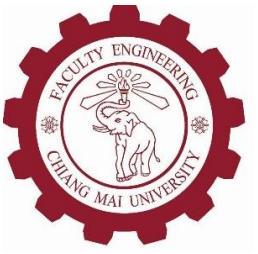


รายงานการค้นคว้าและทดลอง

ผลกระทบของ **Neural Network ที่เกิดจากการเปลี่ยนแปลงตัวแปรในโมเดล**

โดย

นายศุภกฤต ก่องคำ

650610858

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261456

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่1 ปีการศึกษา 2566

มหาวิทยาลัยเชียงใหม่

บทคัดย่อ

การศึกษาค้นคว้าและทดลองในครั้งนี้มีวัตถุประสงค์เพื่อ . เพื่อทดสอบ Validity, ความเร็วในการ Converge และความถูกต้อง (Accuracy)ของ Neural Network ที่เกิดขึ้นจากการเปลี่ยนแปลงของ Hidden layer, Leaning rate และ Momentum rate

ผลการศึกษาพบว่า การศึกษาค้นคว้าและทดลองเรื่อง Neural Network มีจุดประสงค์เพื่อเสริมสร้างองค์ความรู้และความเข้าใจในเรื่อง Neural Network และนำความรู้นี้ไปประยุกต์ใช้ในชีวิตประจำวัน โดยมีการดำเนินงาน คือการจำลองโมเดลและสุ่มค่าตัวแปรที่มีผลต่อโมเดล จากนั้นทำการทดลองปรับค่าตัวแปรที่สำคัญ เช่น Momentum rate, Hidden nodes และ Learning rate เป็นต้น ทำการติดตามผลของการทดลองและบันทึกข้อมูลผลลัพธ์จากการปรับค่าต่างข้างต้น จากการศึกษา ค้นคว้าและทดลองเรื่อง Neural Network ทำให้ผู้จัดทำมีความเข้าใจเรื่องนั้นๆ ได้ดียิ่งขึ้น และหวังเป็นอย่างยิ่งว่าโครงงานฉบับนี้จะมีประโยชน์แก่ผู้พบเห็นเป็นอย่างมาก

บทนำ

ที่มาและความสำคัญ

เนื่องจากรายวิชา 261456 (INTRODUCTION OF COMPUTING INTELLIGENCE) ได้มีการมุ่งเน้นหลักสูตรที่ให้นักศึกษาทำการศึกษาค้นคว้าและสร้างองค์ความรู้เกี่ยวกับความฉลาดและการเรียนรู้เชิงลึกทางคอมพิวเตอร์ ข้าพเจ้าจึงมีความสนใจในการศึกษาค้นคว้าเกี่ยวกับ Neural Network ที่เป็นหนึ่งในการเรียนรู้เชิงลึกของคอมพิวเตอร์ ส่งผลให้คอมพิวเตอร์สามารถคาดการณ์ผลลัพธ์ด้วยตัวเองได้ โดยวัตถุประสงค์ของระบบนี้เพื่อใช้ในการลดแรงงานมนุษย์รวมไปถึงการช่วยเหลือมนุษย์ในการทำสิ่งต่างๆให้สะดวกขึ้น

ซึ่งในปัจจุบันมนุษย์กับปัญญาประดิษฐ์รวมไปถึงระบบทางคอมพิวเตอร์ ต่างมีอิทธิพลต่อการดำรงชีวิตมากขึ้นอย่างต่อเนื่อง ซึ่งในปัจจุบันมนุษย์ใช้โมเดล Neural Network ในการคิดวิเคราะห์และคำนวณสิ่งต่างๆในชีวิตประจำวัน ด้วยเหตุนี้จึงทำให้จุดประกายทางความคิดและเกิดความสนใจจนเป็นที่มาของการศึกษาค้นคว้าและทดลองเกี่ยวกับ Neural Network ว่ามีหลักการทำงานรวมไปถึงข้อบกพร่องอย่างไร และเราสามารถพัฒนาอย่างไรจนทำให้เกิดประสิทธิภาพจนนำไปใช้งานได้

วัตถุประสงค์ของการศึกษาค้นคว้าและทดลอง

เพื่อทดสอบ Validity, ความเร็วในการ Converge และความถูกต้อง(Accuracy)และผลกระทบโดยรวมของ Neural Network ที่เกิดขึ้นจากการเปลี่ยนแปลงของ Hidden layer, Leaning rate และ Momentum rate

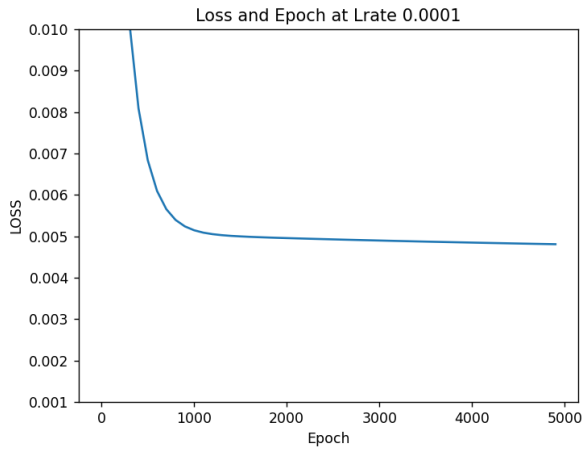
ขอบเขตการทดลอง

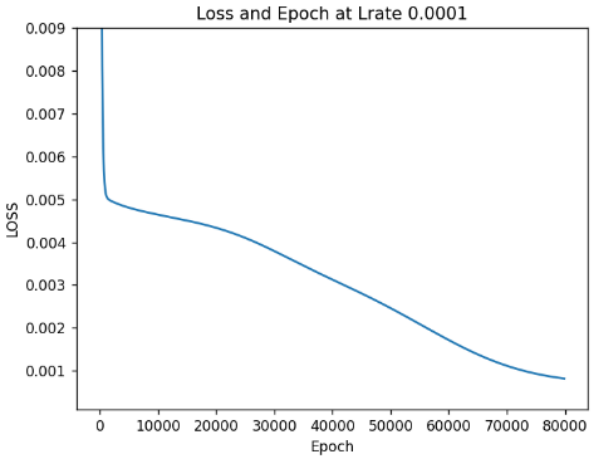
ข้อมูลระดับน้ำจาก 2 สถานีที่ย้อนหลังไปอีก 3 ชั่วโมง ซึ่งได้แก่ สถานีที่ 1 t-3, สถานีที่ 1 t-2, สถานีที่ 1 t-1, สถานีที่ 1 t-0, สถานีที่ 2 t-3, สถานีที่ 2 t-2, สถานีที่ 2 t-1 และสถานีที่ 2 t-0

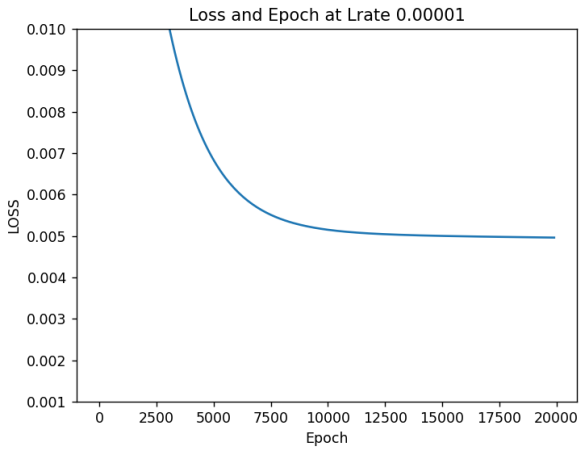
1.การทดลองผลกระทบที่เกิดจาก Learning Rate

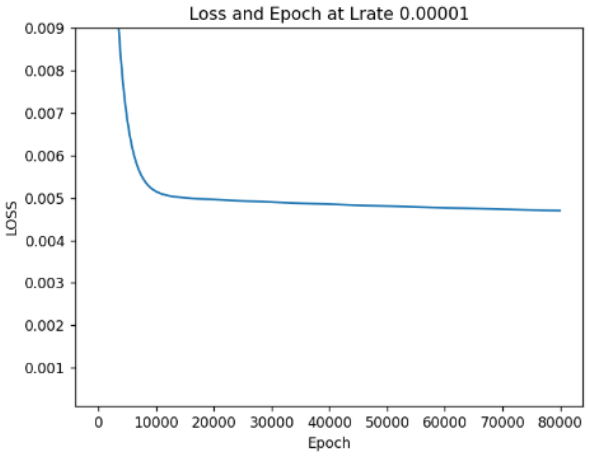
กำหนดให้ Momentum rate อยู่ที่ 0.00 และ Hidden Node จำนวน 2 Node

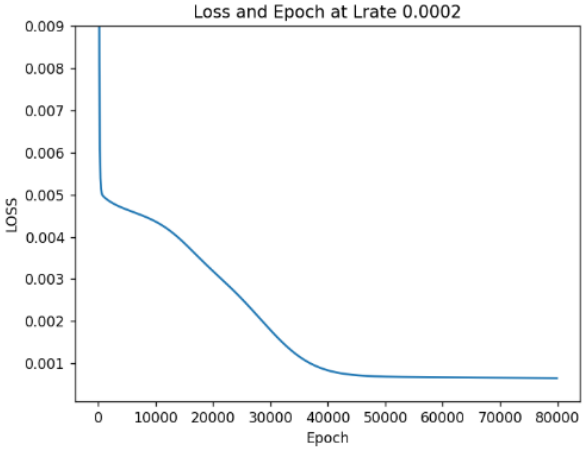
จากนั้นพลอตกราฟระหว่าง Loss และ Epoch และสังเกตกราฟที่ได้จาก Learning Rate ที่ต่างกัน

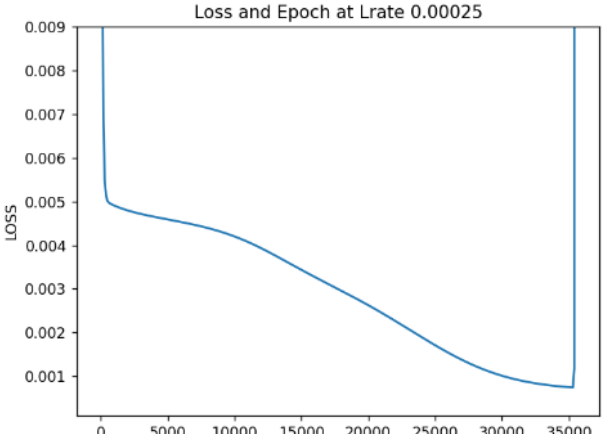








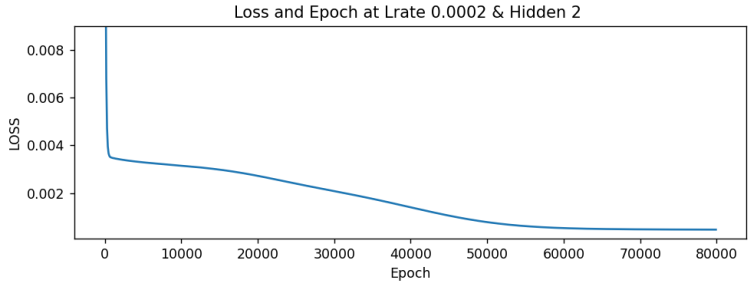


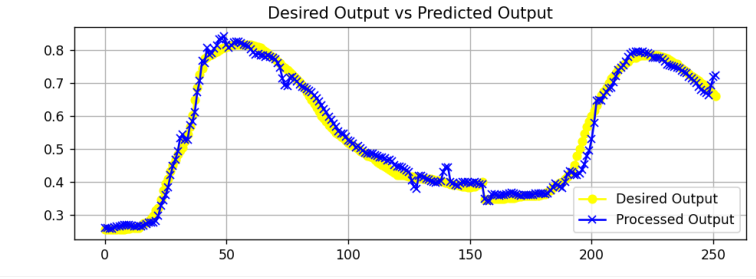


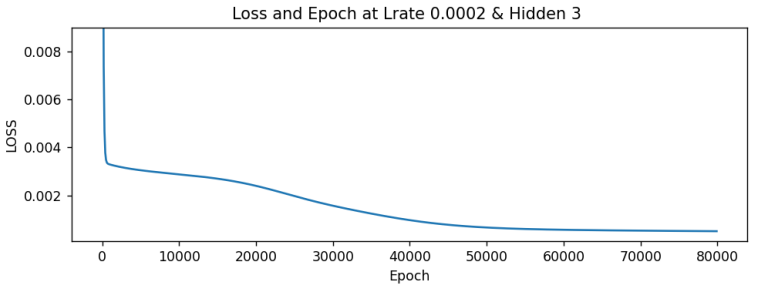
จากการทดลองเปลี่ยนแปลงค่า **Learning Rate** เป็นค่าต่างๆโดยที่ใช้ **Hidden Node** 2 **Node** พบว่าเมื่อ **Learning Rate** มีค่ามากเกินไปหรือมีค่าน้อยเกินไปในระดับหนึ่ง จะทำให้ **Neural Network** ไม่สามารถทำงานได้ โดยจากการทดลองยังพบอีกว่าเมื่อ **Learning Rate** มีค่าน้อยจะต้องใช้ **Epoch** จำนวนมากขึ้นเพื่อทำให้โมเดลมีประสิทธิภาพเทียบเท่ากับ **Learning Rate** ที่พอดีและใช้ **Epoch** ที่น้อยกว่า

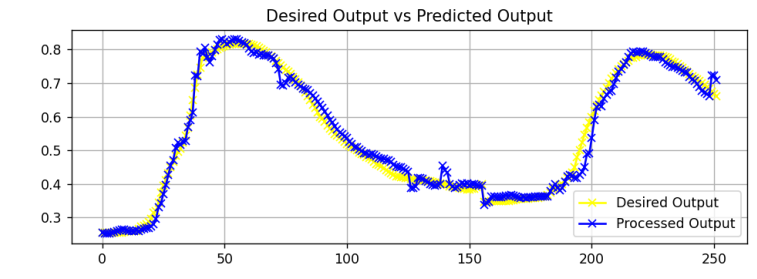
2.การทดลองผลกระทบที่เกิดจากการปรับ **Hidden Node**

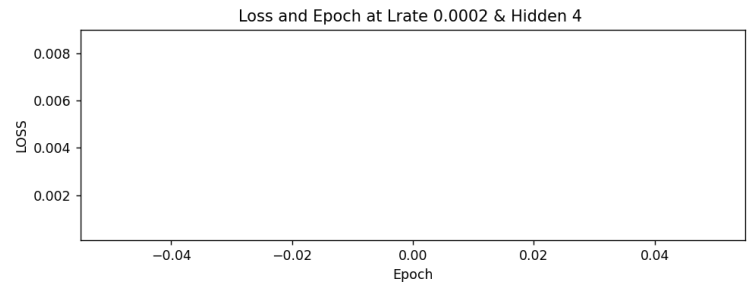
กำหนดให้ **Momentum rate = 0.00** จากนั้นพลอตกราฟระหว่าง **Loss** และ **Epoch** และสังเกตกราฟที่ได้จากจำนวณ **Hidden Node** ที่ต่างกัน

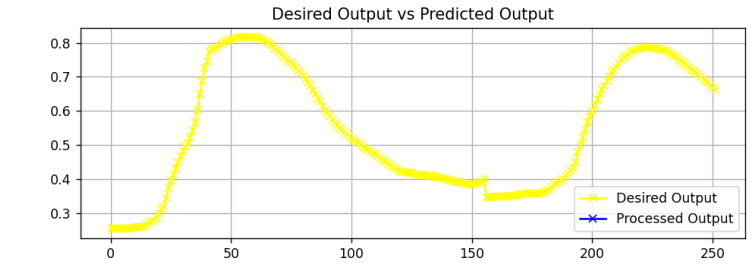


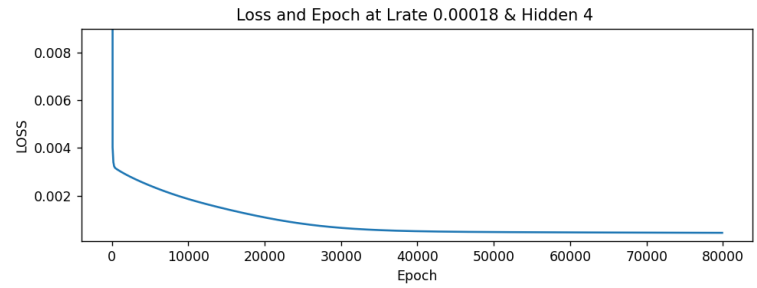


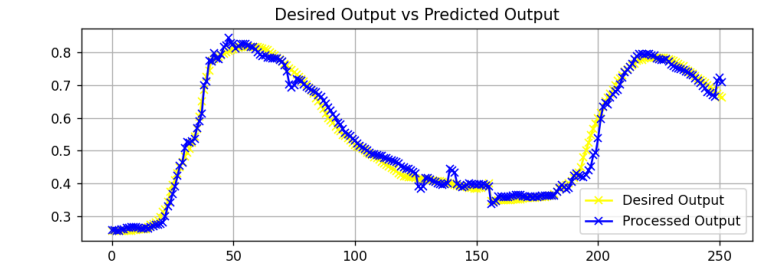








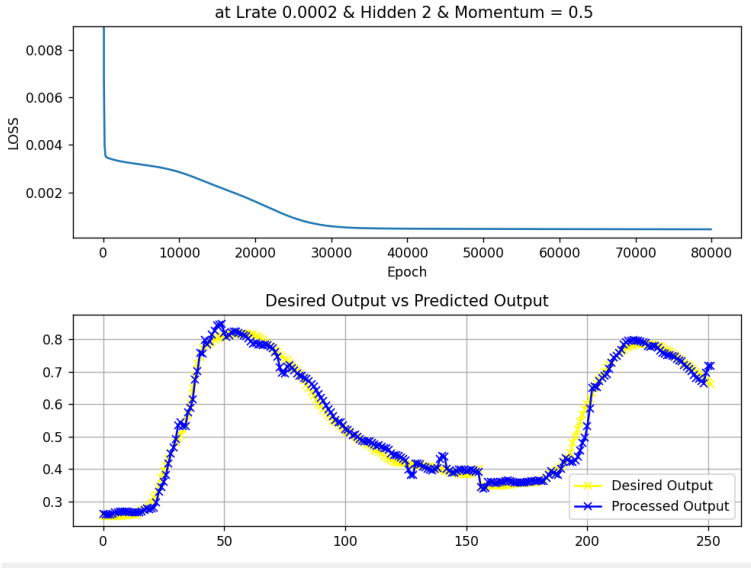


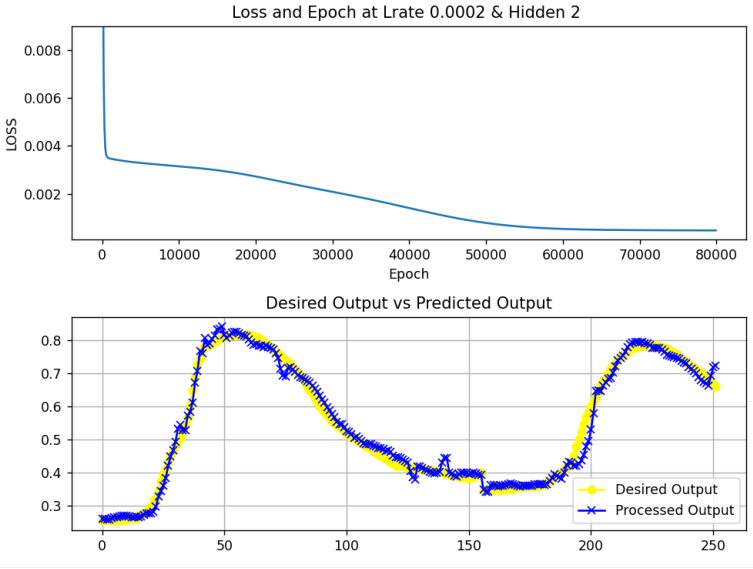


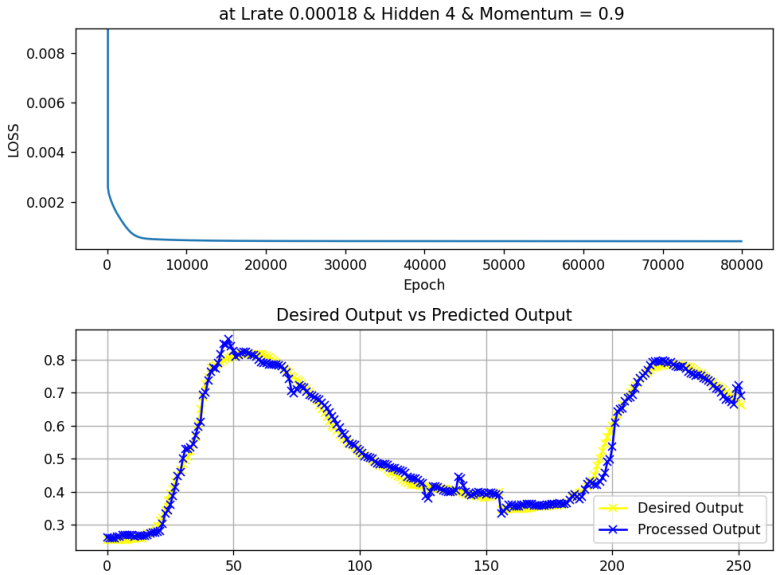
จากการทดลองเปลี่ยนแปลงจำนวณ Hidden Node โดยกำหนดค่า Momentum Rate= 0 พบว่า เมื่อ Hidden Node มีจำนวณมากขึ้น จะยิ่งทำให้โมเดลเรียนรู้ได้มีประสิทธิภาพ แต่เมื่อมากเกินไปโมเดลก็ไม่สามารถทำงานตามที่ต้องการได้ และยังมีความสัมพันธ์กับ Learning Rate โดยเมื่อ Hidden Node มีจำนวณมากขึ้น Learning Rate ก็ต้องปรับให้พอดีถึงจะสามารถทำงานได้ และเมื่อเทียบกันแล้วโมเดลที่สัดส่วน Hidden Node จำนวนน้อยแล้ว Learning Rate มีค่ามาก จะแม่นยำน้อยกว่า โมเดลที่ Hidden Node จำนวนมาก และ Learning Rate ที่มีค่าน้อย

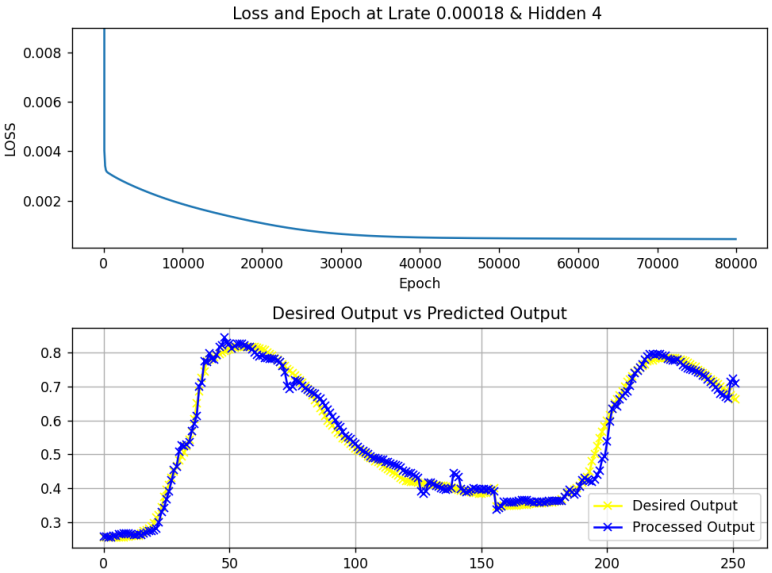
3.การทดลองผลกระทบที่เกิดจาก Momentum Rate

โดยการทดลองนี้จะปรับค่า Learning Rate และจำนวณ Hidden Node ตามความเหมาะสมเพื่อสังเกตความสัมพันธ์ระหว่างตัวแปรข้างต้นกับค่า Momentum Rate





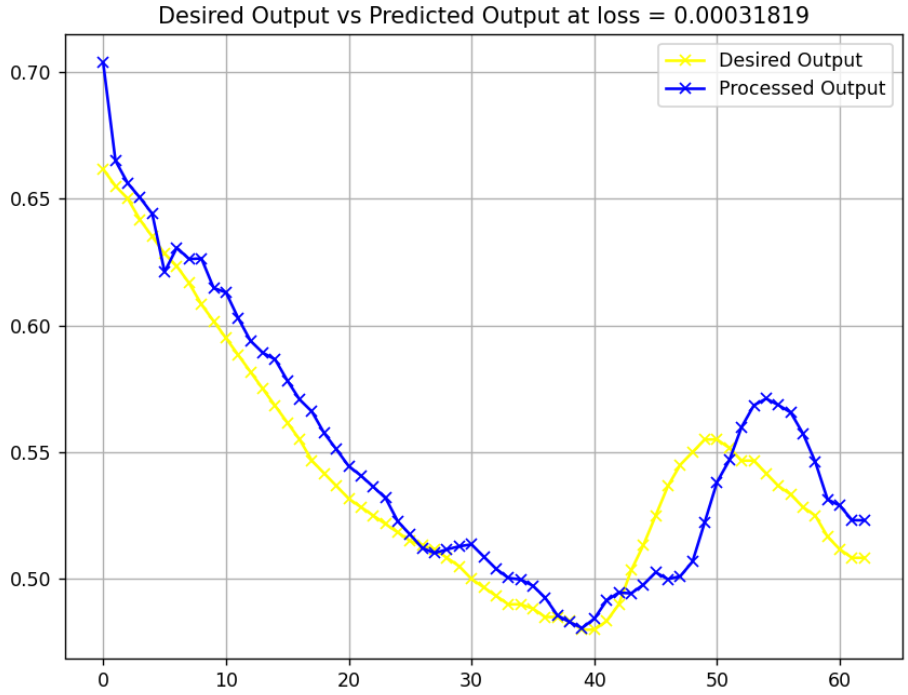




จากการทดลองเปลี่ยนแปลงค่า Momentum Rate พบว่า เมื่อมีค่า Momentum Rate มากขึ้น โมเดลก็จะมีความเร็วในการเรียนรู้มากขึ้น นั่นก็คือใช้ Epoch ที่น้อยลง แต่เมื่อ Momentum Rate มีค่ามากเกินไป ก็จะทำให้โมเดลเกิดผิดพลาดได้ หรือทำให้ความแม่นยำลดลงได้ ดังนั้นค่า Momentum Rate ที่พอดีจะทำให้โมเดลทำงานได้เร็วมากขึ้น

4. ทดลองคาดการระดับน้ำจากข้อมูลที่ไม่เคยใช้ฝึกโมเดล

กำหนดให้ใช้ Hidden Node จำนวน 4 Nodes, Learning Rate = 0.00018, และ Momentum Rate = 0.5 ในการเทรนโมเดลแล้วใช้ค่า weight ที่ได้ในการใช้งานจริง โดยสังเกตว่าโมเดลยังมีปัญหาเรื่องอะไรและสรุปผล



จากการคาดการณ์ระดับน้ำจากข้อมูลที่ไม่เคยเห็น พบว่าโมเดลยังคลาดเคลื่อนในระดับหนึ่งจากการโดยอาจจะเกิดได้หลายสาเหตุ เช่น ข้อมูลไม่เกาะกลุ่มกัน, ข้อมูลไม่มากพอ และ โมเดลยังมีโครงสร้างที่ไม่รัดกุม เป็นต้น โดยแต่ละสาเหตุก็จะมีเทคนิคในการแก้ไขแต่ละปัญหา ตัวอย่างในเคสนี้คือการแสดงให้เห็นเบื้องต้นถึงการทำงานแบบ Neural Network เท่านั้น

ภาคผนวก

LINK : https://drive.google.com/drive/folders/1H1Q3W2JxVBHRph3nDvggLc4TNH9ftrrN?usp=sharing

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

def data_matrix(file_path, rows, columns):
    data = np.zeros((rows, columns))
    l = 0
    with open(file_path, 'r') as file:
        for line in file:
            if l < rows:
                values = line.strip().split()
                for column in range(columns):
                    value = float(values[column])
                    data[l, column] = value
                l += 1

    return data


source_data ='flood_data_set.txt'
data = data_matrix(source_data,252,9) #training set fold1 not swap
data_normalized = data / 600
input_data = data_normalized[:,:8]
output_data = data_normalized[:,8:]

source_data2 ='flood_data_test.txt'
unsee_data = data_matrix(source_data2,63,9)
unsee_data_normalized = unsee_data / 600
unsee_input_data = unsee_data_normalized[:,:8]
unsee_output_data = unsee_data_normalized[:,8:]

#architecture
hidden_size = 3
input_size = input_data.shape[1]
output_size = output_data.shape[1]

np.random.seed(42)
weights_hidden = np.random.rand(input_size, hidden_size)
biases_hidden = np.random.rand(hidden_size)
weights_output = np.random.rand(hidden_size, output_size)
biases_output = np.random.rand(output_size)

# Initialize previous weight changes with zeros
prev_weights_output_change = np.zeros_like(weights_output)
prev_biases_output_change = np.zeros_like(biases_output)
prev_weights_hidden_change = np.zeros_like(weights_hidden)
prev_biases_hidden_change = np.zeros_like(biases_hidden)


def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x > 0, 1, 0)

epochs = 80000
learning_rate = 0.0001
momentum_rate = 0.2
i = 0
plot_epoch=[]
plot_loss=[]
```

```
93
94        # Update the previous weight changes
95        prev_weights_output_change = weights_output_change
96        prev_biases_output_change = biases_output_change
97        prev_weights_hidden_change = weights_hidden_change
98        prev_biases_hidden_change = biases_hidden_change
99
00        if(epoch % 100 == 0):
01            plot_epoch.append(epoch)
02            plot_loss.append(loss)
03
04        #print("Loss >> ",round(loss,8))
05        #print("Predicted output : \n",predicted_output)
06
07    print("Loss >> ",round(loss,8))
08    predicted_output = predicted_output *600
09    print("Predicted output : \n",predicted_output)
10
11    final_bias_hidden = biases_hidden
12    final_weights_hidden =  weights_hidden
13    fianl_bias_output = biases_output
14    fianl_weights_output = weights_output
15
16    print("------------------------------\n")
17    print("final_bias_hidden : \n", biases_hidden)
18    print("final_weights_hidden : \n", weights_hidden)
19    print("fianl_bias_output :\n", biases_output)
20    print("fianl_weights_output : \n", weights_output)
21
```

```
print("started_bias_hidden :\n", biases_hidden)
print("started_weights_hidden : \n", weights_hidden)
print("started_bias_output :\n", biases_output)
print("started_weights_output : \n", weights_output)
print("------------------------------")

for epoch in range(epochs) :
    #forward propagation
    hidden_layer_input = np.dot(input_data, weights_hidden) + biases_hidden
    hidden_layer_output = relu(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_output) + biases_output
    predicted_output = output_layer_input

    #calculate the loss (mean squared error)
    loss = np.mean((predicted_output - output_data) ** 2)
    loss = round(loss,8)

    # Backpropagation
    output_error = predicted_output - output_data
    output_gradient = output_error
    weights_output_change = (learning_rate * np.dot(hidden_layer_output.T, output_gradient)) + (momentum_rate * prev_weights_outp
    biases_output_change = (learning_rate * np.sum(output_gradient, axis=0)) + (momentum_rate * prev_biases_output_change)
    weights_output -= weights_output_change
    biases_output -= biases_output_change

    hidden_error = np.dot(output_gradient, weights_output.T) * relu_derivative(hidden_layer_input)
    weights_hidden_change = (learning_rate * np.dot(input_data.T, hidden_error)) + (momentum_rate * prev_weights_hidden_change)
    biases_hidden_change = (learning_rate * np.sum(hidden_error, axis=0)) + (momentum_rate * prev_biases_hidden_change)
    weights_hidden -= weights_hidden_change
    biases_hidden -= biases_hidden_change
```

```
122  '''
123  plt.figure(figsize=(8, 6))
124
125  plt.subplot(2, 1, 1)
126  plt.plot(plot_epoch,plot_loss)
127  plt.xlabel("Epoch")
128  plt.ylabel("LOSS")
129  plt.ylim(0.0001,0.009)
130  plt.title("at Lrate {} & Hidden {} & Momentum = {} ".format(learning_rate,hidden_size,momentum_rate))
131
132  plt.subplot(2, 1, 2)
133  plt.plot(output_data, label="Desired Output", marker='x',color ='yellow')
134  plt.plot(predicted_output/600, label="Processed Output", marker='x',color ='blue')
135  plt.title("Desired Output vs Predicted Output")
136  plt.legend()
137  plt.grid(True)
138
139  plt.tight_layout()
140  plt.show()'''
141
142  #forward propagation
143  unsee_hidden_layer_input = np.dot(unsee_input_data, final_weights_hidden) + final_bias_hidden
144  unsee_hidden_layer_output = relu(unsee_hidden_layer_input)
145  unsee_output_layer_input = np.dot(unsee_hidden_layer_output, fianl_weights_output) + fianl_bias_output
146  unsee_predicted_output = unsee_output_layer_input
147
148  #calculate the loss (mean squared error)
149  loss = np.mean((unsee_predicted_output - unsee_output_data) ** 2)
150  loss = round(loss,8)
```



```
141
142  #forward propagation
143  unsee_hidden_layer_input = np.dot(unsee_input_data, final_weights_hidden) + final_bias_hidden
144  unsee_hidden_layer_output = relu(unsee_hidden_layer_input)
145  unsee_output_layer_input = np.dot(unsee_hidden_layer_output, fianl_weights_output) + fianl_bias_output
146  unsee_predicted_output = unsee_output_layer_input
147
148  #calculate the loss (mean squared error)
149  loss = np.mean((unsee_predicted_output - unsee_output_data) ** 2)
150  loss = round(loss,8)
151
152  plt.figure(figsize=(8, 6))
153  plt.plot(unsee_output_data, label="Desired Output", marker='x',color ='yellow')
154  plt.plot(unsee_predicted_output, label="Processed Output", marker='x',color ='blue')
155  plt.title("Desired Output vs Predicted Output at loss = {}".format(loss))
156  plt.legend()
157  plt.grid(True)
158
159  plt.show()
160
```