CS6910 Fundamentals of Deep Learning Code

Assignment -1

Group Number : 11

Arabhi Subhash - cs17b005
Allada Praharsh - cs17b036
Allumalla Ravi Kiran -cs17b038

February 2020

1 Function Approximation

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math

data=pd.read_csv('func/train100.txt',delimiter=' ',header=None)
txt = pd.DataFrame(data).to_numpy()

X_train = txt[:, :2]
Y_train = txt[:, 2]

# plot of given data
ax = plt.axes(projection='3d')
ax.scatter3D(txt[:,0],txt[:,1],txt[:,2])
plt.show()


# Parameters
lam=1
learning_rate=0.01
learning_Rate=0.01
num_iterations=30000
delta_cost = 1e-6
beta = 0.9
```

```
m=100


# Random initialization of weights
W1 = np.random.randn(30,2) *np.sqrt(2/2)
b1 = np.zeros(shape=(30, 1))
W2 = np.random.randn(10, 30) * np.sqrt(2/30)
b2 = np.zeros(shape=(10, 1))
W3 = np.random.randn(1, 10) *np.sqrt(2/10)
b3 = np.zeros(shape=(1, 1))

costs=[]
cost = 0

# initialisation of velocities
vW1=np.zeros_like(W1)
vW2=np.zeros_like(W2)
vW3=np.zeros_like(W3)
vb1=np.zeros_like(b1)
vb2=np.zeros_like(b2)
vb3=np.zeros_like(b3)

i = 0
while(True):

#   forward propagation
    Z1 = np.dot(W1, X_train.T) + b1
    A1 = np.tanh(Z1)
    D1 = np.random.rand(A1.shape[0], A1.shape[1])
    Z2 = np.dot(W2, A1) + b2
    A2 = np.tanh(Z2)
    D2 = np.random.rand(A2.shape[0], A2.shape[1])
    Z3 = np.dot(W3, A2) + b3
    A3 = Z3

#   cost calculation
    prev_cost = cost
    cost = (np.sum((Y_train-A3)**2)/m)+(lam* (np.sum(np.square(W1)) + np.sum(np.square(W2))
+ np.sum(np.square(W3))) / (2 * m))
    if i % 100 == 0:
        costs.append(cost)

#     backpropagation
```

```
    dZ3 = A3 - Y_train
    dW3 = (1 / m) * np.dot(dZ3, A2.T)+((lam * W3) / m)
    db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)
    dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.power(A2, 2))
    dW2 = (1 / m) * np.dot(dZ2, A1.T)+((lam * W2) / m)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = (1 / m) * np.dot(dZ1, X_train)+((lam * W1) / m)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

#     update perameters
    vW1 = beta * vW1 + (1 - beta) * dW1
    vb1 = beta * vb1 + (1 - beta) * db1
    vW2 = beta * vW2 + (1 - beta) * dW2
    vb2 = beta * vb2 + (1 - beta) * db2
    vW3 = beta * vW3 + (1 - beta) * dW3
    vb3 = beta * vb3 + (1 - beta) * db3

    dW1 = vW1
    db1 = vb1
    dW2 = vW2
    db2 = vb2
    dW3 = vW3
    db3 = vb3

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    W3 = W3 - learning_rate * dW3
    b3 = b3 - learning_rate * db3

    if(i%1000==0):
        print("cost ",cost)

    if(abs(prev_cost - cost) < delta_cost or i >= num_iterations):
        break

    i += 1
```

```
# Avg. error vs Epoch plot

print(i)
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Avg. error vs Epoch plot - Function approx.")
plt.show()

# Checking Train on the Model

m=X_train.T.shape[1]

Z1 = np.dot(W1, X_train.T) + b1
# print(np.shape(Z1))
A1 = np.tanh(Z1)
D1 = np.random.rand(A1.shape[0], A1.shape[1])
# print(np.shape(A1))
Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)
D2 = np.random.rand(A2.shape[0], A2.shape[1])
Z3 = np.dot(W3, A2) + b3
A3 = Z3
# print(np.shape(A3))
# print(A3)

cost = np.sum((Y_train.T-A3)**2)/m
print(cost)

# Scatter plot

pred = A3.flatten()
plt.plot(Y_train,pred,".")
plt.xlabel("True")
plt.ylabel("Predicted")
plt.title("Scatter plot - Function approx.")
plt.plot([-70 , 80], [-70, 80])
plt.show()

# Surface plot

def plotting(x, y):
   p = np.array([[x,y]]).T
```

```
    #print(np.shape(p))
    #J1 = np.dot(W1, [[x],[y]]) + b1
    J1 = []
    for u in W1:
        val = u[0]*x + u[1]*y
        J1.append(val)
    J1 = np.array(J1)
    O1 = np.tanh(J1)
    #J2 = np.dot(W2, O1) + b2
    J2 = []
    for u in W2:
        val = 0
        for y in range(len(u)):
            val = val + O1[y]*u[y]
        J2.append(val)
    J2 = np.array(J2)
    O2 = np.tanh(J2)
    #J3 = np.dot(W3, O2) + b3
    J3 = []
    for u in W3:
        val = 0
        for y in range(len(u)):
            val = val + O2[y]*u[y]
        J3.append(val)
    return J3[0]

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
Z = plotting(X,Y)
ax = plt.axes(projection="3d")
ax.scatter3D(txt[:,0],txt[:,1],txt[:,2],c='r')
ax.plot_surface(X,Y,Z)
plt.title("Surface plot Training - Function approx.")
plt.show()

# Development Data

data=pd.read_csv('func/val.txt',delimiter=' ',header=None)

text = pd.DataFrame(data).to_numpy()
```

```
X_val = text[:, :2]
Y_val = text[:, 2]

# Surface Plot

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
Z = plotting(X,Y)
ax = plt.axes(projection="3d")
ax.scatter3D(text[:,0],text[:,1],text[:,2],c='r')
ax.plot_surface(X,Y,Z)
plt.title("Surface plot Development - Function approx.")
plt.show()

# Working on Development data

m=X_val.T.shape[1]

Z1 = np.dot(W1, X_val.T) + b1
# print(np.shape(Z1))
A1 = np.tanh(Z1)
D1 = np.random.rand(A1.shape[0], A1.shape[1])
# print(np.shape(A1))
Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)
D2 = np.random.rand(A2.shape[0], A2.shape[1])
Z3 = np.dot(W3, A2) + b3
A3 = Z3
# print(np.shape(A3))
# print(A3)

cost = np.sum((Y_val.T-A3)**2)/m
print(cost)
```

2 2D Non-Linear Data

```
# importing required libraries;
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd

# reading data from csv and extracting data points belonging to each class and restructuring
# and using first 100 points from each class as train and rest 50 each class as test;

data=np.genfromtxt("traingroup11.csv",delimiter=',',skip_header=1)
print(data.shape)
c0=data[data[:,2]==0.0]
c1=data[data[:,2]==1.0]
c2=data[data[:,2]==2.0]
data1=np.zeros((0,3))
for i in range(451):
    if i%3==0:
        data1=np.append(data1,c0[int(i/3)].reshape(1,3),axis=0)
    elif i&3==1:
        data1=np.append(data1,c1[int(i/3)].reshape(1,3),axis=0)
    else:
        data1=np.append(data1,c2[int(i/3)].reshape(1,3),axis=0)
test=data1[301:,:]
train=data1[0:300,:]
X_train = train[:, :2]
Y_train = train[:, 2]
X_train1=X_train
Y_train1=Y_train
expected_out=Y_train


plt.scatter(X_train1[:,0],X_train1[:,1], c=Y_train1, s=30, cmap=plt.cm.Spectral);

l1_nodes=7
l2_nodes=5

plt.scatter(data1[:,0],data1[:,1], c=data1[:,2], s=30, cmap=plt.cm.Spectral);

X_val = test[:, :2]
Y_val = test[:, 2]
expected_out1=Y_val

X_train=X_train.T
Y_train=Y_train.T
Y_train.shape
a=Y_train
```

```
Y_train=Y_train.astype(int)

#converting Y_train to different usable form;

temp = np.zeros((Y_train.size, Y_train.max()+1))
temp[np.arange(Y_train.size),Y_train] = 1
Y_train=temp

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

# neural network model function
def mlffnn(X, Y,  num_iterations ,print_cost=False,learning_rate = 0.01,beta = 0.9):
    req_wts=[]

    #initializing weights;

    W1 = np.random.randn(l1_nodes,2) * np.sqrt(2/l1_nodes)
    b1 = np.zeros(shape=(l1_nodes, 1))
    W2 = np.random.randn(l2_nodes, l1_nodes) * np.sqrt(2/l2_nodes)
    b2 = np.zeros(shape=(l2_nodes, 1))
    W3 = np.random.randn(3, l2_nodes) * np.sqrt(2/3)
    b3 = np.zeros(shape=(3, 1))


    vW1 = np.zeros_like(W1)
    vb1 = np.zeros_like(b1)
    vW2 = np.zeros_like(W2)
    vb2= np.zeros_like(b2)
    vW3 = np.zeros_like(W3)
    vb3= np.zeros_like(b3)


    costs=[]
    for i in range(0, num_iterations):

    #farward propagation steps, parameter calculation;
        Z1 = np.dot(W1, X) + b1
        A1 = np.tanh(Z1)

        Z2 = np.dot(W2, A1) + b2
        A2 = np.tanh(Z2)
```

```
Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)


m=300

#calculating cost;
cost = cost = (-1/m)*np.sum(np.multiply(np.log(A3), Y) + np.multiply((1 - Y), np.log(1 - A3)))
if i % 100 == 0:
    costs.append(cost)

# Backpropagation steps, parameter calculation;
dZ3 = A3 - Y
dW3 = (1 / m) * np.dot(dZ3, A2.T)
db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)
dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.power(A2, 2))
dW2 = (1 / m) * np.dot(dZ2, A1.T)
db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
dW1 = (1 / m) * np.dot(dZ1, X.T)
db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

# parameter updating using generalised delta method;
vW1 = beta * vW1 + (1 - beta) * dW1
vb1 = beta * vb1 + (1 - beta) * db1
vW2 = beta * vW2 + (1 - beta) * dW2
vb2 = beta * vb2 + (1 - beta) * db2
vW3 = beta * vW3 + (1 - beta) * dW3
vb3 = beta * vb3 + (1 - beta) * db3

W1 = W1 - learning_rate * vW1
b1 = b1 - learning_rate * vb1
W2 = W2 - learning_rate * vW2
b2 = b2 - learning_rate * vb2
W3 = W3 - learning_rate * vW3
b3 = b3 - learning_rate * vb3


if(print_cost and i%1000==0):
    print("cost ",cost)

weights = {"W1": W1,
```

```
                "b1": b1,
                "W2": W2,
                "b2": b2,
                "W3": W3,
                "b3": b3
               }
        if (i==1 or i==2 or i==10 or i==50 or i==60000-1):
            req_wts+=[weights]
    return weights,costs,req_wts

weights,costs,req_wts = mlffnn(X_train, Y_train.T,  num_iterations = 60000, print_cost=True)

# plot of epochs vs cost;
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.show()

def predict(weights, X):

    W1 = weights['W1']
    b1 = weights['b1']
    W2 = weights['W2']
    b2 = weights['b2']
    W3 = weights['W3']
    b3 = weights['b3']



    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)

    Z2 = np.dot(W2, A1) + b2
    A2 = np.tanh(Z2)

    Z3 = np.dot(W3, A2) + b3
    A3 = sigmoid(Z3)
    predictions = np.argmax(A3,axis=0)

    return predictions

pred=predict(weights, X_train)
count=0
```

```
count1=0
pred1=predict(weights,X_val.T)
i=0
j=0
while i<300:
    if(pred[i]==expected_out[i]):
        count=count+1
    i=i+1
while j<150:
    if(pred1[j]==expected_out1[j]):
        count1=count1+1
    j=j+1
print("train accuracy =",count/301)
print("test accuracy =",count1/150)

# ploting decision boundary overlaped with train data;
def plot_decision_boundary(weights, X):
    # Set min and max values and give it some padding
    x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
    y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
    h = 0.1
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole grid
    Z = (np.c_[xx.ravel(), yy.ravel()])
    Z = (np.c_[xx.ravel(), yy.ravel()])
    pred=predict(weights, Z.T)
    pred=pred.reshape(xx.shape)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.scatter(xx, yy, c=pred, cmap=plt.cm.Spectral)
    plt.scatter(c0[0:105,0],c0[0:105,1] , s=30)
    plt.scatter(c1[0:105,0],c1[0:105,1] , s=30)
    plt.scatter(c2[0:105,0],c2[0:105,1] , s=30)
    plt.title("Decision Region with train-data points for 2d-data")
plot_decision_boundary(weights, X_train)



# ploting decision boundary overlaped with test data;
def plot_decision_boundary(weights, X):
    # Set min and max values and give it some padding
    x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
```

```
    y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
    h = 0.1
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole grid
    Z = (np.c_[xx.ravel(), yy.ravel()])
    Z = (np.c_[xx.ravel(), yy.ravel()])
    pred=predict(weights, Z.T)
    pred=pred.reshape(xx.shape)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.scatter(xx, yy, c=pred, cmap=plt.cm.Spectral)
    plt.scatter(c0[105:,0],c0[105:,1] , s=30)
    plt.scatter(c1[105:,0],c1[105:,1] , s=30)
    plt.scatter(c2[105:,0],c2[105:,1] , s=30)
    plt.title("Decision Region with test-data points for 2d-data")
plot_decision_boundary(weights, X_train)
```

```
# functions for plotting node output of each layer;
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

def last_layer(j,wts,epoch):

    W1 = wts['W1']
    b1 = wts['b1']
    W2 = wts['W2']
    b2 = wts['b2']
    W3 = wts['W3']
    b3 = wts['b3']
    x_min, x_max = X_train[0, :].min() - 1, X_train[0, :].max() + 1
    y_min, y_max = X_train[1, :].min() - 1, X_train[1, :].max() + 1
    b = np.arange(x_min,x_max, 0.2)
    d = np.arange(y_min, y_max, 0.2)

    B, D = np.meshgrid(b, d)
    X =np.c_[B.ravel(), D.ravel()]
    x1=np.tanh(np.dot(W1, X.T) + b1)
    x2=np.tanh(np.dot(W2, x1) + b2)
```

```
    x3=sigmoid(np.dot(W3, x2) + b3)
    nu=x3[j]
    nu=nu.reshape(B.shape[0],B.shape[1])
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(B, D, nu,cmap=plt.cm.Spectral)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title("epochs-"+epoch+" outputlayer, node-"+str(j+1))
    plt.show()
#     plt.savefig("epochs-"+epoch+" outputlayer-node-"+str(j+1)+".jpg")
def first_layer(j,wts,epoch):

    W1 = wts['W1']
    b1 = wts['b1']
    W2 = wts['W2']
    b2 = wts['b2']
    W3 = wts['W3']
    b3 = wts['b3']
    x_min, x_max = X_train[0, :].min() - 1, X_train[0, :].max() + 1
    y_min, y_max = X_train[1, :].min() - 1, X_train[1, :].max() + 1
    b = np.arange(x_min,x_max, 0.2)
    d = np.arange(y_min, y_max, 0.2)

    B, D = np.meshgrid(b, d)
    X =np.c_[B.ravel(), D.ravel()]
    x1=np.tanh(np.dot(W1, X.T) + b1)
    nu=x1[j]
    nu=nu.reshape(B.shape[0],B.shape[1])
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(B, D, nu,cmap=plt.cm.Spectral)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title("epochs-"+epoch+" hiddenlayer-1, node-"+str(j+1))
    plt.show()
#     plt.savefig("epochs-"+epoch+" hiddenlayer1-node-"+str(j+1)+".jpg")
def second_layer(j,wts,epoch):

    W1 = wts['W1']
    b1 = wts['b1']
    W2 = wts['W2']
    b2 = wts['b2']
```

```
    W3 = wts['W3']
    b3 = wts['b3']
    x_min, x_max = X_train[0, :].min() - 1, X_train[0, :].max() + 1
    y_min, y_max = X_train[1, :].min() - 1, X_train[1, :].max() + 1
    b = np.arange(x_min,x_max, 0.2)
    d = np.arange(y_min, y_max, 0.2)

    B, D = np.meshgrid(b, d)
    X =np.c_[B.ravel(), D.ravel()]
    x1=np.tanh(np.dot(W1, X.T) + b1)
    x2=np.tanh(np.dot(W2, x1) + b2)
    nu=x2[j]
    nu=nu.reshape(B.shape[0],B.shape[1])
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(B, D, nu,cmap=plt.cm.Spectral)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title("epochs-"+epoch+" hiddenlayer-2, node-"+str(j+1))
    plt.show()
#     plt.savefig("epochs-"+epoch+" hiddenlayer2-node-"+str(j+1)+".jpg")




def print_every_layer_node_outputs(p,i,e):

    print("first layer 8 nodes-",e)
    print (" iterations")
    first_layer(0,p,e)
    first_layer(1,p,e)
    first_layer(2,p,e)
    first_layer(3,p,e)
    first_layer(4,p,e)
    first_layer(5,p,e)
    first_layer(6,p,e)
#     first_layer(7,p)
    print("second layer 5 nodes-",e)
    second_layer(0,p,e)
    second_layer(1,p,e)
    second_layer(2,p,e)
    second_layer(3,p,e)
    second_layer(4,p,e)
```

```
    print("last layer 3 nodes-",e)
    last_layer(0,p,e)
    last_layer(1,p,e)
    last_layer(2,p,e)



def print_all_required_images():
    for i in range(len(req_wts)):
        temp=0
        if(i==0):
            temp='1'
        elif(i==1):
            temp='2'
        elif(i==2):
            temp='10'
        elif(i==3):
            temp='50'
        elif(i==4):
            temp='end'
        print_every_layer_node_outputs(req_wts[i],i,temp)



print_all_required_images()
```

3 Image Classification

```
def sigmoid(x):
  return 1/(1+np.exp(-x))

import os
import numpy as np
import matplotlib.pyplot as plt
from random import shuffle

path = './Feature_extraction_2D/'
class_names = os.listdir(path)

print(class_names)
```

```
val = 0
data_points = []
data_points_class = []
for i in class_names:
    load_name = os.path.join(path,i)
    extracted_features = np.load(load_name)
    for j in extracted_features:
        data_points.append(j)
        data_points_class.append(val)
    print(val,i)
    val += 1

temp = list(zip(data_points,data_points_class))
shuffle(temp)

data_points,data_points_class = zip(*temp)
data_points = np.asanyarray(data_points)
data_points_class = np.asanyarray(data_points_class)
print(data_points_class)

np.shape(data_points)

# setting nodes

dim_hidden_1 = 40
dim_hidden_2 = 28
pca_components = 512
C = 0.01

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(data_points, data_points_class,test_size = 0.2)

# from sklearn import decomposition

# pca1 = decomposition.PCA(n_components = pca_components)
# pca1.fit(X_train)
# X_train = pca1.transform(X_train)
# print(X_train.shape)

# pca2 = decomposition.PCA(n_components = pca_components)
# pca2.fit(X_test)
# X_test = pca2.transform(X_test)
# print(X_test.shape)
```

```
ytrain = Y_train
b = np.zeros((Y_train.size, Y_train.max()+1))
b[np.arange(Y_train.size),Y_train] = 1
Y_train=b
print(Y_train)

# Random Initialization and Parameters


learning_rate=0.0005
learning_Rate=0.0005
num_iterations=60000
delta_cost = 1e-8
beta = 0.9
gamma = 0.99
m=1200
lam=0.4
A_W1 = np.random.randn(dim_hidden_1,pca_components) * np.sqrt(2/pca_components)
A_b1 = np.zeros(shape=(dim_hidden_1, 1))
A_W2 = np.random.randn(dim_hidden_2, dim_hidden_1) * np.sqrt(2/dim_hidden_1)
A_b2 = np.zeros(shape=(dim_hidden_2, 1))
A_W3 = np.random.randn(5, dim_hidden_2) * np.sqrt(2/dim_hidden_2)
A_b3 = np.zeros(shape=(5, 1))


opcode = 2

W1=A_W1
W2=A_W2
W3=A_W3
b1=A_b1
b2=A_b2
b3=A_b3

costs=[]
cost = 0
vW1 = np.zeros_like(W1)
vW2 = np.zeros_like(W2)
vW3 = np.zeros_like(W3)
vb1 = np.zeros_like(b1)
vb2 = np.zeros_like(b2)
vb3 = np.zeros_like(b3)
```

```
rvW1 = np.zeros_like(W1)
rvW2 = np.zeros_like(W2)
rvW3 = np.zeros_like(W3)
rvb1 = np.zeros_like(b1)
rvb2 = np.zeros_like(b2)
rvb3 = np.zeros_like(b3)

# Training

for i in range(0, num_iterations):
# forward prop
    Z1 = np.dot(W1, X_train.T) + b1
    A1 = np.tanh(Z1)

    Z2 = np.dot(W2, A1) + b2
    A2 = np.tanh(Z2)

    Z3 = np.dot(W3, A2) + b3
    A3 = sigmoid(Z3)

#  cost
    prev_cost = cost
    logprobs = np.multiply(np.log(A3), Y_train.T) + np.multiply((1 - Y_train.T), np.log(1 - A3))
    cost =( - np.sum(logprobs) / m)+(lam * (np.sum(np.square(W1)) + np.sum(np.square(W2)) +
np.sum(np.square(W3))) / (2 * m))
    cost = float(np.squeeze(cost))
    if i % 100 == 0:
        costs.append(cost)

# back prop
    dZ3 = A3 - Y_train.T
    dW3 = (1 / m) * np.dot(dZ3, A2.T)+((lam* W3) / m)
    db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)
    dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.power(A2, 2))
    dW2 = (1 / m) * np.dot(dZ2, A1.T)+((lam * W2) / m)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = (1 / m) * np.dot(dZ1, X_train)+(lam * W1) / m
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    if(opcode == 0):
```

```
    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    W3 = W3 - learning_rate * dW3
    b3 = b3 - learning_rate * db3

else:
    vW1 = beta * vW1 + (1 - beta) * dW1
    vb1 = beta * vb1 + (1 - beta) * db1
    vW2 = beta * vW2 + (1 - beta) * dW2
    vb2 = beta * vb2 + (1 - beta) * db2
    vW3 = beta * vW3 + (1 - beta) * dW3
    vb3 = beta * vb3 + (1 - beta) * db3
    if(opcode == 1):

        dW1 = vW1
        db1 = vb1
        dW2 = vW2
        db2 = vb2
        dW3 = vW3
        db3 = vb3

        W1 = W1 - learning_rate * dW1
        b1 = b1 - learning_rate * db1
        W2 = W2 - learning_rate * dW2
        b2 = b2 - learning_rate * db2
        W3 = W3 - learning_rate * dW3
        b3 = b3 - learning_rate * db3

    else:
        rvW1 = gamma * rvW1 + (1 - gamma) * np.power(dW1,2)
        rvb1 = gamma * rvb1 + (1 - gamma) * np.power(db1,2)
        rvW2 = gamma * rvW2 + (1 - gamma) * np.power(dW2,2)
        rvb2 = gamma * rvb2 + (1 - gamma) * np.power(db2,2)
        rvW3 = gamma * rvW3 + (1 - gamma) * np.power(dW3,2)
        rvb3 = gamma * rvb3 + (1 - gamma) * np.power(db3,2)

        E = 1e-08

        vvW1=np.zeros_like(W1)
        vvW2=np.zeros_like(W2)
        vvW3=np.zeros_like(W3)
```

```
        vvb1=np.zeros_like(b1)
        vvb2=np.zeros_like(b2)
        vvb3=np.zeros_like(b3)

        svW1=np.zeros_like(W1)
        svW2=np.zeros_like(W2)
        svW3=np.zeros_like(W3)
        svb1=np.zeros_like(b1)
        svb2=np.zeros_like(b2)
        svb3=np.zeros_like(b3)


        vvW1 = vW1/(1 - np.power(beta,(i+1)))
        vvb1 = vb1/(1 - np.power(beta,(i+1)))
        vvW2 = vW2/(1 - np.power(beta,(i+1)))
        vvb2 = vb2/(1 - np.power(beta,(i+1)))
        vvW3 = vW3/(1 - np.power(beta,(i+1)))
        vvb3 = vb3/(1 - np.power(beta,(i+1)))

        svW1 = rvW1/(1 - np.power(gamma,(i+1)))
        svb1 = rvb1/(1 - np.power(gamma,(i+1)))
        svW2 = rvW2/(1 - np.power(gamma,(i+1)))
        svb2 = rvb2/(1 - np.power(gamma,(i+1)))
        svW3 = rvW3/(1 - np.power(gamma,(i+1)))
        svb3 = rvb3/(1 - np.power(gamma,(i+1)))

        dW1 = vvW1 / np.sqrt(svW1 + E)
        db1 = vvb1 / np.sqrt(svb1 + E)
        dW2 = vvW2 / np.sqrt(svW2 + E)
        db2 = vvb2 / np.sqrt(svb2 + E)
        dW3 = vvW3 / np.sqrt(svW3 + E)
        db3 = vvb3 / np.sqrt(svb3 + E)

        W1 = W1 - learning_rate * dW1
        b1 = b1 - learning_rate * db1
        W2 = W2 - learning_rate * dW2
        b2 = b2 - learning_rate * db2
        W3 = W3 - learning_rate * dW3
        b3 = b3 - learning_rate * db3

    if(i%1000==0):
        print("cost "+str(i/1000),cost)
```

```
    if(abs(prev_cost - cost) < delta_cost):
        break

# Avg. error vs Epoch plot

plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Avg. error vs Epoch plot - Adam - Image data")
plt.show()

# Working on training data

Z1 = np.dot(W1, X_train.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred = np.argmax(A3,axis=0)

# Working on development data




Z1 = np.dot(W1, X_test.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred1= np.argmax(A3,axis=0)

# accuracy scores

count=0
```

```
i=0
while i<1200:
    if(pred[i]==ytrain[i]):
        count=count+1
    i=i+1
print("train accuracy =",count/1200)

count=0
i=0
while i<300:
    if(pred1[i]==Y_test[i]):
        count=count+1
    i=i+1
print("test accuracy =",count/300)

the_plot_conf(pred,ytrain,'Training - Adam - Image Data')

the_plot_conf(pred1,Y_test,'Development - Adam - Image Data')



opcode = 1

W1=A_W1
W2=A_W2
W3=A_W3
b1=A_b1
b2=A_b2
b3=A_b3

costs=[]
cost = 0
vW1 = np.zeros_like(W1)
vW2 = np.zeros_like(W2)
vW3 = np.zeros_like(W3)
vb1 = np.zeros_like(b1)
vb2 = np.zeros_like(b2)
vb3 = np.zeros_like(b3)

rvW1 = np.zeros_like(W1)
rvW2 = np.zeros_like(W2)
rvW3 = np.zeros_like(W3)
rvb1 = np.zeros_like(b1)
```

```
rvb2 = np.zeros_like(b2)
rvb3 = np.zeros_like(b3)

# Training

for i in range(0, num_iterations):
# forward prop
   Z1 = np.dot(W1, X_train.T) + b1
   A1 = np.tanh(Z1)

   Z2 = np.dot(W2, A1) + b2
   A2 = np.tanh(Z2)

   Z3 = np.dot(W3, A2) + b3
   A3 = sigmoid(Z3)

#  cost
   prev_cost = cost
   logprobs = np.multiply(np.log(A3), Y_train.T) + np.multiply((1 - Y_train.T), np.log(1 - A3))
   cost =( - np.sum(logprobs) / m)+(lam * (np.sum(np.square(W1)) + np.sum(np.square(W2)) +
np.sum(np.square(W3))) / (2 * m))
   cost = float(np.squeeze(cost))
   if i % 100 == 0:
      costs.append(cost)

# back prop
   dZ3 = A3 - Y_train.T
   dW3 = (1 / m) * np.dot(dZ3, A2.T)+((lam* W3) / m)
   db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)
   dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.power(A2, 2))
   dW2 = (1 / m) * np.dot(dZ2, A1.T)+((lam * W2) / m)
   db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
   dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
   dW1 = (1 / m) * np.dot(dZ1, X_train)+(lam * W1) / m
   db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

   if(opcode == 0):

      W1 = W1 - learning_rate * dW1
      b1 = b1 - learning_rate * db1
      W2 = W2 - learning_rate * dW2
      b2 = b2 - learning_rate * db2
      W3 = W3 - learning_rate * dW3
```

```
    b3 = b3 - learning_rate * db3

else:
    vW1 = beta * vW1 + (1 - beta) * dW1
    vb1 = beta * vb1 + (1 - beta) * db1
    vW2 = beta * vW2 + (1 - beta) * dW2
    vb2 = beta * vb2 + (1 - beta) * db2
    vW3 = beta * vW3 + (1 - beta) * dW3
    vb3 = beta * vb3 + (1 - beta) * db3
    if(opcode == 1):

        W1 = W1 - learning_rate * vW1
        b1 = b1 - learning_rate * vb1
        W2 = W2 - learning_rate * vW2
        b2 = b2 - learning_rate * vb2
        W3 = W3 - learning_rate * vW3
        b3 = b3 - learning_rate * vb3

    else:
        rvW1 = gamma * rvW1 + (1 - gamma) * np.power(dW1,2)
        rvb1 = gamma * rvb1 + (1 - gamma) * np.power(db1,2)
        rvW2 = gamma * rvW2 + (1 - gamma) * np.power(dW2,2)
        rvb2 = gamma * rvb2 + (1 - gamma) * np.power(db2,2)
        rvW3 = gamma * rvW3 + (1 - gamma) * np.power(dW3,2)
        rvb3 = gamma * rvb3 + (1 - gamma) * np.power(db3,2)

        E = 1e-08

        vvW1=np.zeros_like(W1)
        vvW2=np.zeros_like(W2)
        vvW3=np.zeros_like(W3)
        vvb1=np.zeros_like(b1)
        vvb2=np.zeros_like(b2)
        vvb3=np.zeros_like(b3)

        svW1=np.zeros_like(W1)
        svW2=np.zeros_like(W2)
        svW3=np.zeros_like(W3)
        svb1=np.zeros_like(b1)
        svb2=np.zeros_like(b2)
        svb3=np.zeros_like(b3)
```

```
        vvW1 = vW1/(1 - np.power(beta,(i+1)))
        vvb1 = vb1/(1 - np.power(beta,(i+1)))
        vvW2 = vW2/(1 - np.power(beta,(i+1)))
        vvb2 = vb2/(1 - np.power(beta,(i+1)))
        vvW3 = vW3/(1 - np.power(beta,(i+1)))
        vvb3 = vb3/(1 - np.power(beta,(i+1)))

        svW1 = rvW1/(1 - np.power(gamma,(i+1)))
        svb1 = rvb1/(1 - np.power(gamma,(i+1)))
        svW2 = rvW2/(1 - np.power(gamma,(i+1)))
        svb2 = rvb2/(1 - np.power(gamma,(i+1)))
        svW3 = rvW3/(1 - np.power(gamma,(i+1)))
        svb3 = rvb3/(1 - np.power(gamma,(i+1)))

        dW1 = vvW1 / np.sqrt(svW1 + E)
        db1 = vvb1 / np.sqrt(svb1 + E)
        dW2 = vvW2 / np.sqrt(svW2 + E)
        db2 = vvb2 / np.sqrt(svb2 + E)
        dW3 = vvW3 / np.sqrt(svW3 + E)
        db3 = vvb3 / np.sqrt(svb3 + E)

        W1 = W1 - learning_rate * dW1
        b1 = b1 - learning_rate * db1
        W2 = W2 - learning_rate * dW2
        b2 = b2 - learning_rate * db2
        W3 = W3 - learning_rate * dW3
        b3 = b3 - learning_rate * db3

    if(i%1000==0):
        print("cost "+str(i/1000),cost)
    if(abs(prev_cost - cost) < delta_cost):
        break

# Avg. error vs Epoch plot

plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Avg. error vs Epoch plot - Gen Delta - Image data")
plt.show()

# Working on training data
```

```
Z1 = np.dot(W1, X_train.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred = np.argmax(A3,axis=0)

# Working on development data



Z1 = np.dot(W1, X_test.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred1= np.argmax(A3,axis=0)

# accuracy scores

count=0
i=0
while i<1200:
    if(pred[i]==ytrain[i]):
        count=count+1
    i=i+1
print("train accuracy =",count/1200)

count=0
i=0
while i<300:
    if(pred1[i]==Y_test[i]):
        count=count+1
    i=i+1
```

```
print("test accuracy =",count/300)

the_plot_conf(pred,ytrain,'Training - gen-Delta - Image Data')

the_plot_conf(pred1,Y_test,'Development - gen-Delta - Image Data')



opcode = 0

W1=A_W1
W2=A_W2
W3=A_W3
b1=A_b1
b2=A_b2
b3=A_b3

costs=[]
cost = 0
vW1 = np.zeros_like(W1)
vW2 = np.zeros_like(W2)
vW3 = np.zeros_like(W3)
vb1 = np.zeros_like(b1)
vb2 = np.zeros_like(b2)
vb3 = np.zeros_like(b3)

rvW1 = np.zeros_like(W1)
rvW2 = np.zeros_like(W2)
rvW3 = np.zeros_like(W3)
rvb1 = np.zeros_like(b1)
rvb2 = np.zeros_like(b2)
rvb3 = np.zeros_like(b3)

# Training

for i in range(0, num_iterations):
# forward prop
   Z1 = np.dot(W1, X_train.T) + b1
   A1 = np.tanh(Z1)

   Z2 = np.dot(W2, A1) + b2
   A2 = np.tanh(Z2)
```

```
    Z3 = np.dot(W3, A2) + b3
    A3 = sigmoid(Z3)

#  cost
    prev_cost = cost
    logprobs = np.multiply(np.log(A3), Y_train.T) + np.multiply((1 - Y_train.T), np.log(1 - A3))
    cost =( - np.sum(logprobs) / m)+(lam * (np.sum(np.square(W1)) + np.sum(np.square(W2)) +
np.sum(np.square(W3))) / (2 * m))
    cost = float(np.squeeze(cost))
    if i % 100 == 0:
        costs.append(cost)

# back prop
    dZ3 = A3 - Y_train.T
    dW3 = (1 / m) * np.dot(dZ3, A2.T)+((lam* W3) / m)
    db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)
    dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.power(A2, 2))
    dW2 = (1 / m) * np.dot(dZ2, A1.T)+((lam * W2) / m)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = (1 / m) * np.dot(dZ1, X_train)+(lam * W1) / m
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    if(opcode == 0):

        W1 = W1 - learning_rate * dW1
        b1 = b1 - learning_rate * db1
        W2 = W2 - learning_rate * dW2
        b2 = b2 - learning_rate * db2
        W3 = W3 - learning_rate * dW3
        b3 = b3 - learning_rate * db3

    else:
        vW1 = beta * vW1 + (1 - beta) * dW1
        vb1 = beta * vb1 + (1 - beta) * db1
        vW2 = beta * vW2 + (1 - beta) * dW2
        vb2 = beta * vb2 + (1 - beta) * db2
        vW3 = beta * vW3 + (1 - beta) * dW3
        vb3 = beta * vb3 + (1 - beta) * db3
        if(opcode == 1):

            dW1 = vW1
            db1 = vb1
```

```
    dW2 = vW2
    db2 = vb2
    dW3 = vW3
    db3 = vb3

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    W3 = W3 - learning_rate * dW3
    b3 = b3 - learning_rate * db3

else:
    rvW1 = gamma * rvW1 + (1 - gamma) * np.power(dW1,2)
    rvb1 = gamma * rvb1 + (1 - gamma) * np.power(db1,2)
    rvW2 = gamma * rvW2 + (1 - gamma) * np.power(dW2,2)
    rvb2 = gamma * rvb2 + (1 - gamma) * np.power(db2,2)
    rvW3 = gamma * rvW3 + (1 - gamma) * np.power(dW3,2)
    rvb3 = gamma * rvb3 + (1 - gamma) * np.power(db3,2)

    E = 1e-08

    vvW1=np.zeros_like(W1)
    vvW2=np.zeros_like(W2)
    vvW3=np.zeros_like(W3)
    vvb1=np.zeros_like(b1)
    vvb2=np.zeros_like(b2)
    vvb3=np.zeros_like(b3)

    svW1=np.zeros_like(W1)
    svW2=np.zeros_like(W2)
    svW3=np.zeros_like(W3)
    svb1=np.zeros_like(b1)
    svb2=np.zeros_like(b2)
    svb3=np.zeros_like(b3)


    vvW1 = vW1/(1 - np.power(beta,(i+1)))
    vvb1 = vb1/(1 - np.power(beta,(i+1)))
    vvW2 = vW2/(1 - np.power(beta,(i+1)))
    vvb2 = vb2/(1 - np.power(beta,(i+1)))
    vvW3 = vW3/(1 - np.power(beta,(i+1)))
    vvb3 = vb3/(1 - np.power(beta,(i+1)))
```

```
            svW1 = rvW1/(1 - np.power(gamma,(i+1)))
            svb1 = rvb1/(1 - np.power(gamma,(i+1)))
            svW2 = rvW2/(1 - np.power(gamma,(i+1)))
            svb2 = rvb2/(1 - np.power(gamma,(i+1)))
            svW3 = rvW3/(1 - np.power(gamma,(i+1)))
            svb3 = rvb3/(1 - np.power(gamma,(i+1)))

            dW1 = vvW1 / np.sqrt(svW1 + E)
            db1 = vvb1 / np.sqrt(svb1 + E)
            dW2 = vvW2 / np.sqrt(svW2 + E)
            db2 = vvb2 / np.sqrt(svb2 + E)
            dW3 = vvW3 / np.sqrt(svW3 + E)
            db3 = vvb3 / np.sqrt(svb3 + E)

            W1 = W1 - learning_rate * dW1
            b1 = b1 - learning_rate * db1
            W2 = W2 - learning_rate * dW2
            b2 = b2 - learning_rate * db2
            W3 = W3 - learning_rate * dW3
            b3 = b3 - learning_rate * db3

    if(i%1000==0):
        print("cost "+str(i/1000),cost)
    if(abs(prev_cost - cost) < delta_cost):
        break

# Avg. error vs Epoch plot

plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Avg. error vs Epoch plot -Delta - Image data")
plt.show()

# Working on training data

Z1 = np.dot(W1, X_train.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)
```

```
Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred = np.argmax(A3,axis=0)

# Working on development data




Z1 = np.dot(W1, X_test.T) + b1
A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = np.tanh(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)

pred1= np.argmax(A3,axis=0)

# accuracy scores

count=0
i=0
while i<1200:
    if(pred[i]==ytrain[i]):
        count=count+1
    i=i+1
print("train accuracy =",count/1200)

count=0
i=0
while i<300:
    if(pred1[i]==Y_test[i]):
        count=count+1
    i=i+1
print("test accuracy =",count/300)

the_plot_conf(pred,ytrain,'Training - Delta - Image Data')

the_plot_conf(pred1,Y_test,'Development - Delta - Image Data')
```