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

* 1. ข้อมูล cross.pat

เป็นชุดข้อมูลที่ประกอบไปด้วย ลำดับข้อมูล(p) , ข้อมูลเลขทศนิยม 2 จำนวน และ จำนวนจริง 2 จำนวน รวมแล้วมีทั้งหมด 5 ข้อมูล

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

ได้ทำการเปลี่ยนแปลงข้อมูล โดยใช้สมการ Standard normal distribution   
() ซึ่งข้อมูลจะอยู่ในช่วงของ [-3,3]

* + 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 ประกอยไปด้วย ข้อมูลจำนวนจริง 0 และ 1ทั้งหมด 2 จำนวน

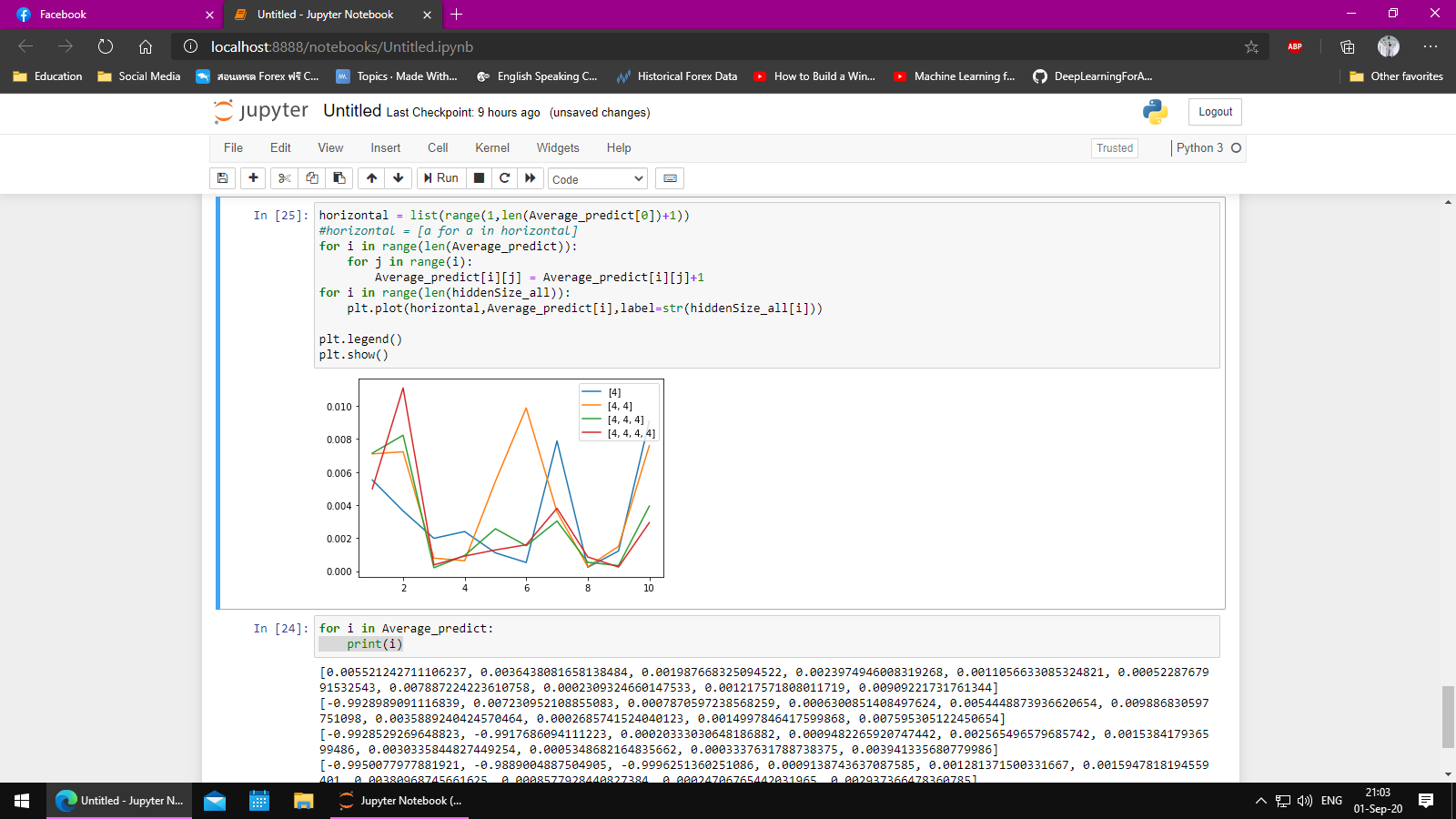
1. **การทดลองประมวลผลข้อมูล**

อธิบายเกี่ยวกับการทดลอง

ในแต่ละการทดลอง ผู้ทดลองได้ทำการ Train ในรูปแบบของ 10-folds Cross validations split ซึ่งผู้ทดลองแสดงผลของการทดลองในรูปแบบกราฟเส้นตรงซึ้นหมายถึง MSE จาก Test set ในแต่ละ Cross validations split โดยแกน X หมายถึง folds ที่ทำทดสอบ และ แกน Y หมายถึง ค่า Error

* 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 4-4-4-4 เป็นจำนวน hidden nodes และ hidden layers ที่เหมาะสมที่สุดในการทดลองครั้งนี้ โดยเปรียบเทียบจากความแม่นยำในการ 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 |

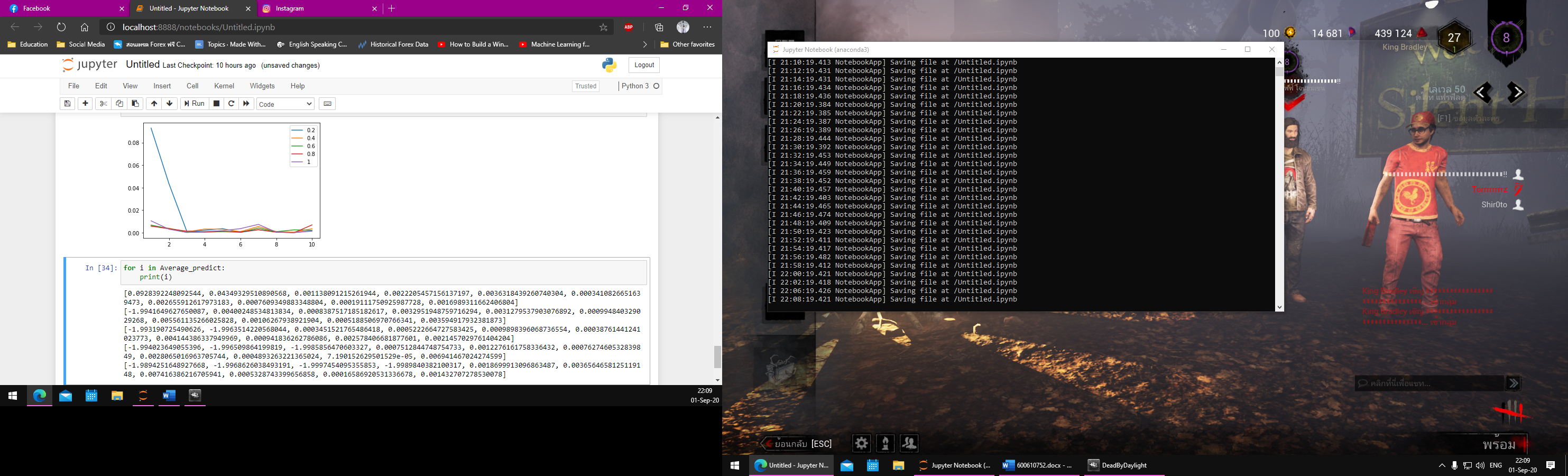
|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 6 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 69 | 31 |
| **[ 0 1 ]** | 9 | 91 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 9 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 73 | 27 |
| **[ 0 1 ]** | 9 | 91 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix [ 12 ] | *[ 1 0 ]* | *[ 0 1 ]* |
| **[ 1 0 ]** | 74 | 26 |
| **[ 0 1 ]** | 10 | 90 |

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

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



พบว่าค่า learning rate ที่เหมาะสมคือ 0.6 และมีค่าที่ใกล้เคียงกันคือ 0.4 โดยอ้างอิงจาก ความแม่นยำในการ Test

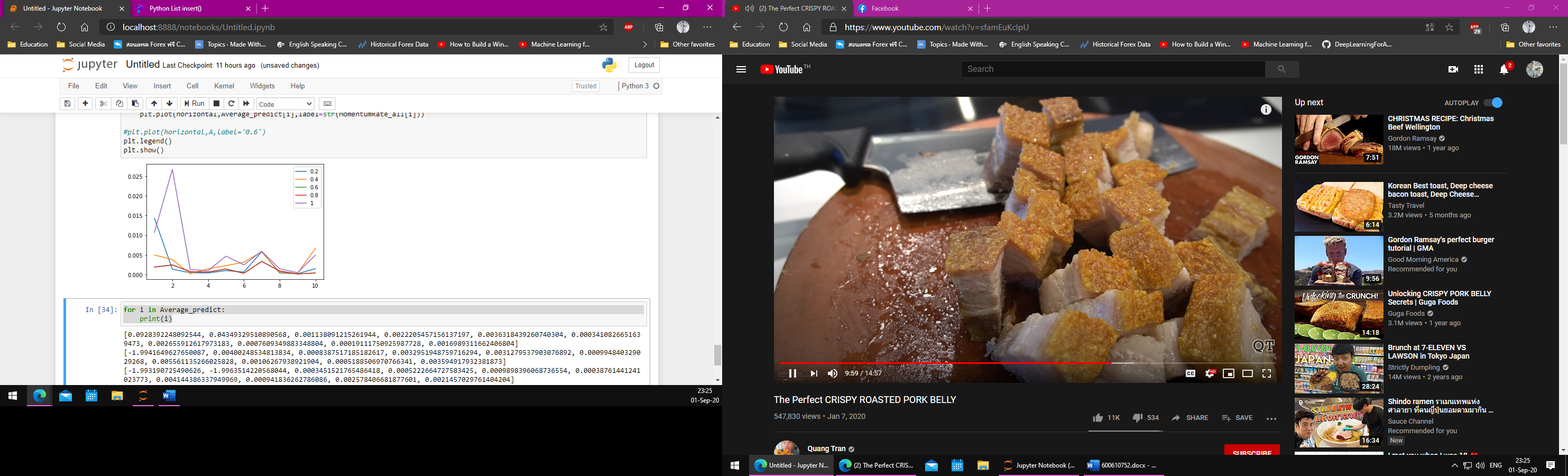
* + 1. **การทดลองเกี่ยวกับ cross.pat**

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 = 0.7 และทำการสุ่ม weight ในช่วง (0,1)

พบว่าค่า learning rate ที่เหมาะสมคือ 0.8 และมีค่าที่ใกล้เคียงกันคือ 0.2 ,0.7, และ 1 ที่มีค่าใกล้เคียงกัน โดยอ้างอิงจาก ความแม่นยำในการ 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 4-4-4-4(อ้างอิงจากข้อ 3.1.1) , learning rate = 0.6(อ้างอิงจากข้อ 3.2.1) และทำการสุ่ม weight ในช่วง (0,1)



พบว่าค่า Momentum rate ที่เหมาะสมคือ 0.6 และ 0.8 เพราะมีค่าเท่ากัน โดยอ้างอิงจาก ความแม่นยำในการ Test

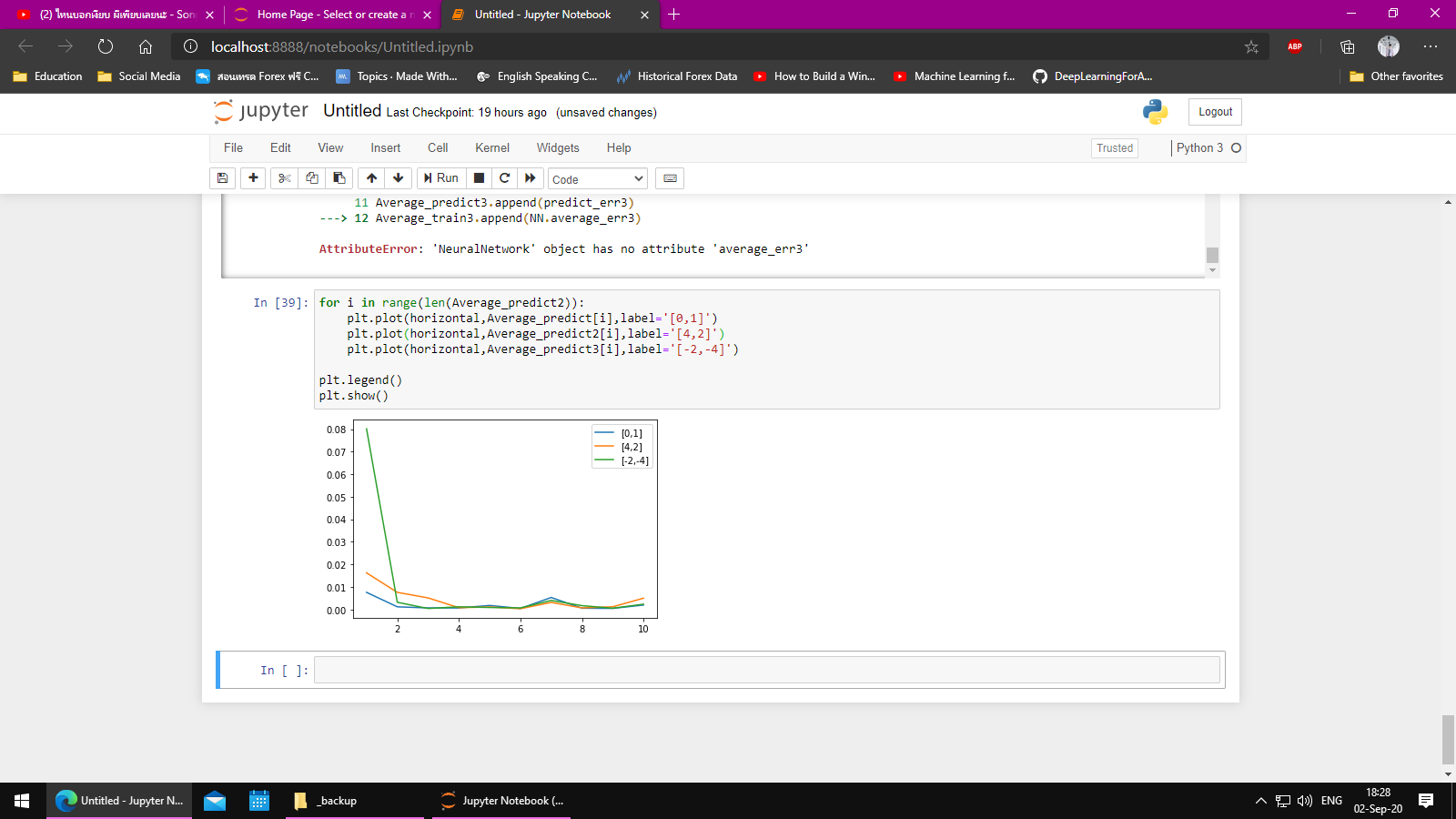
* + 1. **การทดลองเกี่ยวกับ cross.pat**

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) , momentum rate = 0.7(อ้างอิงจากข้อ 3.3.1) และ learning rate = 0.7(อ้างอิงจากข้อ 3.2.1)



พบว่าช่วงของ Weight ที่เหมาะสมคือ 0 ถึง 1 ซึ่งมีความแม่นยำถึง 99.1 % รองลงมา คือ -1 ถึง 0 และ 1 ถึง 2 ตามลำดับ 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) , momentum rate = 0.2(อ้างอิงจากข้อ 3.3.2) และ learning rate = 0.8(อ้างอิงจากข้อ 3.2.2)

พบว่าช่วงของ Weight ที่เหมาะสมคือ 1 ถึง 2 ซึ่งมีความแม่นยำถึง 82% รองลงมา คือ -1 ถึง 0 และ 0 ถึง 1 ตามลำดับ โดยอ้างอิงจาก ความแม่นยำในการ Test โดยเฉลี่ย

1. **สรุปผลการทดลอง**

สรุปผลการทดลองในแต่ละชุดข้อมูลได้ในรูปแบบตารางดังนี้

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

จากการทดลองอาจจะการคลาดเคลื่อนได้เนื่องจากตัวเลขที่หาได้จากแต่ละการทดลองนั้นมีค่าใกล้เคียงกันมาก ค่า Momentum rate ของชุดข้อมูล cross.pat ที่ 0.2 และ 0.4 มีค่าต่างกัน 0.8%

1. **ภาคผนวก**
   1. **ภาษาที่ใช้ในการทดลอง :** Python 3.7.8
   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. # initiate bias
24. bias = []
25. for i in range(len(layers)-1):
26. b = np.random.rand(layers[i+1])
27. bias.append(b)
28. self.bias = 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
37. derivatives = []
38. for i in range(len(layers) - 1):
39. d = np.zeros((layers[i], layers[i + 1]))
40. derivatives.append(d)
41. self.derivatives = derivatives
42. self.derivatives\_old = copy.deepcopy(self.derivatives)
44. def sigmoid(self, s, deriv=False):
45. if (deriv == True):
46. return s \* (1-s)
47. return 1/(1 + np.exp(-s))
49. def feedForward(self, X):
50. activations = X
51. self.activations[0] = X
52. for i, w in enumerate(self.weights):
53. # calculate NN\_input
54. v = np.dot(activations, w)
55. # calculate the activations
56. b = self.bias[i]
57. activations = self.sigmoid(v+b)
58. self.activations[i+1] = activations
59. return activations
61. def backPropagate(self, error):
62. for i in reversed(range(len(self.derivatives))):
64. # get activation for previous layer
65. activations = self.activations[i+1]
67. # apply sigmoid derivative function
68. delta = error \* self.sigmoid(activations, deriv=True)
70. # reshape delta as to have it as a 2d array
71. delta\_re = delta.reshape(delta.shape[0], -1).T
73. # get activations for current layer
74. 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)
80. # save derivative after applying matrix multiplication
81. self.derivatives[i] = np.dot(current\_activations, delta\_re)
83. # backpropogate the next error
84. error = np.dot(delta, self.weights[i].T)
86. def train(self, X, Y, epochs, learning\_rate,momentumRate):
87. # now enter the training loop
88. for i in range(epochs):
89. sum\_errors = 0
91. # iterate through all the training data
92. for j, input in enumerate(X):
93. target = Y[j]
95. # activate the network!
96. output = self.feedForward(input)
98. error = target - output
99. print(output, " - ", target)
100. if i > 0 :
101. self.backPropagate(error)
102. self.derivatives\_old = copy.deepcopy(self.derivatives)
103. else:
104. self.derivatives\_old = copy.deepcopy(self.derivatives)
105. self.backPropagate(error)



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

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

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

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

300. print(sum\_avg\_train/10)
301. print(sum\_avg\_predict/10)