Computer Assignment 1

จัดทำโดย

นายรักษ์พงศ์ ทอหุล 600610769

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของวิชา

CPE 261456 (Introduction to Computational Intelligence)

ภาคเรียนที่ 1 ปีการศึกษา 2563

มหาวิทยาลัยเชียงใหม่

สารบัญ

Method

การทำงานของระบบโดยรวม

1. ระบบจะรับอินพุตดังนี้
    - อาเรย์แสดงจำนวนฮิดเดนเลเยอร์(hidden layer)และจำนวนโหนดของเน็ตเวิร์ค
    - Data_num เป็นเลขที่จะใช้กำหนดว่าจะเอาอินพุตมาจากไฟล์ flood.txt หรือ cross.pat
    - ค่าเลินนิ่งเรท(Learning_rate)
    - ค่าโมเมนตัมเรท(momentum_rate)
    - ฟังก์ชั่นการกระตุ้น(activation)
    - จำนวนรอบการทำงาน(epoch)
2. ทำการสุ่มค่า weight และค่า bias ในแต่ละโหนดใน neural network
3. นำค่าอินพุตจากไฟล์ที่กำหนดมาผ่านการ normalization และ cross validation 10% เป็น train data และ test data
4. นำ train data ที่ได้มาผ่านเข้า neural network โดยนำ input mathix ไปคูณกับ weight แต่ละตัวแล้วนำไปบวกกับ bias จากนั้นนำไปเข้า activation function แล้วนำไปเข้าเป็น input ของ layer ถัดไป กระบวนการนี้เรียกว่า feed forward
5. จะได้ค่าผลลัพธ์สุดท้ายมาเราก็นำมาทพการคำนวณหาค่า error แล้วทำกระบวนการ back propagation เพื่อเปลี่ยนแปลงค่า weight และ bias ให้ได้ค่า error น้อยลงโดยจะมีขั้นตอนดังนี้
    - หาค่า error ของผลลัพธ์จากสมการ $e_j(t) = d_j(t) - y_j(t)$
    - นำค่า error ที่ได้ไปหา gradient จากสมการ $\delta_j(t) = e_j(t)\varphi_j'(v_j(t))$
    - การหา gradient ใน hidden layer หาได้จากสมการ
      $$\delta_j^{(l)}(t) = \varphi_j^{(l)'}\left(v_j^{(l)}(t)\right)\sum_k \delta_k^{(l+1)}(t) w_{kj}^{(l+1)}(t)$$
    - นำค่า gradient ที่ได้ไปปรับค่า weight จากสมการนี้
      $$\Delta w_{ji}^{(l)}(t) = \alpha \Delta w_{ji}^{(l)}(t-1) + \eta \delta_j^{(l)}(t) y_i^{(l-1)}(t)$$
6. ทำการวนลูปทำงานเรื่อย ๆ ตามค่า epoch โดย 1 epoch คือการทำงาน 1 รอบโดยรับ input จาก train data ครบทุกตัว 1 ครั้ง

Result

ในการทดลองนี้จะทำการทดลองเปลี่ยนแปลงค่าต่าง ๆ จาก input ทั้งสองไฟล์ทั้ง flood.txt และ cross.pat ดังนี้

1. ทดลองเปลี่ยนจำนวนโหนดของ hidden layer 10 ค่าโดยมีรายละเอียดดังนี้
    - Hidden layer มี 2 ชั้นโดยจำนวนโหนดจะเปลี่ยนแปลงตามค่า i ที่อยู่ในช่วง 1-10 ตามสมการนี้

      ```
      layer = [(2+(i*2)),(1+(i*2))]
      ```

    - ค่า learning rate คงที่เท่ากับ 0.15
    - ค่า momentum rate คงที่เท่ากับ 0.2
    - Activation เป็น ‘sigmoid’
    - ค่า epoch คงที่เท่ากับ 1000
2. ทดลองเปลี่ยนค่า learning rate 10 ค่าโดยมีรายละเอียดดังนี้
    - Hidden layer มี 2 ชั้นโดยจำนวนโหนดเท่ากับ [6,5]
    - ค่า learning rate เปลี่ยนตามค่า i ที่อยู่ในช่วง 1-10 ตามสมการนี้

      ```
      learning_rate = (0.01*i)+0.1
      ```

    - ค่า momentum rate คงที่เท่ากับ 0.2
    - Activation เป็น ‘sigmoid’
    - ค่า epoch คงที่เท่ากับ 1000
3. ทดลองเปลี่ยนค่า momentum rate 10 ค่าโดยมีรายละเอียดดังนี้
    - Hidden layer มี 2 ชั้นโดยจำนวนโหนดเท่ากับ [6,5]
    - ค่า learning rate คงที่เท่ากับ 0.15
    - ค่า momentum rate เปลี่ยนตามค่า i ที่อยู่ในช่วง 1-10 ตามสมการนี้

      ```
      momentum_rate = (0.01*i)+0.2
      ```

    - Activation เป็น ‘sigmoid’
    - ค่า epoch คงที่เท่ากับ 1000

โดยได้ผลการทดลองดังนี้

1. **การทดลองเปลี่ยนจำนวนโหนดของ hidden layer**

   มีการตั้งค่าตามรายละเอียดดังนี้

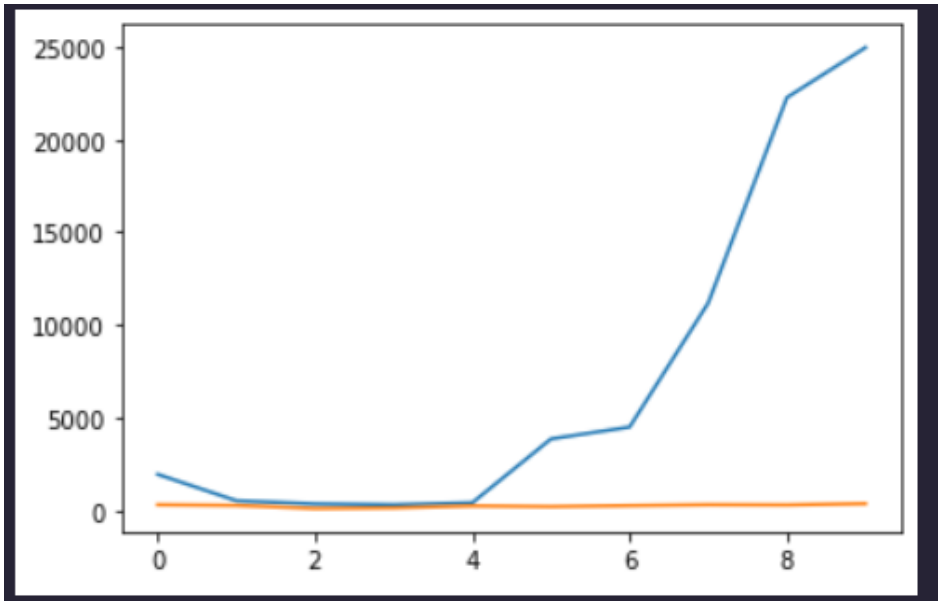
```python
# Find best layer for Flood_dataset.txt
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [(2+(i*2)),(1+(i*2))]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

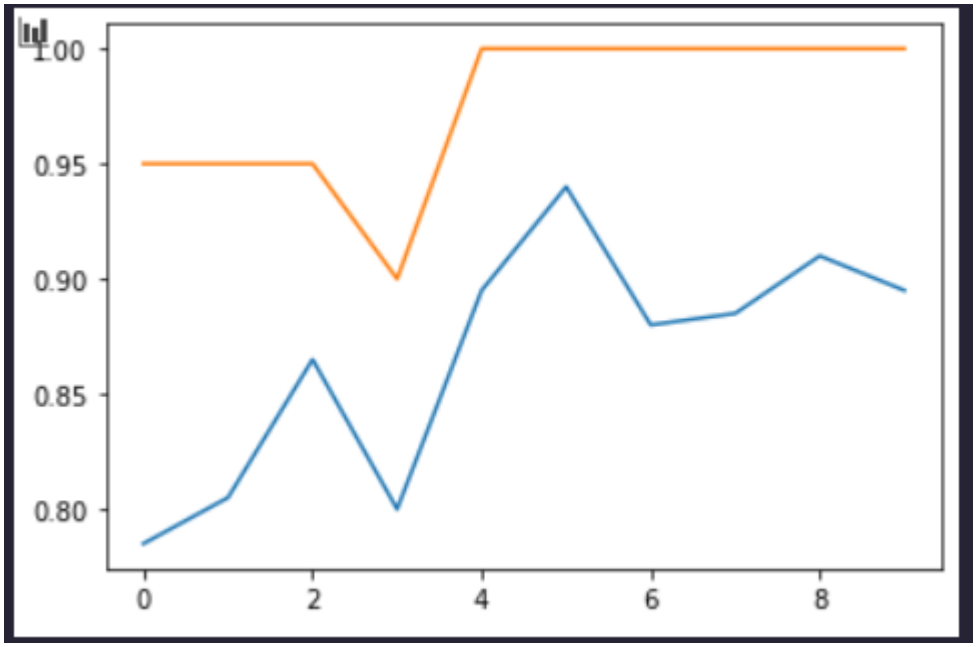
```python
# Find best layer for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [(2+(i*2)),(1+(i*2))]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

โดยจะนำค่า sum square error และ minimum ของ sum square error มาแสดงเป็นกราฟดังนี้



กราฟแสดงค่า min error และ sum square error ของไฟล์ flood.txt



กราฟแสดงค่า min error และ sum square error ของไฟล์ cross.pat

โดยจากการทดลองพบว่าจำนวนโหนดของ hidden layer ที่เหมาะสมในช่วงที่ทำการทดลองในไฟล์ flood.txt อยู่ในช่วง i = 1-4 เพราะมีค่า error ทั้งสองแบบที่น้อยที่สุดในกราฟโดยค่า i แปลงเป็น

จำนวนโหนดของ hidden layer ได้เป็น [4,3] – [10,9] และในการทดลองในไฟล์ cross.pat ได้ค่า i ที่ทำให้ค่า error ทั้งสองแบบน้อยที่สุดเท่ากับ 3 แปลงเป็นจำนวนโหนดของ hidden layer ได้เป็น [8,7]

2. การทดลองเปลี่ยนค่า learning rate

มีการตั้งค่าตามรายละเอียดดังนี้

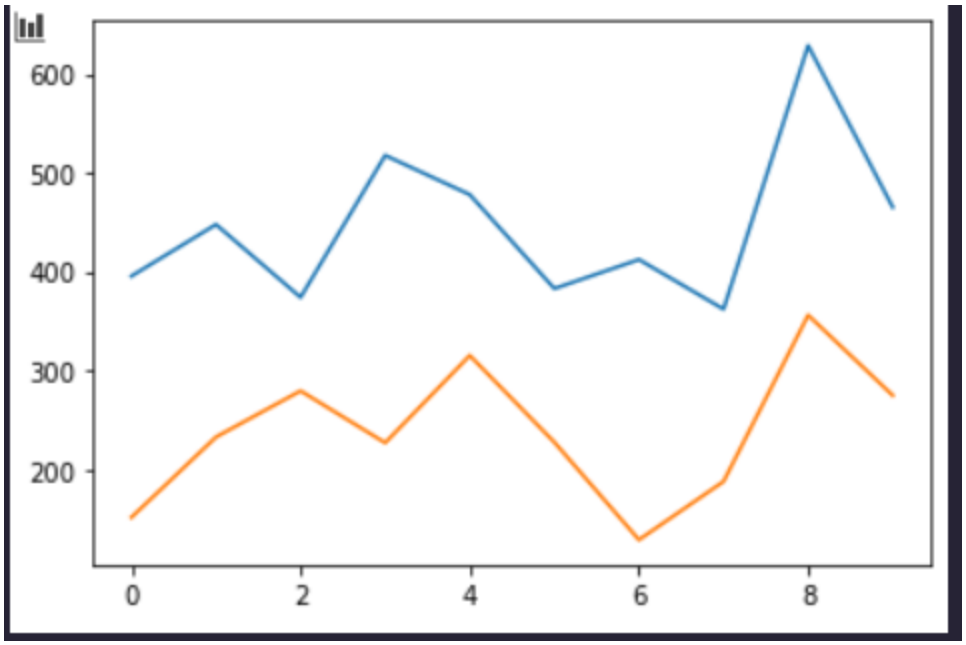
```
# Find best learning rate for Flood_dataset.txt
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = (0.01*i)+0.1
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

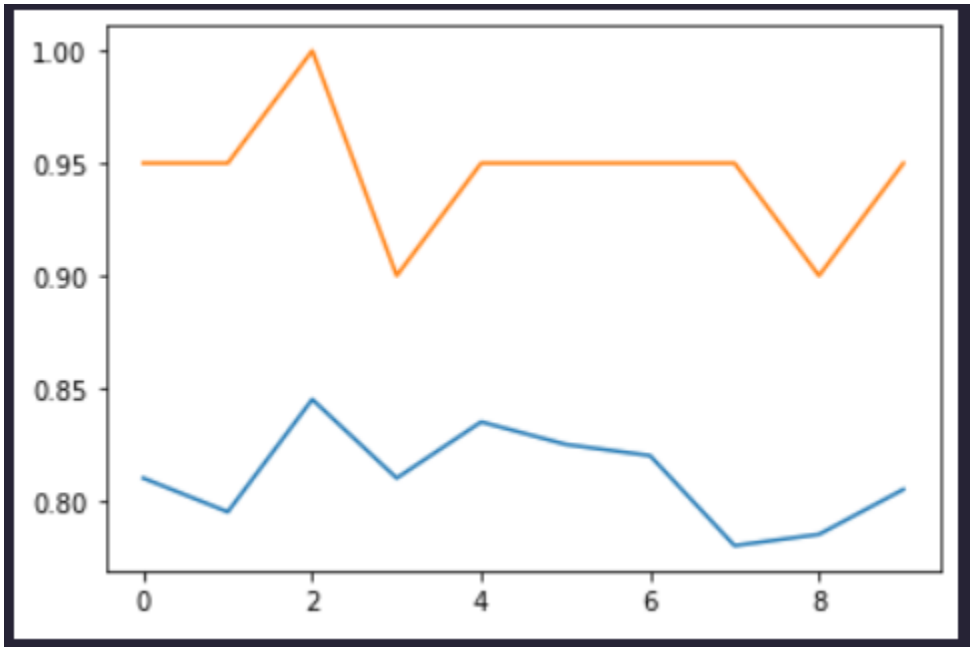
```
# Find best learning rate for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = (0.01*i)+0.1
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

โดยจะนำค่า sum square error และ minimum ของ sum square error มาแสดงเป็นกราฟดังนี้



กราฟแสดงค่า min error และ sum square error ของไฟล์ flood.txt



กราฟแสดงค่า min error และ sum square error ของไฟล์ cross.pat

โดยจากการทดลองพบว่าค่า learing rate ที่เหมาะสมในช่วงที่ทำการทดลองในไฟล์ flood.txt อยู่ในช่วง i = 6-7 แต่ค่า 6 จะมี sum square error เยอะกว่าค่า 7 ค่า i = 7 จึงดีที่สุดจากความคิดของ

ผม โดยค่า i แปลงเป็นค่า learing rate ได้เป็น 0.17 และในการทดลองในไฟล์ cross.pat ได้ค่า i ที่เหมาะสม อยู่ในช่วง i = 7-8 แต่ค่า 7 จะมี sum square error เยอะกว่าค่า 8 ค่า i = 8 จึงดีที่สุดจากความคิดของผม โดยค่า i แปลงเป็นค่า learing rate ได้เป็น 0.18

3. การทดลองเปลี่ยนค่า momentum rate

มีการตั้งค่าตามรายละเอียดดังนี้

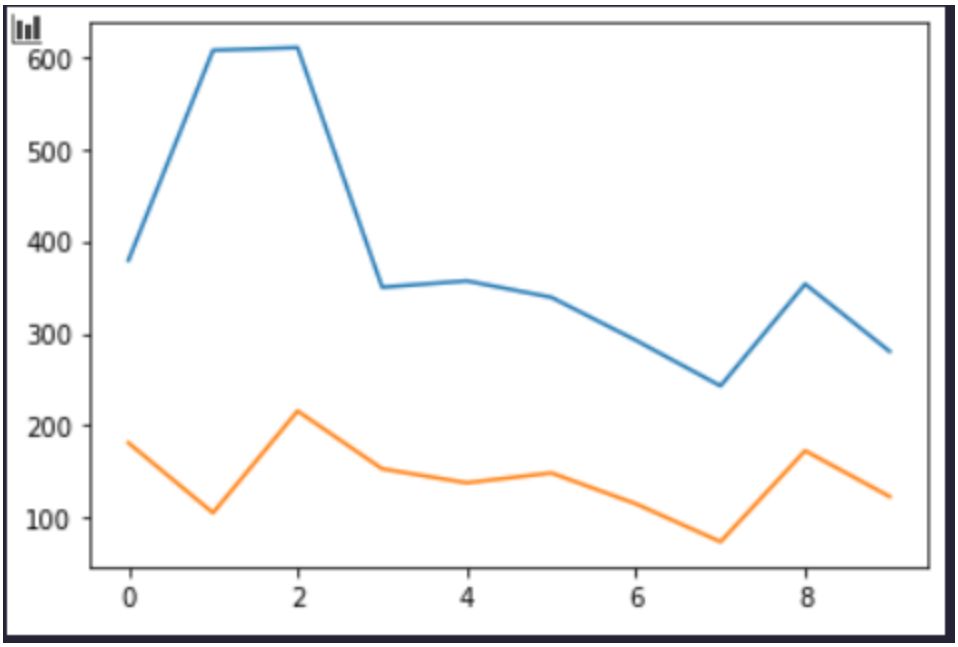
```
# Find best momentum rate for Flood_dataset.txt
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = (0.01*i)+0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

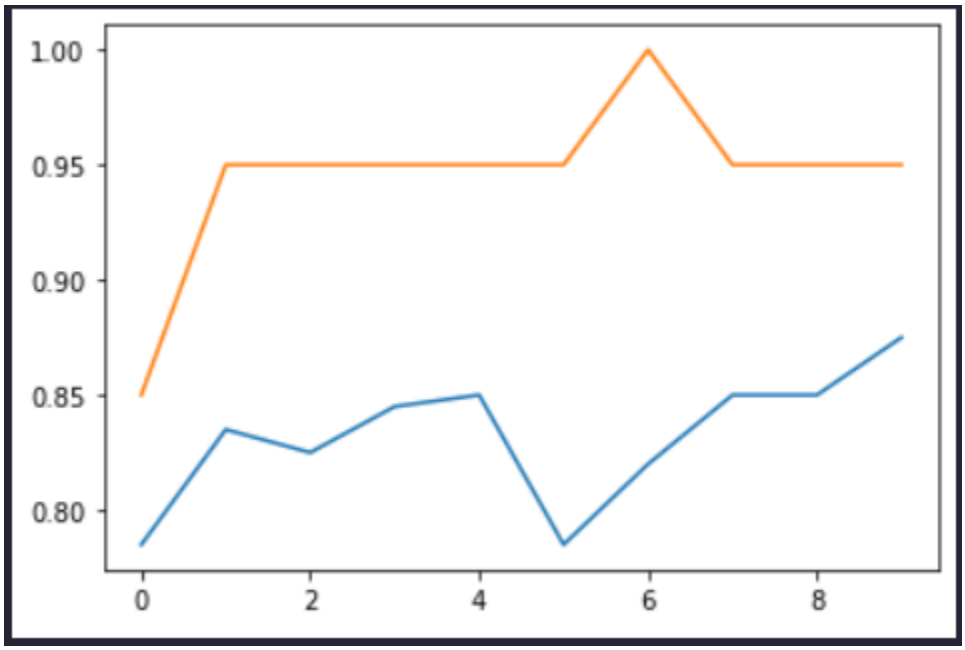
```
# Find best momentum rate for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = (0.01*i)+0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

โดยจะนำค่า sum square error และ minimum ของ sum square error มาแสดงเป็นกราฟดังนี้



กราฟแสดงค่า min error และ sum square error ของไฟล์ flood.txt



กราฟแสดงค่า min error และ sum square error ของไฟล์ cross.pat

โดยจากการทดลองพบว่าค่า momentum rate ที่เหมาะสมในช่วงที่ทำการทดลองในไฟล์ flood.txt คือค่า i = 7 เพราะได้ค่า error ทั้งสองแบบน้อยที่สุดในช่วงการทดลอง โดยค่า i แปลงเป็นค่า

momentum rate ได้เป็น 0.27 และในการทดลองในไฟล์ cross.pat ได้ค่า i ที่เหมาะสม เป็น i = 1 และ i = 5 แต่ค่า 5 จะมี sum square error เยอะกว่าค่า 1 ค่า i = 1 จึงดีที่สุดจากความคิดของผม โดยค่า i แปลงเป็นค่า learing rate ได้เป็น 0.21

ภาคผนวกโปรแกรม

In [ ]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
```

In [ ]:
```
def readfile(file):
    f = open(file, "r")
    if(file[-3:] == 'txt'):
        data = f.readlines()
        dataset = np.zeros((len(data)-2, len(data[2].split('\t'))-1))
        label = np.zeros((len(data)-2))

        for i in range(len(data)-2):
            x = data[i+2].split("\t")
            for j in range(len(x)):
                if j != len(x) - 1:
                    dataset[i][j] = float(x[j])
                else:
                    label[i] = float(x[j][:-1])
    else:
        data = f.readlines()
        n_data = int(len(data)/3)
        dataset = np.zeros((n_data, 2))
        label = np.zeros((n_data, 2))
        j = 0
        count = 0
        for i in range(len(data)):
            if(j == 1):
                dataset[count][0] = float(data[i].split()[0])
                dataset[count][1] = float(data[i].split()[1])
            if(j == 2):
                label[count][0] = int(data[i].split()[0])
                label[count][1] = int(data[i].split()[1])
                count = count + 1
                j = -1
            j += 1
    return dataset, label

def norm(data_r):
    print(data_r)
    data = data_r.copy()
    datanorm = (data - data.min())/(data.max() - data.min())
    return datanorm, data.max(), data.min()

def convert_norm(pred,mx,mn):
    return pred*(mx - mn) + mn

def prepare(dataset,label ,percent, epoch):
    data = dataset.copy()
    la = label.copy()
    train_data = []
    test_data = []
    data_with_label =[]
    for i in range(len(data)):
        dataTemp=[]
        dataTemp.append(data[i])
        dataTemp.append(la[i])
        data_with_label.append(dataTemp)
    random.shuffle(data_with_label)
    for i in range(len(data_with_label)):
        if((i%percent) == ((epoch-1)%percent)):
            test_data.append(data_with_label[i])
```

```
        else:
            train_data.append(data_with_label[i])
    return train_data,test_data

def setMathix(train_data,test_data):
    train_feature = []
    train_labels = []
    test_feature = []
    test_labels = []

    for i in range(len(train_data)):
        train_feature.append(train_data[i][0])
        train_labels.append(train_data[i][1])
    for j in range(len(test_data)):
        test_feature.append(test_data[j][0])
        test_labels.append(test_data[j][1])

    # train_labels_fix = np.zeros((len(train_labels), 1))
    # for i in range(len(train_labels)):
    #     train_labels_fix [i][0] = train_labels[i]

    train_feature_fix = np.zeros((len(train_feature), len(train_feature[0]) ))
    for i in range(len(train_feature)):
        for j in range(len(train_feature[0])):
            train_feature_fix [i][j] = train_feature[i][j]

    # test_labels_fix = np.zeros((len(test_labels), 1))
    # for i in range(len(test_labels)):
    #     test_labels_fix [i][0] = test_labels[i]

    test_feature_fix = np.zeros((len(test_feature), len(test_feature[0]) ))
    for i in range(len(test_feature)):
        for j in range(len(test_feature[0])):
            test_feature_fix [i][j] = test_feature[i][j]
    train_labels_fix = np.asarray(train_labels)
    test_labels_fix = np.asarray(test_labels)
    return train_feature_fix,train_labels_fix,test_feature_fix,test_labels_fix
```

In [ ]:

```
def load_data(name):
    is_norm = False
    if(name  == 1):
        dataset,label = readfile("./Flood_dataset.txt")
        dataset,mx_dataset,mn_dataset = norm(dataset)
        label,mx_label,mn_label = norm(label)
        is_norm = True
        max_min = [mx_dataset,mn_dataset,mx_label,mn_label]
    else :
        dataset,label = readfile("./cross.pat")
    n_sample = np.arange(len(dataset))
    np.random.shuffle(n_sample)
    if(is_norm) :
        return dataset,label,n_sample,max_min
    else :
        return dataset,label,n_sample

def dataset_with_crossvalidation_90(dataset,label):
    train_data,test_data = prepare(dataset, label, percent, epochs)
    train_feature,train_labels,test_feature,test_labels = setMathix(train_data,test_
    return train_feature,train_labels,test_feature,test_labels
```

In [ ]:

```
percent = 10
epochs = 500
file1 = "Flood_dataset.txt"
file2 = "cross.pat"
```

In [ ]:

```
class NN :

    def __init__(self,shape,nueral_shape,acti_funct):
        shape[1:1] = nueral_shape
        self.shape = shape
        self.act_func = acti_funct
        self.weights = self.init_weights(self.shape)
        self.outputs = None
        self.deltas = None
        self.del_old_weights = None

    def init_old_weights(self,network_shape):
        weight_arrays = []
        for i in range(0, len(network_shape) - 1):
            cur_idx = i
            next_idx = i + 1
            weight_array = np.zeros((network_shape[next_idx], network_shape[cur_idx]
            weight_arrays.append(weight_array)

        return weight_arrays

    def init_weights(self,network_shape):
        weight_arrays = []
        for i in range(0, len(network_shape) - 1):
            cur_idx = i
            next_idx = i + 1
            weight_array = 2*np.random.rand(network_shape[next_idx], network_shape[c
            weight_arrays.append(weight_array)

        return weight_arrays


    def predict(self,sample):

        current_input =  (sample.copy()).T
        outputs = []
        for network_weight in self.weights:
            current_output_temp = np.dot(network_weight, current_input)
            current_output = self.acti_funct(current_output_temp)
            outputs.append(current_output)
            current_input = current_output

        if(self.shape[-1] == 1) :
            return current_output.T
        else :
            tp = None
            fp = None
            for i in range(len(outputs[-1])):
                if( i == 0) :
                    tp = outputs[-1][i]
                else :
                    fp = outputs[-1][i]
                    tp = np.vstack((tp, fp)).T
            return np.argmax(tp, axis=1)

    def train(self,sample, d_out, training_rate,momentum_rate,epoch,show=True):
        # print(type(d_out))
        # print(np.shape(d_out))
        sample_T = (sample.copy()).T
        d_out_T = (d_out.copy()).T
        for i in range(epoch):
            self.FW_NN(sample_T)
            self.BW_NN(d_out_T)
            self.update_weights(sample_T,learning_rate,momentum_rate,i)
```

```
            sqe = self.sum_sqaure_error(self.predict(sample),d_out_T)
            if(show and i % 10 == 0):
                print('Epoch : #'+str(i)+',  Sum Square Error : '+str(sqe))
            if sqe < np.finfo(np.float32).eps :
                break

    def FW_NN(self,input):

        current_input = input
        outputs = []
        for w in self.weights:
            current_output_tmp = np.dot(w, current_input)
            current_output = self.acti_funct(current_output_tmp)
            outputs.append(current_output)
            current_input = current_output
        self.outputs = outputs

    def BW_NN(self,d_out):

        deltas = []
        O_error = d_out - self.outputs[len(self.outputs)-1]
        O_delta = O_error *self.derivertive_acti_funct(self.outputs[len(self.outputs
        deltas.append(O_delta)

        cur_delta = O_delta
        back_idx = len(self.outputs) - 2

        for w in self.weights[::-1][:-1]:
            hidd_error = np.dot(w.T, cur_delta)
            hidd_delta = hidd_error * self.derivertive_acti_funct(self.outputs[back_
            deltas.append(hidd_delta)
            cur_delta = hidd_delta
            back_idx -= 1

        self.deltas = deltas

    def update_weights(self,sample,learning_rate,momentum_rate,count):
        index_current_weight = len(self.weights) - 1
        current_dels = []
        for d in self.deltas:
            sample_used = None
            if index_current_weight - 1 < 0:
                sample_used = sample
            else:
                sample_used = self.outputs[index_current_weight - 1]

            current_delta = learning_rate*np.dot(d, sample_used.T)

            if(count == 0) :
                self.weights[index_current_weight] +=  current_delta
            else :
                self.weights[index_current_weight] +=  momentum_rate*self.del_old_we
            index_current_weight -= 1
            current_dels.insert(0, current_delta)

        self.del_old_weights = current_dels

    def acti_funct(self,v):
        if self.act_func == 'sigmoid' :
            return 1 / (1 + np.exp(-v))
        if self.act_func == 'tanh' :
            return np.tanh(v)
        if self.act_func == 'linear' :
            return v
        return v
```

```
    def derivertive_acti_funct(self,v):
        if self.act_func == 'sigmoid' :
            return v * (1 - v)
        if self.act_func == 'tanh' :
            return 1 - (v ** 2)
        if self.act_func == 'linear' :
            return 1
        return v


    def sum_sqaure_error(self,pred,real):
        real_m = real.copy()
        sums = 0
        if(real.ndim > 1) :
            tp = None
            fp = None
            for i in range(len(real_m)):
                if( i == 0) :
                    tp = real_m[i]
                else :
                    fp = real_m[i]
                    tp = np.vstack((tp, fp)).T
            real_m = np.argmax(tp, axis=1)
        for i in range(len(pred)):
            sums = sums + np.square(pred[i]-real_m[i])
        return sums/2

    def conf_matrix(self,pred,true,is_norm=False,confuse=True,Table=True):
        true_m = np.zeros(len(true))
        if(true.ndim > 1) :
            for i in range(len(true)):
                true_m[i] = np.argmax(true[i], axis=0)
        if(is_norm):
            sqr_error = 0
            if(Table):
                print('Desired Output\t\t|\tPredict\t\t\t|\tError')
                print('------------------------------------------------------------
            for i in range(len(true)):
                error = round(true[i] - round(pred[i][0],8),2)
                if(Table):
                    print(str(int(true[i]))+'\t\t\t\t|\t'+str(format(round(pred[i][0
                sqr_error = sqr_error + (error * error)
            if(Table):
                print('------------------------------------------------------------
                print('\t\t Sum Square Error = '+str(round(sqr_error/len(true),6)))
                print('======================================')
            return round(sqr_error/len(true),6)
        else :
            if(Table):
                print('Desired Output\t\t|\tPredict\t\t\t')
                print('------------------------------------------------')
                for i in range(len(true)):
                    print(str(int(true_m[i]))+'\t\t\t\t|\t'+str(pred[i]))
                print('------------------------------------------------')
        if(confuse):
            print('\n\t\t Confusion Matrix')
            TP = 0
            FN = 0
            FP = 0
            TN = 0
            for i in range(len(true)):
                if((pred[i] == 0) and ( true_m[i] == 0)):
                    TN = TN + 1
                elif((pred[i] == 1) and ( true_m[i] == 1)):
```

```
                TP = TP + 1
            elif((pred[i] == 1) and ( true_m[i] == 0)):
                FP = FP + 1
            else :
                FN = FN + 1

        print(' ---------------------------------------')
        for i in range(8):
            print('|\t\t\t|\t\t\t|')
            if(i == 1):
                print('|\t    '+str(TN)+'\t\t|\t    '+str(FP)+'\t\t|\t '+str(FP+
            if(i == 3):
                print(' ---------------------------------------')
            if(i == 5):
                print('|\t    '+str(FN)+'\t\t|\t    '+str(TP)+'\t\t|\t '+str(FN+
        print(' ---------------------------------------')
        print(' \t    '+str(TN+FN)+'\t\t       '+str(FP+TP)+'\t\t\t'+str(TN+FP+F
        print('')
        print('Accuracy : '+str((TN+TP)/(TN+FP+FN+TP)))
        return((TN+TP)/(TN+FP+FN+TP))
```

In [ ]:

```
def MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num) :
    if(data_num == 0):
        print('------------------- Variable -------------------\n')
        dataset,label,n_sample = load_data(data_num)
        data_name = 'cross.pat'
    else :
        print('------------------------------- Variable ----------------------------
        dataset,label,n_sample,max_min = load_data(data_num)
        data_name = 'Flood data set'
    print('Datafile : ' +str(data_name),end='\n')
    print('layer number: '+str(len(dataset[0]))+'-',end='')
    for i in range(len(layer)):
        print(str(layer[i])+'-',end='')
    print(label.ndim,end='\n')
    print('Learning rate : '+str(learning_rate),end='\n')
    print('Momentum rate : '+str(momentum_rate),end='\n')
    print('Activaion Function : ' +str(activation),end='\n')
    print('Cross validation : 90',end='\n')
    print('#Epoch : '+str(epoch),end='\n')
    error_avg = []
    acc_avg = []
    for i in range(10):
        # test_data = n_sample[i*n_test_per_round:i*n_test_per_round+n_test_per_roun
        # train_data = list(set(n_sample) - set(test_data))
        train_feature,train_labels,test_feature,test_labels = dataset_with_crossvali
        nn = NN([len(dataset[0]),label.ndim],layer,activation)
        nn.train(train_feature,train_labels,learning_rate,momentum_rate,epoch,False)
        pred = nn.predict(test_feature)
        if(data_num == 1):
#             print('\n------------------------------- Round : '+str(i)+' ---------
            pred = convert_norm(pred,max_min[2],max_min[3])
            test_label = convert_norm(test_labels,max_min[2],max_min[3])
            error_avg.append(nn.conf_matrix(pred,test_label,is_norm=True,confuse=Fal
        else :
#             print('\n----------------- Round : '+str(i)+' -----------------')
            acc_avg.append(nn.conf_matrix(pred,test_labels,is_norm=False,confuse=Tru
    if(data_num == 1):
        print('\n********  Sum Square Error Average : ' + str(round(np.sum(error_avg
        print(np.min(error_avg))
        return round(np.sum(error_avg)/len(error_avg),4),np.min(error_avg)
    else :
        print('\n********  Accuracy Average : ' + str(round(np.sum(acc_avg)/len(acc_
        return round(np.sum(acc_avg)/len(acc_avg),4),np.max(acc_avg)
```

In [ ]:
```
# Visualization Flood_dataset.txt
dataset, label = readfile("./Flood_dataset.txt")
plt.plot(label, label="Y")
plt.plot(dataset)
plt.show()
```

In [ ]:
```
# Visualization cross.pat
dataset, label = readfile("./cross.pat")

setAx = []
setAy = []
setBx = []
setBy = []

for i in range(len(label)):
    if label[i][0] == 1:
        setAx.append(dataset[i][0])
        setAy.append(dataset[i][1])
    if label[i][0] == 0:
        setBx.append(dataset[i][0])
        setBy.append(dataset[i][1])

plt.scatter(setAx, setAy, label='SetA')
plt.scatter(setBx, setBy, label='SetB')
```

In [ ]:
```
# Find best layer for Flood_dataset.txt
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [(2+(i*2)),(1+(i*2))]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]:
```
# Find best layer for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [(2+(i*2)),(1+(i*2))]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]:
```
# Find best learning rate for Flood_dataset.txt
error_arr = np.zeros(10)
```

```
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = (0.01*i)+0.1
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]:

```
# Find best learning rate for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = (0.01*i)+0.1
    momentum_rate = 0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]:

```
# Find best momentum rate for Flood_dataset.txt
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 1 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = (0.01*i)+0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]:

```
# Find best momentum rate for cross.pat
error_arr = np.zeros(10)
minn_arr = np.zeros(10)
for i in range(1,11):
    layer = [6,5]
    data_num = 0 # 0 = cross.pat , 1 = flood data set
    learning_rate = 0.15
    momentum_rate = (0.01*i)+0.2
    activation = 'sigmoid'
    epoch = 1000
    error,minn = MLP(layer,learning_rate,momentum_rate,activation,epoch,data_num)
    error_arr[i-1] = error
```

```
    minn_arr[i-1] = minn

plt.plot(error_arr, label="Error Avg")
plt.plot(minn_arr, label="Min Error")
plt.show()
```

In [ ]: