Statistical learning: Second assignment

Ali Zamani(96123035)

November 11, 2018

0.1 Import packages

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  5 17:55:23 2018

@author: ali
"""
#Start import packages
import tensorflow as tf
import numpy as np
from sklearn.metrics import accuracy_score as acc
import matplotlib.pyplot as plt
#End import packages
```

0.2 Load Data Set:

In this section we define function loadDataSet for loading data.Inputs of this function are:
1)directory: address of data set file.
2)trainNum: number of training data.
3)validNum: number of validation data.

```
#%%
#Start Function to load notMNist dataset
def loadDataSet(directory,trainNum , validNum):
    with np.load(directory) as data :
        Data, Target=data["images"], data["labels"]
        posClass = 2
        negClass = 9
        dataIndx = (Target==posClass) + (Target==negClass)
        Data = Data[dataIndx]/255.
        Target = Target[dataIndx].reshape(-1, 1)
        Target[Target==posClass] = 1
        Target[Target==negClass] = 0
        np.random.seed(521)
        randIndx = np.arange(len(Data))
        np.random.shuffle(randIndx)
        Data, Target = Data[randIndx], Target[randIndx]
        Data = Data.reshape([-1, 784])
        Target= Target.reshape([-1, 1])
        trainData, trainTarget = Data[:trainNum], Target[:trainNum]
        validData, validTarget = Data[trainNum:validNum+trainNum],
    Target[trainNum:validNum+trainNum]
        testData, testTarget = Data[validNum+trainNum:], Target[
    validNum+trainNum:]
        return trainData, trainTarget, validData, validTarget,
    testData, testTarget
#End Function to load notMNist dataset
```

0.3 Define variables , constants and placeholders of the logistic regression model

First we set epochNum=5000 , batchSize=500 , trainNum=3500 ,validNum=100 and learningRate=1e-6 for maximum likelihood and 1e-6 for binary cross-entropy and regularized binary cross-entropy.
Deta set was loded by calling the loadDataSet function.
we must define to placeholders, one of them for input(named as x) and another one for output(named as y).
Two variables w and b are also defined.

```
#%%
#Start main program
epochNum = 5000
batchSize = 500
trainNum=3500
validNum=100
learningRate = 1e-4
print ("dataset is loading....")
trainData, trainTarget, validData, validTarget, testData,
    testTarget=loadDataSet("DataSet/notMNIST.npz",trainNum ,
    validNum)
print ("dataset was loaded!!!")
# Define placeholder x for input
x = tf.placeholder(dtype=tf.float64, shape=[None, 784], name="x")
# Define placeholder y for output
y = tf.placeholder(dtype=tf.float64, shape=[None, 1], name="y")
# Define variable w and fill it with random number
w = tf.Variable(tf.random_normal(shape=[784, 1], stddev=0.1, dtype=
    tf.float64), name="weights", dtype=tf.float64)
# Define variable b and fill it with zero
b = tf.Variable(tf.zeros(1, dtype=tf.float64), name="bias", dtype=
    tf.float64)
```

0.4 Define logistic regression model, loss functions and optimizers

ypredicted was defined for modeling.
Three loss functions lossML, lossBCE, lossRBCE were also defined for maximum likelihood, binary cross-entropy and regularized binary cross-entropy.
Gradient Descent Optimizer was used for optimizing.

```
# Define logistic Regression
logit = tf.matmul(x, w) + b
yPredicted = 1.0 / (1.0 + tf.exp(-logit))
# Define maximum likelihood(ML) loss function
lossML = -1 * tf.reduce_sum(y * tf.log(yPredicted) + (1 - y) * tf.
    log(1 - yPredicted))
# Define binary cross-entropy(BCE) loss function
```

```
lossBCE = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
    labels=y, logits=logit))
# Define Regularized Binary Cross-Entropy(RBCE) loss function
lossRBCE =  tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
    labels=y, logits=logit)) +\
            tf.constant(0.5 *learningRate, dtype=tf.float64) * tf.
    pow(tf.linalg.norm(w), 2)
# Define optimizer: GradientDescent
optimizer = tf.train.GradientDescentOptimizer(learning_rate=
    learningRate).minimize(lossBCE)
```

0.5 Run Session

We run the session for three different loss functions ans report the results.

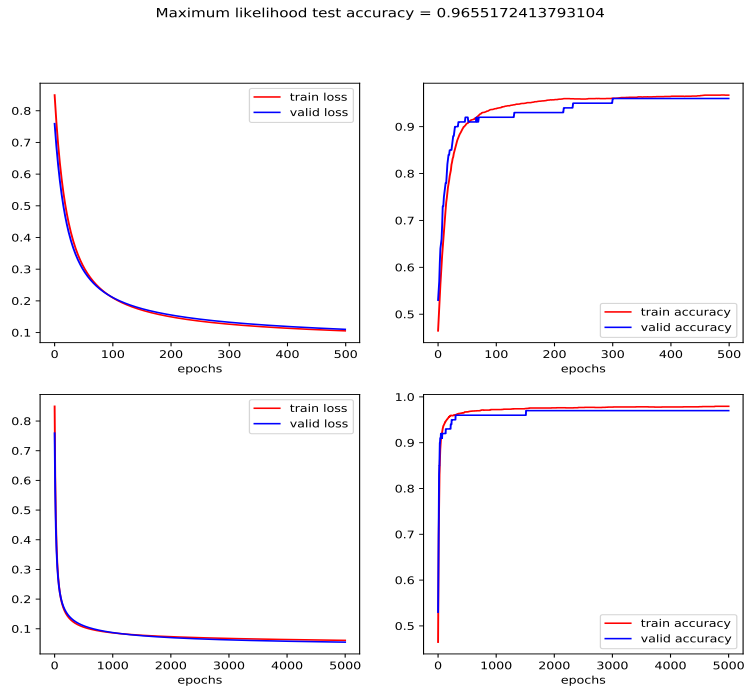
```
#%%
print("Parameters were initialized, Session is runing ...")
trainErrorList = []
validErrorList = []
trainAccList = []
validAccList = []
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(epochNum):
        trainLoss = 0
        for idx in range(trainNum//batchSize):
            InputList = {x: trainData[idx*batchSize:(idx+1)*
    batchSize],
                         y: trainTarget[idx*batchSize:(idx+1)*
    batchSize]}
            _, trainL = sess.run([optimizer, lossBCE], feed_dict=
    InputList)
            trainLoss += trainL
        trainAccList.append(acc(trainTarget, np.round(sess.run(
    yPredicted, feed_dict={x:trainData}))))
        validAccList.append(acc(validTarget, np.round(sess.run(
    yPredicted, feed_dict={x:validData}))))
        trainErrorList.append(trainLoss/trainNum)#number should be
    used as constant
        validErrorList.append(sess.run(lossML, feed_dict={x:
    validData,y:validTarget})/100)

        print("train accuracy =".format(i), trainAccList[i])
        w_value, b_value = sess.run([w, b])
        testAcc = acc(testTarget, np.round(sess.run(yPredicted,
    feed_dict={x: testData})))
        print("accuracy =", testAcc)
#titles=["Maximum likelihood", "Cross Entropy", "Regularized Cross
    Entropy"]
```

0.6 Results

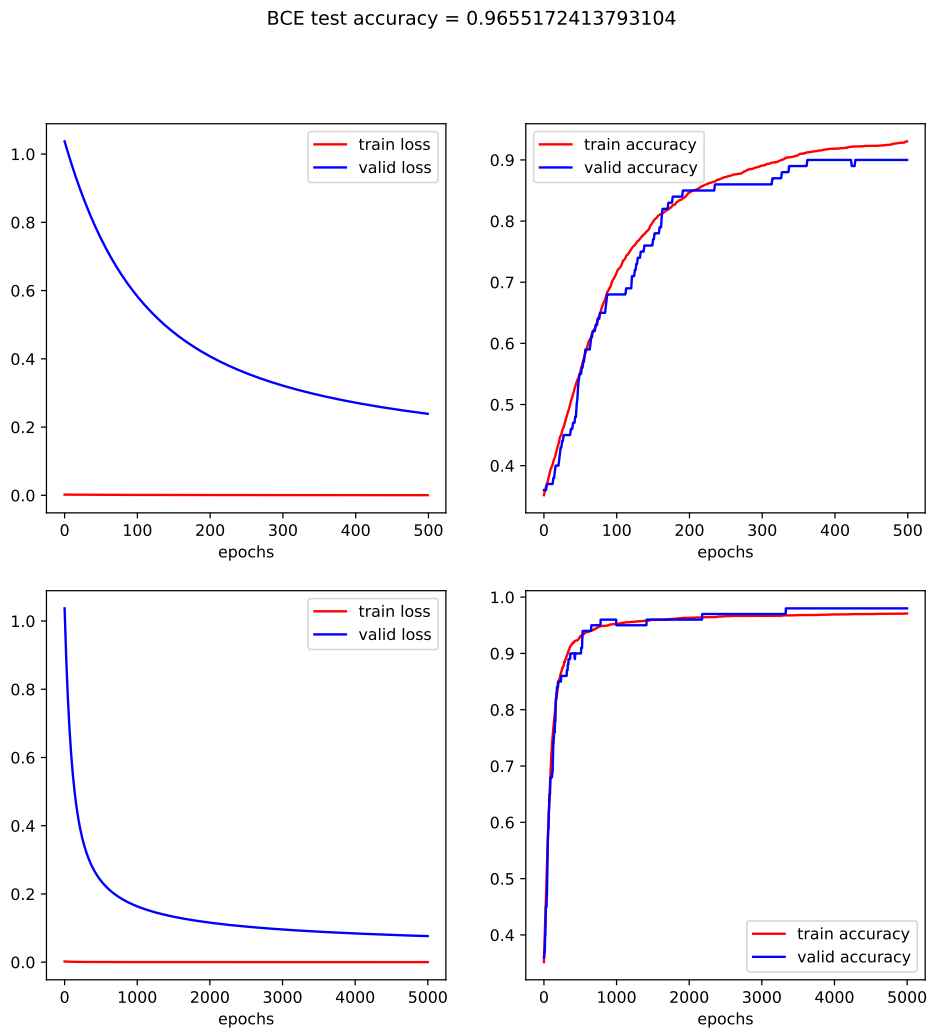
```
#%%
```

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle("BCE "+"test accuracy = " + str(testAcc))
for a in ax.reshape(-1,1):
    a[0].set_xlabel("epochs")
ax[0][0].plot(trainErrorList[:500], color='red', label='train loss')
ax[0][0].plot(validErrorList[:500], color='blue', label='valid loss')
ax[0][0].legend()
ax[1][0].plot(trainErrorList, color='red', label='train loss')
ax[1][0].plot(validErrorList, color='blue', label='valid loss')
ax[1][0].legend()
ax[0][1].plot(trainAccList[:500], color='red', label='train accuracy')
ax[0][1].plot(validAccList[:500], color='blue', label='valid accuracy')
ax[0][1].legend()
ax[1][1].plot(trainAccList, color='red', label='train accuracy')
ax[1][1].plot(validAccList, color='blue', label='valid accuracy')
ax[1][1].legend()
plt.savefig("BCE"+".pdf")
#End main program
```

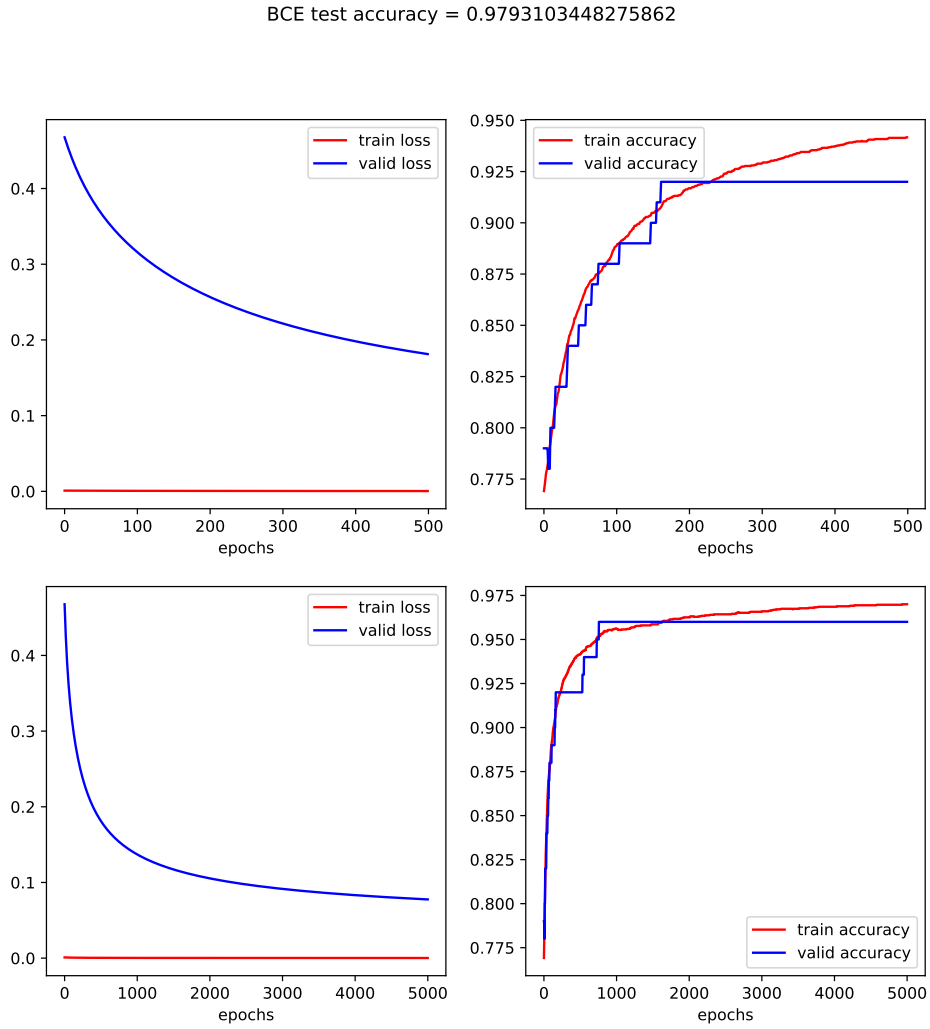
maximum likelihood:



binary cross-entropy:



regularized binary cross-entropy:



Source Code