

**รายงานการทดลอง**

ผลกระทบของการเปลี่ยนแปลงตัวแปรในโมเดล ที่มีผลต่อ Neural Network

**โดย**

นายณปพน วงค์คม

650610834

**เสนอ**

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261456

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

**บทคัดย่อ**

Neural Network หรือเครือข่ายประสาทเทียมเป็นหนึ่งในเครื่องมือที่สำคัญและมีประสิทธิภาพสูงในการสร้างปัญญาประดิษฐ์ (AI) โดยเฉพาะในการจำแนกและทำนายข้อมูล อย่างไรก็ตาม ประสิทธิภาพของ Neural Network ขึ้นอยู่กับการตั้งค่าตัวแปรต่างๆ ในโมเดล เช่น จำนวนชั้น (layers) จำนวนโหนดในแต่ละชั้น (nodes) อัตราการเรียนรู้ (learning rate) และอัตราโมเมนตัม (momentum rate) เป็นต้น

รายงานนี้มุ่งเน้นศึกษาและวิเคราะห์ผลกระทบของการเปลี่ยนแปลงตัวแปรเหล่านี้ต่อการทำงานและผลลัพธ์ของ Neural Network มีการดำเนินงานโดยแบ่งออกเป็น 2 แบบ คือ regression และ classification แล้วทำการจำลองโมเดล จากนั้นสุ่มค่าตัวแปรต่างๆ เช่น learning rate, momentum rate และ hidden layer ที่มีผลต่อ neural network มาเปรียบเทียบกัน โดยการบันทึกผลการทดลองต่างๆ จากการปรับเปลี่ยนค่าตัวแปรข้างต้น การทดลองและการวิเคราะห์ในรายงานนี้แสดงให้เห็นว่าการเปลี่ยนแปลงตัวแปรต่างๆ มีผลกระทบต่อ validity ความแม่นยำ (accuracy) และความเร็วในการ converge การปรับตัวแปรเหล่านี้ให้เหมาะสมจึงมีความสำคัญอย่างยิ่งในการปรับปรุงและเพิ่มประสิทธิภาพของระบบ Neural Network ในอนาคต

**บทนำ**

**ที่มาและความสำคัญ**

ในยุคที่เทคโนโลยีสารสนเทศและการประมวลผลข้อมูลเจริญก้าวหน้าอย่างรวดเร็ว การใช้ปัญญาประดิษฐ์ (Artificial Intelligence หรือ AI) ได้เข้ามามีบทบาทสำคัญในหลายๆ ด้านของชีวิตประจำวัน หนึ่งในเทคนิคที่มีประสิทธิภาพและได้รับความนิยมมากที่สุดในการสร้าง AI คือ Neural Network หรือเครือข่ายประสาทเทียม

Neural Network เป็นระบบการเรียนรู้ของเครื่องที่ได้แรงบันดาลใจจากโครงสร้างและการทำงานของสมองมนุษย์ ซึ่งมีความสามารถในการจำแนกและทำนายข้อมูลได้อย่างแม่นยำ อย่างไรก็ตาม ประสิทธิภาพของ Neural Network ขึ้นอยู่กับหลายปัจจัย หนึ่งในปัจจัยสำคัญคือการตั้งค่าตัวแปรต่างๆ ในโมเดล เช่น จำนวนชั้น (layers) จำนวนโหนดในแต่ละชั้น (nodes) อัตราการเรียนรู้ (learning rate) อัตราโมเมนตัม (momentum rate) และการเลือกใช้ฟังก์ชัน activation เป็นต้น

การเปลี่ยนแปลงตัวแปรเหล่านี้มีผลกระทบต่อการทำงานและผลลัพธ์ของ Neural Network เป็นอย่างมาก การศึกษาผลกระทบของการเปลี่ยนแปลงตัวแปรในโมเดลจึงมีความสำคัญ เพื่อให้สามารถปรับปรุงและเพิ่มประสิทธิภาพของระบบได้อย่างเหมาะสม

**วัตถุประสงค์**

เพื่อศึกษาการทำงานของ neural network และ ทดสอบ validity, ความเร็วในการ converge และความถูกต้อง accuracy ของ neural network และเพื่อทดสอบผลกระทบที่เกิดขึ้นกับ neural network จากการเปลี่ยนแปลงของ hidden layer, learning rate และ momentum rate ทั้งแบบ regression และ classification

**ขอบเขตการทดลอง**

1.ข้อมูลระดับน้ำที่สถานี 1 และ สถานี 2 ที่เวลาปัจจุบัน และย้อนหลังไป 3 ชั่วโมง โดยมีดังนี้ สถานี 1 เวลา *t* -3, สถานี 1 เวลา *t* -2, สถานี 1 เวลา *t* -1, สถานี 1 เวลา *t* -0, สถานี 2 เวลา *t* -3, สถานี 2 เวลา *t* -2, สถานี 2 เวลา *t* -1, สถานี 2 เวลา *t* -0

2.ข้อมูลสำหรับการแบ่งแยกคลาส เช่นมี input เป็น 0.0902, 0.2690 และมี output เป็น 1, 0

**วิธีดำเนินการ**

1.เขียนโปรแกรมเพื่อสร้างโมเดล neural network สำหรับทดสอบ validity, ความเร็วในการ converge และความถูกต้อง accuracy

2.กำหนดค่า hidden layer, learning rate และ momentum rate ที่เหมาสม โดยแบ่งการทดลองออกเป็น 2 การทดลองคือ regression และ classification

3.นำข้อมูลที่มีไปทำการ k-fold validation เพื่อฝึกฝนโมเดล neural network

4.นำข้อมูลที่มีไปทำการ k-fold validation เพื่อนำข้อมูลมาทดสอบกับโมเดล neural network

5.บันทึกผลการทดสอบใรรูปของกราฟ

6.ทำตามข้อ 2. ถึง 5. ซ้ำเรื่อยๆ โดยเปลี่ยนแปลงค่า hidden layer, learning rate และ momentum rate จนได้ผลการทดลองที่เพียงพอ

**ผลการทดลอง**

1.ผลการทดลอง regression ที่แปรตาม learning rate โดยกำหนดให้ momentum rate คือ 0.00 hidden layer คือ 1 layer 16 node จำนวน epoch คือ 10000 epoch

A graph of data and a graph of data

Description automatically generated with medium confidenceA graph of data and a graph of data

Description automatically generated ที่ learning rate 1.0 ที่ learning rate 0.1

A graph of data and a graph of data

Description automatically generated with medium confidenceA graph of data and a line graph

Description automatically generated ที่ learning rate 0.01 ที่ learning rate 0.001

จากการทดลอง เมื่อเปลี่ยนแปลงค่า learning rate เป็นค่าต่างๆ โดยให้ momentum rate, hidden node และจำนวน epoch คงเดิม จะได้ว่ายิ่งค่า learning rate มากขึ้น จะทำให้ใช้จำนวน epoch ในการ converge น้อยลง แต่ถ้าหาก learning rate มีค่าน้อยไปหรือมากไปจะทำให้ neural network ทำงานได้ไม่ตรงตามที่ต้องการ โดยหาก learning rate มีค่ามากเกินไปอาจทำให้การ converge มีการแปรปรวนได้ หาก learning rate มีค่าน้อยเกินไปอาจทำให้ จำเป็นต้องเพิ่มปริมาณ epoch เพื่อให้ ข้อมูลแม่นยำมากขึ้น

A graph of data and a graph of data

Description automatically generated2.ผลการทดลอง regression ที่แปรตาม momentum rate โดยกำหนดให้ learning rate คือ 1.0 hidden layer คือ 1 layer 16 node จำนวน epoch คือ 10000 epoch

A graph of data and a line graph

Description automatically generated with medium confidence ที่ momentum rate 1.0 ที่ momentum rate 0.1

A graph of data and data

Description automatically generated with medium confidence

A graph of data and a graph of data

Description automatically generated ที่ momentum rate 0.01 ที่ momentum rate 0.001

จากการทดลอง เมื่อเปลี่ยนแปลงค่า momentum rate เป็นค่าต่างๆ โดยให้ learning rate, hidden node และจำนวน epoch คงเดิม จะได้ว่ายิ่งค่า momentum rate มาก ความเร็วในการ converge จะมาก หาก momentum rate มีค่ามากเกินไป อาจส่งผลให้ neural network ทำงานได้ไม่แม่นยำ แต่ถ้า momentum rate มีค่าน้อยเกินไป จะทำให้เกิดความไม่คงที่ในการ converge ของ neural network

3.ผลการทดลอง regression ที่แปรตาม hidden node โดยกำหนดให้ learning rate คือ 1.0 momentum rate คือ 0.00 จำนวน epoch คือ 10000 epoch

A graph of data and a line graph

Description automatically generated with medium confidenceA graph of a graph with numbers and lines

Description automatically generated with medium confidence เมื่อ hidden node คือ 64 เมื่อ hidden node คือ 16

A graph of data and a line graph

Description automatically generated with medium confidenceA graph of data and a graph of data

Description automatically generated เมื่อ hidden node คือ 8 เมื่อ hidden node คือ 2

จากการทดลอง เมื่อเปลี่ยนแปลง hidden node เป็นค่าต่างๆ โดยให้ learning rate, momentum rate และจำนวน epoch คงเดิม จะได้ว่ายิ่งจำนวน hidden node มาก ความแม่นยำของ neural network จะมากขึ้นด้วย หาก hidden node มีมากเกินไปจะทำให้การ converge ของ neural network ไม่คงที่ ถ้าหาก hidden node มีน้อยไป อาจทำให้ข้อมูลไม่แม่นยำในบางจุด

4.ผลการทดลอง classification ที่แปรตาม learning rate โดยกำหนดให้ momentum rate คือ 0.00 hidden layer คือ 1 layer 16 node จำนวน epoch คือ 10000 epoch

A blue squares with red numbers

Description automatically generatedA graph of a function

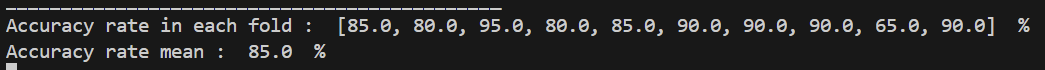
Description automatically generatedที่ learning rate 1.0



A diagram of confusion matrix

Description automatically generatedA graph with numbers and a line

Description automatically generatedที่ learning rate 0.1



A diagram of a confusion matrix

Description automatically generatedA graph with numbers and a line

Description automatically generatedที่ learning rate 0.01



จากการทดลอง เมื่อเปลี่ยนแปลงค่า learning rate เป็นค่าต่างๆ โดยให้ momentum rate, hidden node และจำนวน epoch คงเดิม จะได้ว่ายิ่งค่า learning rate มากขึ้น จะทำให้ใช้จำนวน epoch ในการ converge น้อยลง แต่ถ้าหาก learning rate มีค่าน้อยไปจะทำให้ neural network ทำงานได้ไม่ตรงตามที่ต้องการ โดยถ้า learning rate มีค่าน้อยเกินไปอาจทำให้ จำเป็นต้องเพิ่มปริมาณ epoch เพื่อให้ ข้อมูลแม่นยำมากขึ้น

5.ผลการทดลอง classification ที่แปรตาม momentum rate โดยกำหนดให้ learning rate คือ 1.0 hidden layer คือ 1 layer 16 node จำนวน epoch คือ 10000 epoch

A blue squares with red numbers

Description automatically generatedA graph with numbers and a line

Description automatically generated with medium confidenceที่ momentum rate 1.0



A diagram of a confusion matrix

Description automatically generatedA graph of a function

Description automatically generatedที่ momentum rate 0.1



A diagram of a confusion matrix

Description automatically generatedA graph of a function

Description automatically generatedที่ momentum rate 0.0001



จากการทดลอง เมื่อเปลี่ยนแปลงค่า momentum rate เป็นค่าต่างๆ โดยให้ learning rate, hidden node และจำนวน epoch คงเดิม จะได้ว่ายิ่งค่า momentum rate มาก ความเร็วในการ converge จะมาก หาก momentum rate มีค่ามากเกินไป อาจส่งผลให้ neural network ทำงานได้ไม่แม่นยำ แต่ถ้า momentum rate มีค่าน้อยเกินไป จะทำให้เกิดความไม่คงที่ในการ converge ของ neural network

6.ผลการทดลอง classification ที่แปรตาม hidden node โดยกำหนดให้ learning rate คือ 1.0 momentum rate คือ 0.00 จำนวน epoch คือ 10000 epoch

A diagram of a confusion matrix

Description automatically generatedA graph with numbers and a line

Description automatically generated with medium confidenceเมื่อ hidden node คือ 64



A blue squares with red dots

Description automatically generatedA graph of a function

Description automatically generatedเมื่อ hidden node คือ 16



A diagram of a confusion matrix

Description automatically generatedA graph with numbers and a line

Description automatically generated with medium confidenceเมื่อ hidden node คือ 2



จากการทดลอง เมื่อเปลี่ยนแปลง hidden node เป็นค่าต่างๆ โดยให้ learning rate, momentum rate และจำนวน epoch คงเดิม จะได้ว่ายิ่งจำนวน hidden node มาก ความแม่นยำของ neural network จะมากขึ้นด้วย หาก hidden node มีมากเกินไปจะทำให้การ converge ของ neural network ไม่คงที่ ถ้าหาก hidden node มีน้อยไป อาจทำให้ข้อมูลไม่แม่นยำ

**วิเคราะห์ผลการทดลอง**

จากผลการทดลอง เพื่อศึกษา validity, ความเร็วในการ converge และความถูกต้อง Accuracy ของ neural network ทำให้ทราบว่า จำนวน hidden layer ค่า learning rate ค่า momentum rate ต่างมีผลต่อ validity, ความเร็วในการ converge และความถูกต้อง Accuracy ของ neural network โดยหากตัวแปรต่างๆ มีค่ามากเกินไปหรือน้อยเกินไป จะส่งผลให้ neural network ไม่ได้ประสิทธิภาพ มีความแม่นยำต่ำ ความเร็วในการ converge ช้า ดังนั้นการจะให้ neural network มีประสิทธิภาพ มีความแม่นยำสูง มีความเร็วในการ converge ที่เร็ว จะต้องมีค่าของตัวแปรต่างๆที่เหมาะสม ไม่มากเกินไป ไม่น้อยเกินไป

**ภาคผนวก**

Github : [Introduction-to-Computational-Intelligence/Assignment1/Computer Assignment1 650610834.py at main · napapon-wongkom/Introduction-to-Computational-Intelligence (github.com)](https://github.com/napapon-wongkom/Introduction-to-Computational-Intelligence/blob/main/Assignment1/Computer%20Assignment1%20650610834.py#L60)

#Coding by Napapon Wongkom 650610834

import numpy as np

from matplotlib import pyplot as plt

class NeuralNetwork:

    def \_\_init\_\_(self, layer\_sizes, learning\_rate=0.01, epochs=10000, eps=0.001, beta=0.9, datatype = '1'):

        self.layer\_sizes = layer\_sizes

        self.learning\_rate = learning\_rate

        self.epochs = epochs

        self.eps = eps

        self.beta = beta

        self.parameters = self.initialize\_parameters()

        self.velocities = self.initialize\_velocities()

        self.error = []

        self.datatype = datatype

        self.accuracy\_rate = None

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-x))

    def sigmoid\_derivative(self, x):

        return x \* (1 - x)

    def linear(self, x):

        return x

    def linear\_derivative(self, x):

        return 1

    def initialize\_parameters(self):

        parameters = []

        for i in range(len(self.layer\_sizes) - 1):

            W = np.random.randn(self.layer\_sizes[i], self.layer\_sizes[i + 1])

            b = np.zeros((1, self.layer\_sizes[i + 1]))

            parameters.append((W, b))

        return parameters

    def initialize\_velocities(self):

        velocities = []

        for i in range(len(self.layer\_sizes) - 1):

            vW = np.zeros((self.layer\_sizes[i], self.layer\_sizes[i + 1]))

            vb = np.zeros((1, self.layer\_sizes[i + 1]))

            velocities.append((vW, vb))

        return velocities

    def forward\_propagation(self, X):

        A = X

        caches = []

        for i in range(len(self.parameters) - 1):

            W, b = self.parameters[i]

            Z = np.dot(A, W) + b

            A = self.sigmoid(Z)

            caches.append((A, Z))

        W, b = self.parameters[-1]

        Z = np.dot(A, W) + b

        A = self.sigmoid(Z)

        caches.append((A, Z))

        return caches

    def backward\_propagation(self, X, Y, caches):

        m = Y.shape[0]

        grads = []

        A, Z = caches[-1]

        dZ = A - Y

        dW = (1/m) \* np.dot(caches[-2][0].T, dZ) if len(caches) > 1 else (1/m) \* np.dot(X.T, dZ)

        db = (1/m) \* np.sum(dZ, axis=0)

        grads.append((dW, db))

        for i in range(len(caches) - 2, -1, -1):

            A, Z = caches[i]

            dA = np.dot(dZ, self.parameters[i + 1][0].T)

            dZ = dA \* self.sigmoid\_derivative(A)

            dW = (1/m) \* np.dot(caches[i - 1][0].T, dZ) if i > 0 else (1/m) \* np.dot(X.T, dZ)

            db = (1/m) \* np.sum(dZ, axis=0)

            grads.append((dW, db))

        grads.reverse()

        for i in range(len(self.parameters)):

            W, b = self.parameters[i]

            dW, db = grads[i]

            vW, vb = self.velocities[i]

            vW = self.beta \* vW + (1 - self.beta) \* dW

            vb = self.beta \* vb + (1 - self.beta) \* db

            W -= self.learning\_rate \* vW

            b -= self.learning\_rate \* vb

            self.parameters[i] = (W, b)

            self.velocities[i] = (vW, vb)

    def normalize(self, X, Y):

        epsilon = 1e-8

        normalize\_X = (X - np.min(X)) / (np.max(X) - np.min(X) + epsilon)

        normalize\_Y = (Y - np.min(Y)) / (np.max(Y) - np.min(Y) + epsilon)

        return normalize\_X, normalize\_Y

    def train(self, X, Y):

        if self.datatype == '1':

            X, Y = self.normalize(X, Y)

        for epoch in range(self.epochs):

            caches = self.forward\_propagation(X)

            self.backward\_propagation(X, Y, caches)

            loss = np.mean((Y - caches[-1][0]) \*\* 2)

            print(f'Epoch {epoch}, Loss: {loss}')

            if loss <= 100:

                self.error.append(loss)

            if(loss <= self.eps):

                break

        plt.subplot(2,1,1)

        plt.plot(self.error)

        plt.xlabel('epoch')

        plt.ylabel('MSE')

        plt.grid()

    def test(self, Xtest, Ytest, i):

        if self.datatype == '1':

            Xnor, Ynor = self.normalize(Xtest, Ytest)

            predictions = self.predict(Xnor)

            plt.subplot(2,1,2)

            plt.plot(Ynor, marker = '.')

            plt.plot(predictions, marker = '.', color = "orange")

            plt.xlabel('Data')

            plt.ylabel('Data Value')

            plt.legend(["Desire Output","Actual Output"])

            plt.grid()

        elif self.datatype == '2':

            predictions = self.predict(Xtest)

            predictions = np.round(predictions, decimals = 0)

            plt.subplot(2,1,2)

            plt.plot(Ytest, 'o', ms = 10)

            plt.plot(predictions, 'x', ms = 10)

            plt.xlabel('Data')

            plt.ylabel('Data Value')

            plt.legend(["Desire Output1", "Desire Output2","Actual Output1","Actual Output2"])

            plt.grid()

            plt.figure(i + 10)

            self.plot\_confusion\_matrix(Ytest,predictions)

    def predict(self, X):

        Y = 0

        X, Y = self.normalize(X, Y)

        caches = self.forward\_propagation(X)

        return caches[-1][0]

    def plot\_confusion\_matrix(self,desired\_output,prediction):

        if self.datatype == '2':

            y\_true = np.argmax(desired\_output, axis=1)

            y\_pred = np.argmax(prediction, axis=1)

            confusion\_matrix = np.zeros((2, 2), dtype=int)

            for true, pred in zip(y\_true, y\_pred):

                confusion\_matrix[true, pred] += 1

            TP = confusion\_matrix[0, 0]  # True Positive

            FP = confusion\_matrix[0, 1]  # False Positive

            FN = confusion\_matrix[1, 0]  # False Negative

            TN = confusion\_matrix[1, 1]  # True Negative

            accuracy = (TP + TN) / (TP + TN + FP + FN)

            self.accuracy\_rate = accuracy \* 100

            # Plot the confusion matrix

            fig, ax = plt.subplots()

            cax = ax.matshow(confusion\_matrix, cmap=plt.cm.Blues)

            plt.colorbar(cax)

            # Annotate the confusion matrix with the counts

            for (i, j), val in np.ndenumerate(confusion\_matrix):

                plt.text(j, i, val, ha='center', va='center', color='red')

            plt.xlabel('Predicted')

            plt.ylabel('True')

            plt.title('Confusion Matrix')

            plt.xticks([0, 1], ['[1,0]', '[0,1]'])

            plt.yticks([0, 1], ['[1,0]', '[0,1]'])

#Data handle code section

def import\_dataset(file\_path):

    with open(file\_path, 'r') as file:

        lines = file.readlines()

    data\_lines = lines[2:]

    data = []

    for line in data\_lines:

        data.append([int(value) for value in line.split()])

    dataset = np.array(data)

    return dataset

def import\_cross(file\_path):

    with open(file\_path, 'r') as file:

        lines = file.readlines()

    Cdata = []

    current\_input = []

    current\_output = []

    for line in lines:

        line = line.strip()

        if line.startswith('p'):

            if current\_input and current\_output:

                # Combine input and output into a single row

                combined\_data = current\_input + current\_output

                Cdata.append(combined\_data)

                current\_input = []

                current\_output = []

        else:

            if not current\_input:

                current\_input = [float(num) for num in line.split()]

            else:

                current\_output = [int(num) for num in line.split()]

    if current\_input and current\_output:

        combined\_data = current\_input + current\_output

        Cdata.append(combined\_data)

    return Cdata

def tranfer(dataset):

    np.random.shuffle(dataset)

    X = np.array(dataset[2: , : -1])

    y = np.array(dataset[2: , -1])

    Y = []

    for i in y:

        Yarray = [i]

        Y.append(Yarray)

    Y = np.array(Y)

    return X, Y

def tranfer\_cross(Cdata):

    Cdata = np.array(Cdata)

    np.random.shuffle(Cdata)

    Xc = np.array(Cdata[: ,0:int(len(Cdata[0]) / 2)])

    Yc = np.array(Cdata[: ,-int(len(Cdata[0]) / 2):])

    return Xc, Yc

def k\_fold\_validation(X, Y, i):

    Xset = np.array\_split(X,k)

    Yset = np.array\_split(Y,k)

    Xtest = Xset[i]

    Ytest = Yset[i]

    Xtrain = np.concatenate([Xset[j] for j in range(k) if j != i])

    Ytrain = np.concatenate([Yset[j] for j in range(k) if j != i])

    return Xtrain, Ytrain, Xtest, Ytest

if \_\_name\_\_ == '\_\_main\_\_':

    data\_type = input("Select data type 1.Regression 2.Classification :")

    # Import dataset

    file = 'dataset.txt'

    file2 = 'cross.txt'

    dataset = import\_dataset(file)

    c\_dataset = import\_cross(file2)

    X, Y = tranfer(dataset)

    Xc, Yc = tranfer\_cross(c\_dataset)

    # Hyperparameters

    layer\_sizes = [8, 16, 1]  # Input size, hidden layer sizes, output size for Regression

    C\_layer\_sizes = [2, 16, 2]  # Input size, hidden layer sizes, output size for Classification

    epochs = 10000

    learning\_rate = 0.9

    eps = 0.001

    beta = 0.95  # Momentum term

    #K-fold validation

    k = 10

    if data\_type == '1':

        for i in range(k):

            nn = NeuralNetwork(layer\_sizes, learning\_rate, epochs, eps, beta, data\_type) #Parameter for specific regression

            Xtrain, Ytrain, Xtest, Ytest = k\_fold\_validation(X, Y, i)

            nn.error = []

            print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            plt.figure(i + 1)

            nn.train(Xtrain, Ytrain)

            nn.test(Xtest, Ytest ,i)

    elif data\_type == '2':

        acc = []

        for i in range(k):

            classification = NeuralNetwork(C\_layer\_sizes, 0.85, epochs, eps, 0.9, data\_type) #Parameter for specific classification

            Xctrain, Yctrain, Xctest, Yctest = k\_fold\_validation(Xc, Yc, i)

            classification.error = []

            print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            plt.figure(i + 1)

            classification.train(Xctrain, Yctrain)

            classification.test(Xctest, Yctest, i)

            acc.append(classification.accuracy\_rate)

        print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

        print("Accuracy rate in each fold : ", acc, " %")

        accuracy = np.mean(acc)

        print("Accuracy rate mean : ", accuracy," %")

    else:

        raise Exception("plese enter only 1 or 2 !")

    plt.show()