類神經網路第二次作業

(1) 程式介面說明

兩個檔案：new_simulator.py 和 NN_HW2.py

從 new_simulator.py 執行即可，NN_HW2.py 已經 import 進去了，執行後選擇檔案再按下開始鍵就會有動畫

(2) 程式碼說明

定義一個 class 叫做 neuron，會用 weight(array random)初始化它，裡面有一個 function 叫 forward，負責前饋

```
class neuron:
    def __init__(self, weight):
        self.weight = weight

    def forward(self, feature):
        v = self.weight.dot(feature)
        v = float(v)
        y = sigmoid(v)
        return y
```

定義一個 class 叫做 Network，以下是裡面所有的 function

```
class Network:
    def __init__(self, hidden_layer_neuron, output_layer_neuron):···

    def feedforward(self, feature):···

    def delta_hiddenlayer(self, delta_out, hidden_output):···

    def delta_outputlayer(self, d, predict):···

    def modify_out_w(self, learning_rate, delta, hidden_output):···

    def modify_hidden_w(self, learning_rate, delta, feature):···

    def train(self, true, features):···
```

__init__:用隱藏層(8 個 neuron)和輸出層(1 個 neuron)初始化它

```
def __init__(self, hidden_layer_neuron, output_layer_neuron):
    self.hidden_layer = hidden_layer_neuron
    self.output = output_layer_neuron
```

feedforward:呼叫 neuron 裡面的 forward 進行前饋，隱藏層輸出的結果會存到 hidden_layer_output 的 list 並加入-1(bias)再轉成 array，把這個結果給輸出層再進行一次前饋

```
def feedforward(self, feature):
    hidden_layer_output = []
    for i in range(0, len(self.hidden_layer)):
        res = self.hidden_layer[i].forward(feature)
        hidden_layer_output.append(res)

    hidden_layer_output.insert(0, -1.)
    hidden_layer_output = np.array(hidden_layer_output)
    final = self.output.forward(hidden_layer_output)

    return hidden_layer_output, final
```

delta_hiddenlayer:計算隱藏層 delta 值

$$\delta_j(n) = y_j(n)\left(1 - y_j(n)\right)\sum_k \delta_k(n) w_{k_j}(n)$$

```
def delta_hiddenlayer(self, delta_out, hidden_output):
    delta_hidden = []
    for i in range(1, len(hidden_output)):
        delta_hidden.append(
            (hidden_output[i])*(1-hidden_output[i])*delta_out*(self.output.weight[0][i]))
    return delta_hidden
```

delta_outputlayer:計算輸出層 delta 值

$$\delta_j(n) = \left(d_j(n) - O_j(n)\right) O_j(n) \left(1 - O_j(n)\right)$$

```
def delta_outputlayer(self, d, predict):
    delta_out = (d-predict)*predict*(1-predict)
    return delta_out
```

modify_out_w:計算輸出層鍵結值要調整多少

```
def modify_out_w(self, learning_rate, delta, hidden_output):
    mod_out_w = learning_rate*delta*hidden_output
    mod_out_w = mod_out_w.reshape(1, 9)
    return mod_out_w
```

modify_hidden_w:計算隱藏層鍵結值要調整多少

```
def modify_hidden_w(self, learning_rate, delta, feature):
    tmp = feature.reshape(1, 4)
    mod_hidden_w = []
    for i in range(0, len(delta)):
        mod_hidden_w.append(learning_rate*delta[i]*tmp)
    return mod_hidden_w
```

train：前饋->倒傳遞->調整鍵結值

Epoch 和 learning rate 皆可以調整

```
def train(self, true, features):
    learning_rate = 0.5
    epoch = 100
    for i in range(epoch):
        for j in range(0, len(true)):
            # forward
            hidden_output, predict = self.feedforward(features[j])

            if predict != true[j]:
                # update output layer
                delta_out = self.delta_outputlayer(true[j], predict)

                mod_out_w = self.modify_out_w(
                    learning_rate, delta_out, hidden_output)

                # update hidden layer
                delta_hidden = self.delta_hiddenlayer(
                    delta_out, hidden_output)

                mod_hidden_w = self.modify_hidden_w(
                    learning_rate, delta_hidden, features[j])

                for k in range(0, len(self.hidden_layer)):
                    self.hidden_layer[k].weight = self.hidden_layer[k].weight + \
                        mod_hidden_w[k]

                self.output.weight = self.output.weight+mod_out_w
                # calculate loss

    return self.hidden_layer, self.output.weight
```

test：測試資料進行正規化->用 training 最後輸出的鍵結值進行前饋->取消正規化->輸出預測值

```
def test(hiddenlayer_weight, outputlayer_weight, parms, test_data):
    test_hidden_out = []
    test_data.insert(0, -1)
    test_arr = np.array(test_data)
    test_arr = test_arr.astype(float)

    for i in range(1, 4):
        test_arr[i] = (test_arr[i]-parms[1])/(parms[0]-parms[1])

    for j in range(0, len(hiddenlayer_weight)):
        h_out = sigmoid(hiddenlayer_weight[j].weight.dot(test_arr))
        test_hidden_out.append(h_out)

    test_hidden_out.insert(0, -1.)
    test_hidden_out_arr = np.array(test_hidden_out)

    angle = sigmoid(outputlayer_weight.dot(test_hidden_out_arr))
    angle_recover = angle*(parms[2]-parms[3])+parms[3]
    # f.write(str(angle_recover)+'\n')
    return angle_recover
```

sigmoid: activation function

```
def sigmoid(v):
    res = 1/(1+math.exp((-1)*v))
    return res
```

read_data: 讀資料->對 input data 和期望輸出進行正規化

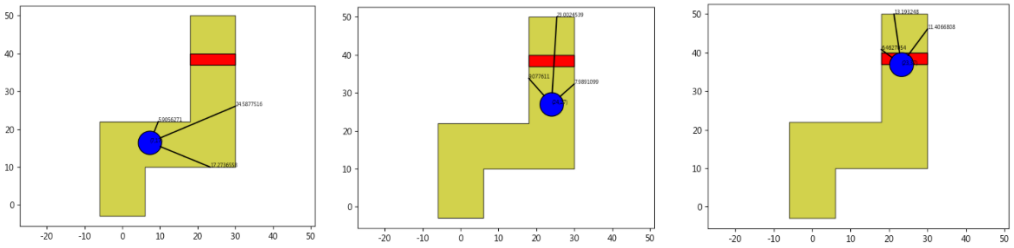
```
def read_data(address):
    Input = open(address)
    data = Input.readlines()
    x_train = []
    x_test = []
    a = []
    b = []
    for line in data:
        buff = line.split(' ')
        for i in range(0, 4):
            if i == 3:
                b.append(float(buff[i]))
            else:
                a.append(float(buff[i]))

    a = np.array(a)
    b = np.array(b)

    train_max = a.max()
    test_max = b.max()
    train_min = a.min()
    test_min = b.min()
```

```
for line in data:
    buff = line.split(' ')

    stringArr = np.array([-1., buff[0], buff[1], buff[2]])
    floatArr = stringArr.astype(float)
    for i in range(1, 4):
        floatArr[i] = (floatArr[i]-train_min)/(train_max-train_min)
    x_train.append(floatArr)

    stringD = buff[3]
    floatD = float(stringD)
    x_test.append((floatD-test_min)/(test_max-test_min))

return x_train, x_test, train_max, train_min, test_max, test_min
```

主程式:建立隱藏層和輸出層的神經元->丟進 Network 裡面->呼叫 train function 進行訓練->得到訓練過後的鍵結值->給 test function

```
# main
parms = []
hidden_layer_neuron = []
for i in range(0, 8):
    hidden_layer_neuron.append(neuron(np.random.rand(1, 4)))

output_layer_neuron = neuron(np.random.rand(1, 9))

network = Network(hidden_layer_neuron, output_layer_neuron)

features, d, train_max, train_min, test_max, test_min = read_data(
    'C:\\Users\\angela_cheng\\Downloads\\NN+HW2_Dataset\\train4dAll.txt')

parms.append(train_max)
parms.append(train_min)
parms.append(test_max)
parms.append(test_min)

hiddenlayer_weight, outputlayer_weight = network.train(d, features)
```

(3) 實驗結果(移動軌跡截圖)



(4) 分析

1. Learning rate：
   原本訓練時 learning rate 都只用 0.01，怎麼訓練都一直撞牆，把訓練資料的 predict 值輸出後發現表現也很差，介於-10~10 度之間。後來將 learning rate 調成 0.5，結果明顯變好，猜測因為一開始的鍵結值是用亂數產生的，那些值都小數點後好幾位，可能 learning rate 用 0.01 更新鍵結的速度太慢了。
2. Epoch：
   因為我的 function 一個 epoch 就會看過全部 1475 筆資料，所以跑 10 個 epoch 就相當於鍵結值已經調整 14750 次了，所以 epoch 不用設成上萬次。
3. 隱藏層神經元數目：
   目前隱藏層使用的神經元有 8 個，之前用 6 個以下的神經元去訓練效果不是很好，可能是因為這樣座標上分隔的空間數量太少。
4. 正規化：
   因為是用 sigmoid function 的關係，它的輸出值只會介在 0~1 之間，因此 training data 和期望輸出都要各自進行正規化表現會比較好

$$\text{new} = \frac{\text{data} - \text{min}}{\text{max} - \text{min}}$$

   不過最後 output 的時候要記得乘回來！