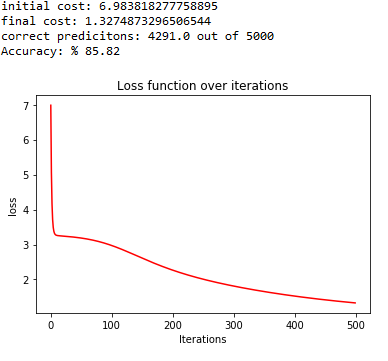
Suhail Basalama - Machine Learning - D. Lu Zhang – HW3

Implementation of Digit Recognition Neural Network

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



1. 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'**)**