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.

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

* + 1. 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 และการทดลองในข้อถัดๆไปจะใช้ ค่าของ **Hyperparameter** ที่ดีที่สุดในข้อก่อนหน้ามาเป็นค่าหลักของการทดลองข้อถัดๆไป
      6. แสดงผลการทดลองซึ่งใช้สมการ **Mean squared error** ในการวิเคราะห์ผล
         * สำหรับการทดลอง **Flood\_dataset** จะแสดงผลในรูปของกราฟและ แผนภูมิแท่ง
         * สำหรับการทดลอง **Cross** จะแสดงผลในรูปของ **Confusion matrix** และแผนภูมิแท่ง
      7. สรุปผลการทดลอง

อธิบายเกี่ยวกับการทดลอง

ในแต่ละการทดลอง ผู้ทดลองได้ทำการ Train ในรูปแบบของ 10-folds Cross validations split ซึ่งผู้ทดลองแสดงผลของการทดลองในรูปแบบแผนภูมิแท่ง ประกอบไปด้วย แท่งสีน้ำเงิน คือ ค่าความถูกต้องโดยเฉลี่ย(Average Accuracy) ในการเทรน(Train) และ แท่งสีส้ม คือ ค่าความถูกต้องโดยเฉลี่ย(Average Accuracy) ในการทดสอบ(Test) ซึ่งแกน X คือ ค่าที่กำหนดในการทดลอง และ แกน Y คือ Average Accuracy

* 1. **การทดลองเปลี่ยนแปลงจำนวน Hidden Layer**
     1. **การทดลองเกี่ยวกับ flood\_dataset.txt**

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Hidden Layer ซึ่งแต่ละ Hidden Layer จะประกอบไปด้วย 4 Nodes โดยทำการทดลองโดยใช้ epochs = 1,000 , learning rate = 0.7 ,   
momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)

พบว่า 4-4-4 เป็นจำนวน hidden nodes และ hidden layers ที่เหมาะสมที่สุดในการทดลองครั้งนี้ โดยเปรียบเทียบจากความแม่นยำโดยเฉลี่ยในการ Test โดยการแบ่ง Cross validation

หมายเหตุ : การทดลองไม่ได้ทำนับรวมกับ จำนวน Input node และ Output node

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

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Hidden Layer ซึ่งแต่ละ Hidden Layer จะประกอบไปด้วย 4 Nodes โดยทำการทดลองโดยใช้ epochs = 1,000 , learning rate = 0.8 ,   
momentum rate = 0.2 และทำการสุ่ม weight ในช่วง (0,1)

พบว่า จำนวน Hidden layer = 1 , Hidden node = 9 เป็นปริมาณ Hidden layer และ Hidden node ที่เหมาะสมที่สุดในการทดลองครั้งนี้ โดยเปรียบเทียบจากความแม่นยำในการ Test ซึ่งมีความแม่นยำถึง 86.6%

หมายเหตุ : การทดลองไม่ได้ทำนับรวมกับ จำนวน Input node และ Output node

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 3 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 47 | 53 |
| **[ 0 1 ]** | 6 | 94 |
| **ความเร็วในการ converge** | **332.52 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 6 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 69 | 31 |
| **[ 0 1 ]** | 9 | 91 |
| **ความเร็วในการ converge** | **356.85 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 9 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 73 | 27 |
| **[ 0 1 ]** | 9 | 91 |
| **ความเร็วในการ converge** | **381.15 seconds** | |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 12 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 74 | 26 |
| **[ 0 1 ]** | 10 | 90 |
| **ความเร็วในการ converge** | **406.38 seconds** | |

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

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Learning rate โดยการทดลองจะมีค่า Learning rate ตั้งแต่ 0.2 – 1 เพิ่มขึ้นที่ละ 0.1 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [4-4-4](อ้างอิงจากข้อ 3.1.1) , momentum rate = 0.5 และทำการสุ่ม weight ในช่วง (0,1)

พบว่าค่า learning rate ที่เหมาะสมคือ 0.7 โดยอ้างอิงจาก ความแม่นยำเฉลี่ยในการ Test

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

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Learning rate โดยการทดลองจะมีค่า Learning rate ตั้งแต่ 0.1 – 1 เพิ่มขึ้นที่ละ 0.1 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [3](อ้างอิงจากข้อ 3.1.2) , momentum rate = 0.2 และทำการสุ่ม weight ในช่วง (0,1)

พบว่าค่า learning rate ที่เหมาะสมคือ 0.8 และมีค่าที่ใกล้เคียงกันคือ 0.2 ,0.7, และ 1 ที่มีค่าใกล้เคียงกัน โดยอ้างอิงจาก ความแม่นยำเฉลี่ยในการ Test

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

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Momentum rate โดยการทดลองจะมีค่า Momentum rate ตั้งแต่ 0.2 – 1 เพิ่มขึ้นที่ละ 0.2 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer 4-4-4(อ้างอิงจากข้อ 3.1.1) , learning rate = 0.7(อ้างอิงจากข้อ 3.2.1) และทำการสุ่ม weight ในช่วง (0,1)

พบว่าค่า Momentum rate ที่เหมาะสมคือ 0.2 โดยอ้างอิงจาก ความแม่นยำเฉลี่ยนการ Test

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

ผู้ทดลองได้ทำการทดลองเปลี่ยนแปลงจำนวณ Momentum rate โดยการทดลองจะมีค่า Momentum rate ตั้งแต่ 0.1 – 1 เพิ่มขึ้นที่ละ 0.1 ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [3](อ้างอิงจากข้อ 3.1.2) , momentum rate = 0.8(อ้างอิงจากข้อ 3.2.2) และทำการสุ่ม weight ในช่วง (0,1)

พบว่าค่า Momentum rate ที่เหมาะสมคือ 0.2 และมีค่าที่ใกล้เคียงกันคือ 0.7, 0.8 และ 0.4 ที่มีค่าใกล้เคียงกัน โดยอ้างอิงจาก ความแม่นยำในการ Test โดยเฉลี่ย

* 1. **การทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน**
     1. **การทดลองเกี่ยวกับ flood\_dataset.txt**

ผู้ทดลองได้ทำการทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน โดยการทดลองจสุ่ม Weight ในช่วง[ 0 ถึง 1 ] , [ 1 ถึง 2 ], และ [ -1 ถึง 1 ] ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [4-4-4](อ้างอิงจากข้อ 3.1.1) , learning rate = 0.7(อ้างอิงจากข้อ 3.2.1) และ momentum rate = 0.7(อ้างอิงจากข้อ 3.3.1)

พบว่าช่วงของ Weight ที่เหมาะสมคือ 0 ถึง 1 ซึ่งมีความแม่นยำถึง 92.5 โดยอ้างอิงจาก ความแม่นยำในการ Test โดยเฉลี่ย

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

ผู้ทดลองได้ทำการทดลองสุ่มช่วงของ Weight ที่แตกต่างกัน โดยการทดลองจสุ่ม Weight ในช่วง[ 0 ถึง 1 ] , [ 1 ถึง 2 ], และ [ -1 ถึง 1 ] ซึ่งทำการทดลองโดยใช้ epochs = 1,000 , Hidden layer [3](อ้างอิงจากข้อ 3.1,2) , learning rate = 0.8(อ้างอิงจากข้อ 3.2.2) และ momentum rate = 0.2(อ้างอิงจากข้อ 3.3.2)

พบว่าช่วงของ Weight ที่เหมาะสมคือ 1 ถึง 2 ซึ่งมีความแม่นยำถึง 82% รองลงมา โดยอ้างอิงจาก ความแม่นยำในการ Test โดยเฉลี่ย

1. **สรุปผลการทดลอง**

สรุปผลการทดลองในแต่ละชุดข้อมูลได้ในรูปแบบตารางดังนี้

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ชุดข้อมูล** | **การทดลอง** | | | | |
| **NN Layer** | **Learning rate** | **Momentum rate** | **Weight** | **Accuracy** |
| **flood\_dataset.txt** | 8 – 4 – 4 – 4 – 1 | 0.7 | 0.7 | ( 0 ถึง 1 ) | 92.5% |
| **cross.pat** | 2 – 9 – 2 | 0.8 | 0.2 | ( 1 ถึง 2 ) | 82% |

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

1. **ภาคผนวก**
   1. **ภาษาที่ใช้ในการทดลอง :** Python 3.7
   2. **Source code :** [Click](https://github.com/pannawit2541/CI_Assignment/blob/master/Assignment%201/600610752.py)
2. import numpy as np
3. import copy
4. import sys

7. class NeuralNetwork(object):
8. def \_\_init\_\_(self, hiddenSize, inputSize, outputSize):
9. # initiate layers
10. self.inputSize = inputSize
11. self.outputSize = outputSize
12. self.hiddenSize = hiddenSize
14. layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
16. # initiate weights
17. weights = []
18. for i in range(len(layers)-1):
19. w = np.random.rand(layers[i], layers[i+1])
20. weights.append(w)
21. self.weights = weights
23. weights\_t-1 = copy.deecopy(weights)
25. # initiate bias
26. bias = []
27. for i in range(len(layers)-1):
28. b = np.random.rand(layers[i+1])
29. bias.append(b)
30. self.bias = bias
32. # initiate activations
33. activations = []
34. for i in range(len(layers)):
35. a = np.zeros(layers[i])
36. activations.append(a)
37. self.activations = activations
39. derivatives = []
40. for i in range(len(layers) - 1):
41. d = np.zeros((layers[i], layers[i + 1]))
42. derivatives.append(d)
43. self.derivatives = derivatives
44. self.derivatives\_old = copy.deepcopy(self.derivatives)
46. def sigmoid(self, s, deriv=False):
47. if (deriv == True):
48. return s \* (1-s)
49. return 1/(1 + np.exp(-s))
51. def feedForward(self, X):
52. activations = X
53. self.activations[0] = X
54. for i, w in enumerate(self.weights):
55. # calculate NN\_input
56. v = np.dot(activations, w)
57. # calculate the activations
58. b = self.bias[i]
59. activations = self.sigmoid(v+b)
60. self.activations[i+1] = activations
61. return activations
63. def backPropagate(self, error):
64. for i in reversed(range(len(self.derivatives))):
66. # get activation for previous layer
67. activations = self.activations[i+1]
69. # apply sigmoid derivative function
70. 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
75. # get activations for current layer
76. current\_activations = self.activations[i]
78. # reshape activations as to have them as a 2d column matrix
79. current\_activations = current\_activations.reshape(
80. current\_activations.shape[0], -1)
82. # save derivative after applying matrix multiplication
83. self.derivatives[i] = np.dot(current\_activations, delta\_re)
85. # backpropogate the next error
86. error = np.dot(delta, self.weights[i].T)
88. def train(self, X, Y, epochs, learning\_rate,momentumRate):
89. # now enter the training loop
90. for i in range(epochs):
91. sum\_errors = 0
93. # iterate through all the training data
94. for j, input in enumerate(X):
95. target = Y[j]
97. # activate the network!
98. output = self.feedForward(input)
100. error = target - output
101. print(output, " - ", target)
102. if i > 0 :
103. self.backPropagate(error)
104. self.derivatives\_old = copy.deepcopy(self.derivatives)
105. else:
106. self.derivatives\_old = copy.deepcopy(self.derivatives)
107. self.backPropagate(error)



112. # now perform gradient descent on the derivatives
113. # (this will update the weights
115. self.gradient\_descent(learning\_rate,momentumRate)
117. # keep track of the MSE for reporting later
118. sum\_errors += self.\_mse(target, output)
120. # Epoch complete, report the training error
121. print("Error: {} at epoch {}".format(round(sum\_errors / len(X) , 5), i+1))
122. self.sum\_all\_err = sum\_errors / len(X)
123. print("Training complete! : ",sum\_errors/len(X))
124. print("=====")
126. def gradient\_descent(self, learningRate=1,momentumRate=1):
127. # update the weights by stepping down the gradient
128. for i in range(len(self.weights)):
129. weights = self.weights[i]
130. bias = self.bias[i]
131. derivatives = self.derivatives[i]
132. derivatives\_old = self.derivatives\_old[i]
133. delta = (derivatives \* learningRate) + ((derivatives-derivatives\_old)\*momentumRate)
134. weights += delta
135. delta = np.dot(delta.T,np.ones(delta.T.shape[1]))
136. bias += delta
138. def \_mse(self, target, output):
139. return np.average((target - output) \*\* 2)
141. def convert\_output(max,min,data,flag = False):
142. if flag == True:
143. return ( data\*(max-min)) + min
144. return (data - min) / (max - min)
146. def convert\_input(data):
147. mean = data.mean(axis = 0)
148. sd = data.std(axis = 0)
149. return (data- mean)/ sd
151. def Preprocessing():
153. # import data set
154. with open("Flood\_dataset.txt", "r") as f:
155. content = f.readlines()
156. del content[0:3]
158. # split data set
159. data = []
160. for X in content:
161. data.append(X.split())
163. # convert data to list
164. output = [list(map(int, X[8:])) for X in data]
165. input = [list(map(int, X[:8])) for X in data]
167. input = np.array(input)
168. output = np.array(output)
169. #print(output)
170. inputSize = input.shape[1]
171. outputSize = output.shape[1]
173. return input, output, inputSize, outputSize
175. def Preprocessing\_Cross():
176. # import data set
177. with open("cross.pat", "r") as f:
178. content = f.readlines()
179. del content[0:3]
181. # split data set
182. output = []
183. input = []
184. for i,X in enumerate(content):
185. if X[0] != 'p':
186. if (i+1)%3 == 0:
187. a,b = X.split()
188. output.append([int(a),int(b)])
189. else:
190. a,b = X.split()
191. input.append([float(a),float(b)])
192. input = np.array(input)
193. output = np.array(output)
195. inputSize = input.shape[1]
196. print(input.shape)
197. outputSize = output.shape[1]
199. return input, output, inputSize, outputSize

202. def cross\_validations\_split(dataset,output\_dataset,folds):
203. fold\_size = int(dataset.shape[0] \* folds/100)
204. k = 0
205. index = []
206. for i in range(1,folds+1):
207. if i < folds:
208. index.append([k,i\*fold\_size])
209. else:
210. index.append([k,dataset.shape[0]])
211. k = i\*fold\_size
212. return index
214. def confusion\_matrix(act,predict):
215. label\_data = []
216. label\_predict = []
217. for i in range(act.shape[0]):
218. if predict[i][0] > predict[i][1]:
219. label\_predict.append(0)
220. elif predict[i][0] <= predict[i][1]:
221. label\_predict.append(1)
222. if act[i][0] > act[i][1]:
223. label\_data.append(0)
224. elif act[i][0] <= act[i][1]:
225. label\_data.append(1)
227. a1 = 0
228. a2 = 0
229. b1 = 0
230. b2 = 0
231. for i in range(len(label\_data)):
232. if label\_data[i] == 0:
233. if label\_predict[i] != label\_data[i]:
234. a2 += 1
235. else:
236. a1 += 1
237. else:
238. if label\_predict[i] != label\_data[i]:
239. b1 += 1
240. else:
241. b2 += 1
243. print("== 0 ==== 1")
244. print("0 = ",a1," == ",a2)
245. print("1 = ",b1," == ",b2)
247. model = 'choose model'
249. while(True):
250. print(' -- Please press one to training --> A or B -- ')
251. print(' -- A : flood\_dataset , B : cross\_dataset -- ')
252. print(' -- q : to exit -- ')
253. model = input()
254. print(type(model))
255. if model == 'q' or model == 'Q':
256. sys.exit()
257. elif model == 'A' or model == 'a' or model == 'B' or model == 'b' or model == 'q' :
258. break

261. print("What Size of Hidden layer Neural Network ?")
262. print(" -- Example : '4-2-2' --")
263. print(" -- Hidden layer have 3 layers and 4,2,2 nodes respectively -- ")
265. hiddenSizeStr = input('Size of Hidden layer : ')
266. learningRate = input('Learning Rate : ')
267. learningRate = float(learningRate)
268. momentumRate = input('Momentum Rate : ')
269. momentumRate = float(momentumRate)
270. epochs = input('Epochs : ')
271. epochs = int(epochs)
272. hiddenSize = hiddenSizeStr.split("-")
273. hiddenSize = list(map(int, hiddenSize))

276. if model == 'A' or model == 'a':
277. X, Y, inputSizeX, outputSizeY = Preprocessing()
278. max,min = Y.max(),Y.min()
279. y = convert\_output(max,min,Y)
280. x = convert\_input(X)
281. index\_flood = cross\_validations\_split(x,y,10)
282. NN\_flood = NeuralNetwork(hiddenSize, inputSizeX, outputSizeY)
283. for a,b in index\_flood:
284. inTest = np.concatenate((x[:a],x[b+1:]))
285. outTest = np.concatenate((y[:a],y[b+1:]))
286. NN\_flood.train(inTest, outTest, epochs , learningRate,momentumRate)
287. sum\_avg\_train += NN\_flood.sum\_all\_err
288. sum\_avg\_predict += np.sum(NN\_flood.\_mse(NN\_flood.feedForward(x[a:b,:]),y[a:b,:]),axis=0)
289. else:
290. A, B, inputSizeA, outputSizeB = Preprocessing\_Cross()
291. index\_cross = cross\_validations\_split(A,B,10)
292. NN\_cross = NeuralNetwork(hiddenSize, inputSizeA, outputSizeB)
293. for a,b in index\_cross:
294. inTest = np.concatenate((A[:a],A[b+1:]))
295. outTest = np.concatenate((B[:a],B[b+1:]))
296. NN\_cross.train(inTest, outTest, epochs , learningRate,momentumRate)
297. sum\_avg\_train += NN\_cross.sum\_all\_err
298. sum\_avg\_predict += np.sum(NN\_cross.\_mse(NN\_cross.feedForward(A[a:b,:]),B[a:b,:]),axis=0)
299. confusion\_matrix(B,NN\_cross.feedForward(A))

302. print(sum\_avg\_train/10)
303. print(sum\_avg\_predict/10)