],1)), o_2))

   y_p  = tanh(o_22 @ w_3)

   return y_p


#Training including forward and backward pass for one epoch

def train(w_1, w_2, w_3, x, y, batch_size, rate, nBatches):

   for i in range(nBatches):

      #Load batch
```

```
        batch_x = x[i*batch_size:(i+1)*batch_size]
        batch_y = y[i*batch_size:(i+1)*batch_size]
        #Forward pass
        o_1  = tanh(batch_x @ w_1)
        o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
        o_2  = tanh(o_11 @ w_2)
        o_22 = np.hstack((np.ones((o_2.shape[0],1)), o_2))
        y_p  = tanh(o_22 @ w_3)
        #Backward pass
        delta_3 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
        grad_w3 = o_22.T @ delta_3
        delta_2 = (delta_3 @ w_3[1:].T) * (1 - o_2 ** 2)
        grad_w2 = o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
        grad_w1 = batch_x.T @ delta_1
        #Gradient descent
        w_1 = w_1 - rate * grad_w1
        w_2 = w_2 - rate * grad_w2
        w_3 = w_3 - rate * grad_w3
    return w_1, w_2, w_3


#Calculate accuracy
def accuracy(w_1, w_2, w_3, x_tr, y_tr, samples):
    y_p   = forward(w_1, w_2, w_3, x_tr)
    y_pr  = (2 * (y_p > 0)) - 1
    trues = np.sum(y_pr == y_tr)
    acc   = 100 * (trues / samples)
    return acc
```

Abdul Basit Anees
21600659
EEE-443

```
#Plot data with labels
def plot(quantity, lists, epochs, ylabel, title):
    plt.figure()
    for num in lists:
        plt.plot(np.arange(epochs),quantity[str(num)], label = str(num))
    py.legend(loc = 'upper left')
    plt.xlabel('Epochs')
    plt.ylabel(ylabel)
    plt.title(title)

mse_tr      = {}
mse_test    = {}
accur_train = {}
accur_test  = {}

#Train for different rates
for rate in L:
    w_1 = w1
    w_2 = w2
    w_3 = w3
    for nEpoch in range(epochs):
        w_1, w_2, w_3     = train(w_1, w_2, w_3, x_tr, y_tr, batch_size, rate, nBatches)
        yp_tr             = forward(w_1, w_2, w_3, x_tr)
        yp_test           = forward(w_1, w_2, w_3, x_test)
        J_train[nEpoch]   = (1 / (0.5 * x_tr.shape[0])) * (y_tr - yp_tr).T @ (y_tr - yp_tr)
        J_test[nEpoch]    = (1 / (0.5 * x_test.shape[0])) * (y_test - yp_test).T @ (y_test - yp_test)
        acc_train[nEpoch] = accuracy(w_1, w_2, w_3, x_tr, y_tr, 1900)
        acc_test[nEpoch]  = accuracy(w_1, w_2, w_3, x_test, y_test, 1000)
    mse_tr[str(rate)]     = np.copy(J_train)
```

```
        mse_test[str(rate)]    = np.copy(J_test)

        accur_train[str(rate)] = np.copy(acc_train)

        accur_test[str(rate)]  = np.copy(acc_test)


    #Plot observations

    plot(mse_tr, L, epochs, 'MSE', 'Training MSE over epochs part d')

    plot(mse_test, L, epochs, 'MSE', 'Test MSE over epochs part d')

    plot(accur_train, L, epochs, 'Classification error', 'Training Classification error over epochs part d')

    plot(accur_test, L, epochs, 'Classification error', 'Test Classification error over epochs part d')


def Q2e():

    #Load data

    data   = scipy.io.loadmat("assign2_data1.mat")

    x_tr   = data["trainims"].transpose((2,0,1))

    x_test = data["testims"].transpose((2,0,1))

    y_tr   = np.copy(data["trainlbls"].T).astype(np.int8)

    y_test = np.copy(data["testlbls"].T).astype(np.int8)


    #Manipulate data for Neural Network

    x_tr   = x_tr.reshape((x_tr.shape[0], 32 * 32))/ 255

    x_tr   = np.hstack((np.ones((x_tr.shape[0],1)), x_tr))

    x_test = x_test.reshape((x_test.shape[0], 32 * 32))/ 255

    x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))

    y_tr[y_tr == 0]     = -1

    y_test[y_test == 0] = -1


    #Shuffle data

    random = np.arange(x_tr.shape[0])

    np.random.shuffle(random)
```

```
x_tr = x_tr[random]

y_tr = y_tr[random]


#Initialize network parameters

N1        = 16

N2        = 32

epochs     = 500

batch_size = 256

std       = 0.01

L         = 0.2

B         = [0, 0.25, 0.5]

nBatches   = int(np.ceil(x_tr.shape[0]/batch_size))

w1        = np.random.normal(0, std, ((32 * 32) + 1, N1))

w2        = np.random.normal(0, std, (N1 + 1, N2))

w3        = np.random.normal(0, std, (N2 + 1, 1))

J_train    = np.zeros((epochs, 1))

J_test     = np.zeros((epochs, 1))

acc_train  = np.zeros((epochs, 1))

acc_test   = np.zeros((epochs, 1))


#Activation function

def tanh(x):

   return (np.exp(2*x) - 1) / (np.exp(2*x) + 1)


#Forwrard pass

def forward(w_1, w_2, w_3, x_tr):

   o_1  = tanh(x_tr @ w_1)

   o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))

   o_2  = tanh(o_11 @ w_2)
```

```
    o_22 = np.hstack((np.ones((o_2.shape[0],1)), o_2))
    y_p  = tanh(o_22 @ w_3)
    return y_p


#Training including forward and backward pass for one epoch
def train(w_1, w_2, w_3, x, y, batch_size, rate, nBatches, B):
    v1 = 0
    v2 = 0
    v3 = 0
    for i in range(nBatches):
        #Load batch
        batch_x = x[i*batch_size:(i+1)*batch_size]
        batch_y = y[i*batch_size:(i+1)*batch_size]
        #Forward pass
        o_1  = tanh(batch_x @ w_1)
        o_11 = np.hstack((np.ones((o_1.shape[0],1)), o_1))
        o_2  = tanh(o_11 @ w_2)
        o_22 = np.hstack((np.ones((o_2.shape[0],1)), o_2))
        y_p  = tanh(o_22 @ w_3)
        #Backward pass
        delta_3 = - (1 / batch_size) * (batch_y - y_p) * (1 - y_p ** 2)
        grad_w3 = o_22.T @ delta_3
        delta_2 = (delta_3 @ w_3[1:].T) * (1 - o_2 ** 2)
        grad_w2 = o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (1 - o_1 ** 2)
        grad_w1 = batch_x.T @ delta_1
        #Momentum
        v1 = B * v1 - rate * grad_w1
        v2 = B * v2 - rate * grad_w2
```

```
        v3 = B * v3 - rate * grad_w3
        #Gradient descent
        w_1 += v1
        w_2 += v2
        w_3 += v3
    return w_1, w_2, w_3

#Calculate accuracy
def accuracy(w_1, w_2, w_3, x_tr, y_tr, samples):
    y_p   = forward(w_1, w_2, w_3, x_tr)
    y_pr  = (2 * (y_p > 0)) - 1
    trues = np.sum(y_pr == y_tr)
    acc   = 100 * (trues / samples)
    return acc

#Plot data with labels
def plot(quantity, lists, epochs, ylabel, title):
    plt.figure()
    for num in lists:
        plt.plot(np.arange(epochs),quantity[str(num)], label = str(num))
    py.legend(loc = 'upper left')
    plt.xlabel('Epochs')
    plt.ylabel(ylabel)
    plt.title(title)

mse_tr      = {}
mse_test    = {}
accur_train = {}
accur_test  = {}
```

Abdul Basit Anees
21600659
EEE-443

```
    #Train for different rates
    rate = L
    for momentum in B:
        w_1 = w1
        w_2 = w2
        w_3 = w3
        for nEpoch in range(epochs):
            w_1, w_2, w_3     = train(w_1, w_2, w_3, x_tr, y_tr, batch_size, rate, nBatches, momentum)
            yp_tr             = forward(w_1, w_2, w_3, x_tr)
            yp_test           = forward(w_1, w_2, w_3, x_test)
            J_train[nEpoch]   = (1 / (0.5 * x_tr.shape[0])) * (y_tr - yp_tr).T @ (y_tr - yp_tr)
            J_test[nEpoch]    = (1 / (0.5 * x_test.shape[0])) * (y_test - yp_test).T @ (y_test - yp_test)
            acc_train[nEpoch] = accuracy(w_1, w_2, w_3, x_tr, y_tr, 1900)
            acc_test[nEpoch]  = accuracy(w_1, w_2, w_3, x_test, y_test, 1000)
        mse_tr[str(momentum)]      = np.copy(J_train)
        mse_test[str(momentum)]    = np.copy(J_test)
        accur_train[str(momentum)] = np.copy(acc_train)
        accur_test[str(momentum)]  = np.copy(acc_test)

    #Plot observations
    plot(mse_tr, B, epochs, 'MSE', 'Training MSE over epochs part e')
    plot(mse_test, B, epochs, 'MSE', 'Test MSE over epochs part e')
    plot(accur_train, B, epochs, 'Classification error', 'Training Classification error over epochs part e')
    plot(accur_test, B, epochs, 'Classification error', 'Test Classification error over epochs part e')

def Q3a(D,P):
    #Load data
    with h5py.File("assign2_data2.h5", 'r') as f:
```

```
    keys = list(f.keys())
    y_test = f[keys[0]].value
    x_test = f[keys[1]].value
    y_train = f[keys[2]].value
    x_train = f[keys[3]].value
    y_val = f[keys[4]].value
    x_val = f[keys[5]].value
    words = (f[keys[6]].value).astype('U13')
    f.close()

#Convert to one hot encoding
def to_categorical(y):
    out = np.zeros((len(y),250))
    out[np.arange(len(y)),y-1] = 1
    return out

#Sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

#Softmax function
def softmax(x):
    return np.exp(x) / (np.sum(np.exp(x), axis = 1).reshape((len(x),1)))

#Forward propagation
def forward(C, w_1, w_2, batch_x):
    embed_layer = np.hstack((np.ones((len(batch_x),1)), C[batch_x-1].reshape((len(batch_x), 3*D)) ))
    o_1  = sigmoid(embed_layer @ w_1)
    o_11 = np.hstack((np.ones((len(batch_x), 1)), o_1))
```

```
    y_p  = softmax(o_11 @ w_2)
    return y_p


#Traininf function for one epoch
def train(C, w_1, w_2, x, y, nBatches, L, B, batch_size):
    v_1 = 0
    v_2 = 0
#    v_c = 0
    for i in range(nBatches):
        #Forward
        batch_x = x[i*batch_size:(i+1)*batch_size]
        batch_y = y[i*batch_size:(i+1)*batch_size]
        batch_size = len(batch_y)
#        unique, count = np.unique(batch_x, return_counts = True)
#        counts = dict(zip(unique, count))
        embed_layer = np.hstack((np.ones((batch_size,1)), C[batch_x-1].reshape((batch_size, 3*D)) ))
        o_1  = sigmoid(embed_layer @ w_1)
        o_11 = np.hstack((np.ones((batch_size, 1)), o_1))
        y_p  = softmax(o_11 @ w_2)
        #Backward
        delta_2 = (y_p - batch_y)
        grad_w2 = (1 / batch_size) * o_11.T @ delta_2
        delta_1 = (delta_2 @ w_2[1:].T) * (o_1*(1 - o_1))
        grad_w1 = (1 / batch_size) * embed_layer.T @ delta_1
        grad_c  = (delta_1 @ w_1[1:].T).reshape((batch_size, 3, D))
        #Momentum
        v_1 = B * v_1 - L * grad_w1
        v_2 = B * v_2 - L * grad_w2
#        vc = B * vc - L * grad_c
```

```
        #Gradient descent
        w_1 += v_1
        w_2 += v_2
        C[batch_x-1] = C[batch_x-1] - L * (grad_c)
    return C, w_1, w_2

#Cross entropy error
def loss_CE(y_p, y_tr):
    J = - y_tr * np.log(y_p)
    J[np.isnan(J)] = 0
    return np.sum(J)/len(y_p)

#Accuracy calcuation
def accuracy(C, w_1, w_2, batch_x, y_tr, samples):
    y_p = forward(C, w_1, w_2, batch_x)
    y_pr = (np.argmax(y_p, 1).reshape((len(batch_x)))) + 1
    trues = np.sum(y_pr == y_tr)
    acc = 100 * (trues / samples)
    return acc

#Top k accuracy, separate one made becasue this is computationanlly more expensive for k != 1
def accuracy_topK(C, w_1, w_2, batch_x, y_tr, samples, k):
    if k == 1:
        return accuracy(C, w_1, w_2, batch_x, y_tr, samples)
    y_p   = forward(C, w_1, w_2, batch_x)
    y_pr  = (np.argsort(y_p, axis = 1))[:,250-k:] + 1
    trues = np.sum(np.any(y_pr == y_tr.reshape((samples,1)), axis = 1))
    acc   = 100 * (trues / samples)
    return acc
```

Abdul Basit Anees
21600659
EEE-443

```
#Initialize network parameters
L   = 0.25
B   = 0.85
k   = 1
epochs     = 10
batch_size = 200
nBatches   = int(np.ceil(x_train.shape[0]/batch_size))
#Initialize weights
std = 0.01
C   = np.random.normal(0, std, (250, D))
w_1 = np.random.normal(0, std, (3*D + 1, P))
w_2 = np.random.normal(0, std, (P + 1, 250))
#Initialize arrays to store errors and accuracies
loss_tr   = np.zeros((epochs))
loss_test = np.zeros((epochs))
loss_val  = np.zeros((epochs))
acc_tr    = np.zeros((epochs))
acc_test  = np.zeros((epochs))
acc_val   = np.zeros((epochs))
#Convert labels to one hot encoding
y_train1  = to_categorical(y_train)
y_val1    = to_categorical(y_val)
y_test1   = to_categorical(y_test)
#Start training
for j in range(epochs):
    C, w_1, w_2 = train(C, w_1, w_2, x_train, y_train1, nBatches, L, B, batch_size)
    acc_val[j] = accuracy_topK(C, w_1, w_2, x_val, y_val, len(y_val), k)
    acc_tr[j] = accuracy_topK(C, w_1, w_2, x_train, y_train, len(y_train), k)
```

```
    y_p_tr = forward(C, w_1, w_2, x_train)
    loss_tr[j] = loss_CE(y_p_tr, y_train1)
    y_p_val = forward(C, w_1, w_2, x_val)
    loss_val[j] = loss_CE(y_p_val, y_val1)
    print(j)


def plot(a, b, metric, alabel, blabel):
    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel(metric)
    plt.plot(np.arange(epochs), a, label = alabel)
    plt.plot(np.arange(epochs), b, label = blabel)
    plt.title(''.join((metric,' vs Iterations')))
    py.legend(loc = 'upper right')
#Plot results
plot(loss_tr, loss_val, 'Cross Entropy Loss', 'Train', 'Val')
plot(acc_tr, acc_val, 'Accuracy', 'Train', 'Val')
#Save final CE error in dictionary
ce_final = {}
test_p          = forward(C, w_1, w_2, x_test)
ce_final['Train'] = loss_tr[epochs - 1]
ce_final['Val']   = loss_val[epochs - 1]
ce_final['Test']  = loss_CE(test_p, y_test1)
#Save final top k accuracies in dictionary
k = 10
acc_final = {}
acc_final['Train'] = accuracy_topK(C, w_1, w_2, x_train, y_train, len(y_train), k)
acc_final['Val']   = accuracy_topK(C, w_1, w_2, x_val, y_val, len(y_val), k)
acc_final['Test']  = accuracy_topK(C, w_1, w_2, x_test, y_test, len(y_test), k)
```

Abdul Basit Anees
21600659
EEE-443

```
    test_pr = (np.argmax(test_p, 1).reshape((len(x_test)))) + 1
    for i in range(50):
        a = (x_test[i]-1).tolist()
        print(words[a[0]], words[a[1]], words[a[2]], " | ", words[test_pr[i]-1])

    for i in range(50):
        a = (x_test[i]-1).tolist()
        print(words[a[0]], words[a[1]], words[a[2]], "      |", words[y_test[i]-1])
        print("The top 10 candidates:")
        b = np.argsort(test_p[i])[len(test_p[i])-10:]
        print(words[b])
        print("")

output = AbdulBasit_Anees_21600659_hw1(question)
```