Iowa State University

Department of Electrical and Computer Engineering

Deep Machine Learning: Theory and Practice

EE 526X

Homework 2

Author:
Vishal Deep

Instructor:
Dr. Zhengdao Wang

October 9, 2019

1 Problem 1

We have a softplus function,

$$f(z) = \log(1 + e^z) \tag{1}$$

Taking first derivative of the function:

$$f'(z) = \frac{1}{(1 + e^z)} e^z \tag{2}$$

multiplying and dividing by e^{-z},

$$f'(z) = \frac{e^z.e^{-z}}{(1.e^{-z} + e^z.e^{-z})} = \frac{1}{(1 + e^{-z})} \tag{3}$$

Now taking second derivative of softplus function,

$$f''(z) = \frac{(1 + e^{-z}).0 + e^{-z}}{(1 + e^{-z})^2} \tag{4}$$

$$f''(z) = \frac{e^{-z}}{(1 + e^{-z})^2} \tag{5}$$

From eq. (5), numerator is always positive because exponential is a positive number and any power to a positive is number is always a positive number. The denominator is always positive because is squared. Therefore we can say that second derivative of a softplus function is always positive for every value of z. This implies that a softplus function is convex in z.

2 Problem 2

We have a vector $z = [z1, z2, ..., zn]^T$ and $p = [p1, p2, ..., pn]^T$, where p is output after z is applied to a softmax function.

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{6}$$

The jacobian matrix will have two types of elements diagonal and off-diagonal, we will calculate both and then generalize the result for all elements.

Diagonal row elements (i=j):

$$\frac{\partial p_i}{\partial z_i} = \frac{\sum_{j=1}^{n} e^{z_j}.e^{z_i} - e^{z_i}.e^{z_i}}{(\sum_{j=1}^{n} e^{z_j})^2}$$

$$\frac{\partial p_i}{\partial z_i} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} - \left(\frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}\right)^2$$

$$\frac{\partial p_i}{\partial z_i} = p_i - p_i^2 \tag{7}$$

off-diagonal row elements (i≠j):

$$\frac{\partial p_i}{\partial z_j} = \frac{\sum_{j=1}^{n} e^{z_j}.0 - e^{z_i}.e^{z_j}}{(\sum_{j=1}^{n} e^{z_j})^2}$$

$$\frac{\partial p_i}{\partial z_j} = -\frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \times \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

$$\frac{\partial p_i}{\partial z_j} = -p_i p_j \tag{8}$$

The Jacobian matrix is given by

$$\frac{\partial p}{\partial z} = \begin{bmatrix} \frac{\partial p_0}{\partial z_0} & \frac{\partial p_0}{\partial z_1} & \cdots & \frac{\partial p_0}{\partial z_n} \\ \frac{\partial p_1}{\partial z_0} & \frac{\partial p_1}{\partial z_1} & \cdots & \frac{\partial p_1}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial z_0} & \frac{\partial p_n}{\partial z_1} & \cdots & \frac{\partial p_n}{\partial z_n} \end{bmatrix} \tag{9}$$

using eq. (7) and eq. (8), we get

$$\frac{\partial p}{\partial z} = \begin{bmatrix} p_0 - p_0^2 & -p_0 p_1 & \cdots & -p_0 p_n \\ -p_1 p_0 & p_1 - p_1^2 & \cdots & -p_1 p_n \\ \vdots & \vdots & \ddots & \vdots \\ -p_n p_0 & -p_n p_1 & \cdots & p_n - p_n^2 \end{bmatrix} \tag{10}$$

3 Problem 3

We have $y = [y1, y2, ..., yn]^T$, a correct probability vector. And the cross entropy is given by

$$J(z) = -\sum_{i=1}^{n} y_i \log p_i \tag{11}$$

eq. (11) is a dot product or element-wise product of y_i and $\log p_i$. Taking derivative of eq. (11) w.r.t. p_i,

$$\frac{\partial J}{\partial p_i} = -\frac{\partial}{\partial p_i} y_i \log p_i = -y_i \frac{\partial \log p_i}{\partial p_i}$$

$$\frac{\partial J}{\partial p_i} = -\frac{y_i}{p_i} \tag{12}$$

Also from eqs. (7) and (8) we know $\frac{\partial p_i}{\partial z_j}$,

$$\frac{\partial p_i}{\partial z_j} = \begin{cases} p_i - p_i^2 & i = j \\ -p_i p_j & i \neq j \end{cases} \tag{13}$$

from eqs. (12) and (13), we can find $\frac{\partial J}{\partial z_i}$

$$\begin{aligned}
\frac{\partial J}{\partial z_i} &= \sum_{j=1}^{n} \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_i} \\
&= \frac{\partial J}{\partial p_i}\frac{\partial p_i}{\partial z_i} + \sum_{i \neq j} \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_i} \\
&= -\frac{y_i}{p_i}(p_i - p_i^2) + \sum_{i \neq j}(-\frac{y_j}{p_j})(-p_i p_j) \\
&= -y_i(1 - p_i) + \sum_{i \neq j} y_j p_i \\
&= -y_i + y_i p_i + p_i \sum_j y_j \\
&= p_i \left(y_i + \sum_j y_j \right) - y_i \\
&= p_i - y_i
\end{aligned}$$

Because y is one hot encoded and $\sum_j y_j = 1$

$$\frac{\partial J}{\partial z} = \begin{bmatrix} p_0 - y_0 \\ p_1 - y_1 \\ \vdots \\ p_n - y_n \end{bmatrix} \tag{14}$$

4 Problem 4

We have

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{15}$$

$$\boldsymbol{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{16}$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{17}$$

1st Iteration:

$$\begin{aligned}
z^{[0]} &= W^{[0]}x + b^{[0]}.1 \\
&= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}
\end{aligned}$$

$$z^{[0]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{18}$$

We are using a softmax function, therefore p matrix is given by

$$p^{[0]} = \frac{e^{z_i}}{\sum_{i=1}^{4} e^{z_i}}$$

$$p^{[0]} = \begin{bmatrix} \frac{e^0}{e^0+e^0} & \frac{e^0}{e^0+e^0} \\ \frac{e^0}{e^0+e^0} & \frac{e^0}{e^0+e^0} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \tag{19}$$

and cost J is given by cross entropy function,

$$\begin{aligned} J^{[0]} &= -\sum_{i=1}^{2} y_i \log p_i \\ &= -\sum_{i=1}^{2} y_i \log p_i \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} \log 0.5 & \log 0.5 \\ \log 0.5 & \log 0.5 \end{bmatrix} \end{aligned}$$

$$J^{[0]} = -\sum_{i=1}^{2} \begin{bmatrix} \log 0.5 & 0 \\ 0 & \log 0.5 \end{bmatrix} \approx 0.6931 \tag{20}$$

Now doing back propagation, from eq. (14)

$$\boldsymbol{dz}^{[0]} = \frac{1}{m}(\boldsymbol{p} - \boldsymbol{y}) = \frac{1}{2} \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} \tag{21}$$

$$\boldsymbol{dW}^{[0]} = \boldsymbol{dz^{[0]} X^T} = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} \tag{22}$$

$$\boldsymbol{db}^{[0]} = \boldsymbol{dz^{[0]}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{23}$$

Now update weights and bias matrices with learning rate = 1,

$$\boldsymbol{W}^{[1]} = \boldsymbol{W^{[0]}} - 1.\boldsymbol{dW^{[0]}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} = \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix} \tag{24}$$

$$\boldsymbol{b}^{[1]} = \boldsymbol{b^{[0]}} - 1.\boldsymbol{db^{[0]}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{25}$$

2nd Iteration:

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]}.1 \\ &= \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \end{aligned}$$

$$z^{[1]} = \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix} \tag{26}$$

Softmax function with $z^{[1]}$,

$$p^{[1]} = \frac{e^{z_i}}{\sum_{i=1}^{2} e^{z_i}} = \begin{bmatrix} \frac{e^{0.25}}{e^{0.25}+e^{-0.25}} & \frac{e^{-0.25}}{e^{0.25}+e^{-0.25}} \\ \frac{e^{-0.25}}{e^{0.25}+e^{-0.25}} & \frac{e^{0.25}}{e^{0.25}+e^{-0.25}} \end{bmatrix} = \begin{bmatrix} 0.62 & 0.38 \\ 0.38 & 0.62 \end{bmatrix} \tag{27}$$

$$\begin{aligned} J^{[1]} &= -\sum_{i=1}^{2} y_i \log p_i \\ &= -\sum_{i=1}^{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \log 0.62 & \log 0.38 \\ \log 0.38 & \log 0.62 \end{bmatrix} \end{aligned}$$

$$J^{[1]} = -\sum_{i=1}^{2} \begin{bmatrix} \log 0.62 & 0 \\ 0 & \log 0.62 \end{bmatrix} \approx 0.4741 \tag{28}$$

Now doing back propagation, from eq. (14)

$$\boldsymbol{dz}^{[1]} = \frac{1}{m}(\boldsymbol{p} - \boldsymbol{y}) = \frac{1}{2}\begin{bmatrix} 0.62 & 0.38 \\ 0.38 & 0.62 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.19 & 0.19 \\ 0.19 & -0.19 \end{bmatrix} \tag{29}$$

$$\boldsymbol{dW}^{[1]} = \boldsymbol{dz^{[1]}X^{T}} = \begin{bmatrix} -0.19 & 0.19 \\ 0.19 & -0.19 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.19 & 0.19 \\ 0.19 & -0.19 \end{bmatrix} \tag{30}$$

$$\boldsymbol{db}^{[1]} = \boldsymbol{dz^{[1]}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.19 & 0.19 \\ 0.19 & -0.19 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{31}$$

Now update weights and bias matrices with learning rate = 1,

$$\boldsymbol{W}^{[2]} = \boldsymbol{W^{[1]}} - 1.\boldsymbol{dW^{[1]}} = \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix} - \begin{bmatrix} -0.19 & 0.19 \\ 0.19 & -0.19 \end{bmatrix} = \begin{bmatrix} 0.44 & -0.44 \\ -0.44 & 0.44 \end{bmatrix} \tag{32}$$

$$\boldsymbol{b}^{[2]} = \boldsymbol{b^{[1]}} - 1.\boldsymbol{db^{[1]}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{33}$$

5 Problem 5

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 4 11:03:55 2019
Problem: 5, Homework 2
@author: vishal Deep
"""
import time
from DNN import *
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.utils import to_categorical
```

```
def trainAndTestSplit(spamEmails, notSpamEmails):
    # convert data into numpy arrays
    spamEmailsArr =spamEmails.to_numpy()
    notSpamEmailsArr =notSpamEmails.to_numpy()
    # Calculate number of total spam and not spam emails
    totalSpam =spamEmails.shape[0]
    totalNotSpam =notSpamEmails.shape[0]
    # Calculate number of spam and not spam train and test
    numSpamTrain =int(np.floor((2/3) *totalSpam))
    numNotSpamTrain =int(np.floor((2/3) *totalNotSpam))
    numSpamTest =totalSpam -numSpamTrain
    numNotSpamTest =totalNotSpam -numNotSpamTrain
    # Seperate spam and not spam train and test arrays
    trainSpam =spamEmailsArr[0: numSpamTrain, :]
    trainNotSpam =notSpamEmailsArr[0: numNotSpamTrain, :]
    testSpam =spamEmailsArr[numSpamTrain:numSpamTrain+numSpamTest, :]
    testNotSpam =notSpamEmailsArr[numNotSpamTrain:numNotSpamTrain+numNotSpamTest, :]
    # Combine test and train data
    trainData =np.vstack((trainSpam, trainNotSpam))
    testData =np.vstack((testSpam, testNotSpam))
    return trainData, testData

def generateLabel(y_pred):
    y_pred[y_pred <0] =0
    y_pred[y_pred >0] =1
    return np.transpose(y_pred)

def calcAccuracy(p, y):
    y_pred =generateLabel(p)
    perfArr =np.equal(y_pred, y)
    accuracy =(np.sum(perfArr)/np.size(perfArr)) *100
    return accuracy

# read data from the file
spamBaseData =pd.read_csv("spambase/spambase.data", header=None)
# Extract label y
y =spamBaseData.iloc[:, -1]
# filter spam
spamEmails =spamBaseData.loc[y ==1]
# filtet not spam
notSpamEmails =spamBaseData.loc[y ==0]
# split the train and test data
[trainData, testData] =trainAndTestSplit(spamEmails, notSpamEmails)
# seperate X and y of training and test data
Xtrain =trainData[:, 0:-1]
ytrain =trainData[:, -1]
Xtest =testData[:, 0:-1]
ytest =testData[:, -1]

X =np.transpose(Xtrain)

y =ytrain.reshape(1, 3066)

D =57 # Input dimension
Odim =1 # number of outputs
layers=[ (100, ReLU), (40, ReLU), (Odim, Linear) ]
```

```python
# initialize Neural Network with D inputs and layers
nn =NeuralNetwork(D, layers)
# select crossentropy as objective function
CE =ObjectiveFunction('logistic')
eta =[0.01, 0.1, 0.5, 1]

for eta in eta:
    # set random weights with maximum size of 0.1
    nn.setRandomWeights(0.1)
    # Print value of eta being used
    print(f"Learning rate: {eta}\n")
    # Record start time
    startTime =np.round(time.time(), decimals=4)
    for i in range(10000):
        p =nn.doForward(X)
        J =CE.doForward(p, y)
        dz =CE.doBackward(y)
        dx =nn.doBackward(dz)
        nn.updateWeights(eta)
        if (i%2000==0):
            accuracy =np.round(calcAccuracy(p, y), decimals=4)
            J_rounded =np.round(J, decimals=4)
            print(f'Iterations: {i}, J={J_rounded}, Training Accuracy: {accuracy} % \n')

    # Calculate time taken to train
    stopTime =np.round(time.time(), decimals=4)
    totalTime =(np.round(((stopTime -startTime)/60), decimals=4))
    print(f'Training time: {totalTime} minutes \n')

    # Test dataset Prediction accuracy
    X_test =np.transpose(Xtest)
    p =nn.doForward(X_test)
    accuracy =np.round(calcAccuracy(p, ytest), decimals=4)
    print(f'Test dataset accuracy: {accuracy} % \n')
```

```
Learning rate: 0.01

Iterations: 0, J=113.6054, Training Accuracy: 39.3999 %

Iterations: 2000, J=0.557, Training Accuracy: 60.6001 %

Iterations: 4000, J=0.5551, Training Accuracy: 60.6001 %

Iterations: 6000, J=0.5582, Training Accuracy: 60.6001 %

Iterations: 8000, J=0.5588, Training Accuracy: 60.6001 %

Training time: 1.6898 minutes

Test dataset accuracy: 60.5863 %

Learning rate: 0.1

Iterations: 0, J=121.2302, Training Accuracy: 39.3999 %

Iterations: 2000, J=0.5596, Training Accuracy: 60.6001 %

Iterations: 4000, J=0.5596, Training Accuracy: 60.6001 %

Iterations: 6000, J=0.5596, Training Accuracy: 60.5863 %

Iterations: 8000, J=0.5596, Training Accuracy: 60.5863 %

Training time: 1.6917 minutes

Test dataset accuracy: 60.5863 %

Learning rate: 0.5
```

```
Iterations: 0, J=104.0481, Training Accuracy: 39.3999 %

Iterations: 2000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 4000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 6000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 8000, J=0.5643, Training Accuracy: 60.6001 %

Training time: 1.6989 minutes

Test dataset accuracy: 60.5863 %

Learning rate: 1

Iterations: 0, J=113.7425, Training Accuracy: 39.3999 %

Iterations: 2000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 4000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 6000, J=0.5643, Training Accuracy: 60.6001 %

Iterations: 8000, J=0.5643, Training Accuracy: 60.6001 %

Training time: 1.6626 minutes

Test dataset accuracy: 60.5863 %
```

6 Problem 6

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 4 11:03:55 2019
Problem: 6, Homework 2
@author: vishal Deep
"""
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from DNN import *
from keras.utils import to_categorical
from sklearn import preprocessing
import time

def calcAccuracy(logp, y):
    yhat =logp.argmax(axis=0)
    perfArr =np.equal(yhat, y)
    accuracy =(np.sum(perfArr)/np.size(perfArr)) *100
    return accuracy

# create a list containing mini-batches
def createMiniBatches(X, y, batch_size):
    mini_batches =[]
    X =np.transpose(X)
    y =y.reshape((-1, 1))
    data =np.hstack((X,y))
    np.random.shuffle(data)
    n_minibatches =data.shape[0] //batch_size
    for i in range(n_minibatches):
        mini_batch =data[i *batch_size:(i +1)*batch_size, :]
        X_mini =np.transpose(mini_batch[:, :-1])
```

```
        Y_mini =mini_batch[:, -1]
        mini_batches.append((X_mini, Y_mini))
    return mini_batches

testFlag =False
# Get data from tensor flow
mnist =tf.keras.datasets.mnist
# Split data between train and test data
(x, y),(x_test, y_test) =mnist.load_data()
# flatten the image data for all 60000 images
X =x.transpose((1, 2, 0)).reshape(784, -1)
# Normalize data
X_norm =np.divide(X, 255)

if testFlag:
    X_norm =X_norm[:, 0:10000]
    y =y[0:10000]

# Create mini batches of 100
mini_batches =createMiniBatches(X_norm, y, 100)

D =784 # Input dimension
Odim =10 # number of outputs

layers_a =[(Odim, Linear)]
layers_b =[(50, ReLU), (50, ReLU), (Odim, Linear)]
layers_c =[(100, ReLU), (50, ReLU), (Odim, Linear)]

# initialize Neural Network with D inputs and layers
nn_a =NeuralNetwork(D, layers_a)
nn_b =NeuralNetwork(D, layers_b)
nn_c =NeuralNetwork(D, layers_c)

# set random weights with maximum size of 0.1
nn_a.setRandomWeights(0.1)
nn_b.setRandomWeights(0.1)
nn_c.setRandomWeights(0.1)

# select crossentropy as objective function
CE_a =ObjectiveFunction('crossEntropyLogit')
CE_b =ObjectiveFunction('crossEntropyLogit')
CE_c =ObjectiveFunction('crossEntropyLogit')

numOfIterations =10000
batchCount =1
eta =1e-2
print(f"Learning rate: {eta}\n")

for batch in mini_batches:
    X =batch[0]
    y =batch[1]
    # One hot encoding
    y =np.transpose(to_categorical(y))

    # Train network a with training data
    startTime =np.round(time.time(), decimals=4)
```

```
    for i in range(numOfIterations):
        logp_a =nn_a.doForward(X)
        J_a =CE_a.doForward(logp_a, y)
        dz_a =CE_a.doBackward(y)
        dx_a =nn_a.doBackward(dz_a)
        nn_a.updateWeights(eta)

    stopTime =np.round(time.time(), decimals=4)
    totalTime_a =np.round(((stopTime -startTime)/60), decimals=4)
# print(f'Training time NN_a: {totalTime} minutes \n')

    # Train network b with training data
    startTime =np.round(time.time(), decimals=4)
    for i in range(numOfIterations):
        logp_b =nn_b.doForward(X)
        J_b =CE_b.doForward(logp_b, y)
        dz_b =CE_b.doBackward(y)
        dx_b =nn_b.doBackward(dz_b)
        nn_b.updateWeights(eta)

    stopTime =np.round(time.time(), decimals=4)
    totalTime_b =np.round(((stopTime -startTime)/60), decimals=4)
# print(f'Training time NN_b: {totalTime} minutes \n')

    # Train network c with training data
    startTime =np.round(time.time(), decimals=4)
    for i in range(numOfIterations):
        logp_c =nn_c.doForward(X)
        J_c =CE_c.doForward(logp_c, y)
        dz_c =CE_c.doBackward(y)
        dx_c =nn_c.doBackward(dz_c)
        nn_c.updateWeights(eta)

    stopTime =np.round(time.time(), decimals=4)
    totalTime_c =np.round(((stopTime -startTime)/60), decimals=4)
# print(f'Training time NN_c: {totalTime} minutes \n')

    if (batchCount%20 ==0):
        print(f"Batch Completed: {batchCount}\n")
        print(f'Training time NN_a: {totalTime_a} minutes \n')
        print(f'Training time NN_b: {totalTime_a} minutes \n')
        print(f'Training time NN_c: {totalTime_c} minutes \n')
        accuracy_a =np.round(calcAccuracy(logp_a, y), decimals=4)
        J_a =np.round(J_a, decimals=4)
        print(f'Iterations: {i}, J={J_a}, Training Accuracy: {accuracy_a} % \n')
        accuracy_b =np.round(calcAccuracy(logp_b, y), decimals=4)
        J_b =np.round(J_b, decimals=4)
        print(f'Iterations: {i}, J={J_b}, Training Accuracy: {accuracy_b} % \n')
        accuracy_c =np.round(calcAccuracy(logp_c, y), decimals=4)
        J_c =np.round(J_c, decimals=4)
        print(f'Iterations: {i}, J={J_c}, Training Accuracy: {accuracy_c} % \n')

    batchCount +=1

# Test Data Prediction accuracy
X_test =np.transpose(x_test)
```

```python
X_test =X_test.transpose((1, 2, 0)).reshape(784, -1)
# Normalize data
X_test_norm =np.divide(X_test, 255)

p_a =nn_a.doForward(X_test_norm)
p_b =nn_b.doForward(X_test_norm)
p_c =nn_c.doForward(X_test_norm)

accuracy_a =calcAccuracy(p_a, y_test)
accuracy_b =calcAccuracy(p_b, y_test)
accuracy_c =calcAccuracy(p_c, y_test)
print(f'Test Accuracy NN_a: {accuracy_a}%')
print(f'Test Accuracy NN_b: {accuracy_b}%')
print(f'Test Accuracy NN_c: {accuracy_c}%')
```

```
Learning rate: 0.01

Batch Completed: 20

Training time NN_a: 0.0902 minutes

Training time NN_b: 0.0902 minutes

Training time NN_c: 0.3624 minutes

Iterations: 9999, J=0.0069, Training Accuracy: 8.5 %

Iterations: 9999, J=0.0002, Training Accuracy: 8.5 %

Iterations: 9999, J=0.0003, Training Accuracy: 8.5 %

Batch Completed: 40

Training time NN_a: 0.0816 minutes

Training time NN_b: 0.0816 minutes

Training time NN_c: 0.3494 minutes

Iterations: 9999, J=0.007, Training Accuracy: 7.3 %

Iterations: 9999, J=0.0002, Training Accuracy: 7.3 %

Iterations: 9999, J=0.0002, Training Accuracy: 7.3 %

Batch Completed: 60

Training time NN_a: 0.087 minutes

Training time NN_b: 0.087 minutes

Training time NN_c: 0.3454 minutes

Iterations: 9999, J=0.0065, Training Accuracy: 11.9 %

Iterations: 9999, J=0.0001, Training Accuracy: 11.9 %

Iterations: 9999, J=0.0002, Training Accuracy: 11.9 %

Batch Completed: 80

Training time NN_a: 0.1017 minutes

Training time NN_b: 0.1017 minutes

Training time NN_c: 0.3458 minutes

Iterations: 9999, J=0.0054, Training Accuracy: 14.9 %

Iterations: 9999, J=0.0001, Training Accuracy: 14.9 %

Iterations: 9999, J=0.0001, Training Accuracy: 14.9 %

Batch Completed: 100

Training time NN_a: 0.0844 minutes

Training time NN_b: 0.0844 minutes
```

```
Training time NN_c: 0.3437 minutes

Iterations: 9999, J=0.0065, Training Accuracy: 7.2 %

Iterations: 9999, J=0.0001, Training Accuracy: 7.2 %

Iterations: 9999, J=0.0001, Training Accuracy: 7.2 %

Test Accuracy NN_a: 10.05%
Test Accuracy NN_b: 10.09%
Test Accuracy NN_c: 9.86%
```