**Train Multilayer Perceptron**

**with**

**Particle Swarm Optimization (PSO)**

**จัดทำโดย**

นายปัณณวิชญ์ พันธ์วงศ์

600610752

**เสนอ**

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของวิชา

CPE 261456 (Introduction to Computational Intelligence)

ภาคเรียนที่ 1 ปีการศึกษา 2563

มหาวิทยาลัยเชียงใหม่

# สารบัญ

[**สารบัญ 1**](#_Toc55857322)

[**ลักษณะการขั้นตอนการทำงาน 2**](#_Toc55857323)

[**การทำเตรียมข้อมูล 2**](#_Toc55857324)

[**อัลกอริทึม Particle swarm optimization 4**](#_Toc55857325)

[**ผลการทดลอง และการวิเคราะห์ 5**](#_Toc55857326)

[**การทดลองทำนาย Benzene concentration 5 วันล่วงหน้า 5**](#_Toc55857327)

[**การทดลองทำนาย Benzene concentration 10 วันล่วงหน้า 8**](#_Toc55857328)

[**สรุปผลการทดลอง 11**](#_Toc55857329)

[**โปรแกรม 12**](#_Toc55857330)

# ลักษณะการขั้นตอนการทำงาน

## การทำเตรียมข้อมูล

สำหรับ Features ที่ทำการเทรนระบบนั้นได้แก่ 'PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH' และ 'AH' มีทั้งหมด 8 Features และในส่วนของ Output เป็นค่าของ 'C6H6(GT)' หรือ Benzene concentration ในอีก 5 วัน และ 10 วัน ล่วงหน้า โดยทำการ Min-max normalization ให้อยู่ในช่วงของ 0 ถึง 1 โดยจัดเรียงข้อมูลเป็นตารางได้ดังนี้

**ตารางที่ 1** ตารางแสดงการเตรียมข้อมูลของระบบเพื่อทำนาย 5 วันล่วงหน้า

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Features (Input)** | | | | | | | | **Outputs** |
| PT08.S1(CO) | PT08.S2(NMHC) | PT08.S3(NOx) | PT08.S4(NO2) | PT08.S5(O3) | T | RH | AH | C6H6(GT)\_5 |

**ตารางที่ 2** ตารางแสดงการเตรียมข้อมูลของระบบเพื่อทำนาย 10 วันล่วงหน้า

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Features (Input)** | | | | | | | | **Outputs** |
| PT08.S1(CO) | PT08.S2(NMHC) | PT08.S3(NOx) | PT08.S4(NO2) | PT08.S5(O3) | T | RH | AH | C6H6(GT)\_10 |

เนื่องจากข้อมูลที่ได้รับมา มี Noise value ซึ่งผู้จัดทำได้ทำการเปลี่ยน Noise value โดยใช้ทำการแทนที่ด้วยค่าที่เรียงจาก Record ล่าสุดที่ไม่เป็น Noise value ไปจนถึง Record ถัดไปที่ไม่เป็น Noise value ยกตัวอย่างดังนี้

**ตารางที่ 2** ข้อมูลก่อนทำการกำจัด Noise value โดยใช้คอลัมน์ RH และ AH

|  |  |  |
| --- | --- | --- |
| **Features (Input)** | | |
| No. | RH | AH |
| 1 | 68.4 | 53.2 |
| 2 | -200 | -200 |
| 3 | -200 | -200 |
| 4 | -200 | 63.8 |
| 5 | 60.2 | 67.9 |

สมมุติให้ ค่า -200 คือ Noise value ที่อยู่ข้อมูล ซึ่งผู้จัดทำได้แทนที่ค่าเหล่านี้ด้วยค่าที่เรียงจาก Record ที่ไม่เป็น Noise value เช่น คอลัมน์ RH โดย Record ที่ 1 ไม่เป็น Noise value และ Record ที่ 5 ไม่เป็น Noise value เช่นกัน จึงได้ทำกันเรียงค่าจาก 68.4 ไปถึง 60.2 โดยลดค่าลงที่ละ 2 ซึ่งเป็นค่าที่ทำการเฉลี่ยเพื่อให้สอดคล้องกันซึ่งได้ผลลัพธ์ดังตารางที่ 3

**ตารางที่ 3** ข้อมูลหลังทำการกำจัด Noise value โดยใช้คอลัมน์ RH และ AH

|  |  |  |
| --- | --- | --- |
| **Features (Input)** | | |
| No. | RH | AH |
| 1 | 68.4 | 53.2 |
| 2 | 66.4 | 56.2 |
| 3 | 64.4 | 59.2 |
| 4 | 62.4 | 63.8 |
| 5 | 60.2 | 67.9 |

## อัลกอริทึม Particle swarm optimization

หลังจากทำในข้อ 1.1 และ 1.2 แล้ว ระบบจะทำการเอาข้อมูลที่ได้มาทำการเทรนแบบ 10-folds Cross-validation โดยอัลกอริทึมของการหาค่าที่เหมาะสมที่สุดโดยกลุ่มของอนุภาค ใช้ อัลกอริทึมดีที่สุดแบบรวม (Global Best) โดยมี Objective function คือ Mean absolute error (MAE) โดยจะหาค่า Min เพื่อกำหนดค่า pbest และ gbestในการปรับตำแหน่ง(Position) และความเร็ว(Velocity) ของแต่ละ Particle

ทั้งนี้ได้กำหนดขอบเขตของความเร็วไว้ ซึ่งเป็นการป้องกันไม่ให้ อนุภาคเคลื่อนที่เร็วเกินไปจากจุดหนึ่งไปอีกจุดหนึ่งในปริภูมิการค้นหา ซึ่งอัลกอริทึมที่ใช้นั้น ได้ใช้ Optimization ด้วยวิธีการของ Clerc และ Kennedy และใช้ค่าน้ำหนักความเฉื่อย() ในการอัพเดตค่าความเร็วของแต่ละ Particle

โดยค่า วิธีการของ Clerc และ Kennedyมีค่าเท่ากับสมการดังนี้

*โดยที่*

# ผลการทดลอง และการวิเคราะห์

ในแต่ละการทดลองผู้จัดทำได้ทำการกำหนดพารามิเตอร์ดังต่อไปนี้ ทั้งนี้ได้กำหนดขอบเขตของความเร็วไว้ ซึ่งเป็นการป้องกันไม่ให้ อนุภาคเคลื่อนที่เร็วเกินไปจากจุดหนึ่งไปอีกจุดหนึ่งในปริภูมิการค้นหา ซึ่งอัลกอริทึมที่ใช้นั้น ได้ใช้ Optimization ด้วยวิธีการของ Clerc และ Kennedy และใช้ค่าน้ำหนักความเฉื่อย() ในการอัพเดตค่าความเร็วของแต่ละ Particle

**ตารางที่ 4** กำหนดพารามิเตอร์ที่ใช้

|  |  |
| --- | --- |
| **Features (Input)** | |
|  | 2 |
|  | 3 |
|  | 0.2 |
| velocity | ~U(-0.5,0.5) |
| position | ~U(-10,10) |

## การทดลองทำนาย Benzene concentration 5 วันล่วงหน้า

ในการทดลองผู้ทดลองทำนาย Benzene concentration 5 วันล่วงหน้า ได้ทำการทดลองเปลี่ยนแปลงโครงสร้างนิวรอลเน็ตเวิร์คของแต่ละ Swarm โดยได้ทำกำหนดจำนวนของจำนวน Particle โดยทำการทดลองทั้งหมด 3 โครงสร้าง(Swarm) และมีรายละเอียดดังตารางที่ 5

**ตารางที่ 5** ตารางแสดงโครงสร้างของแต่ละ Swarm

|  |  |  |  |
| --- | --- | --- | --- |
| **Swarm** | **Neural Network Architecture** | **Particle** | **Iteration** |
| 1 | 8-4-2 | 12 | 25 |
| 2 | 8-6-4-2 | 12 | 25 |
| 3 | 8-8-6-4-2 | 12 | 25 |

ซึ่งผลลัพธ์การฝึกสอนของแต่ละกลุ่ม Swarm จะนำเสนอในรูปแบบกราฟเส้นตรงและแผนภูมิแท่ง โดยรูปแบบกราฟเส้นตรงจะอธิบายถึง fitness ที่ได้กล่าวไปในหัวข้อ 1.3 ซึ่งกราฟประกอบไปด้วยเส้นตรงที่อธิบายถึง ค่า fitness ของ การเทรนเฉลี่ย, ค่า gbest และ validate เฉลี่ย ในแต่ละ 10-folds cross-validation ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

รูปที่ 1 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 1

รูปที่ 2 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 2

รูปที่ 3 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 3

รูปที่ 4 แผนภูมิแท่งแสดง fitness โดยเฉลี่ย ของแต่ละ Swarm

จากกราฟเส้นตรง ค่า fitness สำหรับการเทรนในแต่ละ Swarm พบว่ามีค่าเฉลียที่ใกล้เคียงกันในช่วงท้าย แต่ในช่วงเริ่มต้น Swarm 3 สามารถปรับค่าใน fold ที่ 2 ได้ดีกว่า Swarm อื่นๆ

สำหรับแผนภูมิแท่งได้แสดงผลลัพธ์ fitness ของข้อมูล validate โดยเฉลี่ย รวม 10-folds cross-validation ซึ่งพบว่า ในกลุ่ม Swarm 3 มีผลลัพธ์ในการ validate ที่ดีกว่าในกลุ่มทั้งหมด เนื่องจากใน fold ที่ 1 พบว่า ข้อมูลสำหรับ validate ที่เริ่มต้น fitness ที่ดีกว่ากลุ่ม Swarm อื่นๆ ทำให้โดยรวมแล้วค่าเฉลี่ยของการ validation มีค่าดีกว่านั่นเอง

## การทดลองทำนาย Benzene concentration 10 วันล่วงหน้า

ในการทดลองผู้ทดลองทำนาย Benzene concentration 10 วันล่วงหน้า ได้ทำการทดลองเปลี่ยนแปลงโครงสร้างนิวรอลเน็ตเวิร์คของแต่ละ Swarm โดยได้ทำกำหนดจำนวนของจำนวน Particle โดยทำการทดลองทั้งหมด 3 โครงสร้าง(Swarm) และมีรายละเอียดดังตารางที่ 6

**ตารางที่ 6** ตารางแสดงโครงสร้างของแต่ละ Swarm

|  |  |  |  |
| --- | --- | --- | --- |
| **Swarm** | **Neural Network Architecture** | **Particle** | **Iteration** |
| 1 | 8-4-2 | 12 | 25 |
| 2 | 8-10-4-2 | 12 | 25 |
| 3 | 8-12-10-4-2 | 12 | 25 |

ซึ่งผลลัพธ์การฝึกสอนของแต่ละกลุ่ม Swarm จะนำเสนอในรูปแบบกราฟเส้นตรงและแผนภูมิแท่ง โดยรูปแบบกราฟเส้นตรงจะอธิบายถึง fitness ที่ได้กล่าวไปในหัวข้อ 1.3 ซึ่งกราฟประกอบไปด้วยเส้นตรงที่อธิบายถึง ค่า fitness ของ การเทรนเฉลี่ย, ค่า gbest และ validate เฉลี่ย ในแต่ละ 10-folds cross-validation ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

รูปที่ 5 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 1

รูปที่ 6 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 2

รูปที่ 7 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 3

รูปที่ 8 แผนภูมิแท่งแสดง fitness โดยเฉลี่ย ของแต่ละ Swarm

จากกราฟเส้นตรง ค่า fitness สำหรับการเทรนในแต่ละ Swarm พบว่ามีค่าเฉลียที่ใกล้เคียงกันในช่วงท้าย แต่ในช่วงเริ่มต้น Swarm 1 สามารถปรับค่าใน fold ที่ 2 ได้ดีกว่า Swarm อื่นๆ ซึ่งสำหรับ Swarm 3 พบกว่า ใน fold ที่ 2 fitness ในการเทรนค่อนข้างแปลกจาก Swarm อื่นๆ

สำหรับแผนภูมิแท่งได้แสดงผลลัพธ์ fitness ของข้อมูล validate โดยเฉลี่ย รวม 10-folds cross-validation ซึ่งพบว่า ในกลุ่ม Swarm 1 มีผลลัพธ์ในการ validate ที่ดีกว่าในกลุ่มทั้งหมด เนื่องจากใน fold ที่ 1 พบว่า ข้อมูลสำหรับ validate ที่เริ่มต้น fitness ที่ดีกว่ากลุ่ม Swarm อื่นๆ ทำให้โดยรวมแล้วค่าเฉลี่ยของการ validation มีค่าดีกว่านั่นเอง

# สรุปผลการทดลอง

เมื่อทำการกำหนดในครั้งแรกก่อนการเทรนซึ่งในแต่ละ Particle ถูกสุ่มตำแหน่งและความเร็ว แบบ uniform distribution ซึ่งตำแหน่งถูกสุ่มในช่วง ~U(-10,10) และ ความเร็วถูกสุ่มในช่วง ~U(-0.5,0.5) และพารามิเตอร์อื่นๆได้กล่าวไปในหัวข้อที่ 2 ไปแล้ว ซึ่งทำให้ แต่ละ Particle ในแต่ละ swarm ถูกกระจายในตำแหน่งตามขอบเขตปริภูมิที่กำหนดไว้ ซึ่งเมื่อทำการเทรนจนสิ้นสุดกระบวนการแล้วพบว่า Particle ในแต่ละ swarm จะปรับตำแหน่งให้เข้สใกล้กับ Particle ที่มีค่า fitness ที่ดีที่สุด

สำหรับผลลัพธ์ในแต่ละการทดลอง ในการทดลองทำนาย Benzene concentration 5 วันล่วงหน้าพบว่า Swarm 3 มีค่า fitness ที่ดีที่สุด และในการทดลองทำนาย Benzene concentration 10 วันล่วงหน้า พบว่า Swarm 1 มีค่า fitness ดีที่สุด ซึ่งทำการสรุปเป็นตารางได้ดังนี้

**ตารางที่ 6** สรุปผลการทดลอง

|  |  |  |  |
| --- | --- | --- | --- |
| **Predict** | **Swarm** | **Neural Network Architecture** | **Fitness** |
| 5 days ahead | 3 | 8-8-6-4-2 | 0.0856 |
| 10 days ahead | 1 | 8-4-2 | 0.0906 |

# โปรแกรม

โดยการ Preprocessing ได้ทำในรูปแบบของ ไฟล์ .csv ไว้ก่อนแล้ว จากนั้นจากทำการ import เข้ามาในรูปของ Matrix numpy

1. import numpy as np
2. import random
3. from random import randint
4. import math
5. class **Particle\_of\_swarm**(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 positions*
13. positions = []
14. for i in **range**(**len**(layers)-1):
15. p = np.random.uniform(-10,10,(layers[i], layers[i+1]))
16. positions.append(p)
17. self.positions = positions
18. self.positions\_best = positions
19. velocitys = []
20. for i in **range**(**len**(layers) - 1):
21. v = np.random.uniform(-10,10,(layers[i], layers[i+1]))
22. velocitys.append(v)
23. self.velocitys = velocitys
24. self.pbest = float('inf')
25. def **feedForward**(self, X):
26. Output\_node = X
27. for i, p in **enumerate**(self.positions):
28. v = np.dot(Output\_node, p)
29. Output\_node = self.sigmoid(v)
30. return Output\_node
31. def **sigmoid**(self, s, deriv=False):
32. if (deriv == True):
33. return s \* (1-s)
34. return 1/(1 + np.exp(-s))
35. def **object\_funct**(self, X, Y):
36. *# Random data*
37. seed = randint(1, 25\*100)
38. np.random.seed(seed)
39. np.random.shuffle(X)
40. np.random.seed(seed)
41. np.random.shuffle(Y)
42. sum\_err = 0
43. for j, **input** in **enumerate**(X):
44. target = Y[j]
45. output = self.feedForward(**input**)
46. sum\_err += self.\_mae(target, output)
47. self.fx = sum\_err/**len**(X)
48. return self.fx
49. def **\_mae**(self, target, output):
50. return np.average(**abs**(target - output))
51. def **cross\_validations\_split**(shape,folds):
52. fold\_size = int(shape \* folds/100)
53. k = 0
54. index = []
55. for i in **range**(1,folds+1):
56. if i < folds:
57. index.append([k,i\*fold\_size])
58. else:
59. index.append([k,shape])
60. k = i\*fold\_size
61. return index
62. Input = np.genfromtxt('data/AirQualityUCI\_input.csv', delimiter=',')
63. Output = np.genfromtxt('data/AirQualityUCI\_output.csv', delimiter=',')
64. particles = []
65. num\_of\_particle = 12
66. for i in **range**(0, num\_of\_particle):
67. par = Particle\_of\_swarm([4], 8, 2)
68. particles.append(par)
69. gbest = [0,float('inf')]
70. gbest\_position = 0
71. *#-------------------------------------------------------*
72. train\_mean\_pbest = []
73. train\_gbest = []
74. test\_mean\_pbest = []
75. *#-------------------------------------------------------*
76. k=1
77. for a,b in cross\_validations\_split(Input.shape[0],10):
78. x\_train = np.concatenate((Input[:a],Input[b+1:]))
79. y\_train = np.concatenate((Output[:a],Output[b+1:]))
80. x\_test = Input[a:b,:]
81. y\_test = Output[a:b]
82. list\_fx = []
83. **print**("-------------------- fold : ",k," --------------------")
84. k+=1
85. for j in **range**(25):
86. for i,p in **enumerate**(particles):
87. fx = p.object\_funct(x\_train, y\_train)
88. list\_fx.append(fx)
89. *# check pbest*
90. if fx < p.pbest:
91. **print**("Update pbest ",i," : ",**round**(fx,3), " / ",**round**(p.pbest,3))
92. p.pbest = fx
93. p.positions\_best = p.positions.copy()
95. *#check gbest*
96. if fx < gbest[1]:
97. gbest[1] = fx
98. gbest[0] = i
99. gbest\_position = p.positions.copy()
100. **print**("Update gbest",i," : ",fx)
101. *# update velocity & position*
102. for i,p in **enumerate**(particles):
103. for c in **range**(0, **len**(p.velocitys)):
104. d1 = 2
105. d2 = 3
106. d = d1+d2
107. g = 1 - (1/(d)) + math.sqrt(**abs**(d\*\*2 - 4\*d))/2
108. p.velocitys[c] = g\*(0.2\*p.velocitys[c] + (d1 \* (p.positions\_best[c]-p.positions[c])) + (d2 \* (gbest\_position[c] - p.positions[c])))
109. p.positions[c] += p.velocitys[c]
110. **print**(" Epoch : " , j+1 ," | err : ", **sum**(list\_fx)/**len**(list\_fx) )
111. train\_mean\_pbest.append(**sum**(list\_fx)/**len**(list\_fx))
112. train\_gbest.append(gbest[1])
114. list\_fx\_test = []
115. for i,p in **enumerate**(particles):
116. fx = p.object\_funct(x\_test, y\_test)
117. list\_fx\_test.append(fx)
118. test\_mean\_pbest.append(**sum**(list\_fx\_test)/**len**(list\_fx\_test))
119. **print**("=========== Test ===========")
120. **print**("Error of test  :  ",**round**(**sum**(list\_fx\_test)/**len**(list\_fx\_test),6))
121. **print**("Error of train :  ",**round**(**sum**(list\_fx)/**len**(list\_fx),6))
122. **print**("gbest : ",**round**(gbest[1],6))