

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,

$$f(z) = \log(1 + e^z) \quad (1)$$

Taking first derivative of the function:

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

multiplying and dividing by e^{-z} ,

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

Now taking second derivative of softplus function,

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

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

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 $z = [z_1, z_2, \dots, z_n]^T$ and $p = [p_1, p_2, \dots, p_n]^T$, where p is output after z is applied to a softmax function.

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

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 ($i=j$):

$$\begin{aligned} \frac{\partial p_i}{\partial z_i} &= \frac{\sum_{j=1}^n e^{z_j} \cdot e^{z_i} - e^{z_i} \cdot e^{z_i}}{(\sum_{j=1}^n e^{z_j})^2} \\ \frac{\partial p_i}{\partial z_i} &= \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} - \left(\frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \right)^2 \\ \frac{\partial p_i}{\partial z_i} &= p_i - p_i^2 \end{aligned} \quad (7)$$

off-diagonal row elements ($i \neq j$):

$$\begin{aligned}\frac{\partial p_i}{\partial z_j} &= \frac{\sum_{j=1}^n e^{z_j} \cdot 0 - e^{z_i} \cdot e^{z_j}}{(\sum_{j=1}^n e^{z_j})^2} \\ \frac{\partial p_i}{\partial z_j} &= -\frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \times \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \\ \frac{\partial p_i}{\partial z_j} &= -p_i p_j\end{aligned}\tag{8}$$

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 $y = [y_1, y_2, \dots, y_n]^T$, 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 ,

$$\begin{aligned}\frac{\partial J}{\partial p_i} &= -\frac{\partial}{\partial p_i} y_i \log p_i = -y_i \frac{\partial \log p_i}{\partial p_i} \\ \frac{\partial J}{\partial p_i} &= -\frac{y_i}{p_i}\end{aligned}\tag{12}$$

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 $\frac{\partial J}{\partial z_i}$

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

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

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

4 Problem 4

We have

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (15)$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (16)$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (17)$$

1st Iteration:

$$\begin{aligned}
z^{[0]} &= W^{[0]}x + b^{[0]}.1 \\
&= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \\
z^{[0]} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}
\end{aligned} \quad (18)$$

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

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

$$p^{[0]} = \begin{bmatrix} \frac{e^0}{e^0+e^0+e^0+e^0} & \frac{e^0}{e^0+e^0+e^0+e^0} \\ \frac{e^0}{e^0+e^0+e^0+e^0} & \frac{e^0}{e^0+e^0+e^0+e^0} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} \quad (19)$$

and cost J is given by cross entropy function,

$$J^{[0]} = - \sum_{i=1}^4 y_i \log p_i = -\mathbf{y}(\log \mathbf{p})^T$$

$$J^{[0]} = - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \log 0.25 & \log 0.25 \\ \log 0.25 & \log 0.25 \end{bmatrix} = - \begin{bmatrix} \log 0.25 & \log 0.25 \\ \log 0.25 & \log 0.25 \end{bmatrix} \quad (20)$$

Now doing back propagation, from eq. (14)

$$dz^{[0]} = \mathbf{p} - \mathbf{y} = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.75 & 0.25 \\ 0.25 & -0.75 \end{bmatrix} \quad (21)$$

$$d\mathbf{W}^{[0]} = dz^{[0]} \mathbf{X}^T = \begin{bmatrix} -0.75 & 0.25 \\ 0.25 & -0.75 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.75 & 0.25 \\ 0.25 & -0.75 \end{bmatrix} \quad (22)$$

$$db^{[0]} = dz^{[0]} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.75 & 0.25 \\ 0.25 & -0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} \quad (23)$$

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

$$\mathbf{W}^{[1]} = \mathbf{W}^{[0]} - 1.d\mathbf{W}^{[0]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} -0.75 & 0.25 \\ 0.25 & -0.75 \end{bmatrix} = \begin{bmatrix} 0.75 & -0.25 \\ -0.25 & 0.75 \end{bmatrix} \quad (24)$$

$$\mathbf{b}^{[1]} = \mathbf{b}^{[0]} - 1.db^{[0]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad (25)$$

2nd Iteration:

$$z^{[1]} = \mathbf{W}^{[1]}x + \mathbf{b}^{[1]}.1$$

$$= \begin{bmatrix} 0.75 & -0.25 \\ -0.25 & 0.75 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$z^{[1]} = \begin{bmatrix} 0.25 & -0.75 \\ -0.75 & 0.25 \end{bmatrix} \quad (26)$$

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

$$p^{[1]} = \frac{e^{z_i}}{\sum_{i=1}^4 e^{z_i}} = \begin{bmatrix} 0.365 & 0.134 \\ 0.134 & 0.365 \end{bmatrix} \quad (27)$$

$$J^{[1]} = - \sum_{i=1}^4 y_i \log p_i = -\mathbf{y}(\log \mathbf{p})^T$$

$$J^{[1]} = - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \log 0.365 & \log 0.134 \\ \log 0.134 & \log 0.365 \end{bmatrix} = - \begin{bmatrix} \log 0.365 & \log 0.134 \\ \log 0.134 & \log 0.365 \end{bmatrix} \quad (28)$$

Now doing back propagation, from eq. (14)

$$d\mathbf{z}^{[1]} = \mathbf{p} - \mathbf{y} = \begin{bmatrix} 0.365 & 0.134 \\ 0.134 & 0.365 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.634 & 0.134 \\ 0.134 & -0.634 \end{bmatrix} \quad (29)$$

$$d\mathbf{W}^{[1]} = d\mathbf{z}^{[1]} \mathbf{X}^T = \begin{bmatrix} -0.634 & 0.134 \\ 0.134 & -0.634 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.634 & 0.134 \\ 0.134 & -0.634 \end{bmatrix} \quad (30)$$

$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.634 & 0.134 \\ 0.134 & -0.634 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} \quad (31)$$

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

$$\mathbf{W}^{[2]} = \mathbf{W}^{[1]} - 1.d\mathbf{W}^{[1]} = \begin{bmatrix} 0.75 & -0.25 \\ -0.25 & 0.75 \end{bmatrix} - \begin{bmatrix} -0.634 & 0.134 \\ 0.134 & -0.634 \end{bmatrix} = \begin{bmatrix} 1.384 & -0.384 \\ -0.384 & 1.384 \end{bmatrix} \quad (32)$$

$$\mathbf{b}^{[2]} = \mathbf{b}^{[1]} - 1.d\mathbf{b}^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (33)$$

5 Problem 5

```
# -*- 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]
# filter 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) ]
# 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 %