Data Science Lab Assignment 8

Name:P Manohar Rao

Roll No: 197158

Section: CSE-A

Implement neural network from scratch using python for the following datasets and predict the values for the following datasets:

1. Boston House prices dataset: https://www.kaggle.com/fedesoriano/the-boston-houseprice-data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

def relu(z):
    a = np.maximum(0,z)
    return a

def initialize_params(layer_sizes):
    params = {}
    for i in range(1, len(layer_sizes)):
        params['W' + str(i)] = np.random.randn(layer_sizes[i], layer_sizes[i-1])*0.01
        params['B' + str(i)] = np.random.randn(layer_sizes[i],1)*0.01
    return params

def forward_propagation(X_train, params):
    layers = len(params)//2
    values = {}
    for i in range(1, layers+1):
        if i==1:
            values['Z' + str(i)] = np.dot(params['W' + str(i)], X_train) + params['B' + st
            values['A' + str(i)] = relu(values['Z' + str(i)])
        else:
            values['Z' + str(i)] = np.dot(params['W' + str(i)], values['A' + str(i-1)]) +
            if i==layers:
                values['A' + str(i)] = values['Z' + str(i)]
            else:
                values['A' + str(i)] = relu(values['Z' + str(i)])
    return values

def compute_cost(values, Y_train):
    layers = len(values)//2
    Y_pred = values['A' + str(layers)]
```

```
    cost = 1/(2*len(Y_train)) * np.sum(np.square(Y_pred - Y_train))
    return cost

def backward_propagation(params, values, X_train, Y_train):
    layers = len(params)//2
    m = len(Y_train)
    grads = {}
    for i in range(layers,0,-1):
        if i==layers:
            dA = 1/m * (values['A' + str(i)] - Y_train)
            dZ = dA
        else:
            dA = np.dot(params['W' + str(i+1)].T, dZ)
            dZ = np.multiply(dA, np.where(values['A' + str(i)]>=0, 1, 0))
        if i==1:
            grads['W' + str(i)] = 1/m * np.dot(dZ, X_train.T)
            grads['B' + str(i)] = 1/m * np.sum(dZ, axis=1, keepdims=True)
        else:
            grads['W' + str(i)] = 1/m * np.dot(dZ,values['A' + str(i-1)].T)
            grads['B' + str(i)] = 1/m * np.sum(dZ, axis=1, keepdims=True)
    return grads

def update_params(params, grads, learning_rate):
    layers = len(params)//2
    params_updated = {}
    for i in range(1,layers+1):
        params_updated['W' + str(i)] = params['W' + str(i)] - learning_rate * grads['W' +
        params_updated['B' + str(i)] = params['B' + str(i)] - learning_rate * grads['B' +
    return params_updated

def model(X_train, Y_train, layer_sizes, num_iters, learning_rate):
    params = initialize_params(layer_sizes)
    for i in range(num_iters):
        values = forward_propagation(X_train.T, params)
        cost = compute_cost(values, Y_train.T)
        grads = backward_propagation(params, values,X_train.T, Y_train.T)
        params = update_params(params, grads, learning_rate)
        print('Cost at iteration ' + str(i+1) + ' = ' + str(cost) + '\n')
    return params

def compute_accuracy(X_train, X_test, Y_train, Y_test, params):
    values_train = forward_propagation(X_train.T, params)
    values_test = forward_propagation(X_test.T, params)
    train_acc = np.sqrt(mean_squared_error(Y_train, values_train['A' + str(len(layer_sizes
    test_acc = np.sqrt(mean_squared_error(Y_test, values_test['A' + str(len(layer_sizes)-1
    return train_acc, test_acc

def predict(X, params):
    values = forward_propagation(X.T, params)
    predictions = values['A' + str(len(values)//2)].T
    return predictions
```

```
#Question 1
#predict the values for Boston House prices dataset
num_iters = 1000
learning_rate = 0.03
layer_sizes=[13,5,5,1]
df =  pd.read_csv('boston.csv')
df.head()
n=df.shape[1]
m=df.shape[0]

x_train=np.array(df.iloc[:int(0.9*m),:n-1])
y_train=np.array(df.iloc[:int(0.9*m),-1])
x_test=np.array(df.iloc[int(0.9*m):,:n-1])
y_test=np.array(df.iloc[int(0.9*m):,-1])
params = model(x_train, y_train, layer_sizes, num_iters, learning_rate)
y_predict = predict(x_test,params)
print(y_predict)
```

```
     Cost at iteration 948 = 37.025782006024556

     Cost at iteration 949 = 37.01875541442267

     Cost at iteration 950 = 37.0117268214678

     Cost at iteration 951 = 37.00469621596984

     Cost at iteration 952 = 36.99766358665111

     Cost at iteration 953 = 36.99062892214705

     Cost at iteration 954 = 36.98359221100676

     Cost at iteration 955 = 36.976553441693504

     Cost at iteration 956 = 36.96951260258542

     Cost at iteration 957 = 36.962469681976046

     Cost at iteration 958 = 36.95542466807493

     Cost at iteration 959 = 36.948377549008235

     Cost at iteration 960 = 36.941328312819294

     Cost at iteration 961 = 36.9342769474693

     Cost at iteration 962 = 36.927223440837864

     Cost at iteration 963 = 36.920167780723624

     Cost at iteration 964 = 36.91310995484488

     Cost at iteration 965 = 36.906049950840206

     Cost at iteration 966 = 36.89898775626909

     Cost at iteration 967 = 36.89192335861254
```

```
     Cost at iteration 968 = 36.88485674527368

     Cost at iteration 969 = 36.87778790357848

     Cost at iteration 970 = 36.87071682077626

     Cost at iteration 971 = 36.863643484040416

     Cost at iteration 972 = 36.85656788046903

     Cost at iteration 973 = 36.84948999708548

     Cost at iteration 974 = 36.84240982083914

     Cost at iteration 975 = 36.83532733860595

     Cost at iteration 976 = 36.82824253718911
```

```
#Question 2
#predict the values for Boston House prices dataset
num_iters = 1000
learning_rate = 0.03
layer_sizes=[7,5,5,1]
df =  pd.read_csv('seeds.csv')
df.head()
n=df.shape[1]
m=df.shape[0]

x_train=np.array(df.iloc[:int(0.9*m),:n-1])
y_train=np.array(df.iloc[:int(0.9*m),-1])
x_test=np.array(df.iloc[int(0.9*m):,:n-1])
y_test=np.array(df.iloc[int(0.9*m):,-1])
params = model(x_train, y_train, layer_sizes, num_iters, learning_rate)
y_predict = predict(x_test,params)
print(y_predict)
```

```
     Cost at iteration 751 = 1.6997005675409078

     Cost at iteration 752 = 1.6992316526262514

     Cost at iteration 753 = 1.698762895049126

     Cost at iteration 754 = 1.6982942947564843

     Cost at iteration 755 = 1.697825851695297

     Cost at iteration 756 = 1.6973575658125544

     Cost at iteration 757 = 1.696889437055262

     Cost at iteration 758 = 1.6964214653704466

     Cost at iteration 759 = 1.6959536507051514

     Cost at iteration 760 = 1.6954859930064372

     Cost at iteration 761 = 1.695018492221385
```

```
Cost at iteration 762 = 1.6945511482970923

Cost at iteration 763 = 1.6940839611806742

Cost at iteration 764 = 1.6936169308192661

Cost at iteration 765 = 1.6931500571600195

Cost at iteration 766 = 1.692683340150105

Cost at iteration 767 = 1.6922167797367103

Cost at iteration 768 = 1.6917503758670422

Cost at iteration 769 = 1.6912841285082751

Cost at iteration 770 = 1.6908180377448312

Cost at iteration 771 = 1.6903521034244544

Cost at iteration 772 = 1.689886325559263

Cost at iteration 773 = 1.6894207039737106

Cost at iteration 774 = 1.6889552386151119

Cost at iteration 775 = 1.6884899294308007

Cost at iteration 776 = 1.688024776368128

Cost at iteration 777 = 1.6875597793744637

Cost at iteration 778 = 1.6870949383971952

Cost at iteration 779 = 1.6866302533837272
```