# 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

# 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 |
|---|---|---|---|---|---|---|---|---|
| $PT_{08.S_1(CO)}$ | $PT_{08.S_2(NMHC)}$ | $PT_{08.S_3(NOx)}$ | $PT_{08.S_4(NO_2)}$ | $PT_{08.S_5(O_3)}$ | T | RH | AH | C6H6(GT)_5 |

**ตารางที่ 2** ตารางแสดงการเตรียมข้อมูลของระบบเพื่อทำนาย 10 วันล่วงหน้า

| Features (Input) | | | | | | | | Outputs |
|---|---|---|---|---|---|---|---|---|
| $PT_{08.S_1(CO)}$ | $PT_{08.S_2(NMHC)}$ | $PT_{08.S_3(NOx)}$ | $PT_{08.S_4(NO_2)}$ | $PT_{08.S_5(O_3)}$ | 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 gbest ในการปรับตำแหน่ง(Position) และความเร็ว(Velocity) ของแต่ละ Particle

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

โดยค่า $K$ วิธีการของ Clerc และ Kennedyมีค่าเท่ากับสมการดังนี้

$$K = 1 - \frac{1}{\rho} + \frac{\sqrt{|\rho^2 - 4\rho|}}{2}$$

โดยที่ $\rho = \rho_1 + \rho_2$

# ผลการทดลอง และการวิเคราะห์

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

**ตารางที่ 4** กำหนดพารามิเตอร์ที่ใช้

| Features (Input) | |
|---|---|
| $\rho_1$ | 2 |
| $\rho_2$ | 3 |
| $\phi$ | 0.2 |
| velocity | ~U(-0.5,0.5) |
| position | ~U(-10,10) |

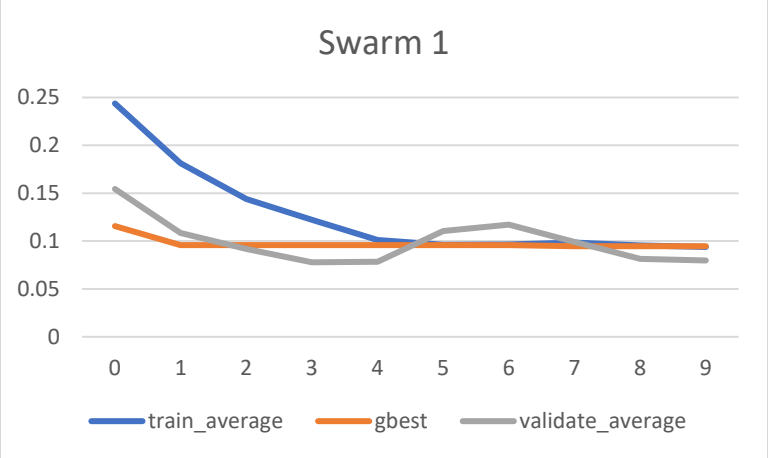
## การทดลองทำนาย Benzene concentration 5 วันล่วงหน้า

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

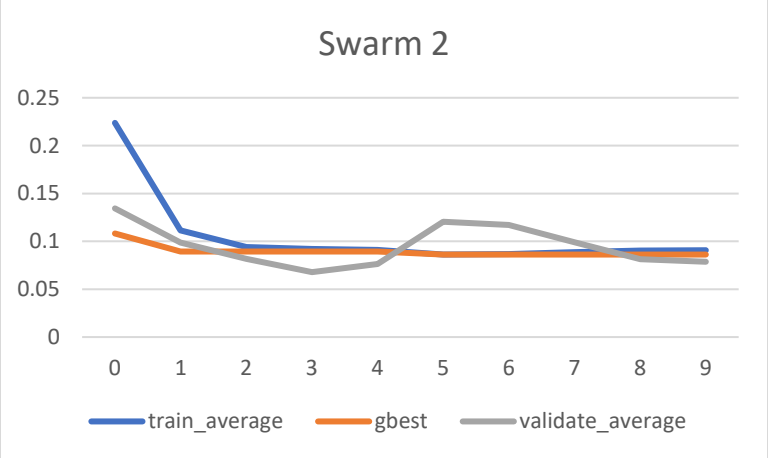
**ตารางที่ 5** ตารางแสดงโครงสร้างของแต่ละ Swarm

| 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 |

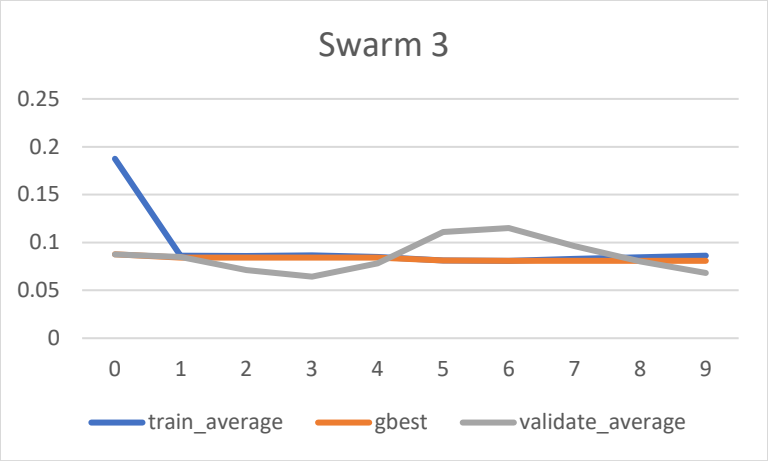
ซึ่งผลลัพธ์การฝึกสอนของแต่ละกลุ่ม Swarm จะนำเสนอในรูปแบบกราฟเส้นตรงและแผนภูมิแท่ง โดยรูปแบบกราฟเส้นตรงจะอธิบายถึง fitness ที่ได้กล่าวไปในหัวข้อ 1.3 ซึ่งกราฟประกอบไปด้วยเส้นตรงที่อธิบายถึงค่า fitness ของ การเทรนเฉลี่ย, ค่า gbest และ validate เฉลี่ย ในแต่ละ 10-folds cross-validation ซึ่งผลลัพธ์ที่ได้เป็นดังนี้



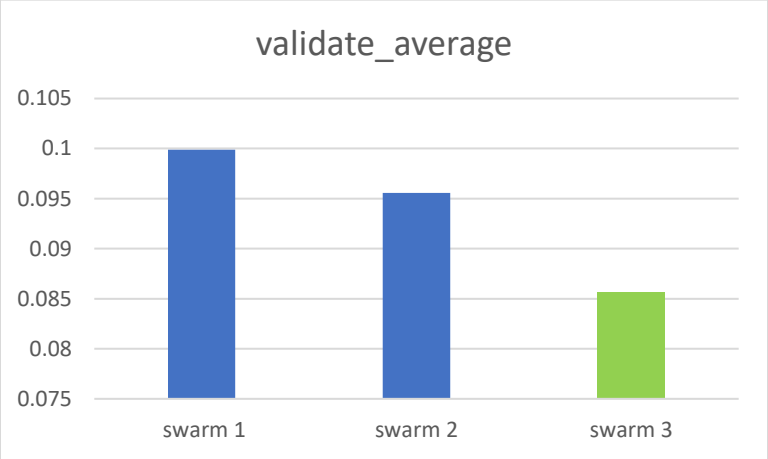

รูปที่ 1 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 1




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

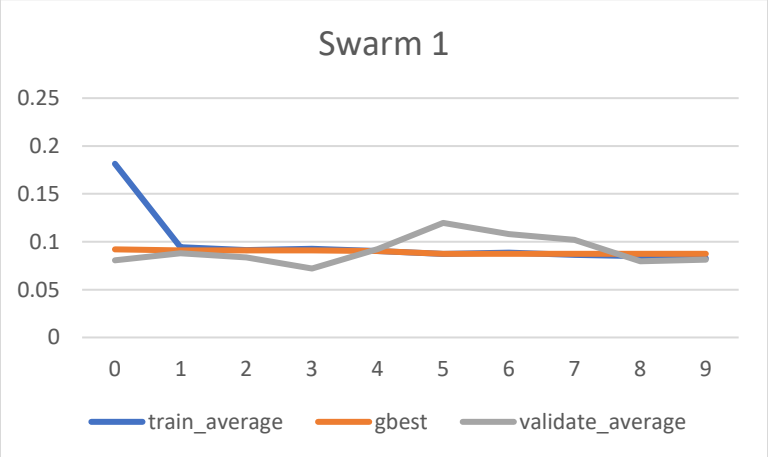
## การทดลองทำนาย Benzene concentration 10 วันล่วงหน้า

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

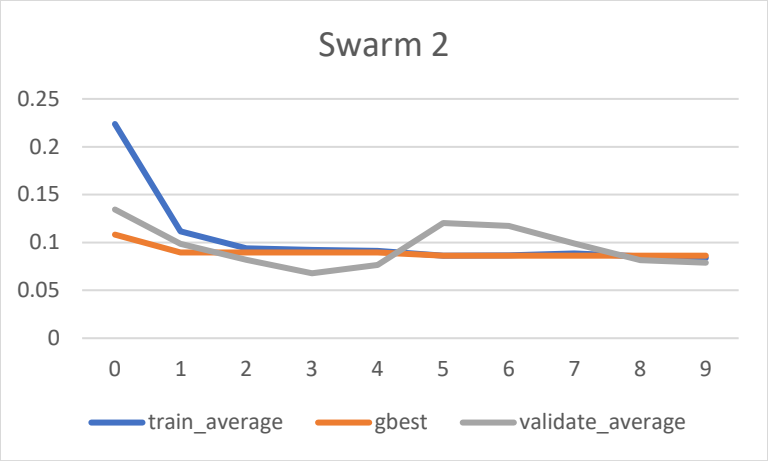
**ตารางที่ 6** ตารางแสดงโครงสร้างของแต่ละ Swarm

| 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 |

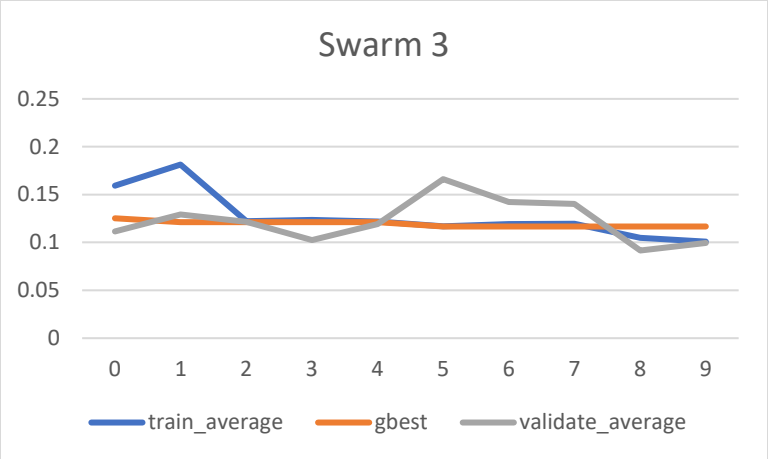
ซึ่งผลลัพธ์การฝึกสอนของแต่ละกลุ่ม Swarm จะนำเสนอในรูปแบบกราฟเส้นตรงและแผนภูมิแท่ง โดยรูปแบบกราฟเส้นตรงจะอธิบายถึง fitness ที่ได้กล่าวไปในหัวข้อ 1.3 ซึ่งกราฟประกอบไปด้วยเส้นตรงที่อธิบายถึงค่า fitness ของ การเทรนเฉลี่ย, ค่า gbest และ validate เฉลี่ย ในแต่ละ 10-folds cross-validation ซึ่งผลลัพธ์ที่ได้เป็นดังนี้



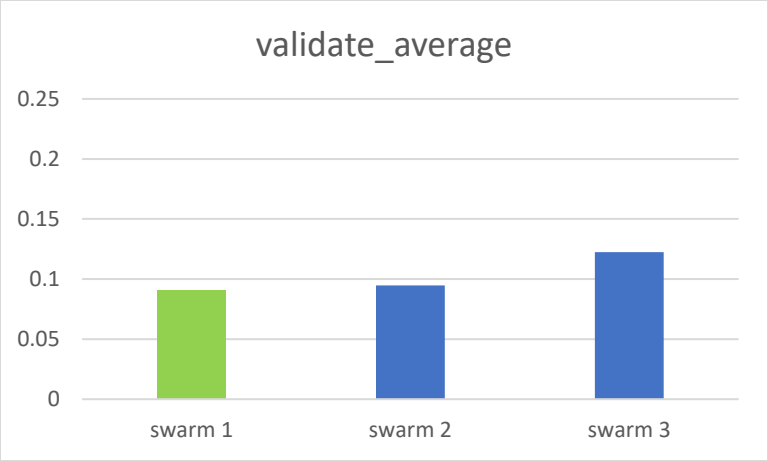

รูปที่ 5 กราฟเส้นตรงแสดงค่า fitness ของ Swarm 1




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 จะปรับตำแหน่งให้เข้าใกล้กับ 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.
6.  class Particle_of_swarm(object):
7.      def __init__(self, hiddenSize, inputSize, outputSize):
8.          # initiate layers
9.          self.inputSize = inputSize
10.         self.outputSize = outputSize
11.         self.hiddenSize = hiddenSize
12.
13.         layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
14.
15.         # initiate positions
16.         positions = []
17.         for i in range(len(layers)-1):
18.             p = np.random.uniform(-10,10,(layers[i], layers[i+1]))
19.             positions.append(p)
20.         self.positions = positions
21.         self.positions_best = positions
22.
23.         velocitys = []
24.         for i in range(len(layers) - 1):
25.             v = np.random.uniform(-10,10,(layers[i], layers[i+1]))
26.             velocitys.append(v)
27.         self.velocitys = velocitys
28.
29.         self.pbest = float('inf')
30.
```

```
31.     def feedForward(self, X):
32.         Output_node = X
33.         for i, p in enumerate(self.positions):
34.
35.             v = np.dot(Output_node, p)
36.             Output_node = self.sigmoid(v)
37.
38.         return Output_node
39.
40.     def sigmoid(self, s, deriv=False):
41.         if (deriv == True):
42.             return s * (1-s)
43.         return 1/(1 + np.exp(-s))
44.
45.     def object_funct(self, X, Y):
46.                     # Random data
47.         seed = randint(1, 25*100)
48.
49.         np.random.seed(seed)
50.         np.random.shuffle(X)
51.
52.         np.random.seed(seed)
53.         np.random.shuffle(Y)
54.
55.         sum_err = 0
56.         for j, input in enumerate(X):
57.             target = Y[j]
58.             output = self.feedForward(input)
59.
60.             sum_err += self._mae(target, output)
61.
62.         self.fx = sum_err/len(X)
63.         return self.fx
64.
65.     def _mae(self, target, output):
```

```
66.         return np.average(abs(target - output))
67.
68. def cross_validations_split(shape,folds):
69.     fold_size = int(shape * folds/100)
70.     k = 0
71.     index = []
72.     for i in range(1,folds+1):
73.         if i < folds:
74.             index.append([k,i*fold_size])
75.         else:
76.             index.append([k,shape])
77.         k = i*fold_size
78.     return index
```

```
79. Input = np.genfromtxt('data/AirQualityUCI_input.csv', delimiter=',')
80. Output = np.genfromtxt('data/AirQualityUCI_output.csv', delimiter=',')
81. particles = []
82. num_of_particle = 12
83.
84. for i in range(0, num_of_particle):
85.     par = Particle_of_swarm([4], 8, 2)
86.     particles.append(par)
87.
88. gbest = [0,float('inf')]
89. gbest_position = 0
90. #---------------------------------------------
91.
92. train_mean_pbest = []
93. train_gbest = []
```

```
94.
95.  test_mean_pbest = []
96.
97.  #---------------------------------------------------
98.  k=1
99.  for a,b in cross_validations_split(Input.shape[0],10):
100.     x_train = np.concatenate((Input[:a],Input[b+1:]))
101.     y_train = np.concatenate((Output[:a],Output[b+1:]))
102.     x_test = Input[a:b,:]
103.     y_test = Output[a:b]
104.     list_fx = []
105.     print("-------------------- fold : ",k," --------------------")
106.     k+=1
107.     for j in range(25):
108.         for i,p in enumerate(particles):
109.             fx = p.object_funct(x_train, y_train)
110.             list_fx.append(fx)
111.
112.         # check pbest
113.             if fx < p.pbest:
114.                 print("Update pbest ",i," : ",round(fx,3), " / ",round(p.pbest,3))
115.                 p.pbest = fx
116.                 p.positions_best = p.positions.copy()
117.
118.
119.
120.         #check gbest
121.             if fx < gbest[1]:
122.                 gbest[1] = fx
123.                 gbest[0] = i
124.                 gbest_position = p.positions.copy()
125.                 print("Update gbest",i," : ",fx)
126.
127.         # update velocity & position
128.         for i,p in enumerate(particles):
```

```
129.             for c in range(0, len(p.velocitys)):
130.                 d1 = 2
131.                 d2 = 3
132.                 d = d1+d2
133.                 g = 1 - (1/(d)) + math.sqrt(abs(d**2 - 4*d))/2
134.                 p.velocitys[c] = g*(0.2*p.velocitys[c] + (d1 * (p.positions_best[c]-
    p.positions[c])) + (d2 * (gbest_position[c] - p.positions[c])))
135.                 p.positions[c] += p.velocitys[c]
136.
137.         print(" Epoch : " , j+1 ," | err : ", sum(list_fx)/len(list_fx) )
138.
139.     train_mean_pbest.append(sum(list_fx)/len(list_fx))
140.     train_gbest.append(gbest[1])
141.
142.     list_fx_test = []
143.     for i,p in enumerate(particles):
144.         fx = p.object_funct(x_test, y_test)
145.         list_fx_test.append(fx)
146.     test_mean_pbest.append(sum(list_fx_test)/len(list_fx_test))
147.     print("=========== Test ===========")
148.     print("Error of test  :  ",round(sum(list_fx_test)/len(list_fx_test),6))
149.     print("Error of train :  ",round(sum(list_fx)/len(list_fx),6))
150.     print("gbest : ",round(gbest[1],6))
```