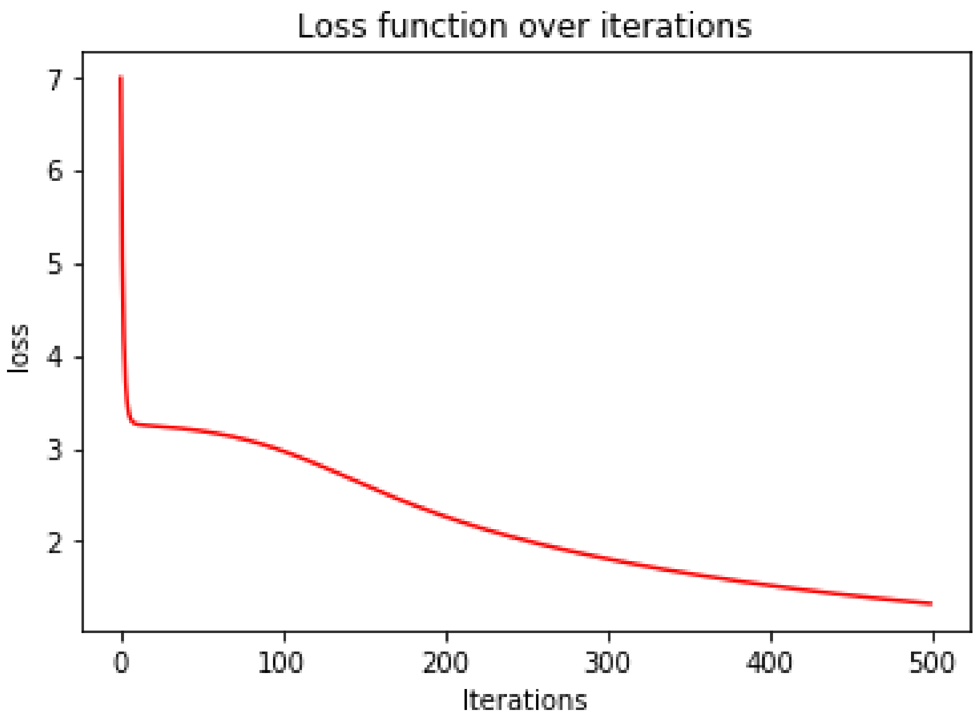
Suhail Basalama - Machine Learning - D. Lu Zhang – HW3

Implementation of Digit Recognition Neural Network

1- Plot of loss function J vs. the number of iterations

```
initial cost: 6.983818277758895
final cost: 1.3274873296506544
correct predicitons: 4291.0 out of 5000
Accuracy: % 85.82
```



2- Source Code

```
#Author: Suhail Basalama
#Date: March/20/2019

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#prepare Y
raw_Y = pd.read_csv('data\Y.csv', header=None)
raw_Y = raw_Y.values
Y =  np.zeros((5000,10))
for i in range(5000):
    Y[i,(raw_Y[i]-1)]=1

#prepare X
X = pd.read_csv('data\X.csv', header=None)
ones = np.ones([X.shape[0],1])
X = np.concatenate((ones,X),axis=1)

#initialize weights
W1 = (pd.read_csv('data/initial_W1.csv',
header=None)).values#(pd.read_csv('data\Initial_W1.csv',
header=None)).values
W2 = (pd.read_csv('data/initial_W2.csv', header=None)).values

#metadata variables
lambd = 3
eta = 0.2

#logistic (sigmoid) function
def Logistic_Function(z):
    return 1 / (1 + np.exp(-z))

#forward propagation
def Forward_Propagation(X,W1,W2):

    Z1 = X@W1.T
    H = Logistic_Function(Z1)

    ones = np.ones([H.shape[0],1])
    H = np.concatenate((ones,H),axis=1)

    Z2 = H@W2.T
    Y_ = Logistic_Function(Z2)

    return H,Y_,Z1

#loss/cost function
```

```
def Loss_Function(X,Y,W1,W2,Y_):
    sum1 = np.sum(-1*Y*np.log(Y_)-(1-Y)*np.log(1-Y_))/len(X)
    sum2 = lambd*(np.sum(W1[:,1:]**2) +
np.sum(W2[:,1:]**2))/(2*len(X))
    return sum1+sum2

#logistic gradient
def Logistic_Gradient(z):
    return Logistic_Function(z)*(1-Logistic_Function(z))

#back propagation

#Gradient of W1 for each example
def GW1Ji(X,Y,H,Y_,Z1,W2):
    B2 = (Y_ - Y)
    B1 = (B2@W2[:,1:])*Logistic_Gradient(Z1)
    GW1J = B1.T @ X
    return GW1J

#Gradient of W2 for each example
def GW2Ji(X,Y,H,Y_,Z1):
    B2 = (Y_ - Y)
    GW2J = B2.T @ H
    return GW2J

#Gradient of W1
def W1_Gradient(W1,H,Y_,Z1,W2):
    tempW = np.array(W1)
    term1 = (1/len(X))*GW1Ji(X,Y,H,Y_,Z1,W2)
    tempW[:,0] = 0
    term2 = (lambd/len(X))*W1
    return term1+term2

#Gradient of W2
def W2_Gradient(W2,H,Y_,Z1):
    tempW = np.array(W2)
    term1 = (1/len(X))*GW2Ji(X,Y,H,Y_,Z1)
    tempW[:,0] = 0
    term2 = (lambd/len(X))*W2
    return term1+term2


#Gradient Descent main algorithm
def Gradient_Descent(X,Y,W1,W2):

    k=0
    cost = []
    while(k<500):

        H,Y_,Z1 = Forward_Propagation(X,W1,W2)
        W1 = W1 - eta*W1_Gradient(W1,H,Y_,Z1,W2)
        W2 = W2 - eta*W2_Gradient(W2,H,Y_,Z1)
```

```
        cost.append(Loss_Function(X,Y,W1,W2,Y_))
        k += 1

    return W1,W2,cost,k

#initial Cost
H,Y_,Z1 = Forward_Propagation(X,W1,W2)
print("initial cost:", Loss_Function(X,Y,W1,W2,Y_))

#training the weights
W1_,W2_,cost,count = Gradient_Descent(X,Y,W1,W2)

#final Cost after training the weights
H,Y_,Z1 = Forward_Propagation(X,W1_,W2_)
print("final cost:", Loss_Function(X,Y,W1_,W2_,Y_))

#measuring the accuracy
Y_Actual = np.array([np.where(r==1)[0][0] for r in Y]).reshape(5000,1)
Y_Predicted = np.array([np.where(r==max(r))[0][0] for r in
Y_]).reshape(5000,1)


Difference = Y_Actual - Y_Predicted

ones = np.ones([5000,1])
zeros = np.zeros([5000,1])

hits = np.where(Difference==0,ones,zeros)

print("correct predicitons:",np.sum(hits),"out of 5000 \nAccuracy:
%",100*(np.sum(hits)/5000))

#output the trained weights
np.savetxt("T_W1.csv", W1_, delimiter=",")
np.savetxt("T_W2.csv", W2_, delimiter=",")

#plot the cost function over the number of iterations
fig, ax = plt.subplots()
ax.plot(np.arange(count), cost, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('loss')
ax.set_title('Loss function over iterations')
```