Programming Assignment 5: Feedforward neural network

1. Group member

   a) Hao Yang, USCID: 5284267375. (Contributions: All)

2. Part 1: Implementation

   1. Programming language: Python 3.7

   2. Libraries used: PIL, numpy, random, matplotlib.pyplot

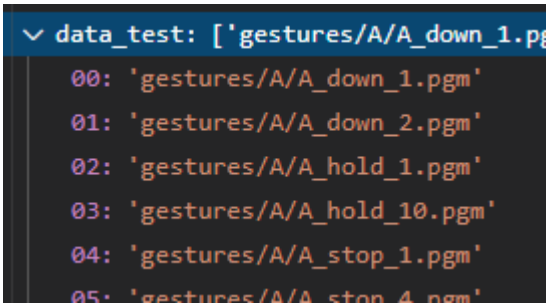
   3. Neural network algorithm: FNN.

   4. Script description:

      1. Data processing: read list file row by row, record content in list.

```
data_train = []
with open('downgesture_train.list','r') as train:
    line = train.readline()
    while line != '':

        data_train.append(line.strip('\n'))
        line = train.readline()
data_test = []
with open('downgesture_test.list','r') as test:
    line = test.readline()
    while line != '':

        data_test.append(line.strip('\n'))
        line = test.readline()
```

Get the list of train data and test data:



According to the list, read corresponding pgm image with PIL.image library, and reshape it to a vector for FNN. Then normalize pixel from 0 to 1. Go through all image in list.

```
def Convert_data(data_train):    # get dataset
    imgfile = "./"+data_train[0]
    img = np.array(Image.open(imgfile).convert('L'), 'f')
    m,n = list(img.shape)        # size of image
    x = (img/255).reshape(1,m*n).T
    y = [1] if "down" in imgfile else [0]
    for i in range(1,len(data_train)):
        imgfile = "./"+data_train[i]
        # size of image is 30 * 32 = 960, convert into vector of 1 * 960
        img = (np.array(Image.open(imgfile).convert('L'), 'f')/255).reshape(1,m*n).T
        x = np.hstack((x,img))
        y.append(1 if "down" in imgfile else 0)
    y = np.array(y).T
    return x,y
```

We get matrix of train data and test data with size of 960x184 and 960x83 correspondingly:

```
x,y = Convert_data(data_train)
x_test,y_test = Convert_data(data_test)
```

```
x: array([[0.        , 0.        ,
  [0:960] : [array([0.        , 0...
  dtype: dtype('float32')
  max: 0.827451
  min: 0.0
  shape: (960, 184)
  size: 176640
  __internals__: {'T': array([[0.
x_test: array([[0.        , 0.
```

2. FNN:

   The function have 5 parameters, x for training dataset, y for labels, epoch for max epoch number, hl for number of neuron in hidden layer, and ini for range of initial weight [-ini, ini].

```
def FNN(x,y,lr=0.1,epoch=1000,hl=100,ini=0.01):
```

Step 1: initialize the weight and bias between -0.01 to 0.01 for each layer:

```
def FNN(x,y,lr=0.1,epoch=1000,hl=100):
    input_size = list(x.shape)[0]           # get input size
    image_num = list(x.shape)[1]
    # initialize weight and bias from input layer to hidden layer
    v = np.mat([random.uniform(-0.01,0.01) for i in range(input_size)])
    r = np.mat(random.uniform(-0.01,0.01))
    for nerual in range(1,hl):
        v = np.vstack((v,np.mat([random.uniform(-0.01,0.01) for i in range(input_size)])))
        r = np.vstack((r,random.uniform(-0.01,0.01)))
    # initialize weight and bias from hidden layer to output layer
    w = np.mat([random.uniform(-0.01,0.01) for i in range(hl)])
    o = random.uniform(-0.01,0.01)
```

Step 2: In each epoch, read all images one by one, and compute their loss with result. Then compute the gradients based on the loss, and update weights and biases to the direction of gradients decent.

```
loss = []
for loop in range(epoch):
    print(str(loop)+'/1000')
    cost = []
    for item in range(image_num):
        b = sigmoid(v*(x[:,item].reshape(960,1)) - r).tolist()
        y_pre = (sigmoid(w * sigmoid(v*(x[:,item].reshape(960,1)) - r) - o)).tolist()[0][0]
        Ek = 0.5 * ((y_pre - y[item])**2)
        cost.append(Ek)
        # start computing gradients
        g = y_pre * (1-y_pre) * (y[item]-y_pre)
        eh = []
        for h in range(hl):
            bh = b[h][0]
            eh.append(bh*(1-bh)*g*(w.tolist()[0][h]))
        # update weights and biases
        w += lr*g*sigmoid(v*(x[:,item].reshape(960,1)) - r).T
        o -= lr * g
        eh = np.mat(eh).reshape(1,100)
        v += (eh.T * x[:,item].reshape(960,1).T)*lr
        r -= lr*eh.T
    loss.append(sum(cost))
```

Return all weights and biases and loss record:

```
return v,r,w,o,loss
```

3. Test:

   Make a prediction for test data, and compare the prediction with the label, compute the accuracy rate:

```
def FNN_test(x,y,v,r,w,o):
    image_num = list(x.shape)[1]
    y_pre = []
    for item in range(image_num):
        pre = (sigmoid(w * sigmoid(v*(x[:,item].reshape(960,1)) - r) - o)).tolist()[0][0]
        result = 1 if pre>=0.5 else 0
        y_pre.append(result)
    y_pre = np.array(y_pre)
    acc = np.sum(np.equal(y_pre, y)==True)


    return y_pre.tolist()[0], acc/image_num
```

5. Optimizations

    1. Use matrix operation instead of compute weights and bias for each neuron one by one, which can improve the implement speed.
    2. Set learning rate, max epoch and hidden layer details as a parameter rather than hard code it. This can help debugging with different network structure.
    3. Record loss in each step, can track if the function works correctly.

6. Challenges faced

    1. Found that the magic num of pgm image in this assignment is P5 rather than P2, which means the content of document can't read in txt file, and must to use PIL.image.convert() to read the pixel value.

```
imgfile = "./"+data_train[0]
img = np.array(Image.open(imgfile).convert('L'), 'f')
```

7. Result of prediction

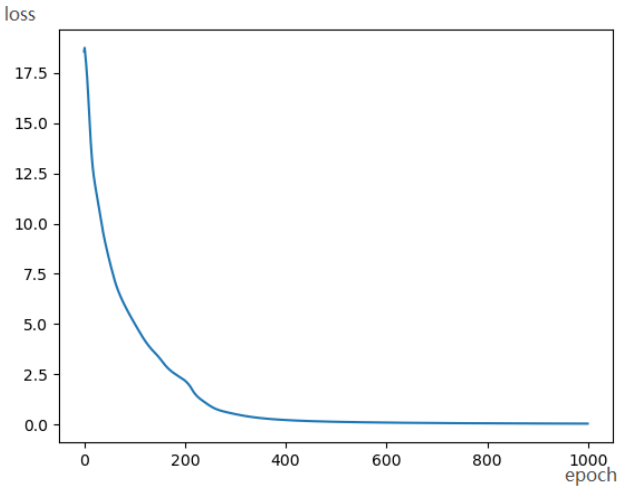
    Run code below:

```
x,y = Convert_data(data_train)
x_test,y_test = Convert_data(data_test)



v,r,w,o,loss = FNN(x,y)
prediction, acc = FNN_test(x_test,y_test,v,r,w,o)
print("Prediction of test data:" + str(prediction))
print("Accuracy rate:" + str(acc))

plt.plot(loss)
plt.show()
```

Get the result of weights and biases, and plot the loss curve:

```
Prediction of test data:[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0
, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

```
Accuracy rate:0.9156626506024096
```



After over 10 times of test, the accuracy rate can achieve 87%-91% with different initial weights.

What's more, I output some error predicted image, found out that they have some shadow and this shadow effects the prediction. The margin of shadow would be recognized as the finger so predicted as "up".

3. Part 2: Software familiarization

1. Library function: sklearn.neural_network.MLPClassifier

2. How to use: use clf=MLPClassifier().fit(x,y) to train data, then use clf.predict(x_test) to test data:

```
clf = MLPClassifier(solver='sgd',activation='logistic',hidden_layer_sizes=(100),learning_rate_init=0.1,max_iter=1000)
clf.fit(x,y)
y_pred = clf.predict(x_test)

acc = (np.sum(np.equal(y_pred, y_test)==True))/y_pred.shape
```

Result:

```
PS D:\Users\yangh\Desktop\552\HW5> & C:/Users/yangh/AppData/Local/Programs/Python/Python37/python.exe d:/Users/yangh/Desktop/552/HW5/test.py
[1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 0]
[0.89156627]
```

3. Comparison:

    1. Difference:

        The function in sklearn have more parameters can set than mine. For example, it can not only choose sgd for gradients decent, it has more choice like “adam” and “lbfgs”, which fits different data size. What’s more, it can choose activation function rather than only sigmoid in my implementation. The most important is it can build different structure of NNs. in my implementation, for time limits, I can only complete the requirement that with one hidden layer with size 100. In sklearn, you can choose any number of hidden layers.

    2. Results:

```
clf = MLPClassifier(solver='sgd',activation='logistic',hidden_layer_sizes=(100),learning_rate_init=0.1,max_iter=1000)
clf.fit(x,y)
y_pred = clf.predict(x_test)

acc = (np.sum(np.equal(y_pred, y_test)==True))/y_pred.shape
```

```
PS D:\Users\yangh\Desktop\552\HW5> & C:/Users/yangh/AppData/Local/Programs/Python/Python37/python.exe d:/Users/yangh/Desktop/552/HW5/test.py
[1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 0]
[0.89156627]
```

```
Prediction of test data:[1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0
, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

```
Accuracy rate:0.9156626506024096
```

In this assignment, I can get accuracy rate of 87%-91% with different initial

weights, and 89%-96% in sklearn implementation with different solver chosen. The results are similar, which can say my implementation works correctly.

4. Part 3: Applications

Neural network is one of the most popular models in artificial intelligence. It has used in our daily life widely. For example, the face recognition function in our cell phone unlock use CNN to train. Many license plate recognitions in toll gate also use CNN to recognize. Can say CNN is the most important algorithm in image recognition.