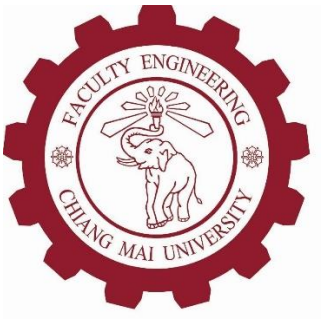


รายงานการทดลอง

Homework 1

261456 (Introduction to Computational Intelligence)

โดย

ปิยะนันท์ ปิยะวรรณ์โณ 650610845

เสนอ

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

บทคัดย่อ

รายงานฉบับนี้มุ่งเน้นการศึกษาโครงข่ายประสาทเทียม (Neural Network) ซึ่งเป็นสถาปัตยกรรมทางคอมพิวเตอร์ที่เลียนแบบการทำงานของสมองมนุษย์ โดยทำเป็น Multi-layer Perceptron โดยใช้วิธี back propagation และมีการประยุกต์ใช้โครงข่ายประสาทเทียมในด้าน การทำนาย(Regression) และ การจัดกลุ่ม(Classification) และมีการทดลองปรับ Hidden layers , learning และ Momentum rate เพื่อทดสอบผลกระทบที่เกิดขึ้น

การทดลองทำนายระดับน้ำที่สะพานนวรัตน์ และการทดลองการแบ่งกลุ่มของข้อมูลโดยใช้ MLP ทำ Back Propagation

วิธีการทดลอง

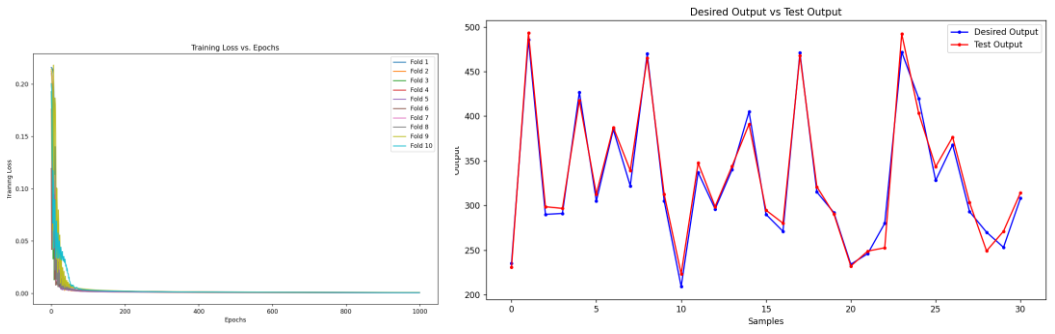
1. นำเข้าข้อมูลจากไฟล์ Flood_dataset.txt และ cross.txt
2. ทำการสุ่มค่า Weight
3. นำข้อมูลไปแบ่งเป็น 10 ส่วน ตาม 10% cross validation
4. ทำการ feed forward ข้อมูลแต่ละส่วน
5. ทำ back propagation โดยหาค่า error แล้วนำมาปรับ Weight และ Bias
6. ทำซ้ำจนกว่าจะครบ Epoch ที่กำหนดไว้ และครบทุกส่วนที่แบ่งไว้
7. นำข้อมูลมา Plot กราฟ
8. Polt Confusion matrix สำหรับการ Classification

ผลการทดลอง

1.การทำนาย (Regression)

1.1. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[5,10] , activation function = sigmoid , Epoch = 1000

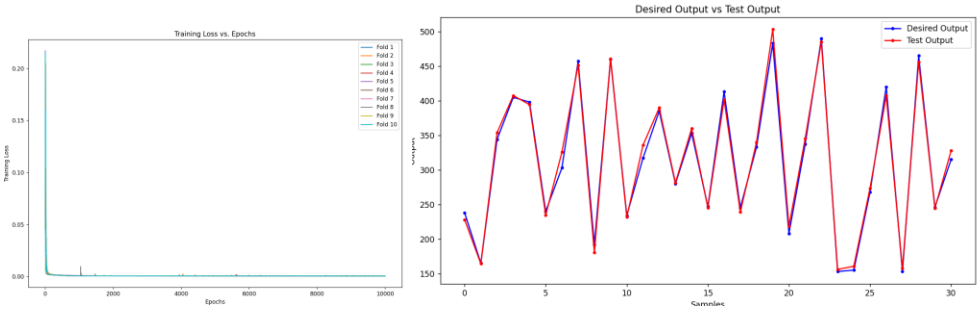
```
Regr = NN("Flood_dataset.txt", "R", [5, 10], "sigmoid", 1000, 0.035, 0.91)
```




```
Loss : 0.00045951673159685274
```

1.2. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[5,10] , activation function = sigmoid , Epoch = 10000

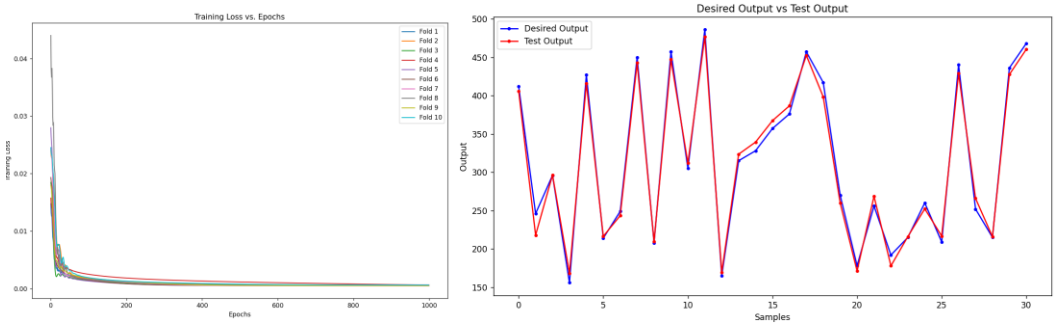
```
Regr = NN("Flood_dataset.txt", "R", [5, 10], "sigmoid", 10000, 0.035, 0.91)
```




```
Loss : 0.0001760732859934178
```

1.3. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[2] , activation function = sigmoid , Epoch = 1000

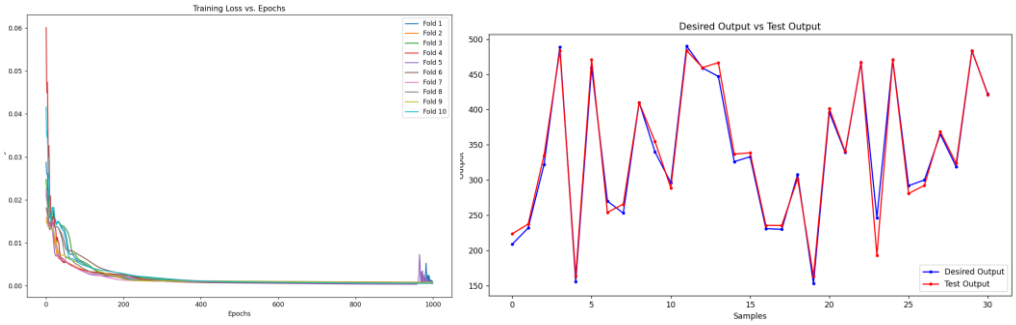
```
Regr = NN("Flood_dataset.txt", "R", [2], "sigmoid", 1000, 0.035, 0.91)
```




```
Loss : 0.0007272658489027741
```

1.4. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[5,150,1] , activation function = sigmoid , Epoch = 1000

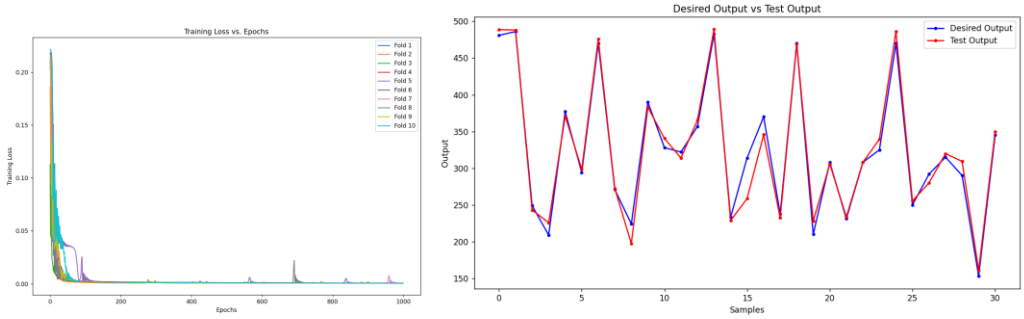
```
Regr = NN("Flood dataset.txt", "R", [5,150,1], "sigmoid", 1000, 0.035, 0.91)
```




```
Loss : 0.0013027228328670 96
```

1.5. กำหนด learning rate = 0.05, momentum rate =0.91 , hidden layer =[5,10] , activation function = sigmoid , Epoch = 1000

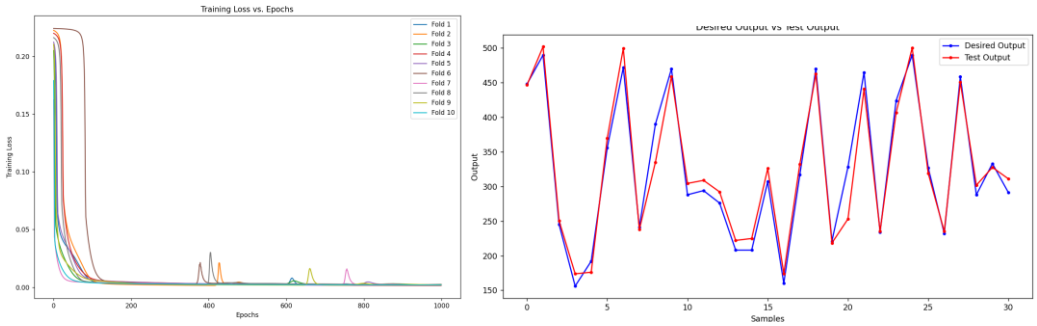
```
Regr = NN("Flood_dataset.txt", "R", [5,10], "sigmoid", 1000, 0.05, 0.91)
```




```
Loss : 0.0006380682138557155
```

1.6. กำหนด learning rate = 0.035, momentum rate =0.08 , hidden layer =[5,10] , activation function = sigmoid , Epoch = 1000

```
Regr = NN("Flood_dataset.txt", "R", [5,10], "sigmoid", 1000, 0.035, 0.08)
```

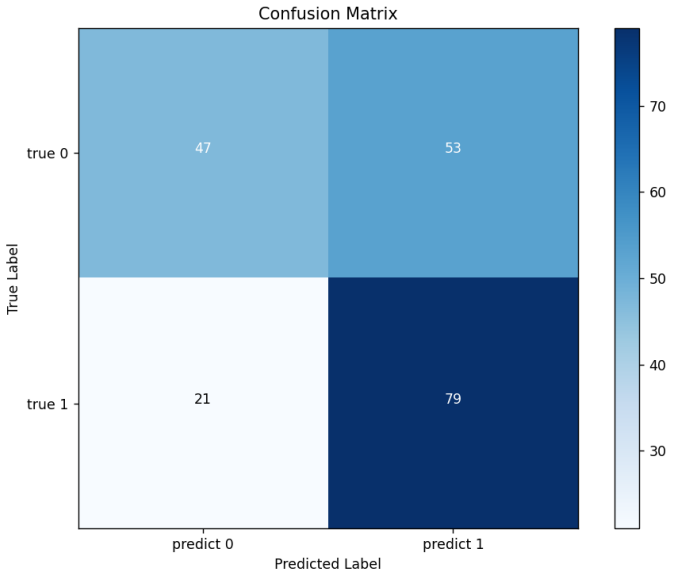



```
Loss : 0.0016258698780213154
```

2.การแบ่งกลุ่ม (Classification)

2.1. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[5,10] , activation function = sigmoid , Epoch = 100

```
Classi = NN("cross.txt", "C", [5,10], "sigmoid", 100, 0.035, 0.91)
```

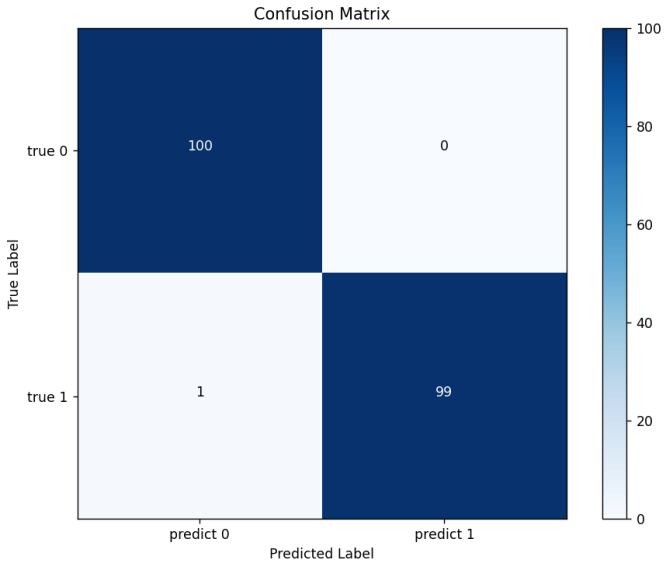


Accurate = 63%

2.2. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[5,10]

activation function = sigmoid , Epoch = 1000

```
Classi = NN("cross.txt", "C", [5,10], "sigmoid", 1000, 0.035, 0.91)
```

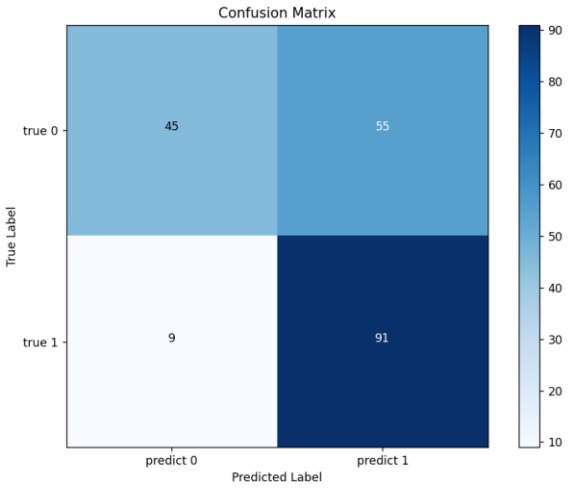


Accurate = 99.5%

2.3. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[1]

activation function = sigmoid , Epoch = 1000

```
Classi = NN("cross.txt", "C", [1], "sigmoid", 1000, 0.035, 0.91)
```

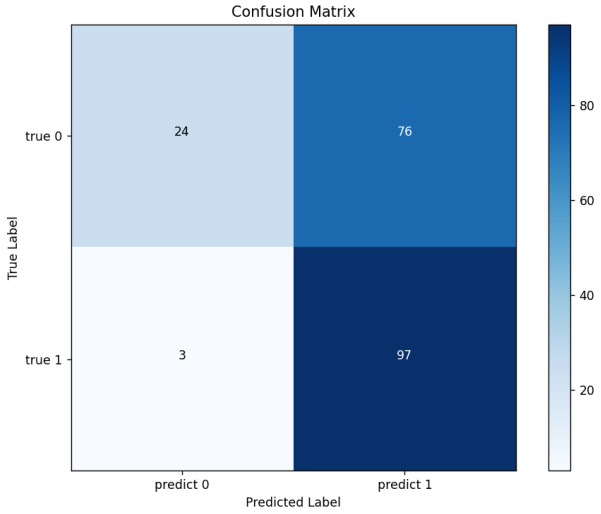


Accurate = 68%

2.4. กำหนด learning rate = 0.035, momentum rate =0.91 , hidden layer =[10,10,30]

activation function = sigmoid , Epoch = 1000

```
Classi = NN("cross.txt", "C", [10,10,30], "sigmoid", 1000, 0.035, 0.91)
```

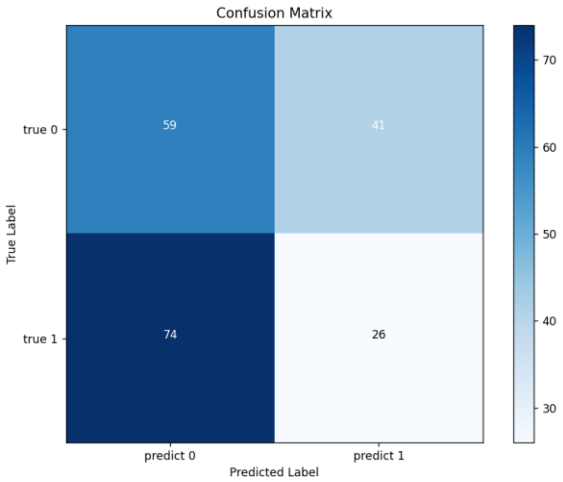


Accurate = 60.5%

2.5. กำหนด learning rate = 0.12, momentum rate =0.91 , hidden layer =[5,10]

activation function = sigmoid , Epoch = 1000

```
Classi = NN("cross.txt", "C", [5,10], "sigmoid", 1000, 0.12, 0.91)
```

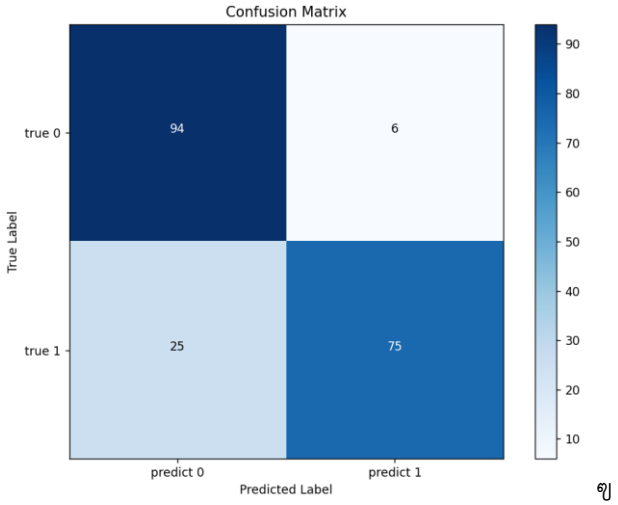


Accurate = 42.5%

2.6. กำหนด learning rate = 0.035, momentum rate =0.015 , hidden layer =[5,10]

activation function = sigmoid , Epoch = 1000

```
Classi = NN("cross.txt", "C", [5,10], "sigmoid", 1000, 0.035, 0.015)
```



Accurate = 84.5%

วิเคราะห์ผลการทดลอง

จากการทดลอง เปลี่ยนค่า Learning rate , Momentum rate , hidden layers และ จำนวน Epoch จะเห็นได้ว่า การเพิ่มจำนวน Epoch จะทำให้ ค่า loss น้อยลง , จำนวนของ hidden layers ส่งผลต่อค่า loss ถ้ามี layers มากเกินไป หรือ น้อยเกินไป จะทำให้ค่า loss เพิ่มขึ้น , ค่า Learning rate และ Momentum rate จะส่งผลต่อค่า loss เช่นกัน ทั้งนี้การปรับเปลี่ยนค่า learning rate , Momentum rate , hidden layers จะแตกต่างกันไปในแต่ละการใช้งาน ไม่สามารถบอกได้ว่าค่าไหนดีที่สุด

สำหรับการทำนาย ยิ่งมีค่า loss น้อยจะทำให้การทำนายใกล้เคียงกับค่าจริงมากขึ้น และ สำหรับการจัดกลุ่ม จะทำให้ มีความแม่นยำสูงขึ้น

ภาคผนวก

https://github.com/SPHSTR/Computer_Intelligence_Flood_Predict/tree/main

```python
import numpy as np
import matplotlib.pyplot as plt

class NN:
    def __init__(self, filename, filetype, hidden_layer_size, function, epoch, learning_rate, momentum_rate):
        self.filename = filename
        self.hidden_layer_size = hidden_layer_size
        self.function = function
        self.epoch = epoch
        self.learning_rate = learning_rate
        self.momentum_rate = momentum_rate
        self.filetype = filetype

    def Activate_Function(self, x):
        if self.function == "sigmoid":
            return 1 / (1 + np.exp(-x))
        if self.function == "relu":
            return np.where(x > 0, x, 0.0)
        if self.function == "tanh":
            return np.tanh(x)

    def Diff_Activate_Function(self, x):
        if self.function == "sigmoid":
            return x * (1 - x)
        if self.function == "relu":
            return np.where(x > 0, 1.0, 0.0)
        if self.function == "tanh":
            return 1 - x**2
```

```
    def Load_data(self):
        if self.filetype == "R":
            file = np.loadtxt(self.filename, dtype=float)
            self.input = file[:, :-1]
            self.desireoutput = file[:, -1].reshape(-1, 1)

            self.filemax = file.max()
            self.filemin = file.min()

        elif self.filetype == "C":
            self.input = []
            self.desireoutput = []

            with open(self.filename, 'r') as f:
                lines = f.readlines()
                for i in range(0, len(lines), 3):
                    coordinates = [float(x) for x in lines[i+1].split()]
                    self.input.append(coordinates)

                    labels = [int(x) for x in lines[i+2].split()]
                    self.desireoutput.append(labels)
```

```
            self.input = np.array(self.input)
            self.desireoutput = np.array(self.desireoutput)

            self.filemax = np.max(self.input)
            self.filemin = np.min(self.input)

        else:
            raise Exception("Invalid filetype")

```

```
    def Random_WeightandBias(self):
        self.actual_layer = [self.input.shape[1]] + self.hidden_layer_size + [self.desireoutput.shape[1]]
        self.weight = []
        self.bias = []
        for i in range(len(self.actual_layer) - 1):
            self.weight.append(np.random.randn(self.actual_layer[i], self.actual_layer[i+1]) - 1)
            self.bias.append(np.zeros((1, self.actual_layer[i+1])))

        self.veloc_weight = []
        self.veloc_bias = []
        for i in range(len(self.actual_layer) - 1):
            self.veloc_weight.append(np.zeros_like(self.weight[i]))
            self.veloc_bias.append(np.zeros_like(self.bias[i]))

```

```
    def Normallization(self):
        self.input = (self.input - self.filemin) / (self.filemax - self.filemin)
        self.desireoutput = (self.desireoutput - self.filemin) / (self.filemax - self.filemin)

```

```
    def DeNormallization(self):
        self.input = (self.input * (self.filemax - self.filemin)) + self.filemin
        self.desireoutput = (self.desireoutput * (self.filemax - self.filemin)) + self.filemin

```

```
    def Value_DeNormallization(self, x):
        return (x * (self.filemax - self.filemin)) + self.filemin

```

```
    def Feed_Forward(self):
        self.layer_output = [self.input]
        for i in range(len(self.weight)):
            net_input = np.dot(self.layer_output[-1], self.weight[i]) + self.bias[i]
            activation_output = self.Activate_Function(net_input)
            self.layer_output.append(activation_output)

    def Back_Propagation(self):
        self.errors = [self.desireoutput - self.layer_output[-1]]
        deltas = [self.errors[-1] * self.Diff_Activate_Function(self.layer_output[-1])]

        for i in range(len(self.weight) - 1, 0, -1):
            error = np.dot(deltas[-1], self.weight[i].T)
            delta = error * self.Diff_Activate_Function(self.layer_output[i])
            self.errors.append(error)
            deltas.append(delta)

        deltas.reverse()

        for i in range(len(self.weight)):
            self.veloc_weight[i] = self.momentum_rate * self.veloc_weight[i] + np.dot(self.layer_output[i].T, deltas[i])
            self.veloc_bias[i] = self.momentum_rate * self.veloc_bias[i] + np.sum(deltas[i], axis=0, keepdims=True)
            self.weight[i] += self.learning_rate * self.veloc_weight[i]
            self.bias[i] += self.learning_rate * self.veloc_bias[i]

    def Train(self):
        self.training_loss = []
        for i in range(self.epoch):
            self.Feed_Forward()
            self.Back_Propagation()
            loss = np.mean(np.square(self.desireoutput - self.layer_output[-1]))
            self.training_loss.append(loss)
            # print("Epoch:", i + 1, " Loss:", loss)
        self.DeNormallization()

    def Test(self, t):
        if self.filetype == "R":
            x = (t - self.filemin) / (self.filemax - self.filemin)
            for i in range(len(self.weight)):
                x = self.Activate_Function(np.dot(x, self.weight[i]) + self.bias[i])
            output = self.Value_DeNormallization(x)
        elif self.filetype == "C":
            t = np.array(t)
            x = t
            for i in range(len(self.weight)):
                x = self.Activate_Function(np.dot(x, self.weight[i]) + self.bias[i])
            output = x
        # print("Test input =", t, "Output =", output)
        return output
```

```
    def K_Fold_Cross_Validation(self, k):
        self.Load_data()
        self.Normallization()
        data = np.hstack((self.input, self.desireoutput))
        np.random.shuffle(data)
        folds = np.array_split(data, k)
        display = []

        plt.figure(figsize=(12, 8))

        if self.filetype == "R" :
            for fold in range(k):
                print(f"Fold {fold + 1}/{k}")

                train_data = np.vstack([folds[i] for i in range(k) if i != fold])
                test_data = folds[fold]

                self.input = train_data[:, :-1]
                self.desireoutput = train_data[:, -1].reshape(-1, 1)

                self.Random_WeightandBias()

                self.Train()

                plt.plot(self.training_loss, label=f'Fold {fold + 1}')

                test_input = test_data[:, :-1]
                test_output = test_data[:, -1].reshape(-1, 1)
                self.input = test_input
                self.desireoutput = test_output
                self.Feed_Forward()
                test_loss = np.mean(np.square(self.desireoutput - self.layer_output[-1]))
                print("Test Loss:", test_loss)
                display.append(test_loss)
                print()
        elif self.filetype == "C":
            for i in range(k):
                self.Random_WeightandBias()

                self.Train()

                self.Feed_Forward()
                test_loss = np.mean(np.square(self.desireoutput - self.layer_output[-1]))
                print("Test Loss:", test_loss)
                display.append(test_loss)
                print()

                test_loss = np.mean(np.square(self.desireoutput - self.layer_output[-1]))
                print("Test Loss:", test_loss)
                print()


        if self.filetype == "R":
            plt.title('Training Loss vs. Epochs')
            plt.xlabel('Epochs')
            plt.ylabel('Training Loss')
            plt.legend()
            plt.show()

        for i in range(k):
            print("Fold", i + 1, " Loss :", display[i])

        self.DeNormallization()
```

```
    def Test_All(self, x):
        testall_output = []
        true_labels = np.argmax(self.desireoutput, axis=1)

        for i in range(len(x)):
            output = self.Test(x[i])
            testall_output.append(output)

        testall_output = np.array(testall_output).reshape(-1, self.desireoutput.shape[1])

        predicted_labels = np.argmax(testall_output, axis=1)

        if self.filetype == "R":
            plt.figure(figsize=(12, 6))

            for i in range(self.desireoutput.shape[1]):
                plt.plot(range(len(self.desireoutput)), self.desireoutput[:, i], label=f"Desired Output {i+1}", marker='.')
                plt.plot(range(len(testall_output)), testall_output[:, i], label=f"Test Output {i+1}", marker='.')


            plt.title('Desired Output vs Test Output')
            plt.xlabel('Samples')
            plt.ylabel('Output')
            plt.legend()
            plt.show()

        if self.filetype == "C" :
            self.plot_confusion_matrix(true_labels, predicted_labels)
```

```
    def plot_confusion_matrix(self, true_labels, predicted_labels):
        # Initialize a 2x2 confusion matrix
        cm = np.zeros((2, 2), dtype=int)

        for t, p in zip(true_labels, predicted_labels):
            cm[t, p] += 1

        plt.figure(figsize=(8, 6))
        plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
        plt.title('Confusion Matrix')
        plt.colorbar()

        tick_marks = np.arange(2)
        plt.xticks(tick_marks, ['predict 0', 'predict 1'])
        plt.yticks(tick_marks, ['true 0', 'true 1'])

        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
```

```
        for i in range(2):
            for j in range(2):
                plt.text(j, i, format(cm[i, j], 'd'),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > cm.max() / 2. else "black")

        plt.tight_layout()
        plt.show()


if __name__ == '__main__':
    Regr = NN("Flood_dataset.txt", "R", [5,10], "sigmoid", 1000, 0.035, 0.08)
    Regr.K_Fold_Cross_Validation(10)
    Regr.Test_All(Regr.input)

    Classi = NN("cross.txt", "C", [5,10], "sigmoid", 1000, 0.035, 0.015)
    Classi.K_Fold_Cross_Validation(10)
    Classi.Test_All(Classi.input)
```