**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. 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. ข้อมูล cross.pat

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

1. **การเตรียมข้อมูลก่อนประมวลผล**
   1. การเตรียมข้อมูล **flood\_dataset.txt**
      1. Input ประกอบไปด้วย ข้อมูลระดับน้ำของทั้งสองสถานีจำนวน 8 ข้อมูล
         * ระดับน้ำปัจจุบัน
         * ระดับน้ำย้อนหลัง 1 ชม.
         * ระดับน้ำย้อนหลัง 2ชม.
         * ระดับน้ำย้อนหลัง 3 ชม.
      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]

* 1. การเตรียมข้อมูล **cross.pat**
     1. Input ประกอบไปด้วย ข้อมูลเลขทศนิยม 2 จำนวน
     2. Output ประกอบไปด้วย ข้อมูลจำนวนจริง 0 และ 1ที่บ่งบอกถึงลักษณะของกลุ่มข้อมูล(Class) 2 กลุ่ม และ ได้ทำการแปลง ค่าจาก 0 เป็น 0.1 และ 1 เป็น 0.9

1. **การทดลองประมวลผลข้อมูล**
   1. **วิธีการทดลอง**
      1. ทำการเตรียมข้อมูลก่อนทำการทดลองตามข้อ 2
      2. ทำการสุ่มลำดับของข้อมูล
      3. ทำการแบ่ง **10 Cross validation** ตามปริมาณของข้อมูล
      4. นำข้อมูลที่แบ่งจากข้อ 3.1.2 เข้า **Multiple Layer Neural Network** โดยแต่ละ **Epoch** ได้ทำการสลับลำดับของข้อมูลในการ **Forward propagation** และ **Backward propagation**
      5. ทำการทดลองในแต่ละหัวข้อ ซึ่งค่าของ **Hyperparameter** เป็นค่าที่สุ่มขึ้นมา และการทดลองในข้อถัดๆไปจะใช้ค่าของ **Hyperparameter** ที่ดีที่สุดในข้อก่อนหน้ามาเป็นค่าหลักของการทดลองข้อนั้นๆ
      6. แสดงผลการทดลองซึ่งใช้สมการ **Mean squared error** ในการวิเคราะห์ผล
         * สำหรับการทดลอง **Flood\_dataset** จะแสดงผลในรูปของกราฟและ แผนภูมิแท่ง
         * สำหรับการทดลอง **Cross** จะแสดงผลในรูปของ **Confusion matrix** และแผนภูมิแท่ง

**อธิบายแผนภูมิแท่ง**

แผนภูมิแท่งสีน้ำเงิน หมายถึง ความแม่นยำในการเทรน(Train) โดยเฉลี่ย และแผนภูมิแท่งสีส้ม หมายถึง ความแม่นยำในการทดสอบ(Test) โดยเฉลี่ย ซึ่งคำนวณจาก **Mean Absolute error** โดยนำข้อมูลเข้าสู่ **Multiple Layer Neural Network** เพื่อหาค่าความแม่นยำในการทดสอบ

* + 1. สรุปผลการทดลอง
  1. **การทดลองเปลี่ยนแปลงจำนวน Hidden Layer**
     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%

จากรูป กราฟเปรียบเทียบระดับน้ำ พบว่า ในช่วงของลำดับข้อมูลที่ 121-151 ,181-211 และ 301-313 ทำนายได้ค่อนข้างผิดพลาด

* + 1. **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%

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 2 ] | *[ 0 1 ]* | *[ 1 0 ]* |
| **[ 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** | |

* 1. **การทดลองเปลี่ยนแปลงจำนวน Learning rate**
     1. **การทดลองเกี่ยวกับ flood\_dataset.txt**

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ **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

จาก กราฟเปรียบเทียบระดับน้ำ พบว่าผลลัพธ์ค่อนข้างดีกว่าข้อ 3.2.1

* + 1. **การทดลองเกี่ยวกับ cross.pat**

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ **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 โดยอ้างอิงจาก   
ความแม่นยำเฉลี่ยในการทดสอบ

|  |  |  |
| --- | --- | --- |
| Confusion Matrix L\_R 0.1 | *[ 0 1 ]* | *[ 1 0 ]* |
| **[ 0 1 ]** | *80* | 20 |
| **[ 1 0 ]** | *21* | 79 |
| **ความเร็วในการ converge** | **381.09 seconds** | |

* 1. **การทดลองเปลี่ยนแปลงจำนวน Momentum rate**
     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

จาก กราฟเปรียบเทียบระดับน้ำ พบว่า ในช่วงลำดับข้อมูลที่ 300-313 มีความคลาดเคลื่อน

* + 1. **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** | |

* 1. **การทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน**
     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 โดยอ้างอิงจาก ความแม่นยำในการทดสอบโดยเฉลี่ย

จาก กราฟเปรียบเทียบระดับน้ำ พบว่าช่วงข้อมูลที่ 300-313 ข้อมูลที่ทำนายได้ค่อนข้างคลาดเคลื่อนจากความเป็นจริง

* + 1. **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** | |

1. **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 ดังนั้นผลลัพธ์ที่ได้ จึงนำค่าที่ได้มากที่สุดมาทำการสรุปผล

1. **ภาคผนวก**

**Source code :** [click](../600610752.py)

1. import numpy as np
2. from random import randint
3. import time
4. import copy
5. class NeuralNetwork(object):
6. def \_\_init\_\_(self, hiddenSize, inputSize, outputSize):
7. # initiate layers
8. self.inputSize = inputSize
9. self.outputSize = outputSize
10. self.hiddenSize = hiddenSize
11. layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
12. # initiate weights
13. weights = []
14. for i in range(len(layers)-1):
15. w = -5\*np.random.rand(layers[i], layers[i+1])+1
16. weights.append(w)
17. self.weights = weights
18. # initiate weights\_t-1
19. self.weights\_befoe = copy.deepcopy(self.weights)
20. self.weights\_next = copy.deepcopy(self.weights)
21. # initiate bias
22. bias = []
23. for i in range(len(layers)-1):
24. b = np.random.rand(layers[i+1])
25. bias.append(b)
26. self.bias = bias
27. # initiate bias\_t-1
28. self.bias\_before = copy.deepcopy(self.bias)
29. self.bias\_next = copy.deepcopy(self.bias)
30. # initiate activations
31. activations = []
32. for i in range(len(layers)):
33. a = np.zeros(layers[i])
34. activations.append(a)
35. self.activations = activations
36. # initiate gradient\_b
37. derivatives\_b = []
38. for i in range(len(layers) - 1):
39. d = np.zeros(layers[i+1])
40. derivatives\_b.append(d)
41. self.derivatives\_b = derivatives\_b
42. # initiate gradient\_w
43. derivatives\_w = []
44. for i in range(len(layers) - 1):
45. d = np.zeros((layers[i], layers[i + 1]))
46. derivatives\_w.append(d)
47. self.derivatives\_w = derivatives\_w
48. # initiate average\_err
49. self.average\_err = 0
50. def sigmoid(self, s, deriv=False):
51. if (deriv == True):
52. return s \* (1-s)
53. return 1/(1 + np.exp(-s))
54. def feedForward(self, X):
55. activations = X
56. self.activations[0] = X
57. for i, w in enumerate(self.weights):
58. # calculate NN\_input
59. v = np.dot(activations, w)
60. # calculate the activations
61. b = self.bias[i]
62. activations = self.sigmoid(v+b)
63. self.activations[i+1] = activations
64. return activations
65. def backPropagate(self, error):
66. for i in reversed(range(len(self.derivatives\_w))):
67. # get activation for previous layer
68. activations = self.activations[i+1]
69. # apply sigmoid derivative function
70. delta = error \* self.sigmoid(activations, deriv=True)
71. # reshape delta as to have it as a 2d array
72. delta\_re = delta.reshape(delta.shape[0], -1).T
73. self.derivatives\_b[i] = copy.deepcopy(delta)
74. # get activations for current layer
75. current\_activations = self.activations[i]
76. # reshape activations as to have them as a 2d column matrix
77. current\_activations = current\_activations.reshape(
78. current\_activations.shape[0], -1)
79. # save derivative after applying matrix multiplication
80. self.derivatives\_w[i] = np.dot(current\_activations, delta\_re)
81. # backpropogate the next error
82. error = np.dot(delta, self.weights[i].T)
83. def train(self, X, Y, epochs, learning\_rate,momentumRate):
84. # now enter the training loop
85. for i in range(epochs):
86. sum\_errors = 0
88. # Random data
89. seed = randint(1, epochs\*100)
90. np.random.seed(seed)
91. np.random.shuffle(X)
92. np.random.seed(seed)
93. np.random.shuffle(Y)
95. # iterate through all the training data
96. for j, input in enumerate(X):
97. target = Y[j]
98. # activate the network!
99. output = self.feedForward(input)
100. error = target - output
102. self.backPropagate(error)
103. # now perform gradient descent on the derivatives
104. # (this will update the weights
106. self.gradient\_descent(learning\_rate,momentumRate)
107. # keep track of the MSE for reporting later
108. sum\_errors += self.\_mse(target, output)
110. # Epoch complete, report the training error
111. print("Error: {} at epoch {}".format(round(sum\_errors / len(X) , 5), i+1))
112. self.average\_err = round(sum\_errors / len(X) , 5)
113. print("Training complete! : ",sum\_errors/len(X))
114. print("=====")
115. def gradient\_descent(self, learningRate=1,momentumRate=1):
116. # update the weights by stepping down the gradient
118. for i in range(len(self.weights)):
119. weights = self.weights[i]
120. weights\_befoe = self.weights\_befoe[i]
121. weights\_next = self.weights\_next[i]
122. bias = self.bias[i]
123. bias\_before = self.bias\_before[i]
124. bias\_next = self.bias\_next[i]
125. derivatives\_w = self.derivatives\_w[i]
126. derivatives\_b = self.derivatives\_b[i]
127. weights\_next += (derivatives\_w \* learningRate) + ((weights-weights\_befoe)\*momentumRate)
128. bias\_next += (derivatives\_b \* learningRate) + ((bias-bias\_before)\*momentumRate)
129. self.weights\_befoe = copy.deepcopy(self.weights)
130. self.weights = copy.deepcopy(self.weights\_next)
131. self.bias\_before = copy.deepcopy(self.bias)
132. self.bias = copy.deepcopy(self.bias\_next)
134. def \_mse(self, target, output):
135. return np.average((target - output) \*\* 2)
136. def \_normalization(NewMax,NewMin,OldMax,OldMin,OldValue):
137. OldRange = (OldMax - OldMin)
138. NewRange = (NewMax - NewMin)
139. NewValue = (((OldValue - OldMin) \* NewRange) / OldRange) + NewMin
141. return NewValue
142. def \_readfile(txt):
143. output = []
144. input = []
145. if txt == "Flood\_dataset.txt":
146. # import data set
147. with open("Flood\_dataset.txt", "r") as f:
148. content = f.readlines()
149. del content[0:3]
150. # split data set
151. data = []
152. for X in content:
153. data.append(X.split())
154. # convert data to list
155. output = [list(map(int, X[8:])) for X in data]
156. input = [list(map(int, X[:8])) for X in data]
157. elif txt == "cross.pat":
158. # import data set
159. with open("cross.pat", "r") as f:
160. content = f.readlines()
161. for i,X in enumerate(content):
162. if X[0] != 'p':
163. if (i+1)%3 == 0:
164. a,b = X.split()
165. output.append([int(a),int(b)])
166. else:
167. a,b = X.split()
168. input.append([float(a),float(b)])
170. else:
171. print("-- Not found a data / Missing data --")
172. # Convert to np\_array
173. input = np.array(input)
174. output = np.array(output)
175. seed = randint(1, len(input)\*100)
176. np.random.seed(seed)
177. np.random.shuffle(input)
178. np.random.seed(seed)
179. np.random.shuffle(output)
180. # Shape input and output
181. inputSize = input.shape[1]
182. outputSize = output.shape[1]
183. return input, output, inputSize, outputSize
184. def cross\_validations\_split(shape,folds):
185. fold\_size = int(shape \* folds/100)
186. k = 0
187. index = []
188. for i in range(1,folds+1):
189. if i < folds:
190. index.append([k,i\*fold\_size])
191. else:
192. index.append([k,shape])
193. k = i\*fold\_size
194. return index
195. def \_confusion\_matrix(predict,actually):
196. def \_create\_matrix(label,act):
197. matrix = np.array([[0, 0], [0, 0]])
198. if act[0] == 0 :
199. if label[0] == 0 :
200. matrix[0][0] += 1
201. else :
202. matrix[0][1] += 1
203. else :
204. if label[0] == 1 :
205. matrix[1][1] += 1
206. else :
207. matrix[1][0] += 1
209. return matrix
210. confusion\_matrix = np.array([[0, 0], [0, 0]])
211. for i in range(len(predict)):
213. if predict[i][0] >= predict[i][1]:
214. label = [1,0]
215. else :
216. label = [0,1]
217. matrix = \_create\_matrix(label,actually[i])
218. confusion\_matrix = np.add(confusion\_matrix,matrix)
220. return confusion\_matrix
221. filename = input('Enter file name : ')
222. X,Y,inputSize,outputSize = \_readfile(filename)
223. print("What Size of Hidden layer Neural Network ?")
224. print(" -- Example : '4-2-2' --")
225. print(" -- Hidden layer have 3 layers and 4,2,2 nodes respectively -- ")
226. hiddenSize = input('Enter hidden size : ')
227. hiddenSize = hiddenSize.split("-")
228. hiddenSize = list(map(int, hiddenSize))
229. learning\_rate = input('Enter learning rate : ')
230. momentum\_rate = input('Enter momentum rate : ')
231. epochs = input('Enter epoch : ')
232. if filename == "Flood\_dataset.txt":
233. X\_train = \_normalization(1,0,X.max(),X.min(),X)
234. Y\_train = \_normalization(1,0,Y.max(),Y.min(),Y)
235. else :
236. X\_train = X
237. Y\_train = \_normalization(0.9,0.1,Y.max(),Y.min(),Y)
238. NN = NeuralNetwork(hiddenSize, inputSize, outputSize)
239. train\_average\_accuracy = 0
240. test\_average\_accuracy = 0
241. for a,b in cross\_validations\_split(X\_train.shape[0],10):
242. inTest = np.concatenate((X\_train[:a],X\_train[b+1:]))
243. outTest = np.concatenate((Y\_train[:a],Y\_train[b+1:]))
244. NN.train(inTest, outTest, int(epochs) , float(learning\_rate) , float(momentum\_rate))
245. train\_average\_accuracy += (1 - NN.average\_err)/10
246. test\_average\_accuracy += (1- np.sum(NN.\_mse(NN.feedForward(X\_train[a:b,:]),Y\_train[a:b,:]),axis=0))/10
247. print("Test\_average : ",test\_average\_accuracy)
248. print("Train\_average : ",train\_average\_accuracy)
249. if filename != "Flood\_dataset.txt":
250. print(\_confusion\_matrix(Y\_predict,Y))
251. #matrix = np.float64(matrix)
     1. **ตัวอย่างการทำงานของโปรแกรม**



