INF 552 MACHINE LEARNING FOR DATA SCIENCE

HOMEWORK 4

Team member:

SANJAY MALLASAMUDRAM SANTHANAM – 3124715393

Part 1:

Language used:

Python 3, jupyter notebook

Data structures used:

Numpy array to store data

Code-level optimization:

Backpropagation is implemented as matrix operations instead of having for loop. It reduced the run time especially when running on GPU, which are specialized for matrix operations.

Challenges:

Adding bias term was a bit tough as they are different compared to other neurons in the sense they have no input and always have an output of 1. It is easy to forget the bias term when coding.

Part 2:

Library used: Scikit learn

Although the Neural network implementations gave good result, the library functions gives a better output because it uses different parameters like regularisation,minibatch gradient descent,inverse scaling learning rate,different optimisers ,momentum etc.

The results of the implementation can be improved by using different local search heuristics like different initial weights and choosing the best output(random restart), mini batch gradient descent,different optimisers like adam,rmsprop,increasing number of iterations,having different number of neurons in hidden layer, increasing the number of hidden layers,using regularization,dropouts etc. to improve the accuracy.

Also, different learning rate can be used to find the most optimal possible solution.

Part 3:

Neural network is used in anomaly detection(bank fraud detection),detect credit card attrition,loan application evaluation,market research,customer behavior modeling etc.

Recurrent neural network is used for natural language processing application like machine translation, question answering,search recommendation,speech recognition etc.

Generative adversarial Network are used for generating images, convert text to image (storyboarding), convert image to text(object detection,commentary generation for cricket) etc.

In [9]:
```
#Done by SANJAY MALLASAMUDRAM SANTHANAM ; USC ID:3124715393
import pandas as pd
import numpy as np
import math
from operator import truediv,add
import copy
import os
from PIL import Image
```

In [10]:
```
#list to store test data's classification label
test_label=[]
#list to store train data's classification label
train_label=[]
#list to store test data
test=[]
#list to store train data
train=[]
#read train data file names
f=open('C:/Users/Lenovo/Desktop/train.txt','r')
train_data_list=f.read()
f.close()
#read test data file names
f=open('C:/Users/Lenovo/Desktop/test.txt','r')
test_data_list=f.read()
f.close()
#split whole chunk of data to separate lines
train_data_list=train_data_list.split("\n")
test_data_list=test_data_list.split("\n")
#convert absolute path to relative path i.e. extract file names alone and discard the root d
irectory names
for i in range(0,len(train_data_list)):
    train_data_list[i]=train_data_list[i].split("/")[-1]
for i in range(0,len(test_data_list)):
    test_data_list[i]=test_data_list[i].split("/")[-1]
#directory where training data images are stored
direc="C:/Users/Lenovo/Desktop/inf ml hw/hw5/gestures"
#get the names of all folders inside the directory
path=os.listdir(direc)
#loop for each folder
for folder in path:
    #loop through all images inside a folder
    for p in os.listdir(direc+"/"+folder+"/"):
        #positive sample if image name contains the word 'down'
        if "down" in p:
            #if image name is in training data list, add it to training data
            if p in train_data_list:
                #+1 denotes label of positive sample
                train_label.append(+1)
                #store image as numpy array
                data=np.asarray(Image.open(direc+"/"+folder+"/"+p))
                #Since images are multi dimensional, convert to 1 D by flattening to so that
we can feed it to neural network
                #as input
                train.append(data.flatten())
            else:
                test_label.append(+1)
                data=np.asarray(Image.open(direc+"/"+folder+"/"+p))
                test.append(data.flatten())
        #negative sample otherwise
        else:
            if p in train_data_list:
                #0 denotes label of negative sample
                train_label.append(0)
                data=np.asarray(Image.open(direc+"/"+folder+"/"+p))
                train.append(data.flatten())
            else:
                test_label.append(0)
                data=np.asarray(Image.open(direc+"/"+folder+"/"+p))
                test.append(data.flatten())
```

In [11]:
```
#number of neurons in input layer. Bias included
L0=len(train[0])+1
#nnumber of neurons in hidden layer including bias
L1=100+1
#Final layer has only one neuron as output .No bias term
L2=1
#initialise weights randomnly between -0.01 and 0.01
w1=-0.01+np.random.random((L1-1,L0))*0.02
w2=-0.01+np.random.random((L2,L1))*0.
#learning rate
lr=0.1
```

In [12]:
```
#1000 epochs
for epoch in range(0,1000):
    #each epoch looks through every data point once.
    for i in range(0,len(train)):
        #forward pass
        #Define input to Neural net
        X=np.ones((L0,1))
        X[1:]=np.reshape(train[i],(L0-1,1))
        #h_net=np.zeros((L1-1,1))
        #h_out=np.zeros((L1,1))

        #Hidden layer
        #compute weighed sum from input layer
        h_net=np.matmul(w1,X)
        #compute sigmoid i.e. non-linearity function, the neuron outputs
        h_out=np.vstack((1,1/(1+np.exp(-h_net))))
        #o_net=np.zeros((L2,1))
        #o_out=np.zeros((L2,1))

        #Output layer
        #compute weighed sum from hidden layer outputs
        o_net=np.matmul(w2,h_out)
        #compute sigmoid i.e. non-linearity function, the neuron output
        o_out=1/(1+np.exp(-o_net))

        #backpropagaton
        #derivative of mean squared error wrt output layer's final  output
        dEdo_out=o_out-train_label[i]
        #derivative of output layer's final output wrt to its net input
        do_outdo_net=o_out*(1-o_out)
        #update value for w2
        w2update=2*dEdo_out*do_outdo_net*h_out.T
        #update value for w1
        w1update=np.matmul((2*dEdo_out*do_outdo_net*w2.T*h_out*(1-h_out))[1:],X.T)

        #update weights between hidden layer and output layer
        w2-=lr*w2update
        #update weights between input layer and hidden layer
        w1-=lr*w1update
```

In [ ]:

In [13]:
```
#prepare test data by appending x0=1 to all test data points
test=np.hstack((np.ones((np.shape(test)[0],1)),test))
```

In [14]:
```
#predict output
y_pred=1/(1+np.exp(-np.matmul(np.hstack((np.ones((np.shape(test)[0],1)),1/(1+np.exp(-np.matmul(test,w1.T))))),w2.T)))
```

In [15]:
```
print("Predictions",y_pred)
```

```
Predictions [[0.98646339]
 [0.98642533]
 [0.98649176]
 [0.95817089]
 [0.9864935 ]
 [0.00169114]
 [0.03236278]
 [0.30543416]
 [0.00168507]
 [0.00168964]
 [0.00244507]
 [0.00289137]
 [0.00289214]
 [0.00289164]
 [0.00289646]
 [0.00406024]
 [0.00547234]
 [0.01430681]
 [0.00175483]
 [0.00232697]
 [0.00284207]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.092277  ]
 [0.10738723]
 [0.11155082]
 [0.11155037]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
```

```
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
[0.11155082]
```

```
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.11155082]
 [0.98649582]
 [0.98649492]
 [0.00759025]
 [0.01386122]
 [0.00289284]
 [0.00289121]
 [0.0138479 ]
 [0.11155082]]
```

In [16]:
```python
#Mean square error of raw neural network output
print("Mean Squared Error:",np.sum(np.square(y_pred-test_label))/len(test))
```

```
Mean Squared Error: 43.29638121504464
```

In [17]:
```python
#round off the output to its nearest label. If output value is > 0.5 it is assigned label 1 else 0.
y_pred_round=np.zeros(np.shape(y_pred)[0])
for i in range(0,len(y_pred)):
    if(y_pred[i]>0.5):
        y_pred_round[i]=1
print(y_pred_round)
```

```
[1. 1. 1. 1. 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. 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. 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. 1. 0. 0. 0. 0. 0. 0.]
```

In [18]:
```python
#Mean square error of rounded off output
print("Mean Squared Error:",np.sum(np.square(y_pred_round-test_label))/len(test))
```

```
Mean Squared Error: 0.21164021164021163
```

In [19]:
```python
#library function
from sklearn.neural_network import MLPClassifier
#train library function neural network
clf = MLPClassifier(activation='logistic',max_iter=1000,hidden_layer_sizes=(100))
clf.fit(train,train_label)
#predict output
pred=clf.predict(test[:,1:])
print(pred)
```

```
[1 1 1 1 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
 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 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 0 0
 0 0 0 0]
```

In [20]:
```python
#mean square error library function's neural network
print("Mean Squared Error:",np.sum(np.square(pred-test_label))/len(test))
```

```
Mean Squared Error: 0.12698412698412698
```

In [21]:
```python
#print actual label for comparison
print(test_label)
```

```
[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0]
```

In [22]:
```
#predict how well the implemented neural network's output compare to the library function
clf.score(test[:,1:],y_pred_round)
```

Out[22]: 0.9153439153439153

In [23]:
```
#predict how well the library neural network's output compare to the library function.it sho
uld be 1 as from
#library function's Point of view, its output is the most accurate
clf.score(test[:,1:],pred)
```

Out[23]: 1.0

In [24]:
```
#find number of miscalssified points for implemented neural network
err=0
for i in range(0,len(test_label)):
    if(y_pred_round[i]!=test_label[i]):
        err+=1
print("Number of Misclassified points:",err)
print("Accuracy :",(len(test_label)-err)/len(test_label))
```

```
Number of Misclassified points: 40
Accuracy : 0.7883597883597884
```

In [25]:
```
#find number of miscalssified points for library function's neural network
err=0
for i in range(0,len(test_label)):
    if(pred[i]!=test_label[i]):
        err+=1
print("Number of Misclassified points:",err)
print("Accuracy :",(len(test_label)-err)/len(test_label))
```

```
Number of Misclassified points: 24
Accuracy : 0.873015873015873
```

In [ ]: