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 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 1ทั้งหมด 2 จำนวน 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%

* + 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 ]** | 69 | 31 |
| **[ 1 0 ]** | 4 | 96 |
| **ความเร็วในการ converge** | **332.52 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 4 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 97 | 3 |
| **[ 0 1 ]** | 4 | 96 |
| **ความเร็วในการ converge** | **356.85 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 6 ] | *[ 0 1 ]* | *[ 1 0 ]* |
| **[ 0 1 ]** | 97 | 3 |
| **[ 1 0 ]** | 2 | 98 |
| **ความเร็วในการ converge** | **381.15 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 8 ] | *[ 0 1 ]* | *[ 1 0 ]* |
| **[ 0 1 ]** | 97 | 3 |
| **[ 1 0 ]** | 4 | 96 |
| **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

* + 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** | **381.15 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

* + 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

* + 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 ดังนั้นผลลัพธ์ที่ได้จริงๆจึงนำค่าที่ได้มากที่สุดมาทำการสรุปผล

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



