INF552: Programming Assignment 5

Yi-Li Chen

Part 1: Implementation

- All imported library

```
1. import csv
2. import os
3. from PIL import Image
4. import numpy as np
5. from sklearn.metrics import accuracy_score
6. import random
7. import matplotlib.pyplot as plt
```

- Read the text file, 'downgesture_test.list', into a list so that we can get the test data by this later

```
1. test_down_labels = []
2. with open('downgesture_test.list.txt') as f:
3.     file = csv.reader(f)
4.     for line in file:
5.         test_down_labels.append(line[0])
```

- Read the text file, 'downgesture_train.list', into a list so that we can get the training data by this later

```
1. train_down_labels = []
2. with open('downgesture_train.list.txt') as f:
3.     file = csv.reader(f)
4.     for line in file:
5.         train_down_labels.append(line[0])
```

- I used the PIL library to read the .pgm file and labeled each data, by the `train_down_labels` list we have built above. Therefore, I got the X and y for training data.

```
1. def read_imgs(data_path, labels):
2.     all_files = os.listdir(data_path)
3.     imgs_data = []
4.     imgs_label = []
5.     for folder in all_files:
6.         folder_path = os.path.join(data_path, folder)
7.         # check if it is folder
8.         if os.path.isdir(folder_path):
```

```
            all_imgs = os.listdir(folder_path)
            for img_name in all_imgs:
                name = 'gestures/'+folder+'/'+img_name
                if name in labels:
                    img_path = os.path.join(folder_path,img_name)
                    img = Image.open(img_path)
                    img_bytes = np.array(img)
                    img_gray_scale = img_bytes/255
                    imgs_data.append(img_gray_scale.flatten())
                    if 'down' in name:
                        imgs_label.append(1)
                    else:
                        imgs_label.append(0)
    return np.array(imgs_data), np.array(imgs_label)
```

- After setting the data path, I get the data and their labels from the above function and give all rows x_0=1.

```
data_path = '/Users/leo/Desktop/Analytics/2020 Fall/DSCI552/HW5_1028due/gestures'
training_imgs_data, training_imgs_label = read_imgs(data_path, train_down_labels)
ytrain = training_imgs_label
Xtrain = np.concatenate((np.ones((len(training_imgs_data),1)),
                         training_imgs_data), axis=1)
```

- Each neuron has a set of weights that need to be maintained. One weight for each input connection and an additional weight for the bias. I initialized the weights within -0.01 to 0.01 randomly and store as array. A network is organized into layers. The input layer is really just a row from our training dataset. The first real layer is the hidden layer. This is followed by the output layer that has one neuron for each class value.

```
def initialize_NN(n_inputs, n_hidden, n_outputs):
    network = []
    np.random.seed(0)
    hidden_layer = np.random.randint(-100,100, size=(n_inputs+1,n_hidden))/10000
    network.append(hidden_layer)
    np.random.seed(0)
    output_layer = np.random.randint(-100,100, size=(n_hidden+1,n_outputs))/10000
    network.append(output_layer)
    return network
```

- Build a function to calculate neuron activation which is the weighted sum of the inputs.

```
def calculate_activation(w, inputs):
    s = inputs.dot(w)
    return s
```

- Transfer an activation function using the sigmoid function as neuron output

```
def sigmoid(s):
    return 1/(1+np.exp(-s))
```

- The function below that implements the forward propagation for a row of data from our dataset with our neural network.

```
def forward_propagate(network, row):
    inputs = row
    output_list = []
    for layer in network:
        s = inputs.dot(layer)
        new_inputs = sigmoid(s)
        output_list.append(new_inputs)
        inputs = np.insert(new_inputs,0,1)
    return new_inputs, output_list
```

- Given an output value from a neuron, we can calculate its derivative and the error for each output neuron with backpropagation. Therefore, I can get all δ for each layer and update all the weights in neuron network.

```
def backpropagate(yhat, output_list, xi, yi, NN):
    delta = 2*(yhat-yi)*(yhat*(1-yhat))
    deltaj = (yhat*(1-yhat))*NN[1]*delta
    deltaj = np.delete(deltaj,0)
    NN[0] = NN[0] - 0.1*xi.reshape(-1,1)*deltaj
    xj = np.insert(output_list[0],0,1)
    NN[1] = NN[1] - 0.1*delta*xj.reshape(-1,1)
    return NN
```

- Train the NN step by step for each row of whole data set. In this case, I run 1000 training epochs and return the result of all weights.

```
def train(X ,y, epochs):
    NN = initialize_NN(960,100,1)
    for epoch in range(epochs):
        for xi, yi in zip(X, y):
```

```
            yhat, output_list = forward_propagate(NN, xi)
            NN = backpropagate(yhat, output_list, xi, yi, NN)
    return NN
```

```
NN = train(Xtrain ,ytrain, 1000)
```

- Build a function to predict the test data and I have set a threshold to judge the img to be 'down' gesture or not. If the final output after at last layer is greater than 0.5, it will be classified as 'down' gesture. Lastly, return the result of predictions and the accuracy.

```
def predict(NN, Xtest, ytest, threshold):
    y_pred_list = []
    for xi, yi in zip(Xtest, ytest):
        yhat, output_list = forward_propagate(NN, xi)
        if yhat > threshold:
            y_pred_list.append(1)
        else:
            y_pred_list.append(0)
    acc = accuracy_score(ytest, y_pred_list)
    return y_pred_list, acc
```

- The result of predictions and accuracy

```
test_imgs_data, test_imgs_label = read_imgs(data_path, test_down_labels)
ytest = test_imgs_label
Xtest = np.concatenate((np.ones((len(test_imgs_data),1)), test_imgs_data), axis=1)
y_pred_list, acc = predict(NN, Xtest, ytest, 0.5)
np.array(y_pred_list), acc
```

Prediction:

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0])
```

Accuracy = 0.8148148148148148

For me, this assignment is kind of a big challenge. This is the first time I define the algorithm of neuron network step by step and use the .pgm image as input files. In the past, I always just call the related library and train the model. But in this assignment, I clearly realized every step of the algorithm in NN.