รายงาน

Home Work 1

261456 (INTRODUCTION OF COMPUTING INTELLIGENCE)

จัดทำโดย

นาย ศิวกร เครื่องคำ

650610857

เสนอ

อาจารย์ประจำวิชา

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

บทคัดย่อ

การศึกษาค้นคว้าและทดลองในครั้งนี้มีวัตถุประสงค์เพื่อ . เพื่อทดสอบ Validity, ความเร็วในการ Converge และความถูกต้อง (Accuracy) ของ Neural Network ที่เกิดขึ้นจากการเปลี่ยนแปลงของ Hidden layer, Leaning rate และ Momentum rate ผลการศึกษาพบว่าการศึกษาค้นคว้าและทดลองเรื่อง Neural Network มีจุดประสงค์เพื่อเสริมสร้างองค์ความรู้และความ เข้าใจในเรื่อง Neural Network และนำความรู้นี้ไปประยุกต์ใช้ในชีวิตประจำวัน โดยมีการดำเนินงาน คือการจำลองโมเดล และสุ่มค่าตัวแปรที่มีผลต่อโมเดล จากนั้นทำการทดลองปรับค่าตัวแปรที่สำคัญ เช่น Momentum rate, Hidden nodes และ Learning rate เป็นต้น ทำการติดตามผลของการทดลองและบันทึกข้อมูลผลลัพธ์จากการปรับค่าต่างข้างต้น จาก การศึกษาค้นคว้าและทดลองเรื่อง Neural Network ทำให้ผู้จัดทำมีความเข้าใจเรื่องนั้นๆ ได้ดียิ่งขึ้น และหวังเป็นอย่างยิ่งว่า โครงงานฉบับนี้จะมีประโยชน์แก่ผู้พบเห็นเป็นอย่างมาก

บทนำ

ที่มาและความสำคัญ เนื่องจากรายวิชา 261456 (INTRODUCTION OF COMPUTING INTELLIGENCE) ได้มีการมุ่งเน้นหลักสูตรที่ให้นักศึกษาทำการศึกษาค้นคว้าและสร้างองค์ความรู้เกี่ยวกับความฉลาดและการเรียนรู้เชิงลึกทางคอมพิวเตอร์ข้าพเจ้าจึงมีความสนใจในการศึกษาค้นคว้าเกี่ยวกับ Neural Network ที่เป็นหนึ่งในการเรียนรู้เชิงลึกของคอมพิวเตอร์ส่งผลให้คอมพิวเตอร์สามารถ คาดการณ์ผลลัพธ์ด้วยตัวเองได้ โดยวัตถุประสงค์ของระบบนี้เพื่อใช้ในการลดแรงงานมนุษย์รวมไปถึงการช่วยเหลือมนุษย์ใน การท าสิ่งต่างๆให้สะดวกขึ้น ซึ่งในปัจจุบันมนุษย์กับปัญญาประดิษฐ์รวมไปถึงระบบทางคอมพิวเตอร์ ต่างมีอิทธิพลต่อการดารงชีวิตมากขึ้นอย่างต่อเนื่อง ซึ่งในปัจจุบันมนุษย์ใช้โมเดล Neural Network ในการคิดวิเคราะห์และคำนวณสิ่งต่างๆในชีวิตประจำวัน ด้วยเหตุนี้จึงทำให้ จุดประกายทางความคิดและเกิดความสนใจจนเป็นที่มาของการศึกษาค้นคว้าและทดลองเกี่ยวกับ Neural Network ว่ามี หลักการทำงานรวมไปถึงข้อบกพร่องอย่างไร และเราสามารถพัฒนาอย่างไรจนทำให้เกิดประสิทธิภาพจนนำไปใช้งานได้

การทดลองเขียน Multi Layer Perceptron เพื่อทดลองการ Predict ระดับน้ำที่สะพานนวรัตน์ ในอีก 7 ชั่วโมงข้างหน้า และ ทดลองกับ cross.pat

โดยวิธีการคือ การนำเข้าข้อมูล Flood data set และ cross.pat ซึ่งมี format แตกต่างกัน จึงต้องแยกฟังก์ชั่นการนำเข้า Data set เป็น 2 ฟังก์ชั่น คือ Read_dataset_1 และ Read_dataset_2 เพื่อนำ Data set มา train และ Test (10% cross validation (Hold out))

โดย สร้าง Class MultilayerPerceptron โดยมีparameter(input Layer , Hidden Layer ,Output Layer) เราจะเริ่มโดยการสุ่มค่า weight และ Bias เริ่มต้นเมื่อเรียกใช้ Class และเขียน Activate Function และ Activate Function diff เพื่อนำไปใช้ใน Forward pass และ Backward pass(ส่วนสำคัญ) และนำ Function Forward pass และ Backward pass ไปทำ Function Train

ขั้นตอนการดำเนินการ

1.**การเตรียมข้อมูล**:

- ใช้ฟังก์ชัน read_dataset_1 อ่านข้อมูลจากไฟล์ Flood_dataset.txt และแยกข้อมูลออกเป็นส่วนของอินพุตและเอาต์พุต
- ใช้ฟังก์ชัน read_dataset_2 อ่านข้อมูลจากไฟล์ cross.txt เพื่อเตรียมข้อมูลสำหรับClassify

2.**การสร้างเครือข่ายประสาทเทียม**:

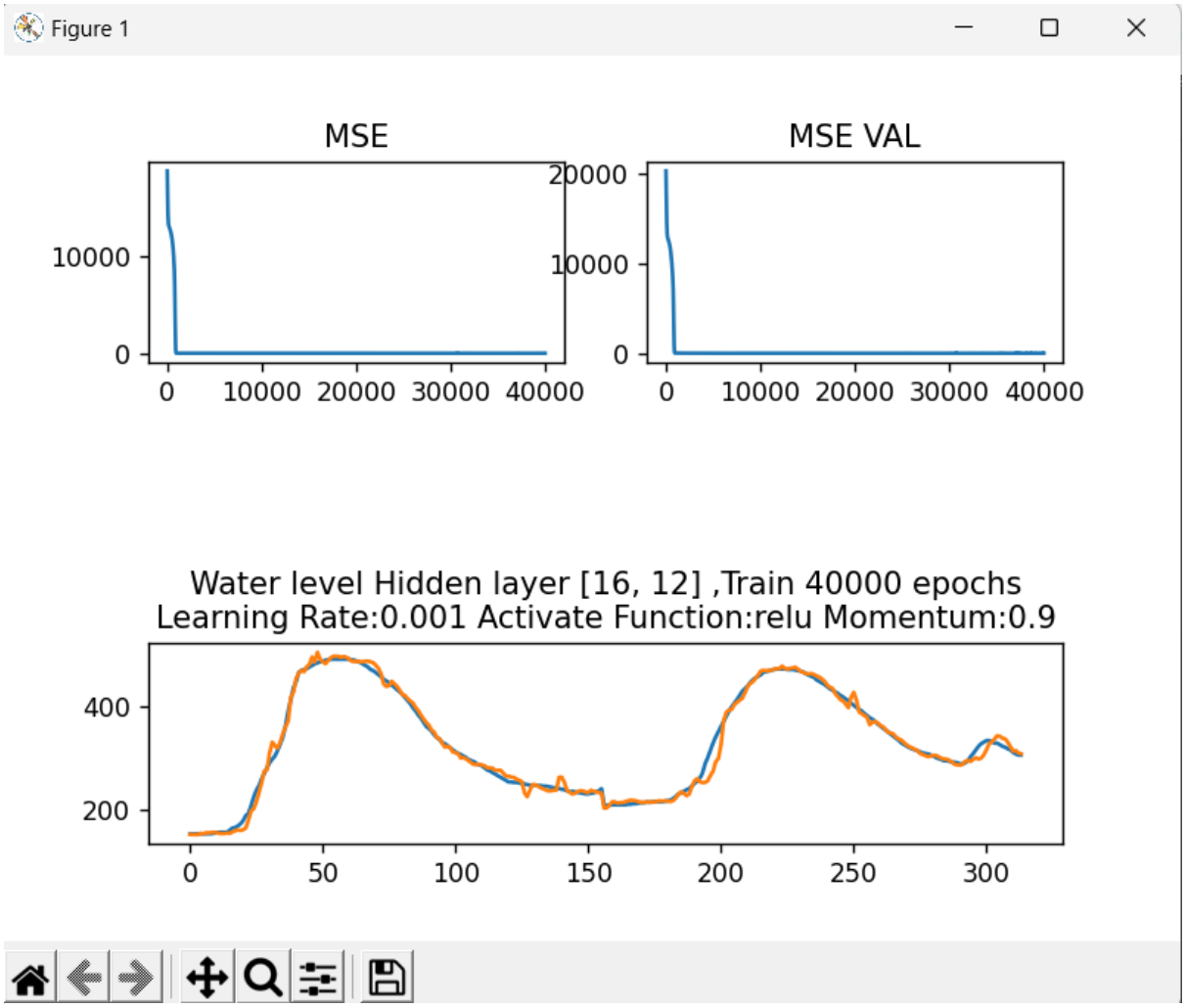
- สร้างคลาส MultiLayerPerceptron เพื่อสร้างโมเดลเครือข่ายประสาทเทียมหลายชั้น โดยมีพารามิเตอร์ที่สามารถปรับแต่งได้ เช่น Input Layer จำนวน Hidden Layer และOutput Layer
- ฟังก์ชันการทำงานในคลาสประกอบด้วย forward pass, backward pass, การคำนวณค่า Mean Squared Error (MSE), และการเทรนโมเดล

3.**การเทรนและทดสอบโมเดล**:

- สำหรับข้อมูลน้ำท่วม ใช้โมเดล MLP โดยมีขนาดอินพุต 8, ขนาดเลเยอร์ที่ซ่อน 2 เลเยอร์ (16, 12)(สามารถปรับแต่งได้) และขนาดเอาต์พุต 1 พร้อมการใช้งาน ReLU เป็น activation function
- สำหรับการจัดประเภท ใช้โมเดล MLP โดยมีขนาดอินพุต 2, ขนาดเลเยอร์ที่ซ่อน 1 เลเยอร์ (16) (สามารถปรับแต่งได้) และขนาดเอาต์พุต 2 พร้อมการใช้งาน Sigmoid เป็น activation function

โดยได้ใช้ 10% Cross validation แบบ Hold out ซึ่งจะนำ 10% ของข้อมูลไปเทสอย่างเดียว

ผลการทดลอง



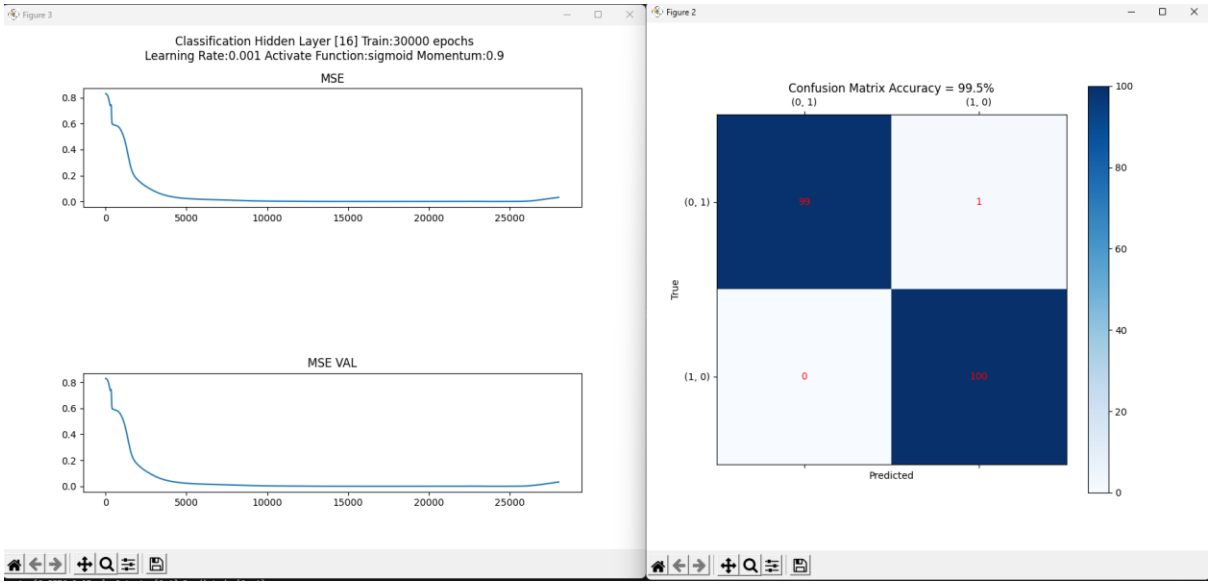


Figure 3
Classification Hidden Layer [16] Train:30000 epochs
Learning Rate:0.001 Activate Function:sigmoid Momentum:0.9
MSE
MSE VAL
Figure 2
Confusion Matrix Accuracy = 99.5%
(0, 1)
(1, 0)
True
Predicted
99
1
0
100

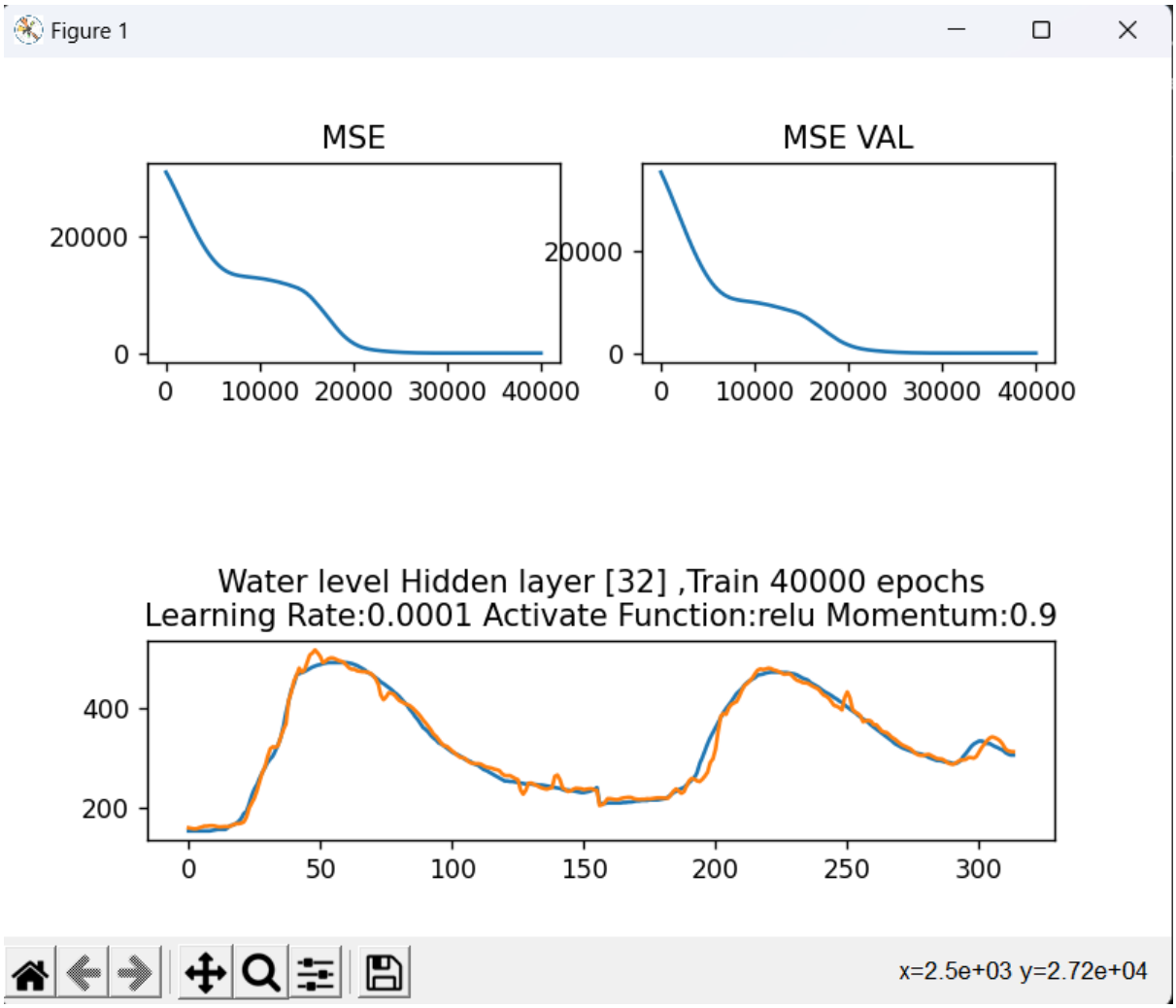


Figure 1
MSE
MSE VAL
Water level Hidden layer [32] ,Train 40000 epochs
Learning Rate:0.0001 Activate Function:relu Momentum:0.9
x=2.5e+03 y=2.72e+04

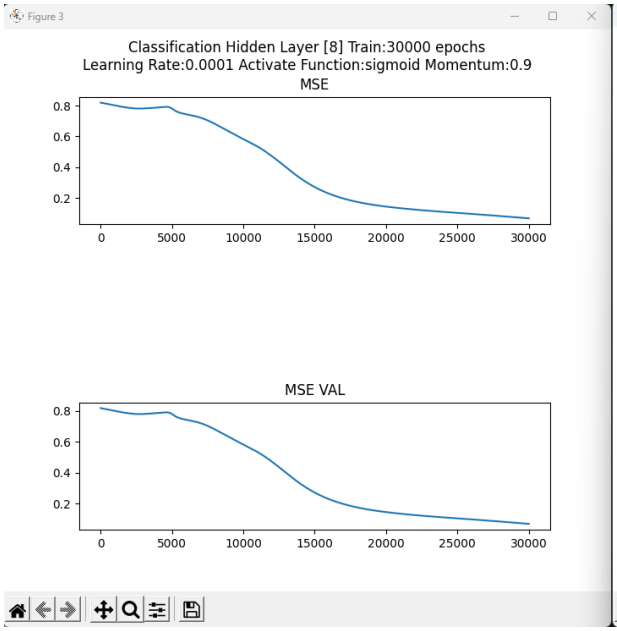


Figure 3
Classification Hidden Layer [8] Train:30000 epochs
Learning Rate:0.0001 Activate Function:sigmoid Momentum:0.9
MSE
MSE VAL

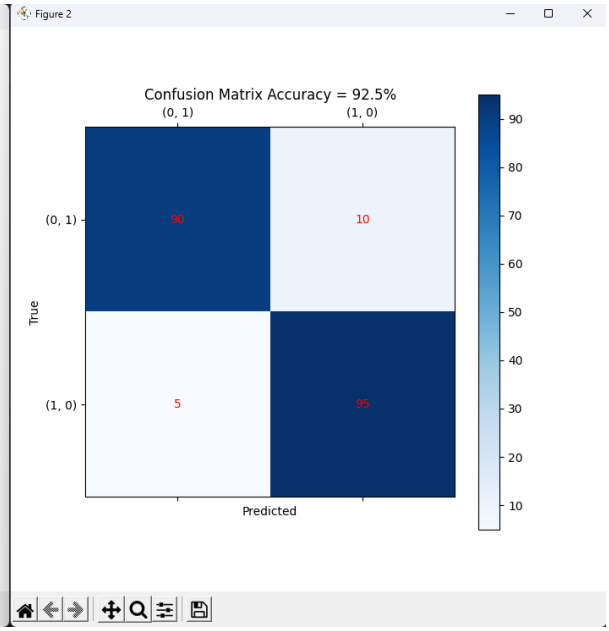


Figure 2
Confusion Matrix Accuracy = 92.5%
(0, 1)
(1, 0)
True
Predicted
90
10
5
95

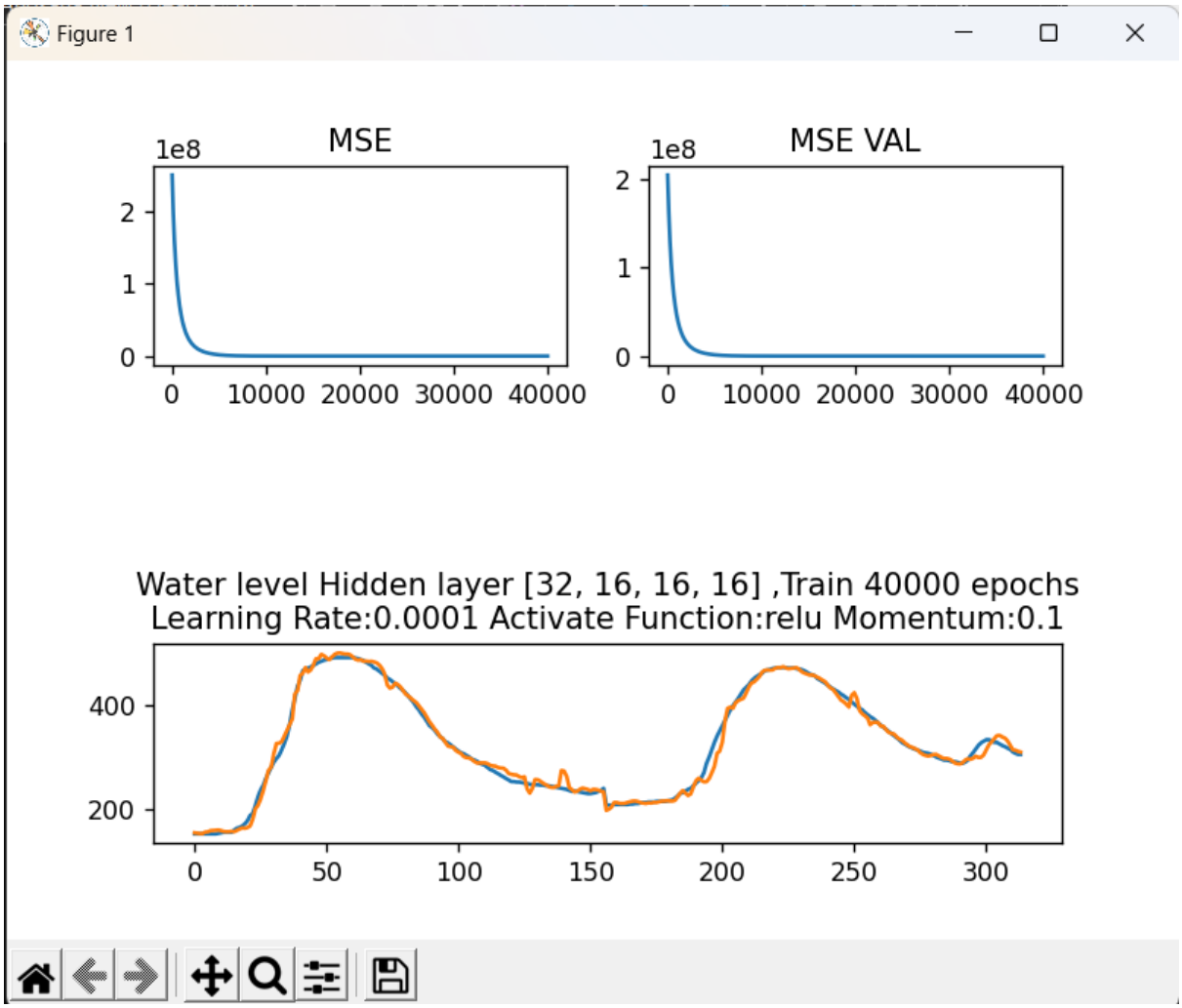


Figure 1
MSE
MSE VAL
1e8
Water level Hidden layer [32, 16, 16, 16] ,Train 40000 epochs
Learning Rate:0.0001 Activate Function:relu Momentum:0.1

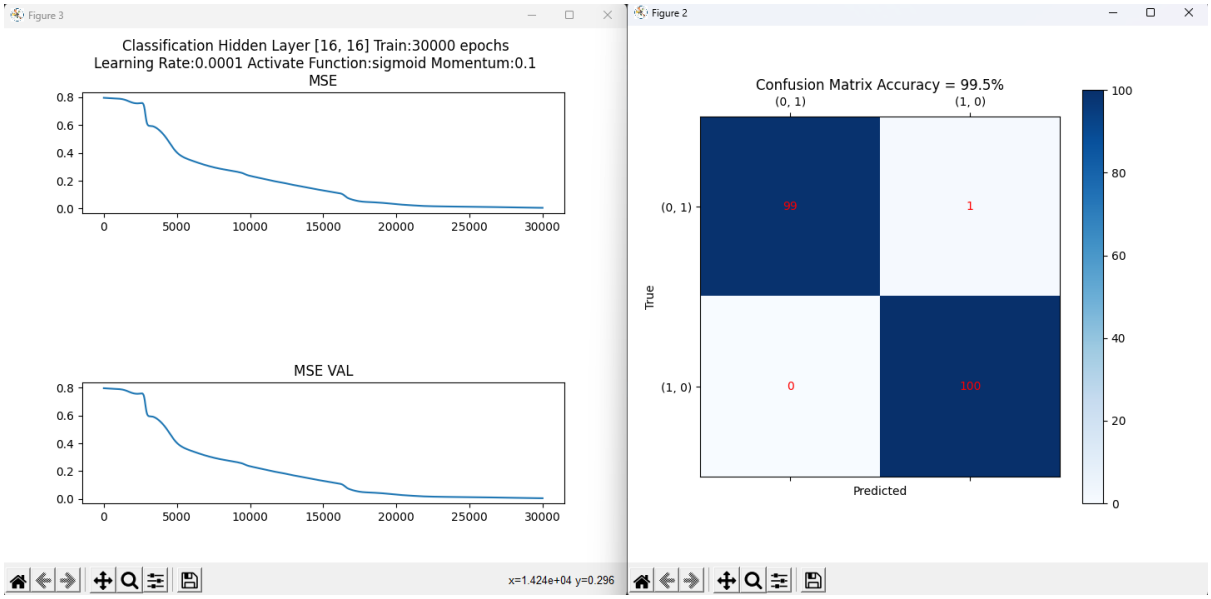


Figure 3
Classification Hidden Layer [16, 16] Train:30000 epochs
Learning Rate:0.0001 Activate Function:sigmoid Momentum:0.1
MSE
MSE VAL
x=1.424e+04 y=0.296
Figure 2
Confusion Matrix Accuracy = 99.5%
(0, 1)
(1, 0)
True
Predicted
99
1
0
100

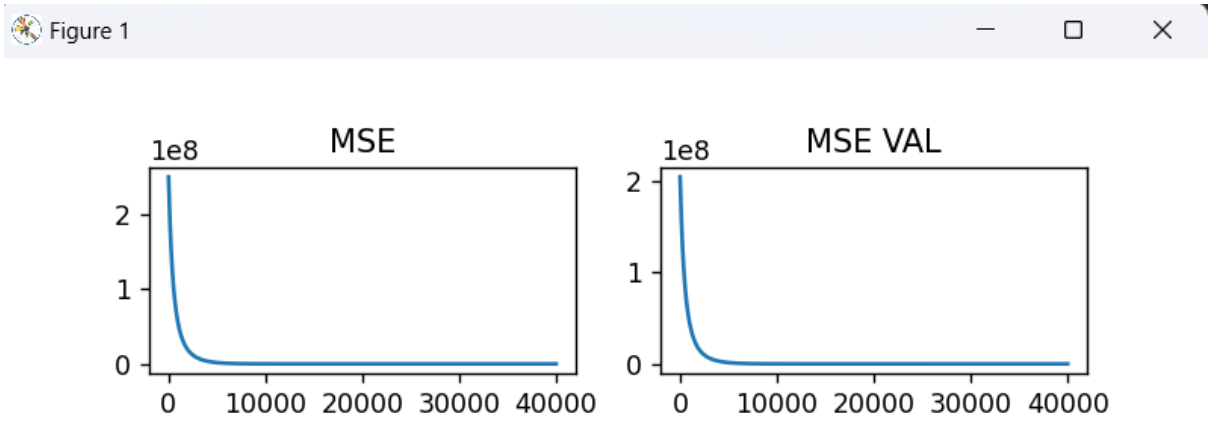
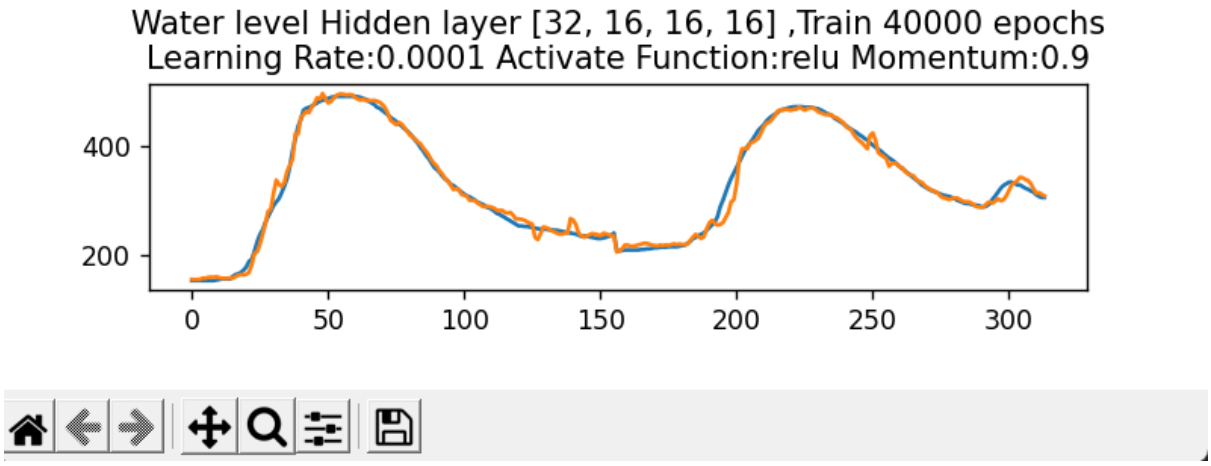
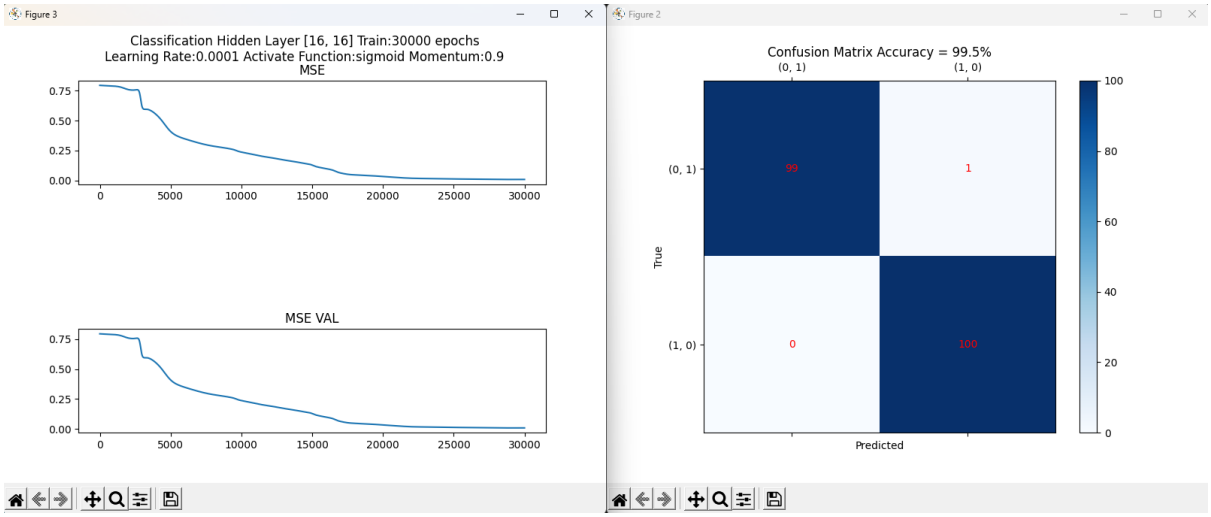
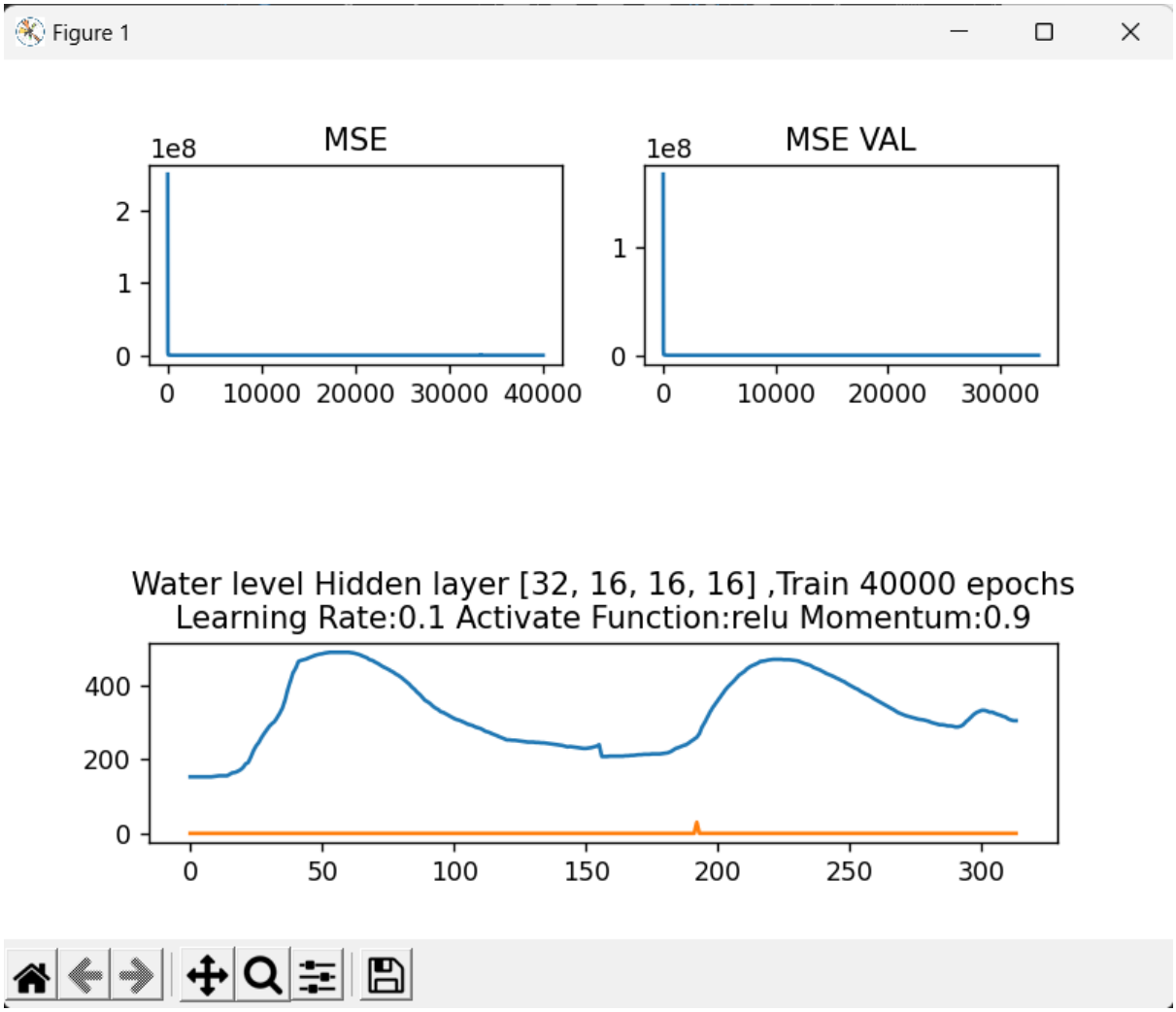


Figure 1
MSE
MSE VAL
1e8


Water level Hidden layer [32, 16, 16, 16] ,Train 40000 epochs
Learning Rate:0.0001 Activate Function:relu Momentum:0.9





Learning Rate มากเกินไป

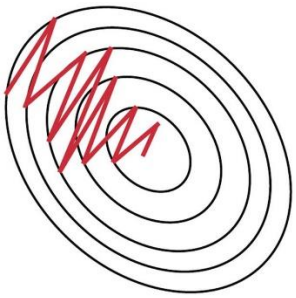
จากการวิเคราะห์ผลการทดลอง

การปรับค่า parameter ต่างๆ เช่น Hidden Layer ,Learning Rate ,Momentum Rate

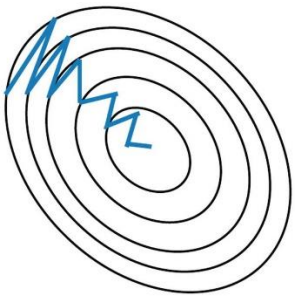
ได้ข้อแตกต่างดังนี้ การปรับ Hidden Layer ทำให้ Model Fit กับข้อมูลได้ดีขึ้น แต่การคำนวณของการ Train จะนานขึ้นเนื่องจาก ต้องคำณวนเพิ่มมากขึ้นจาก Node ที่เพิ่มขึ้น

การปรับ Learning Rate ที่เหมาะสม จะเห็นได้ว่า MSE ลดลงอย่างรวดเร็ว ส่วนการปรับ Momentum Rate จะทำให้ หาค่า Weight ที่ทำให้ Error น้อยลงได้ไวขึ้น

ส่วนการปรับ Learning Rate มากเกินไปจะทำให้ Model ไม่สามารถ ลู่เข้า Data ที่เรานำเข้าไป Train ได้และถ้า Learning Rate น้อยเกินไป ทำให้ต้องใช้ เวลาในการเทรนมากขึ้น จากการ Train หลาย Epochsป



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

ผลสรุป การทดลอง

โมเดลPredictระดับน้ำ:

- โมเดลสามารถทำนายระดับน้ำได้ค่อนข้างดี แต่ยังมีความคลาดเคลื่อนในบางจุด ซึ่งอาจเป็นผลมาจากข้อมูลที่มีการกระจายตัวหรือความไม่สมบูรณ์ของข้อมูล

โมเดลการclassify :

- โมเดลสามารถจัดประเภทได้อย่างแม่นยำ 90% UP ซึ่งแสดงให้เห็นถึงประสิทธิภาพในการใช้งานของโมเดล MLP กับปัญหาการจัดประเภทข้อมูล

ทั้งนี้ยังสามารถเพิ่มประสิทธิภาพต่างๆขึ้นได้โดยการ ปรับค่า parameter ต่างๆเช่น Epochs

Learning rate Hidden Layer, Activate Function และการจัดการข้อมูล เช่นการนำข้อมูลไป Normalization, Standardization เหมาะสมกับข้อมูล เพื่อให้ประสิทธิภาพดียิ่งขึ้น

ภาคผนวก

Github : https://github.com/Siwagon2546/CI_HM1.git

```
import numpy as np

import matplotlib.pyplot as plt


def normalize(X):

    mean = np.mean(X, axis=0)

    std = np.std(X, axis=0)

    return (X - mean) / std, mean, std


class MultiLayerPerceptron:

    def __init__(self, input_size, hidden_layers, output_size,
activation_function,error_stop=0.0001):

        self.input_size = input_size

        self.hidden_layers = hidden_layers

        self.output_size = output_size

        self.activation_function = activation_function

        self.beta1 = 0

        self.weights = []
```

```
self.biases = []

self.error_plot = []

self.error_val_plot = []

self.error_stop = error_stop


self.weights.append(np.random.rand(self.input_size, self.hidden_layers[0]))

self.biases.append(np.random.rand(1, self.hidden_layers[0]))


for i in range(1, len(self.hidden_layers)):

    self.weights.append(np.random.rand(self.hidden_layers[i-1], self.hidden_layers[i]))

    self.biases.append(np.random.rand(1, self.hidden_layers[i]))


self.weights.append(np.random.rand(self.hidden_layers[-1], self.output_size))

self.biases.append(np.random.rand(1, self.output_size))


self.m_weights = [np.zeros_like(w) for w in self.weights]

self.v_weights = [np.zeros_like(w) for w in self.weights]

self.m_biases = [np.zeros_like(b) for b in self.biases]

self.v_biases = [np.zeros_like(b) for b in self.biases]

self.t = 0
```

```
def mean_squared_error(self, y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)


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


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

```
        return 1 - x**2
    elif self.activation_function == "linear":
        return np.ones_like(x)

def forward(self, X):
    self.layer_inputs = [X]
    self.layer_outputs = []

    for i in range(len(self.hidden_layers)):
        layer_input = np.dot(self.layer_inputs[i], self.weights[i]) + self.biases[i]
        layer_output = self.activation(layer_input)
        self.layer_inputs.append(layer_input)
        self.layer_outputs.append(layer_output)

    final_input = np.dot(self.layer_outputs[-1], self.weights[-1]) + self.biases[-1]
    final_output = self.activation(final_input)
    self.layer_inputs.append(final_input)
    self.layer_outputs.append(final_output)

    return final_output
```

```
def backward(self, X, y, learning_rate, beta1, beta2, epsilon):
    self.t += 1
    output_error = y - self.layer_outputs[-1]
    output_delta = output_error * self.activation_diff(self.layer_outputs[-1])
    deltas = [output_delta]

    for i in range(len(self.hidden_layers)-1, -1, -1):
        delta = deltas[-1].dot(self.weights[i+1].T) * self.activation_diff(self.layer_outputs[i])
        deltas.append(delta)

    deltas.reverse()

    for i in range(len(self.weights)):
        layer_output = X if i == 0 else self.layer_outputs[i-1]

        weight_gradient = layer_output.T.dot(deltas[i])
        bias_gradient = np.sum(deltas[i], axis=0, keepdims=True)

        self.m_weights[i] = beta1 * self.m_weights[i] + (1 - beta1) * weight_gradient
```

```
            self.m_biases[i] = beta1 * self.m_biases[i] + (1 - beta1) * bias_gradient

            self.v_weights[i] = beta2 * self.v_weights[i] + (1 - beta2) * (weight_gradient ** 2)
            self.v_biases[i] = beta2 * self.v_biases[i] + (1 - beta2) * (bias_gradient ** 2)

            m_hat_weights = self.m_weights[i] / (1 - beta1 ** self.t)
            m_hat_biases = self.m_biases[i] / (1 - beta1 ** self.t)

            v_hat_weights = self.v_weights[i] / (1 - beta2 ** self.t)
            v_hat_biases = self.v_biases[i] / (1 - beta2 ** self.t)

            self.weights[i] += learning_rate * m_hat_weights / (np.sqrt(v_hat_weights) + epsilon)
            self.biases[i] += learning_rate * m_hat_biases / (np.sqrt(v_hat_biases) + epsilon)

    def train(self, X, y, epochs, learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8):
        X, mean, std = normalize(X)
        self.beta1 = beta1
        self.mean = mean
        self.std = std
```

```
indices = np.arange(X.shape[0])

np.random.shuffle(indices)

split_index = int(0.9 * X.shape[0])

train_indices, val_indices = indices[:split_index], indices[split_index:]


X_train, X_val = X[train_indices], X[val_indices]

y_train, y_val = y[train_indices], y[val_indices]

""" self.error_plot = []

self.error_val_plot = [] """


for epoch in range(epochs):

    indices_xy_train = np.arange(X_train.shape[0])

    np.random.shuffle(indices_xy_train)

    X_train = X_train[indices_xy_train]

    y_train = y_train[indices_xy_train]


    self.forward(X_train)

    self.backward(X_train, y_train, learning_rate, beta1, beta2, epsilon)


    y_t = np.array(normalize(y_train))
```

```
        o_t = np.array(normalize(self.layer_outputs[-1]))
        train_loss = self.mean_squared_error(y_t, o_t)

        val_output = self.forward(X_val)
        v_o_t = np.array(normalize(val_output))
        y_v_t = np.array(normalize(y_val))
        val_loss = self.mean_squared_error(y_v_t, v_o_t)
        """ print(np.mean(train_loss))
        print(55555)
        print(np.mean(val_loss)) """

        self.error_plot.append(np.mean(train_loss))
        self.error_val_plot.append(np.mean(val_loss))
        if np.mean(train_loss) <self.error_stop:
            break
        if epoch % 1000 == 0:

            print(f"Epoch {epoch}, Training MSE: {np.mean(train_loss)}, Validation MSE:
{np.mean(val_loss)}")
```

```
    def predict(self, X):
        X = (X - self.mean) / self.std
        return self.forward(X)
    def predict_classify(self, X):
        X = (X - self.mean) / self.std
        k1= self.forward(X)
        if k1[0][0] >k1[0][1]:
            return [1,0]
        else:
            return [0,1]



def read_dataset_1():
    f = open("Flood_dataset.txt", "r")
    data = []
    for i in f:
        data.append(i.split())


    data = np.array(data[2:len(data)])
```

```
    x_train = []
    y_train = []
    for i in data:
        z = []
        x_train.append(i[:8])
        z.append(i[-1])
        y_train.append(z)

    y_train = np.array(y_train)
    x_train = np.array(x_train)
    y_train = y_train.astype(np.int64)
    x_train = x_train.astype(np.int64)
    return x_train, y_train

def read_dataset_2(filename='cross.txt'):
    data = []
    input = []
    design_output = []
    with open(filename) as f:
        a = f.readlines()
```

```
        for line in range(1, len(a), 3):
            z = np.array([float(element) for element in a[line][:-1].split()])
            zz = np.array([float(element) for element in a[line+1].split()])
            data.append(np.append(z, zz))
    data = np.array(data)
    #np.random.shuffle(data)
    for i in data:
        input.append(i[:-2])
        design_output.append(i[-2:])
    design_output = np.array(design_output)
    input = np.array(input)
    design_output = design_output.astype(np.int16)
    input = input.astype(np.float16)
    return input, design_output


#
np.random.seed(42)
#For Regression
X, y = read_dataset_1()
input_size1 = 8
```

```
hidden_layers1 = [32,16,16,16]

output_size1 = 1

learning_rate1 = 0.0001

epochs1 = 40000


MLP_waterlevel = MultiLayerPerceptron(input_size1, hidden_layers1, output_size1, "relu")

MLP_waterlevel.train(X, y, epochs1, learning_rate1, beta1=0.9)


output_NN1 = []

y_plot1 = []

i = 0


for x in X:

    output = MLP_waterlevel.predict(np.array([x]))

    output_NN1.append(output[0])

    y_plot1.append(y[i][0])

    print(f"Input: {x}, Output: {y[i]} Predicted: {output}")

    i += 1


#For classify
```

```
input2 , design_out2 = read_dataset_2()

input_size2 = 2

hidden_layers2 = [16,16]

output_size2 = 2

learning_rate2 = 0.0001

epochs2 = 30000


MLP_Class = MultiLayerPerceptron(input_size2, hidden_layers2, output_size2, "sigmoid")

MLP_Class.train(input2, design_out2, epochs2, learning_rate2, beta1=0.9)

#print(input2 , design_out2)


# Function to compute confusion matrix


def compute_confusion_matrix(y_true, y_pred):


    # Unique label tuples


    #print(unique_labels)

    label_to_index = {label: index for index, label in enumerate(unique_labels)}

    matrix = np.zeros((len(unique_labels), len(unique_labels)), dtype=int)
```

```
    for true, pred in zip(y_true, y_pred):
        true_idx = label_to_index[true]
        pred_idx = label_to_index[pred]
        matrix[true_idx, pred_idx] += 1
    t = 0
    f = 0
    for i in range(len(matrix[0])):
        for j in range(len(matrix)):
            if i==j:
                t+=matrix[i][j]
                f+=matrix[i][j]
            else:
                f+=matrix[i][j]
    #print(matrix)
    return matrix,(t/f*100)


def plot_confusion_matrix(cm,acc, labels):
    fig, ax = plt.subplots()
    cax = ax.matshow(cm, cmap=plt.cm.Blues)
```

```
    fig.colorbar(cax)


    for (i, j), val in np.ndenumerate(cm):
        ax.text(j, i, val, ha='center', va='center', color='red')


    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')
    ax.set_xticks(np.arange(len(labels)))
    ax.set_yticks(np.arange(len(labels)))
    ax.set_xticklabels(labels)
    ax.set_yticklabels(labels)


    plt.title(f"Confusion Matrix Accuracy = {acc}%")




output_NN2 = []
y_plot2 = []
i = 0
```

```
for x in input2:
    output = MLP_Class.predict_classify(np.array([x]))
    output_NN2.append(output)
    y_plot2.append((np.array(design_out2[i])).tolist())
    print(f"Input: {x}, Output: {design_out2[i]} Predicted: {output}")
    i += 1


#print((y_plot2, output_NN2))
y_plot2 = np.array(y_plot2)
output_NN2 = np.array(output_NN2)
""" y_plot2 = (y_plot2.reshape(len(y_plot2), 2))#.tolist()
output_NN2 = (output_NN2.reshape(len(y_plot2), 2))#.tolist() """
y_plot2_flat =  [tuple(row) for row in y_plot2]
output_NN2_flat =  [tuple(row) for row in output_NN2]
#print((y_plot2), (output_NN2))
#print(type(y_plot2), type(output_NN2))
unique_labels = sorted(set(y_plot2_flat + output_NN2_flat))
#print(unique_labels)
# Flatten the 2D arrays to 1D tuples
```

```
plot1 = plt.subplot2grid((3, 2), (0, 0), colspan=1)

plot2 = plt.subplot2grid((3, 2), (0, 1), colspan=1)

plot3 = plt.subplot2grid((3, 2), (2, 0), rowspan=1, colspan=2)



#print(MLP_waterlevel.error_plot)

p1 =np.array(MLP_waterlevel.error_plot).flatten()

#print(len(p1))

plot1.plot(p1)

plot1.set_title('MSE')



#print(MLP_waterlevel.error_val_plot)

plot2.plot(np.array(MLP_waterlevel.error_val_plot).flatten())

plot2.set_title('MSE VAL')



plot3.plot(y_plot1)

plot3.plot(output_NN1)

plot3.set_title(f'Water level Hidden layer {hidden_layers1} ,Train {epochs1} epochs'

+"\n"+f'Learning Rate:{learning_rate2} Activate

Function:{MLP_waterlevel.activation_function} Momentum:{MLP_waterlevel.beta1}')



""" plt.figure()
```

```
plt.title(f"Water level Hidden layer {hidden_layers1} ,Train {epochs1} epochs")

plt.plot(y_plot1)

plt.plot(output_NN1) """



cm ,acc = compute_confusion_matrix(y_plot2_flat, output_NN2_flat)

plot_confusion_matrix(cm,acc ,unique_labels)



plt.figure(3)



plot4 = plt.subplot2grid((3, 1), (0, 0), colspan=1)

plot5 = plt.subplot2grid((3, 1), (2, 0), colspan=1)



p4 = np.array(MLP_Class.error_val_plot).flatten()

#print(len(p4))

plot4.plot(p4)

plot4.set_title('MSE')



#print(MLP_waterlevel.error_val_plot)

plot5.plot(np.array(MLP_Class.error_val_plot).flatten())

plot5.set_title('MSE VAL')
```

```
plt.suptitle(f"Classification Hidden Layer {hidden_layers2} Train:{epochs2} epochs" +"\n"+
f"Learning Rate:{learning_rate2} Activate Function:{MLP_Class.activation_function}
Momentum:{MLP_Class.beta1}")

plt.show()

""" plt.figure(1)

plt.plot(y_plot2,"bo")

plt.figure(2)

plt.plot(output_NN2,"ro")

plt.show() """
```