Programming Assignment 5: Neural Networks

Member: Luyao Wang, Di Jin, Yingqi Lin

Individual contributions:

(1) Luyao Wang:
Implementation of Neural Networks
The first part of the report

(2) Di Jin:
Implementation of Neural Networks
The second part of the report

(3) Yingqi Lin:
Implementation of Neural Networks
The third part of the report

Part 1: Implementation

(1)Implementation of Neural Networks

Output:

Based on our updated weights, we print our predictions of test data and the final accuracy.

```
prediction_list =  [1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
accuracy =  0.8433734939759037
```

Data Structure:

We choose array and matrix as the storage of our data. This is because most of the following computation in the algorithm is the multiply of matrix or dot of the matrix. Thus, at the first stage, we read images as an (n,m) array, and then change it into (1,(n*m)) array so that we can put all images' data in one array to compute. Finally, we get a (184,961) array as the input data. And in the training and testing algorithm, the weights are also stored as arrays.

We also use a few lists when we predict the output. One is used for storing every prediction, and the other is used for copying y in a list. This is because lists are ordered, and we can easily compare them.

```
f = open("downgesture_train.list.txt")
train_x = []
train_y = []
for line in f.readlines():
    line = line.strip()
    if "down" in line:
        train_y.append(1.0)
    else:
        train_y.append(0.0)
    c = imageio.imread(line)
    m = c.shape[0]*c.shape[1]
    x = list(c.reshape(1,m))
    train_x += x
trainx = np.matrix(train_x) #184*960
trainy = (np.matrix(train_y)).T #184*1
w,w2 = train(trainx,trainy)

f2 = open("downgesture_test.list.txt")
test_x = []
test_y = []
for line in f2.readlines():
    line = line.strip()
    if "down" in line:
        test_y.append(1)
    else:
        test_y.append(0)
    c2 = imageio.imread(line)
    m2 = c2.shape[0]*c2.shape[1]
    x2 = list(c2.reshape(1,m2))
    test_x += x2
testx = np.matrix(test_x) #83*960
esty = (np.matrix(test_y)).T #83*1
```

```
    for j in range(len(prediction)):
        if prediction[j] >= 0.5:
            prediction[j] = 1.0
        else:
            prediction[j] = 0.0
        if prediction[j] == lable[j]:
            count += 1.0
    accuracy = count/updatex.shape[0]
    print ("prediction_list = ", prediction)
    print ("accuracy = ", accuracy)
```

Code-level Optimizations:

(1) Most of the time, we use an array instead of a matrix, so the computation is more clear, and the results are not likely to be wrong.

(2) We write the part of reading images and sigmoid function outside the main body of the algorithm so that there are less repeated codes and the computation is easier to read and check.

(3) We divided the train and the test as two independent parts, so if there is a mistake in one of them, the other has a low risk of being influenced, which means we will find the bug easier.

```
import imageio
import numpy as np
from math import exp

def sig(s):
    if s < 0:
        return 1 - 1 / (1 + exp(s))
    return 1 / (1 + exp(-s))

def trainNN(updatex,updatey):
    #处理数据和初始化
    #x
    a = np.ones(updatex.shape[0])
    updatex = np.insert(updatex, 0, values=a, axis=1) #184*961
    updatex = updatex/255.0
    update_w = np.random.randint(-1, 2, size=[961, 100])  # 961*100
    update_w2 = np.random.randint(-1, 2, size=[101, 1])  # 101*1
#     datanupw = -0.01+0.02*np.random.random((961, 100))  # 961*100
#     datanupw2 = -0.01+0.02*np.random.random((101, 1))  # 101*1

    for i in range(0, 1000):

        #random pickup t
        t = np.random.randint(0, updatex.shape[0], 1)
        #level l-1 :randomx, datanupw, datanup_s
        randomx = updatex[t]  # 1*961
        y = updatey[t]
        y = float(y)
        update_s = (randomx * update_w).tolist() #1*100 #s=x*w的和
        #level l-1:sigmoid
        s_l=[]
```

```
        for j in range(len(update_s[0])):
            l=[]
            l.append(sig(update_s[0][j]))
            s_l.append(l)
        updates=np.array(s_l).T#1*100

        b = np.ones(1)
        #level l: datanupx2, datanupw2, datanup_s2
        updatex2 = np.insert(updates, 0, values=b, axis=1)#1*101
        update_s2 = updatex2.dot(update_w2) #1*1
        #level l:sigmoid
        updatex3 = sig(update_s2)#x3=datanupx3


        #level l: g1
        g1 = 2.0 *(updatex3 - y)* updatex3 * (1.0 - updatex3)
        #更新w2=w2-0.1*g1*x1.T
        w2 = update_w2 - 0.1 * g1 * updatex2.T #101*1 #

        #level l-1: g2 = np.dot((datanupx2.T,(1.0 - datanupx2)), np.dot(w2,g1)) #101*1 ---
        a = updatex2 * (1.0 - updatex2)#1*101*1*101, multiply not dot
        g2 =np.matrix(a.T * np.dot(w2,g1)) #101*1
        g2 = np.delete(g2,0,0) #100*1
        #更新w=w-(g2*x*0.1).T
        w = update_w - np.dot(0.1,g2 * randomx).T#961*100
        update_w2 = w2
        update_w = w
    return w,w2
```

```
def testNN(updatex,updatey,w,w2):
    count = 0
    prediction = []
    lable=updatey
    a = np.ones(updatex.shape[0])
    updatex = np.insert(updatex, 0, values=a, axis=1)  # 83*961
    updatex = updatex / 255.0
    for i in range(0, updatey.shape[0]):
        x = updatex[i,:] #1*961
        y = updatey[i,0]
        y = float(y)
        #datanup_s = np.dot(siglex,w) # 1*100
        update_s =np.dot(x,w).tolist() #1*100 #s=x*w的和
        s_l=[]
        for j in range(len(update_s[0])):
            l=[]
            l.append(sig(update_s[0][j]))
            s_l.append(l)
        updates=np.array(s_l).T #1*100

        b = np.ones(1)
        updatex2 = np.insert(updates, 0, values=b, axis=1) # 1*101
        update_s2 = updatex2.dot(w2)  # 1*1
        updatex3 = sig(update_s2)
        prediction.append(updatex3)
```

```
    for j in range(len(prediction)):
        if prediction[j] >= 0.5:
            prediction[j] = 1.0
        else:
            prediction[j] = 0.0
        if prediction[j] == lable[j]:
            count += 1.0
    accuracy = count/updatex.shape[0]
    print ("prediction_list = ", prediction)
    print ("accuracy = ", accuracy)
```

Challenges:

(1) The first challenge is the math part, we didn't have clear thoughts of the relationship between the hidden layer and input or output. Also, we are not familiar with the back direction computation. So, we search some illustrations and read them until we completely understand the theory of neural networks.

(2) The second challenge is we don't know so many built-in functions of python, so we might write some long and unnecessary codes at the beginning. Then, we read some elegant codes from other people who share them online, and we learn some useful functions and use them as references.

Part 2: Software Familiarization

(1)How to use

Firstly, we use feature scaling so the training data can be uniformly evaluated. Specifically, we use StandardScaler to standardize features by removing the mean and scaling to unit variance, which can make a prediction as real as possible. Secondly, we use MLPClassifier to set the hidden_layer_sizes as 100 and max iteration as 1000. Finally, we will use the result of the MLPClassifier to make predictions on test data. Besides, we will adjust the number of the hidden layers, activation functions and size of the training and testing data to research if we can get a better result.

Code:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
predictions = mlp.predict(X_test)
```

(2)Compare and improve

The sklearn library function of neural networks is very concise and simple. The most important is that the sklearn library function has a prepossessing step on training data and test data, which makes predictions more accurate. In the future, we will improve our code by adding a prepossessing step.

Part 3: Applications

There are two main applications of the neural network. One is image processing and character recognition. We can take in a lot of inputs, process them to infer non-linear relationships. For image recognition, usually, we use facial recognition in social media, cancer detention in medicine. For character recognition, there are a lot of applications in fraud detection and even national security assessments. Another main application is forecasting. In the fields of sales, financial allocation, and capacity utilization, forecasting problem is significant, the neural network algorithm can provide robust alternatives, given its ability to model features and relationships.