เรื่อง Multi Layer Perceptron

จัดทำโดย

นายปัณณวิชญ์ พันธ์วงศ์

600610752

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของวิชา

CPE 261456 (Introduction to Computational Intelligence)

ภาคเรียนที่ 1 ปีการศึกษา 2563

มหาวิทยาลัยเชียงใหม่

1. คำอธิบายข้อมูล

1.1. ข้อมูล flood_dataset.txt

เป็นชุดของข้อมูลรระดับน้ำที่สะพานนวรัตน์ โดยมีข้อมูลที่สถานี 1 และ สถานี 2 ณ เวลาปัจจุบัน ,เวลาย้อนหลังไป 3 ชั่วโมง และระดับน้ำในอีก 7 ชม. ข้างหน้า รวมแล้วมีทั้งหมด 9 ข้อมูล ซึ่งการทดลองเป็นการทำนายระดับน้ำในอนาคตอีก 7 ชม.**(Regression)**

1.2. ข้อมูล cross.pat

เป็นชุดข้อมูลที่ประกอบไปด้วย ลำดับข้อมูล(p) , ข้อมูลเลขทศนิยม 2 จำนวน และจำนวนจริง 2 จำนวน รวมแล้วมีทั้งหมด 5 ข้อมูล(ใช้ทดลอง 4 ข้อมูล) ซึ่งการทดลองเป็นการทำนายกลุ่มของข้อมูล**(Classification)**

2. การเตรียมข้อมูลก่อนประมวลผล

2.1. การเตรียมข้อมูล **flood_dataset.txt**

2.1.1. Input ประกอบไปด้วย ข้อมูลระดับน้ำของทั้งสองสถานี จำนวน 8 ข้อมูล

- ระดับน้ำปัจจุบัน
- ระดับน้ำย้อนหลัง 1 ชม.
- ระดับน้ำย้อนหลัง 2 ชม.
- ระดับน้ำย้อนหลัง 3 ชม.

2.1.2. Output ประกอบไปด้วย ข้อมูลระดับน้ำสะพานนวรัตน์ในอีก 7 ชม. จำนวน 1 ข้อมูล

- ระดับน้ำสะพานนวรัตน์ ณ 7 ชม. ข้างหน้า

เนื่องจากการทดลองใช้ **Activation sigmoid** ในการทดลอง จึงได้ทำการเปลี่ยนแปลงข้อมูล โดยใช้หลักการ **Min-max normalization** ซึ่งได้กำหนดช่วง min และ max อยู่ที่ [0,1]

2.2. การเตรียมข้อมูล **cross.pat**

2.2.1. Input ประกอบไปด้วย ข้อมูลเลขทศนิยม 2 จำนวน

2.2.2. Output ประกอบไปด้วย ข้อมูลจำนวนจริง 0 และ 1 ที่บ่งบอกถึงลักษณะของกลุ่มข้อมูล(Class) 2 กลุ่ม และ ได้ทำการแปลง ค่าจาก 0 เป็น 0.1 และ 1 เป็น 0.9

3. การทดลองประมวลผลข้อมูล

3.1. วิธีการทดลอง

3.1.1. ทำการเตรียมข้อมูลก่อนทำการทดลองตามข้อ 2

3.1.2. ทำการสุ่มลำดับของข้อมูล

3.1.3. ทำการแบ่ง **10 Cross validation** ตามปริมาณของข้อมูล

3.1.4. นำข้อมูลที่แบ่งจากข้อ 3.1.2 เข้า **Multiple Layer Neural Network** โดยแต่ละ **Epoch** ได้ทำการสลับลำดับของข้อมูลในการ **Forward propagation** และ **Backward propagation**

3.1.5. ทำการทดลองในแต่ละหัวข้อ ซึ่งค่าของ **Hyperparameter** เป็นค่าที่สุ่มขึ้นมา และการทดลองในข้อถัดๆไปจะใช้ค่าของ **Hyperparameter** ที่ดีที่สุดในข้อก่อนหน้ามาเป็นค่าหลักของการทดลองข้อนั้นๆ

3.1.6. แสดงผลการทดลองซึ่งใช้สมการ **Mean squared error** ในการวิเคราะห์ผล

- สำหรับการทดลอง **Flood_dataset** จะแสดงผลในรูปของกราฟและ แผนภูมิแท่ง
- สำหรับการทดลอง **Cross** จะแสดงผลในรูปของ **Confusion matrix** และแผนภูมิแท่ง

<u>อธิบายแผนภูมิแท่ง</u>

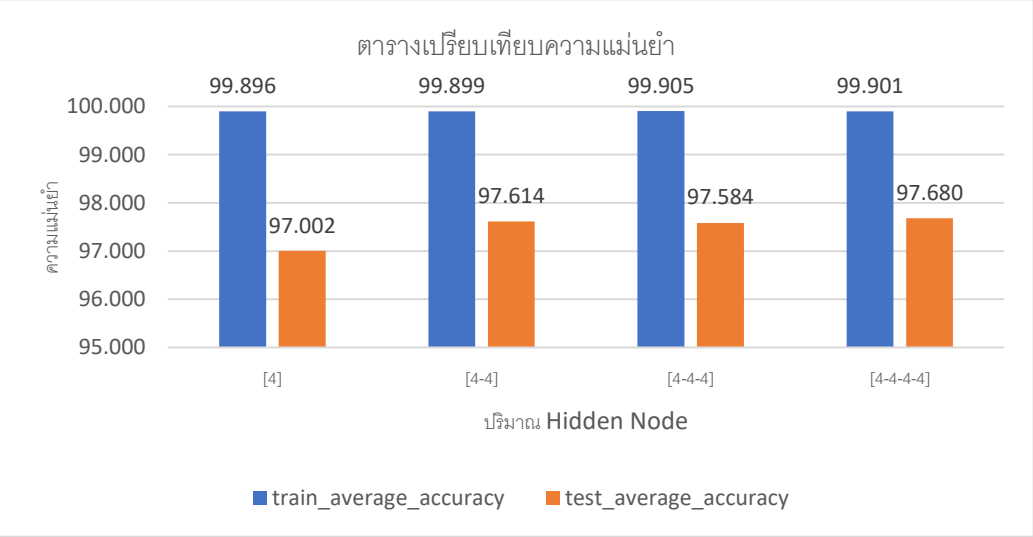
<u>แผนภูมิแท่งสีน้ำเงิน</u> หมายถึง ความแม่นยำในการเทรน(Train) โดยเฉลี่ย และ<u>แผนภูมิแท่งสีส้ม</u> หมายถึง ความแม่นยำในการทดสอบ(Test) โดยเฉลี่ย ซึ่งคำนวณจาก **Mean Absolute error** โดยนำข้อมูลเข้าสู่ **Multiple Layer Neural Network** เพื่อหาค่าความแม่นยำในการทดสอบ

3.1.7. สรุปผลการทดลอง

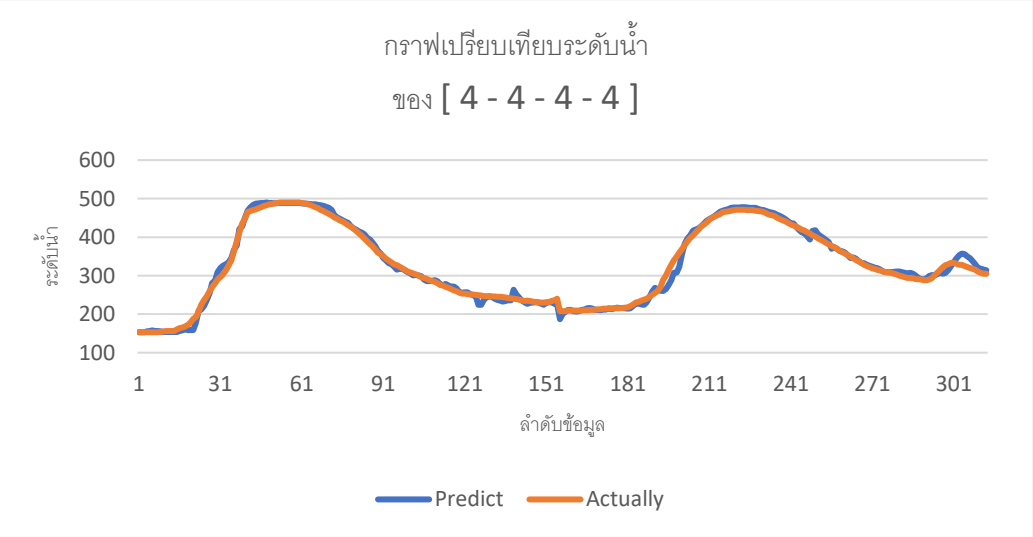
3.2. การทดลองเปลี่ยนแปลงจำนวน Hidden Layer

3.2.1. การทดลองเกี่ยวกับ flood_dataset.txt

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ **Hidden Layer** ซึ่งแต่ละ **Hidden Layer** จะประกอบไปด้วย 4 Nodes โดยทำการทดลองโดยใช้ epochs = 1,000 , learning rate = 0.5 , momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)

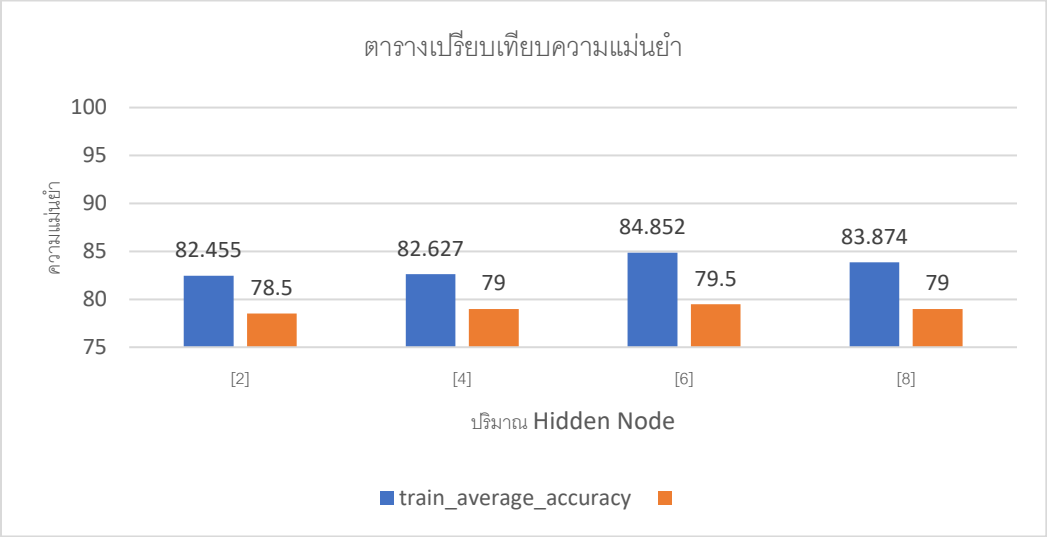


พบว่า 4-4-4-4 เป็นปริมาณ **Hidden nodes และ Hidden layers** ที่เหมาะสมที่สุดในการทดลองครั้งนี้ โดยเปรียบเทียบจากความแม่นยำโดยเฉลี่ยในการทดสอบ ซึ่งได้ความแม่นยำถึง 97.68%



3.2.2. การทดลองเกี่ยวกับ cross.pat

ผู้ทดลองได้ทำการทดลองกำหนดให้มี **1 Hidden Layer** เปลี่ยนแปลงจำนวณ Hidden Node โดยทำการทดลองโดยใช้ epochs = 1,000 , learning rate = 0.5 , momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)



พบว่า จำนวน Hidden layer = 1 , Hidden node = 6 เป็นปริมาณ Hidden layer และ Hidden node ที่เหมาะสมที่สุดในการทดลองครั้งนี้ โดยเปรียบเทียบจากความแม่นยำในการทดสอบ ซึ่งมีความแม่นยำถึง 97.5%

	[0 1]	*[1 0]*
Confusion Matrix [2]		
[0 1]	79	21
[1 0]	22	78
ความเร็วในการ converge	**332.52 seconds**	

Confusion Matrix [4]	*[1 0]*	*[0 1]*
[1 0]	80	20
[0 1]	22	78
ความเร็วในการ converge	**356.85 seconds**	

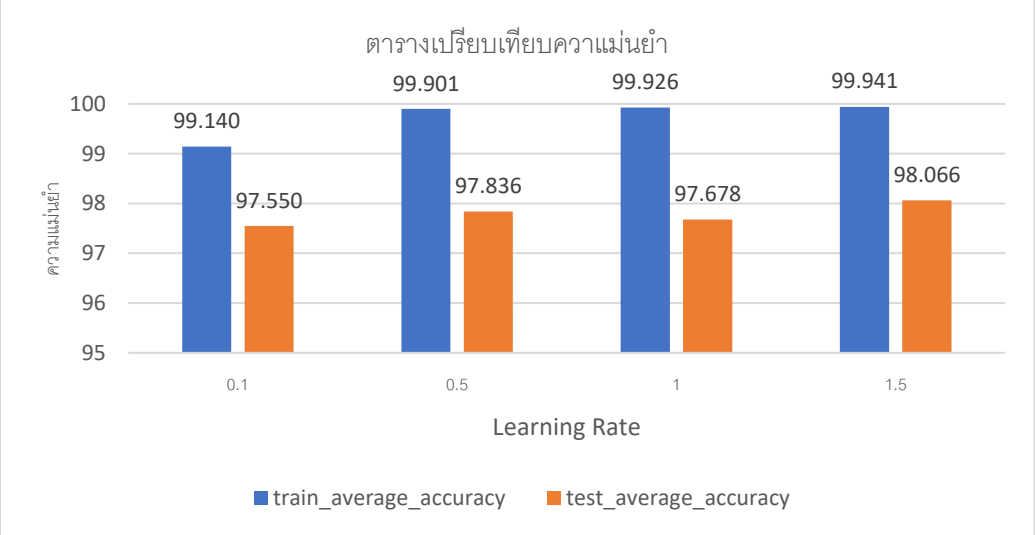
Confusion Matrix [6]	*[0 1]*	*[1 0]*
[0 1]	80	20
[1 0]	21	79
ความเร็วในการ converge	381.15 seconds	

Confusion Matrix [8]	*[0 1]*	*[1 0]*
[0 1]	79	21
[1 0]	21	79
ความเร็วในการ converge	406.38 seconds	

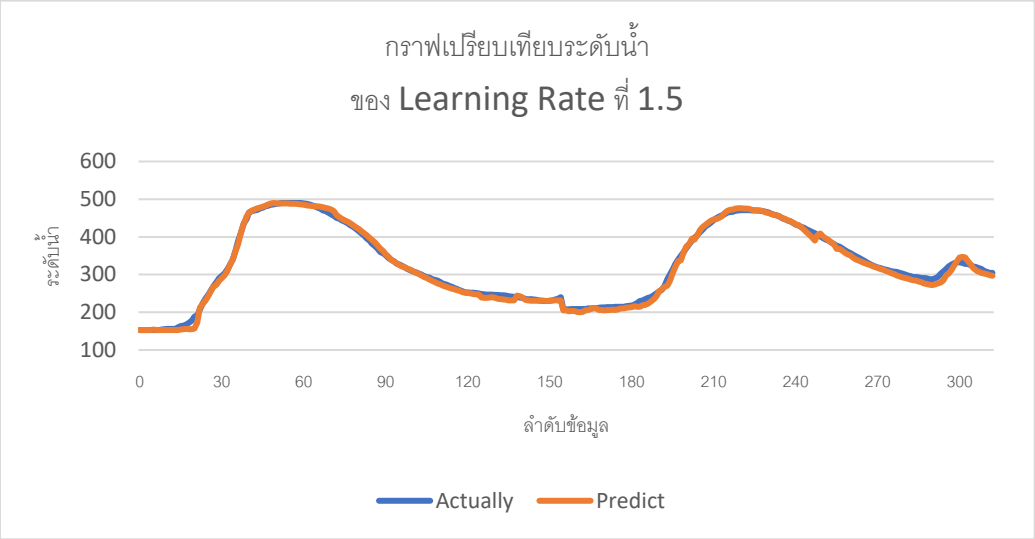
3.3. การทดลองเปลี่ยนแปลงจำนวน Learning rate

3.3.1. การทดลองเกี่ยวกับ flood_dataset.txt

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวน **Learning rate** โดยการทดลองจะมีค่า Learning rate 0.1 ,0.5 ,1 และ 1.5 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [4 – 4 – 4 – 4] (อ้างอิงจากข้อ 3.2.1) , momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)

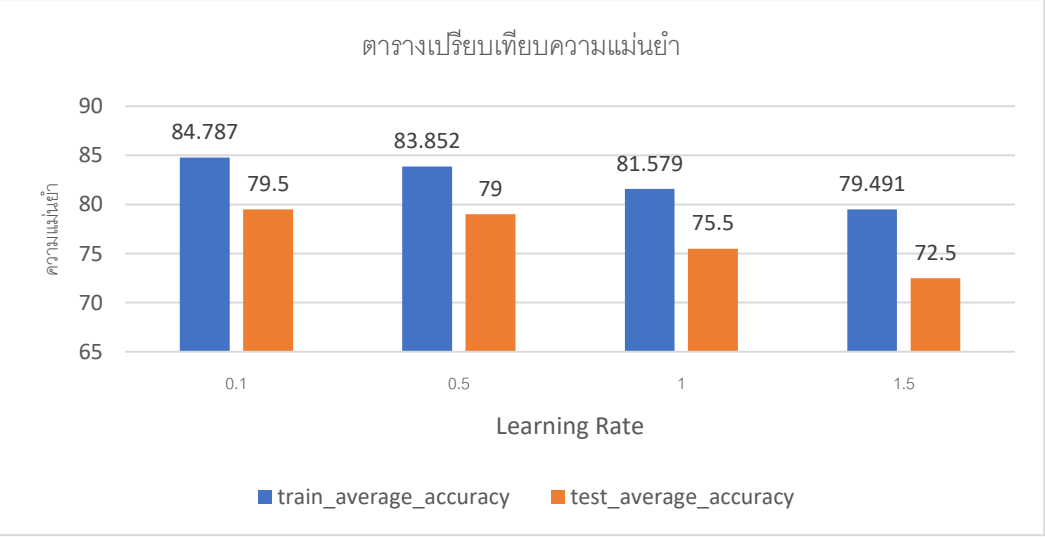


พบว่าค่า learning rate ที่เหมาะสมคือ 1.5 โดยอ้างอิงจาก ความแม่นยำเฉลี่ยในการทดสอบ



3.3.2. การทดลองเกี่ยวกับ cross.pat

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวน **Learning rate** โดยการทดลองจะมีค่า Learning rate ที่ 0.1 ,0.5 ,1 และ 1.5 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [6](อ้างอิงจากข้อ 3.2.2) , momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)



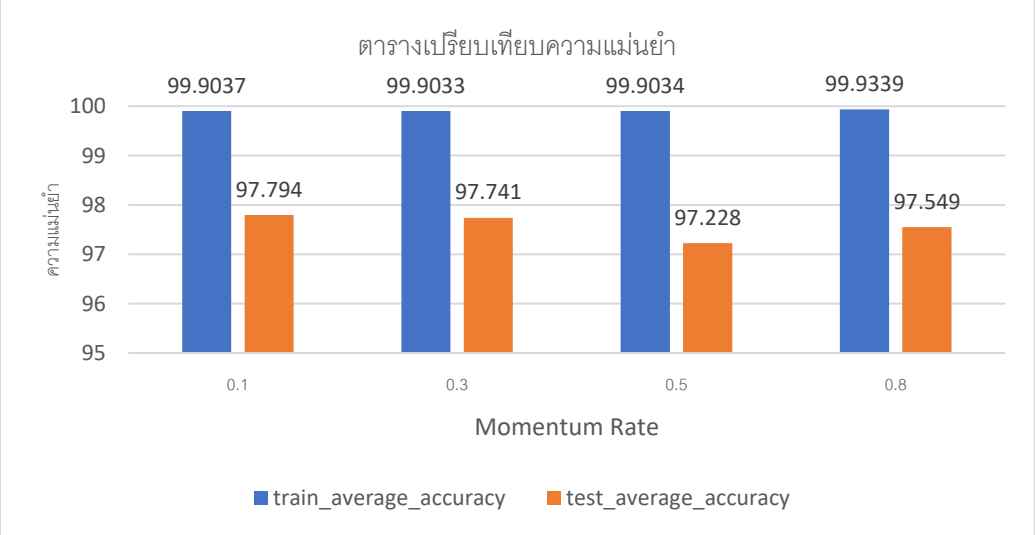
พบว่าค่า learning rate ที่เหมาะสมคือ 0.1 และมีค่าที่ใกล้เคียงกันคือ 0.5 โดยอ้างอิงจากความแม่นยำเฉลี่ยในการทดสอบ

Confusion Matrix L_R 0.1	*[0 1]*	*[1 0]*
[0 1]	*80*	*20*
[1 0]	*21*	*79*
ความเร็วในการ converge	**381.15 seconds**	

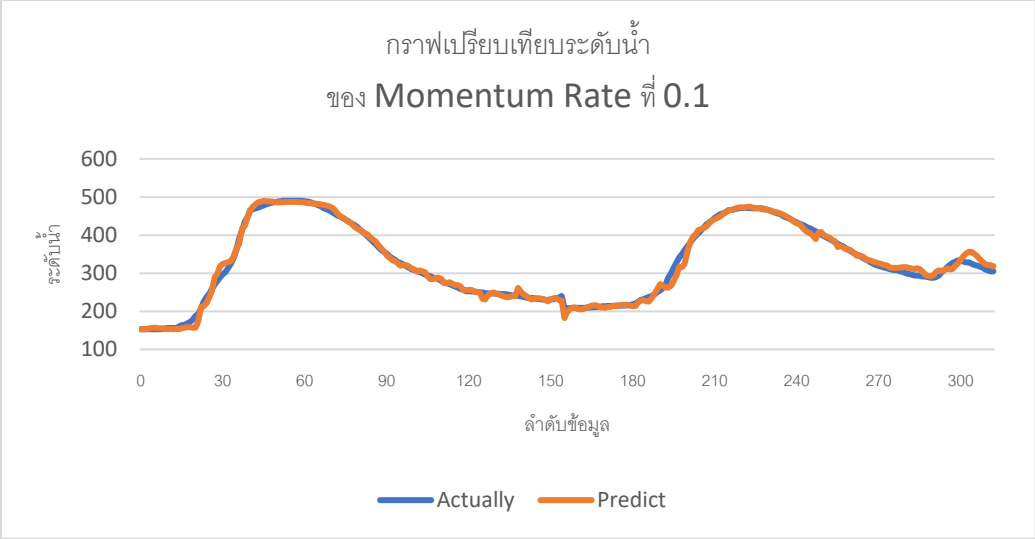
3.4. การทดลองเปลี่ยนแปลงจำนวน Momentum rate

3.4.1. การทดลองเกี่ยวกับ flood_dataset.txt

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ **Momentum rate** โดยการทดลองจะมีค่า Momentum rate ที่ 0.1 ,0.3 ,0.5 และ 0.8 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [4 – 4 – 4 - 4](อ้างอิงจากข้อ 3.2.1) , learning rate = 1.5 (อ้างอิงจากข้อ 3.3.1) และทำการสุ่ม weight ในช่วง (0,1)

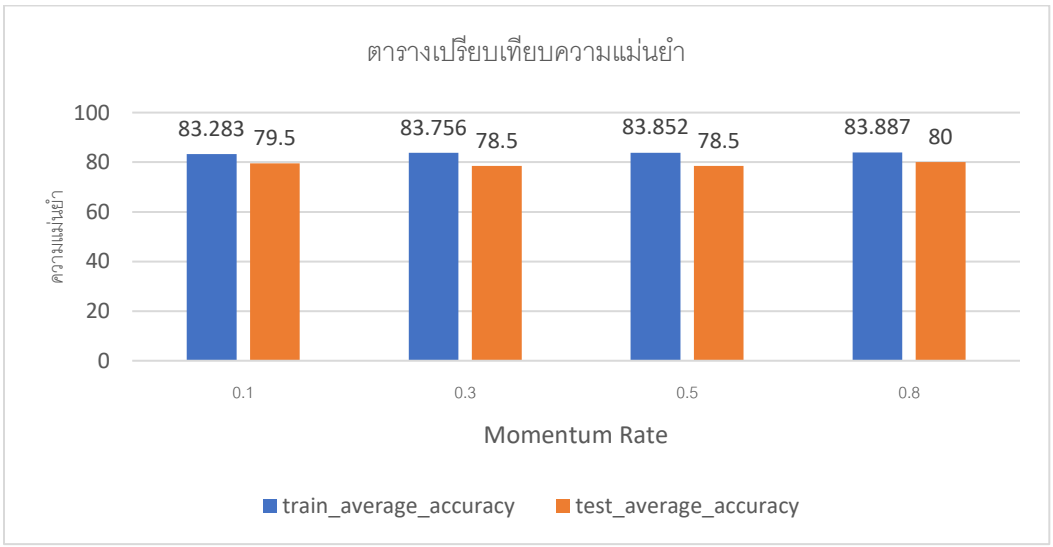


พบว่าค่า Momentum rate ที่เหมาะสมคือ 0.1 โดยอ้างอิงจาก ความแม่นยำเฉลี่ยนการทดสอบ



3.4.2. การทดลองเกี่ยวกับ cross.pat

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Momentum rate โดยการทดลองจะมีค่า Momentum rate ที่ 0.1 ,0.3 ,0.5 และ 0.8 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [6](อ้างอิงจากข้อ 3.2.2) ,Learning rate = 0.1(อ้างอิงจากข้อ 3.3.2) และทำการสุ่ม weight ในช่วง (0,1)



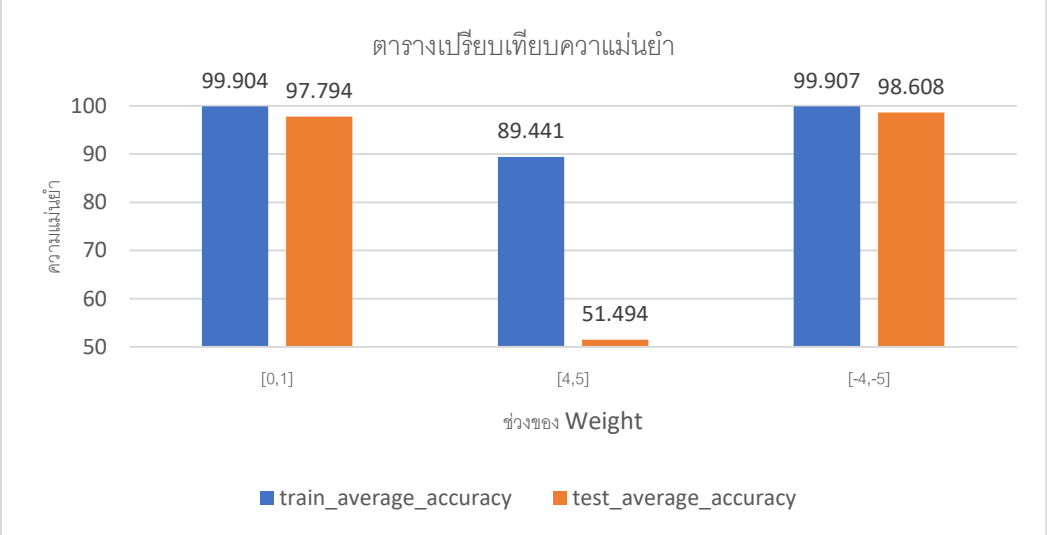
พบว่าค่า Momentum rate ที่เหมาะสมคือ 0.8 และมีค่าที่ใกล้เคียงกันคือ 0.1, 0.5 โดยอ้างอิงจาก ความแม่นยำในการทดสอบโดยเฉลี่ย

Confusion Matrix M_R 0.8	*[0 1]*	*[1 0]*
[0 1]	76	24
[1 0]	16	84
ความเร็วในการ converge	381.10 seconds	

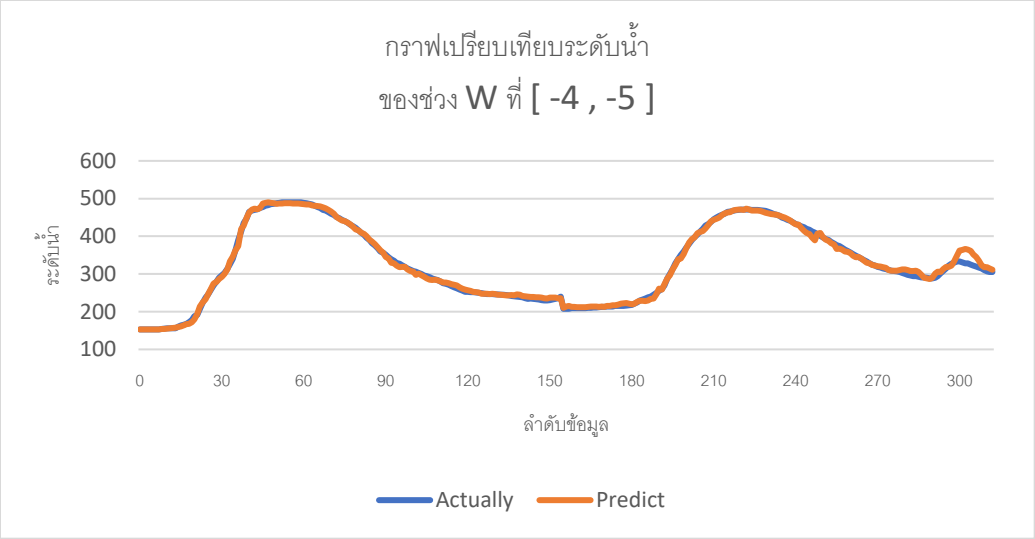
3.5. การทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน

3.5.1. การทดลองเกี่ยวกับ flood_dataset.txt

ผู้ทดลองได้ทำการทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน โดยการทดลองจสุ่ม Weight ในช่วง [0 , 1] , [4 , 5], และ [-4 , -5] ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [4 - 4 - 4 - 4](อ้างอิงจากข้อ 3.2.1) , learning rate = 1.5 (อ้างอิงจากข้อ 3.3.1) และ momentum rate = 0.1(อ้างอิงจากข้อ 3.4.1)

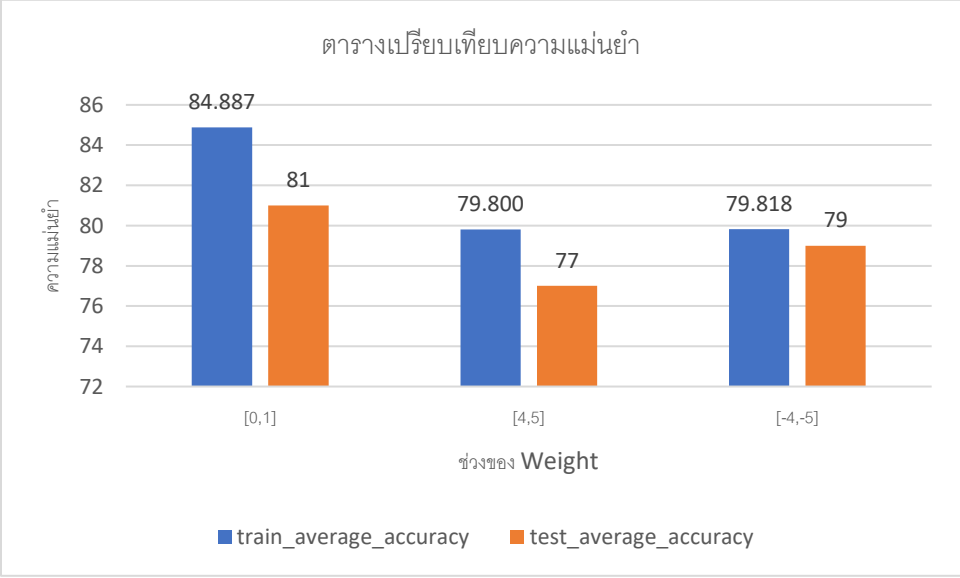


พบว่าช่วงของ Weight ที่เหมาะสมคือ -4 ถึง -5 ซึ่งมีความแม่นยำถึง 98.608 โดยอ้างอิงจากความแม่นยำในการทดสอบโดยเฉลี่ย



3.5.2. การทดลองเกี่ยวกับ cross.pat

ผู้ทดลองได้ทำการทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน โดยการทดลองจสุ่ม Weight ในช่วง [0 , 1] , [4 , 5], และ [-4 , -5] ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [6](อ้างอิงจากข้อ 3.2,2) , learning rate = 0.1(อ้างอิงจากข้อ 3.3.2) และ momentum rate = 0.8(อ้างอิงจากข้อ 3.4.2)



พบว่าช่วงของ Weight ที่เหมาะสมคือ 0 ถึง ซึ่งมีความแม่นยำถึง 81% รองลงมา โดยอ้างอิงจาก ความแม่นยำในการทดสอบโดยเฉลี่ย

Confusion Matrix W [0,1]	*[0 1]*	*[1 0]*
[0 1]	78	22
[1 0]	16	84
ความเร็วในการ converge	380.17 seconds	

4. สรุปผลการทดลอง

สรุปผลการทดลองในแต่ละชุดข้อมูลได้ในรูปแบบตารางดังนี้

ชุดข้อมูล	**การทดลอง**				
	NN Layer	Learning rate	Momentum rate	Weight	Accuracy
flood_dataset.txt	8 – 4 – 4 – 4 - 4– 1	1.5	0.1	[-4 , -5]	98.61%
cross.pat	2 – 6 – 2	0.8	0.2	[0 , 1]	81%

จากผลการทดลองในแต่ละการทดลองผลที่ได้มีค่าค่อนข้างใกล้เคียงกัน ซึ่งแตกต่างกันในหลักจุดทศนิยม อย่างเช่น การทดลองในการหา **Momentum rate** ผลลัพธ์ที่ได้ค่าค่อนข้างใกล้เคียงกัน ดังนั้นผลลัพธ์ที่ได้จริงๆจึงนำค่าที่ได้มากที่สุดมาทำการสรุปผล

5. ภาคผนวก

```
1.  import numpy as np
2.  from random import randint
3.  import time
4.  import copy
5.
6.  class NeuralNetwork(object):
7.      def __init__(self, hiddenSize, inputSize, outputSize):
8.          # initiate layers
9.          self.inputSize = inputSize
10.         self.outputSize = outputSize
11.         self.hiddenSize = hiddenSize
12.
13.         layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
14.
15.         # initiate weights
16.         weights = []
17.         for i in range(len(layers)-1):
18.             w = -5*np.random.rand(layers[i], layers[i+1])+1
19.             weights.append(w)
20.         self.weights = weights
21.
22.         # initiate weights_t-1
23.
24.         self.weights_befoe  = copy.deepcopy(self.weights)
25.         self.weights_next  = copy.deepcopy(self.weights)
26.
27.
28.         # initiate bias
29.         bias = []
30.         for i in range(len(layers)-1):
31.             b = np.random.rand(layers[i+1])
32.             bias.append(b)
33.         self.bias = bias
34.
35.         # initiate bias_t-1
36.         self.bias_before  = copy.deepcopy(self.bias)
37.         self.bias_next  = copy.deepcopy(self.bias)
38.
39.
40.         # initiate activations
41.         activations = []
42.         for i in range(len(layers)):
43.             a = np.zeros(layers[i])
```

```
44.             activations.append(a)
45.         self.activations = activations
46.
47.         # initiate gradient_b
48.         derivatives_b = []
49.         for i in range(len(layers) - 1):
50.             d = np.zeros(layers[i+1])
51.             derivatives_b.append(d)
52.         self.derivatives_b = derivatives_b
53.
54.         # initiate gradient_w
55.         derivatives_w = []
56.         for i in range(len(layers) - 1):
57.             d = np.zeros((layers[i], layers[i + 1]))
58.             derivatives_w.append(d)
59.         self.derivatives_w = derivatives_w
60.
61.         # initiate average_err
62.         self.average_err = 0
63.
64.     def sigmoid(self, s, deriv=False):
65.         if (deriv == True):
66.             return s * (1-s)
67.         return 1/(1 + np.exp(-s))
68.
69.     def feedForward(self, X):
70.         activations = X
71.         self.activations[0] = X
72.         for i, w in enumerate(self.weights):
73.             # calculate NN_input
74.             v = np.dot(activations, w)
75.             # calculate the activations
76.             b = self.bias[i]
77.             activations = self.sigmoid(v+b)
78.             self.activations[i+1] = activations
79.         return activations
80.
81.     def backPropagate(self, error):
82.         for i in reversed(range(len(self.derivatives_w))):
83.
84.             # get activation for previous layer
85.             activations = self.activations[i+1]
86.
87.             # apply sigmoid derivative function
88.             delta = error * self.sigmoid(activations, deriv=True)
89.
90.             # reshape delta as to have it as a 2d array
91.             delta_re = delta.reshape(delta.shape[0], -1).T
```

```
92.
93.             self.derivatives_b[i] = copy.deepcopy(delta)
94.
95.             # get activations for current layer
96.             current_activations = self.activations[i]
97.
98.             # reshape activations as to have them as a 2d column matrix
99.             current_activations = current_activations.reshape(
100.                 current_activations.shape[0], -1)
101.
102.            # save derivative after applying matrix multiplication
103.            self.derivatives_w[i] = np.dot(current_activations, delta_re)
104.
105.
106.            # backpropogate the next error
107.            error = np.dot(delta, self.weights[i].T)
108.
109.    def train(self, X, Y, epochs, learning_rate,momentumRate):
110.        # now enter the training loop
111.        for i in range(epochs):
112.            sum_errors = 0
113.
114.            # Random data
115.            seed = randint(1, epochs*100)
116.
117.            np.random.seed(seed)
118.            np.random.shuffle(X)
119.
120.            np.random.seed(seed)
121.            np.random.shuffle(Y)
122.
123.            # iterate through all the training data
124.            for j, input in enumerate(X):
125.                target = Y[j]
126.
127.                # activate the network!
128.                output = self.feedForward(input)
129.
130.                error = target - output
131.
132.                self.backPropagate(error)
133.                # now perform gradient descent on the derivatives
134.                # (this will update the weights
135.
136.                self.gradient_descent(learning_rate,momentumRate)
137.
138.                # keep track of the MSE for reporting later
139.                sum_errors += self._mse(target, output)
```

```
140.
141.             # Epoch complete, report the training error
142.             print("Error: {} at epoch {}".format(round(sum_errors / len(X) , 5), i+1))
143.
144.         self.average_err = round(sum_errors / len(X) , 5)
145.
146.         print("Training complete! : ",sum_errors/len(X))
147.         print("=====")
148.
149.     def gradient_descent(self, learningRate=1,momentumRate=1):
150.
151.         # update the weights by stepping down the gradient
152.
153.         for i in range(len(self.weights)):
154.
155.             weights = self.weights[i]
156.             weights_befoe = self.weights_befoe[i]
157.             weights_next = self.weights_next[i]
158.
159.             bias = self.bias[i]
160.             bias_before = self.bias_before[i]
161.             bias_next = self.bias_next[i]
162.
163.             derivatives_w = self.derivatives_w[i]
164.
165.             derivatives_b = self.derivatives_b[i]
166.
167.             weights_next += (derivatives_w * learningRate) + ((weights-weights_befoe)*momentumRate)
168.
169.             bias_next += (derivatives_b * learningRate) + ((bias-bias_before)*momentumRate)
170.
171.
172.         self.weights_befoe = copy.deepcopy(self.weights)
173.         self.weights = copy.deepcopy(self.weights_next)
174.
175.         self.bias_before = copy.deepcopy(self.bias)
176.         self.bias = copy.deepcopy(self.bias_next)
177.
178.     def _mse(self, target, output):
179.         return np.average((target - output) ** 2)
180.
181. def _normalization(NewMax,NewMin,OldMax,OldMin,OldValue):
182.
183.     OldRange = (OldMax - OldMin)
184.     NewRange = (NewMax - NewMin)
185.     NewValue = (((OldValue - OldMin) * NewRange) / OldRange) + NewMin
186.
187.     return  NewValue
```

```
188.
189.  def _readfile(txt):
190.
191.      output = []
192.      input = []
193.
194.      if txt == "Flood_dataset.txt":
195.                  # import data set
196.          with open("Flood_dataset.txt", "r") as f:
197.              content = f.readlines()
198.          del content[0:3]
199.
200.          # split data set
201.          data = []
202.          for X in content:
203.              data.append(X.split())
204.
205.          # convert data to list
206.          output = [list(map(int, X[8:])) for X in data]
207.          input = [list(map(int, X[:8])) for X in data]
208.
209.
210.
211.      elif txt == "cross.pat":
212.          # import data set
213.          with open("cross.pat", "r") as f:
214.              content = f.readlines()
215.
216.          for i,X in enumerate(content):
217.              if X[0] != 'p':
218.                  if (i+1)%3 == 0:
219.                      a,b = X.split()
220.                      output.append([int(a),int(b)])
221.                  else:
222.                      a,b = X.split()
223.                      input.append([float(a),float(b)])
224.
225.      else:
226.          print("-- Not found a data / Missing data --")
227.
228.      # Convert to np_array
229.
230.      input = np.array(input)
231.      output = np.array(output)
232.
233.      seed = randint(1, len(input)*100)
234.
235.      np.random.seed(seed)
```

```
236.     np.random.shuffle(input)
237.
238.     np.random.seed(seed)
239.     np.random.shuffle(output)
240.     # Shape input and output
241.
242.     inputSize = input.shape[1]
243.     outputSize = output.shape[1]
244.
245.     return input, output, inputSize, outputSize
246.
247. def cross_validations_split(shape,folds):
248.     fold_size = int(shape * folds/100)
249.     k = 0
250.     index = []
251.     for i in range(1,folds+1):
252.         if i < folds:
253.             index.append([k,i*fold_size])
254.         else:
255.             index.append([k,shape])
256.         k = i*fold_size
257.     return index
258.
259. def _confusion_matrix(predict,actually):
260.
261.     def _create_matrix(label,act):
262.
263.         matrix = np.array([[0, 0], [0, 0]])
264.
265.         if act[0] == 0 :
266.             if label[0] == 0 :
267.                 matrix[0][0] += 1
268.             else :
269.                 matrix[0][1] += 1
270.         else :
271.             if label[0] == 1 :
272.                 matrix[1][1] += 1
273.             else :
274.                 matrix[1][0] += 1
275.
276.         return matrix
277.
278.     confusion_matrix = np.array([[0, 0], [0, 0]])
279.
280.     for i in range(len(predict)):
281.
282.         if predict[i][0] >= predict[i][1]:
283.             label = [1,0]
```

```
284.            else :
285.                label = [0,1]
286.            matrix = _create_matrix(label,actually[i])
287.            confusion_matrix = np.add(confusion_matrix,matrix)
288.
289.        return confusion_matrix
290.
291.
292.    filename = input('Enter file name : ')
293.    X,Y,inputSize,outputSize = _readfile(filename)
294.    print("What Size of Hidden layer Neural Network ?")
295.    print(" -- Example : '4-2-2' --")
296.    print(" -- Hidden layer have 3 layers and 4,2,2 nodes respectively -- ")
297.    hiddenSize = input('Enter hidden size : ')
298.    hiddenSize = hiddenSize.split("-")
299.    hiddenSize = list(map(int, hiddenSize))
300.    learning_rate = input('Enter learning rate : ')
301.    momentum_rate = input('Enter momentum rate : ')
302.    epochs = input('Enter epoch : ')
303.
304.    if filename == "Flood_dataset.txt":
305.        X_train = _normalization(1,0,X.max(),X.min(),X)
306.        Y_train = _normalization(1,0,Y.max(),Y.min(),Y)
307.    else :
308.        X_train = X
309.        Y_train = _normalization(0.9,0.1,Y.max(),Y.min(),Y)
310.
311.
312.    NN = NeuralNetwork(hiddenSize, inputSize, outputSize)
313.
314.    train_average_accuracy = 0
315.    test_average_accuracy = 0
316.
317.    for a,b in cross_validations_split(X_train.shape[0],10):
318.
319.        inTest = np.concatenate((X_train[:a],X_train[b+1:]))
320.        outTest = np.concatenate((Y_train[:a],Y_train[b+1:]))
321.        NN.train(inTest, outTest, int(epochs) , float(learning_rate)  , float(momentum_rate))
322.        train_average_accuracy += (1 - NN.average_err)/10
323.        test_average_accuracy += (1- np.sum(NN._mse(NN.feedForward(X_train[a:b,:]),Y_train[a:b,:]),axis=0))/10
324.
325.    print("Test_average  : ",test_average_accuracy)
326.    print("Train_average : ",train_average_accuracy)
327.
328.    if filename != "Flood_dataset.txt":
329.        print(_confusion_matrix(Y_predict,Y))
330.    #matrix = np.float64(matrix)
```

5.1. ตัวอย่างการทำงานของโปรแกรม

```
Enter file name : Flood_dataset.txt
What Size of Hidden layer Neural Network ?
 -- Example : '4-2-2' --
 -- Hidden layer have 3 layers and 4,2,2 nodes respectively --
Enter hidden size : 4-4-4-4
Enter learning rate : 1.5
Enter momentum rate : 0.4
Enter epoch : 1000
```

```
 Error: 0.00671 at epoch 33
 Error: 0.00665 at epoch 34
 Error: 0.00642 at epoch 35
 Error: 0.00602 at epoch 36
 Error: 0.00587 at epoch 37
 Error: 0.00636 at epoch 38
 Error: 0.00601 at epoch 39
 Error: 0.00574 at epoch 40
 Error: 0.00557 at epoch 41
 Error: 0.00588 at epoch 42
 Error: 0.00587 at epoch 43
 Error: 0.00594 at epoch 44
 Error: 0.0057 at epoch 45
 Error: 0.00501 at epoch 46
 Error: 0.00471 at epoch 47
 Error: 0.00486 at epoch 48
 Error: 0.00537 at epoch 49
```