Computer Intelligence

Kunnapat Thippayapalaphonkul (590612113)

1 December 2019

รายงานผลการทดลองการบ้าน Swarm Intelligence

ในการบ้านรายงานนี้ นั้นได้ทำการเรียกใช้ Libraries 1). numpy 2). copy 3). random 4). Matplotlib 5). pandas โดยที่ numpy นั้นเอาไว้ใช้ในการคำนวณต่างๆ เช่น dot product, mean, random matrix เป็นต้น copy ใช้ทำการ copy List, random ใช้ในการเลือก Chromosome ในขั้นตอนต่างๆ matplotlib เพื่อ Plot Graph และ pandas ใช้เพื่ออ่านไฟล์ Data set เนื่องจากเป็น excel

รายงานผลการทดลองทำกับ Data set Air Quality Data set โดยที่ Data set นี้มี 15 Attributes โดยที่มี 9 Features, Attributes ที่ 5 จะเป็นค่าเฉลย True hourly averaged Benzone concentration in microg/m^3 (reference analyzer) โดยที่ให้ทำการ Regression โดยที่การบ้านนี้นั้นได้ทำการใช้ Neural Network ในการคำนวณหา Error และจากขั้นตอน Backpropagation เปลี่ยนไปใช้ Swarm Intelligence แทนในการบ้านนี้ โดยมีขั้นตอนดังนี้

- Preprocessing Data
- Cross Validation
- Neural Network
- Fitness
- Compare Pbest
- Compare Gbest
- Update Velocity
- Update Position

ขั้นตอน

Preprocessing Data

ในขั้นตอนนี้เช็ค Data ว่าสมบูรณ์รึเปล่าในแต่ละ Attributes จากนั้นใน Data จะมี 15 Attributes ใน Data set ชุดนี้ โดยจะมี Data โดยที่แต่ละ Attributes โดยเริ่มนับจาก Column แรกเริ่มเป็น 0 และที่ใช้ในการ Train ก็คือ Attributes [3, 6, 8, 10, 11, 12, 13, 14] และ Attributes ที่เป็นคำตอบในการ Regression คือ Attribute ที่ 5 ซึ่งตัวอย่าง Data set จะเป็นดัง รูปที่ 1 จะได้เห็นว่า Column 0, 1, 2, 4, 7, 9 นั้นจะไม่ใช้ในการบ้านครั้งนี้ แต่จะใช้ Column ที่เหลือทั้งหมด

Date	Time	CO(GT)	PT08.S1(CO	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
3/10/2004	18:00:00	2.6	1360	150	11.9	1046	166	1056	113	1692	1268	13.6	48.9	0.7578
3/10/2004	19:00:00	2	1292	112	9.4	955	103	1174	92	1559	972	13.3	47.7	0.7255
3/10/2004	20:00:00	2.2	1402	88	9.0	939	131	1140	114	1555	1074	11.9	54.0	0.7502
3/10/2004	21:00:00	2.2	1376	80	9.2	948	172	1092	122	1584	1203	11.0	60.0	0.7867
3/10/2004	22:00:00	1.6	1272	51	6.5	836	131	1205	116	1490	1110	11.2	59.6	0.7888
3/10/2004	23:00:00	1.2	1197	38	4.7	750	89	1337	96	1393	949	11.2	59.2	0.7848
3/11/2004	0:00:00	1.2	1185	31	3.6	690	62	1462	77	1333	733	11.3	56.8	0.7603
3/11/2004	1:00:00	1	1136	31	3.3	672	62	1453	76	1333	730	10.7	60.0	0.7702
3/11/2004	2:00:00	0.9	1094	24	2.3	609	45	1579	60	1276	620	10.7	59.7	0.7648
3/11/2004	3:00:00	0.6	1010	19	1.7	561	-200	1705	-200	1235	501	10.3	60.2	0.7517
3/11/2004	4:00:00	-200	1011	14	1.3	527	21	1818	34	1197	445	10.1	60.5	0.7465
3/11/2004	5:00:00	0.7	1066	8	1.1	512	16	1918	28	1182	422	11.0	56.2	0.7366
3/11/2004	6:00:00	0.7	1052	16	1.6	553	34	1738	48	1221	472	10.5	58.1	0.7353
3/11/2004	7:00:00	1.1	1144	29	3.2	667	98	1490	82	1339	730	10.2	59.6	0.7417
3/11/2004	8:00:00	2	1333	64	8.0	900	174	1136	112	1517	1102	10.8	57.4	0.7408
3/11/2004	9:00:00	2.2	1351	87	9.5	960	129	1079	101	1583	1028	10.5	60.6	0.7691
3/11/2004	10:00:00	1.7	1233	77	6.3	827	112	1218	98	1446	860	10.8	58.4	0.7552
3/11/2004	11:00:00	1.5	1179	43	5.0	762	95	1328	92	1362	671	10.5	57.9	0.7352
3/11/2004	12:00:00	1.6	1236	61	5.2	774	104	1301	95	1401	664	9.5	66.8	0.7951
3/11/2004	13:00:00	1.9	1286	63	7.3	869	146	1162	112	1537	799	8.3	76.4	0.8393
3/11/2004	14:00:00	2.9	1371	164	11.5	1034	207	983	128	1730	1037	8.0	81.1	0.8736
3/11/2004	15:00:00	2.2	1310	79	8.8	933	184	1082	126	1647	946	8.3	79.8	0.8778
3/11/2004	16:00:00	2.2	1292	95	8.3	912	193	1103	131	1591	957	9.7	71.2	0.8569
3/11/2004	17:00:00	2.9	1383	150	11.2	1020	243	1008	135	1719	1104	9.8	67.6	0.8185
3/11/2004	18:00:00	4.8	1581	307	20.8	1319	281	799	151	2083	1409	10.3	64.2	0.8065
3/11/2004	19:00:00	6.9	1776	461	27.4	1488	383	702	172	2333	1704	9.7	69.3	0.8319
3/11/2004	20:00:00	6.1	1640	401	24.0	1404	351	743	165	2191	1654	9.6	67.8	0.8133
3/11/2004	21:00:00	3.9	1313	197	12.8	1076	240	957	136	1707	1285	9.1	64.0	0.7419
3/11/2004	22:00:00	1.5	965	61	4.7	749	94	1325	85	1333	821	8.2	63.4	0.6905
3/11/2004	23:00:00	1	913	26	2.6	629	47	1565	53	1252	552	8.2	60.8	0.6657
3/12/2004	0:00:00	1.7	1080	55	5.9	805	122	1254	97	1375	816	8.3	58.5	0.6438
3/12/2004	1:00:00	1.9	1044	53	6.4	829	133	1247	110	1378	832	7.7	59.7	0.6308

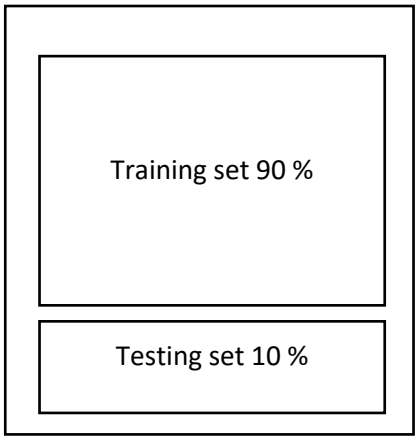
รูปที่ 1 ตัวอย่าง Data set

จาก รูปที่ 1 ใน Column ที่ 5 หรือ Column ที่ชื่อ C6H6 นั้นจะเป็นคำตอบของ Data set ชุดนี้ ซึ่งข้อมูลนั้นเป็นเป็นตัวเลขทั้งหมดเลย ดังนั้นจึงทำ Regression เพื่อที่จะได้ Predict Benzone Concentration และข้อมูลในทุก Column ที่สนใจนั้นเป็นตัวเลขทั้งหมดแต่ในแต่ละ Column นั้นมีระดับที่ต่างกัน จึงทำการ Normalize ในแต่ละ Column เพื่อที่จะให้ Data นั้นอยู่ในระดับเดียวกัน [0, 1] ในแต่ละ Column จะทำการหา Max และ Min เพื่อนำมาคิด x ในแต่ละ Column ดัง [1]

$$\frac{x - x_{min}}{x_{max} - x_{min}} \qquad [\ 1\ ]$$

Cross Validation

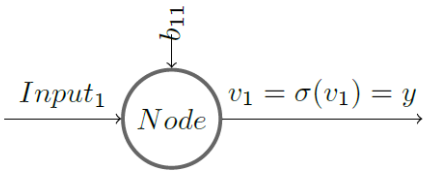
ได้ทำการแบ่ง Data set เป็น 10 Fold โดยที่แต่ละ Fold นั้นแบ่งเป็น Training set 90 % และ Testing set 10 % ดังรูปที่ 2 ทำเป็นจำนวน 10 Fold โดยที่ Data set ชุดนี้มีขนาดจำนวน 9,357 Sample แบ่งเป็นให้ Testing set ประมาณ 10 % เท่ากับ 935 Sample ที่เหลือแบ่งใส่ Training set จำนวน 8,422 Sample โดยที่ Training set จะไม่มีข้อมูลของ Testing set เลยเพื่อเช็คความถูกต้องของ Neural Network ที่ได้ทำการ Train ว่าไม่ได้ Overfitting กับ Training set จนเกินไป



รูปที่ 2 ตัวอย่างการทำ Cross Validation

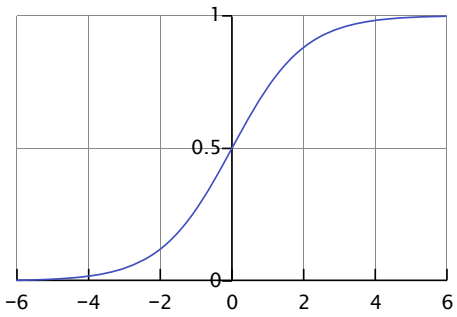
Neural Network

ในขั้นตอนนี้จะทำการสร้าง Neural Network โดยที่จะกำหนดค่าเริ่มต่างๆในกับ Network เพื่อที่จะ Train Network ให้ได้ผลตามที่เราต้องการ Neural Network จะมีการคำนวณดัง รูปที่ 3



รูปที่ 3 Node ของ Neural Network

หลังจากคำนวณแต่ละ Node เสร็จ จะทำการนำค่าที่คำนวณได้ไปทำการผ่าน Activation Function โดยที่ Activation Function ที่ผมได้เลือกใช้ในการบ้านนี้คือ Sigmoid Function โดยผลที่นำไปผ่าน Sigmoid Function จะได้ผลดัง รูปที่ 4



รูปที่ 4 ค่า Sigmoid Function

ผลลัพธ์ที่ได้จาก Sigmoid Function นั้นจะอยู่ระหว่าง [0, 1] ซึ่งทำการคำนวณแต่ละ Node ไปเรื่อย ๆ จนถึง Node สุดท้าย เพื่อที่จะหาค่า Error ของ Neural Network โดยการกำหนดค่าเริ่มต้นต่างๆให้กับ Neural Network มีดังนี้

- Input, Output ของ Neural Network สำหรับ Training set
- กำหนดจำนวน Node ในแต่ละ Layer
- จำนวน Population
- จำนวน Group_population

Fitness

ขั้นตอนนี้หลังการคำนวณ Loss หรือ Error จาก Network โดยปกติเราจะอยากได้ค่า Error ของ Network ให้มีค่าน้อยที่สุดเท่าที่จะทำได้ โดยที่ในการบ้านนี้จะใช้การคำนวณ Error ทั้งหมดคือ Mean Absolute Error (MAE) และค่า Fitness จะเป็นค่าผลลัพธ์ของแต่ละ Group_population ก็จะใช้ค่า MAE เช่นเดียวกับ Error เลยดัง [2]

$$Fitness = \frac{1}{m}\sum_{i=0}^{m}|err_i| \quad [2]$$

Compare Pbest

ในขั้นตอนนี้นั้นจะทำการเปรียบเทียบเพื่อหาค่า Fitness ที่ดีที่สุดในตัวเองที่สามารถทำได้ในแต่ละ Iteration ซึ่งถ้าเจอค่าที่ดีที่สุดแล้วจะทำการอัพเดทค่าที่ดีที่สุดและตำแหน่งที่ดีที่สุด ให้กับตัว Pbest

Compare Gbest

ในขั้นตอนนี้นั้นจะเหมือนกับขั้นตอน Pbest เลยก็คือทำการเทียบค่า Fitness ของทุกคนเพื่อหาค่าของ Fitness ที่ดีที่สุดในสังคมและในทุก Iteration เพื่อที่จะได้ค่า Fitness ที่ดีที่สุดและตำแหน่งที่ดีที่สุดด้วย

Update Velocity

ในขั้นตอนนี้นั้นจะทำการอัพเดทค่าความเร็วของแต่ละกลุ่มและภายในกลุ่มคือแต่ละคนด้วย โดยที่ใช้สมการ Update ความเร็วดัง [3] ซึ่งความเร็วที่ได้ Update ในแต่ iteration นั้นขึ้นกับ x_{pbest_i} และ $\mathrm{x_{gbest}}$ เพื่อกำหนดความเร็วในแต่ละกลุ่มว่าควรเป็นเท่าไหร่จากความเร็วในรอบที่แล้ว

$$v_i(t) = v_i(t-1) + \rho_1\left(x_{pbest_i} - x_i(t)\right) + \rho_2\left(\mathrm{x_{gbest}} - \mathrm{x_i(t)}\right) \qquad [3]$$

Update Position

ในขั้นตอนนี้นั้นจะทำการอัพเดทตำแหน่งของแต่ละกลุ่มโดยที่นำ ความเร็วที่คำนวณมาจาก [3] มาอัพเดทตำแหน่งของแต่ละกลุ่มดัง [4]

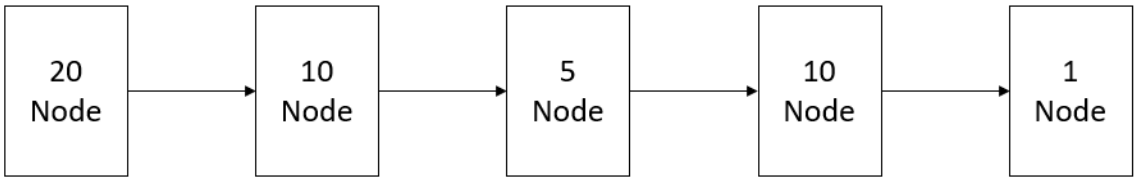
$$x_i(t) = x_i(t-1) + v_i(t) \qquad [4]$$

ผลการทดลอง

การทดลองครั้งนี้นั้นจะทดลองโดยมี 50 Group Population และมี 100 Iteration

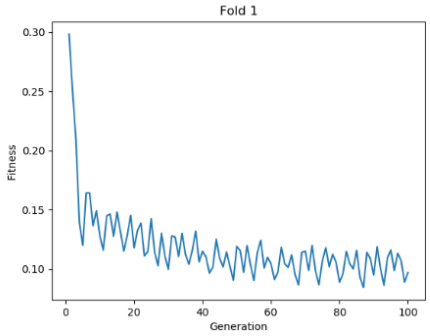
การทดลองครั้งที่ 1

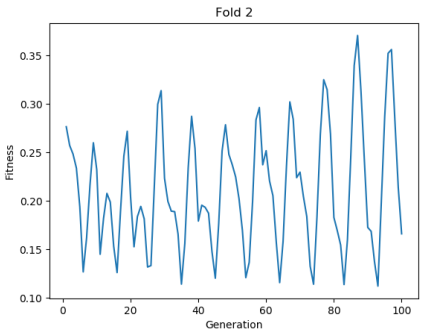
ซึ่งจะทำการทดลองโดยสร้าง Neural Network ที่มี Layer และ Node ดัง รูปที่ 5

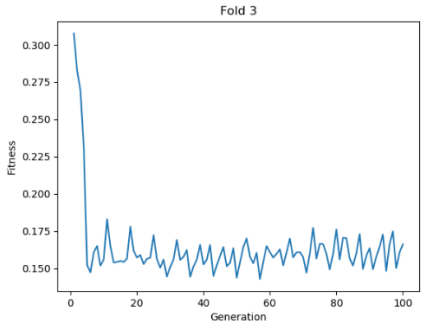


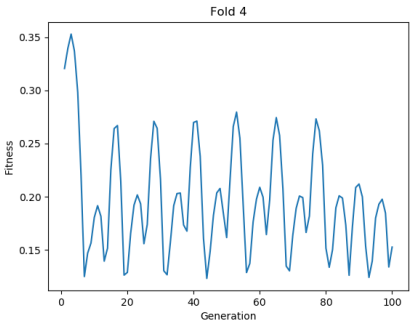
รูปที่ 5 จำนวน Layer และ Node ในแต่ละ Layer

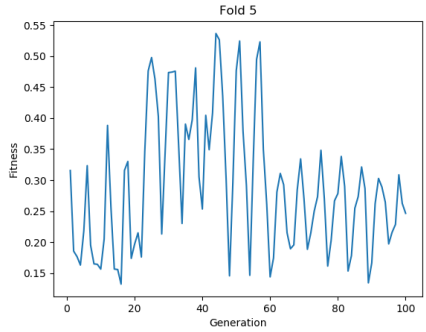
ตาราง 1 ค่า Fitness ของแต่ละ Fold ของการทดลองครั้งที่ 1

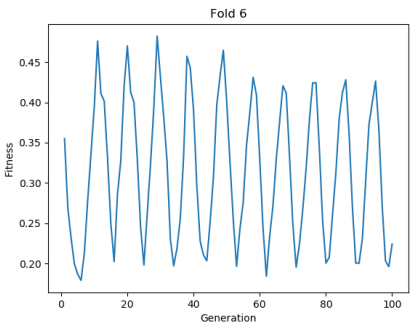


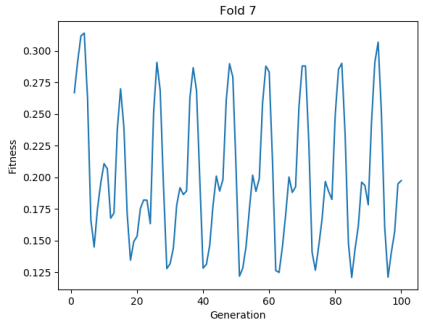


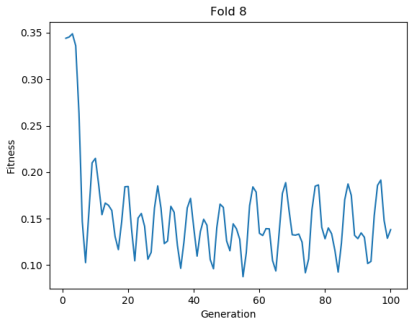


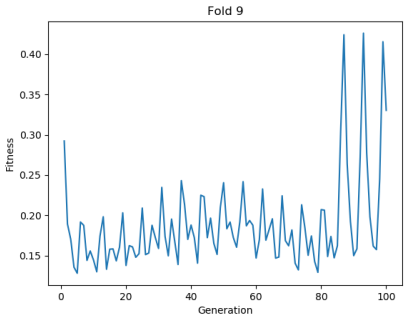


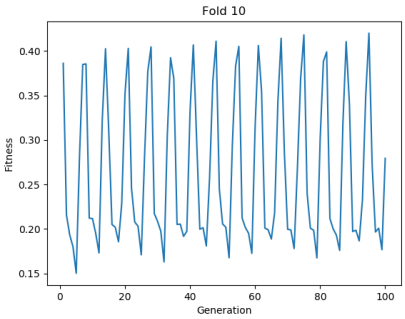












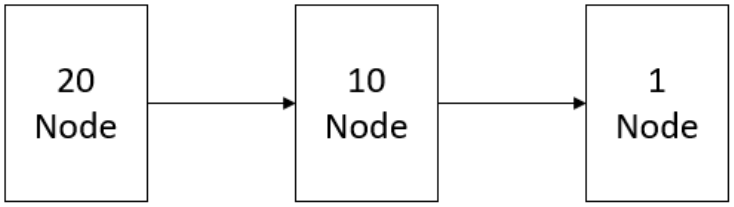
จาก ตาราง 1 จะเห็นได้ว่ามีหลายๆ Fold ที่ลู่เข้า แต่ในหลายๆ Fold ที่ไม่ได้ลู่เข้าคิดว่าเนื่องจากผมได้ทำการ Random ρ_1, ρ_2 1 ครั้งต่อหนึ่ง Fold ซึ่งอาจจะ Random แล้วได้ค่าที่มากจนเกินไปทำให้ค่า MAE ที่คำนวณมาจาก Network นั้นไม่สามารถลู่เข้าได้ในแต่ละ Iteration และจาก ตาราง 2 ค่า Fitness ที่ได้จาก Test set ของแต่ละ Fold ที่ดีที่สุดคือ Fold 1

ตาราง 2 แสดงค่า Fitness ของ Test set แต่ละ Fold ของการทดลองครั้งที่ 1

Fold	Fitness
Fold 1	0.0158610425710746
Fold 2	0.02451764203349384
Fold 3	0.03069327744343776
Fold 4	0.035096333685947785
Fold 5	0.03575401426818805
Fold 6	0.07399093286914757
Fold 7	0.032029285621598354
Fold 8	0.036856660548262116
Fold 9	0.05192566953728086
Fold 10	0.10832223634059243

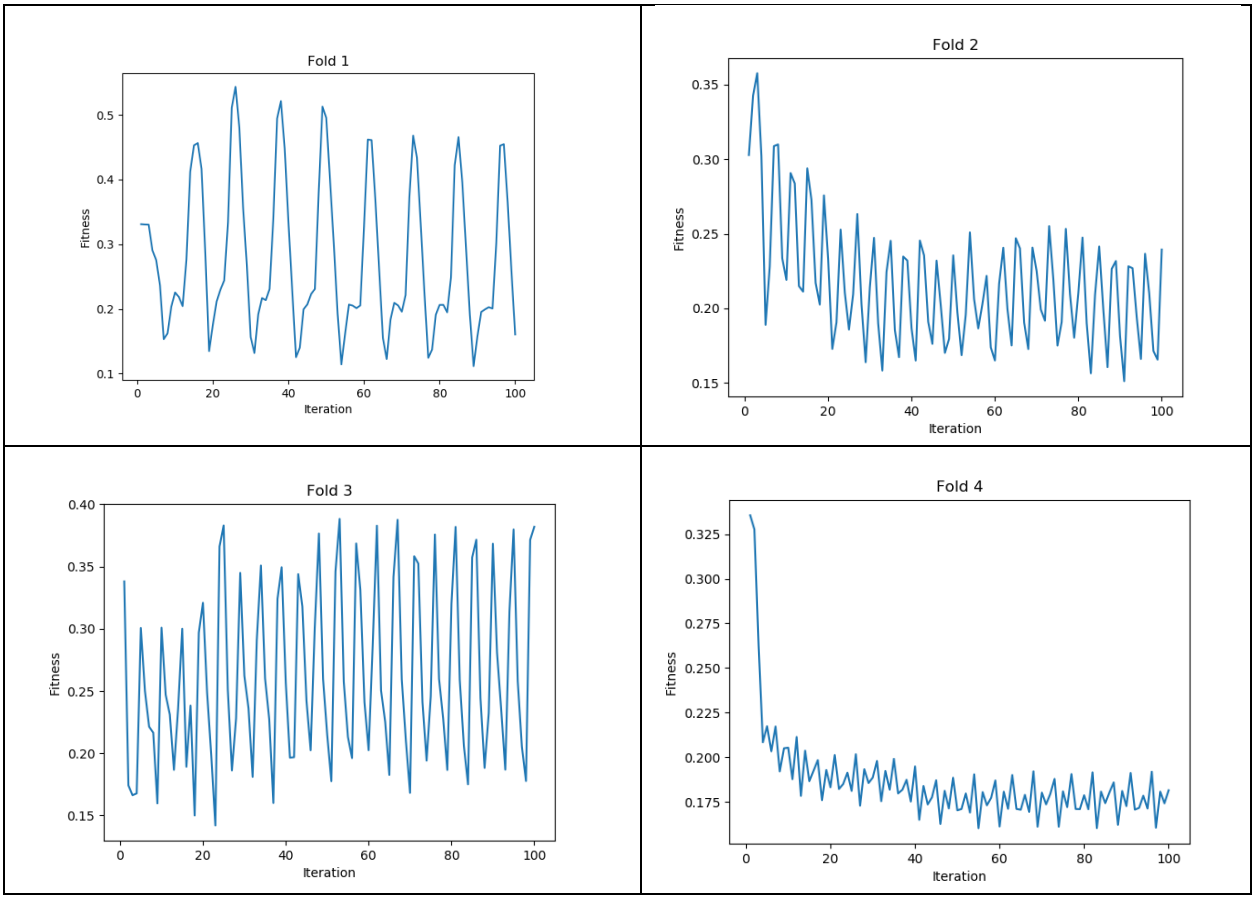
การทดลองครั้งที่ 2

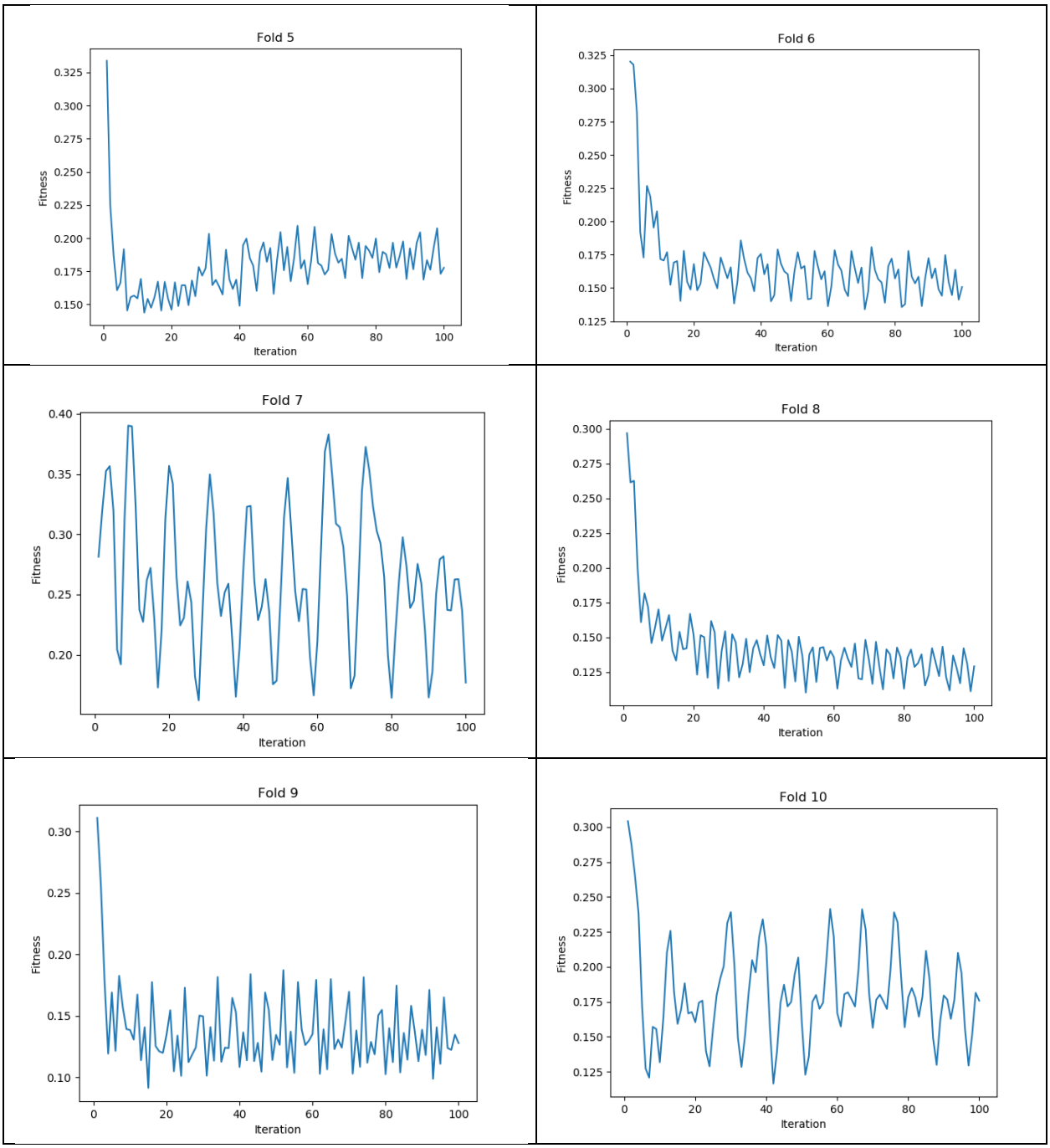
การทดลองครั้งที่ 2 จะทำการลดจำนวน Layer ลงมาดัง รูปที่ 6



รูปที่ 6 จำนวน Layer และ Node ในแต่ละ Layer

ตาราง 3 ค่า Fitness ของแต่ละ Fold ของการทดลองครั้งที่ 2





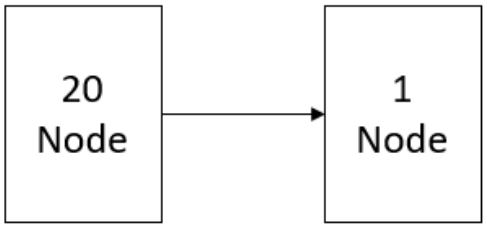
จาก ตาราง 3 จะเห็นว่าหลายๆ Fold นั้นไม่ได้ลู่เข้าเพราะการ Random ρ_1, ρ_2 ขึ้นมานั้นอาจจะเยอะเกินไปทำให้แต่ละ Group_Population นั้นไม่สามารถที่จะลู่เข้าได้ แต่ก็จะมีหลายๆ Fold ที่ลู่เข้าและจาก ตาราง 4 ค่า Fitness ของ Test set ที่น้อยที่สุดจากทั้ง 10 Fold นั้นคือ Fold 1 แต่จาก ตาราง 3 Fold ที่ 1 นั้นไม่ได้ลู่เข้าแต่ทุกๆ Fold นั้นได้ค่า Fitness ที่น้อยทั้งหมด

ตาราง 4 แสดงค่า Fitness ของ Test set แต่ละ Fold ของการทดลองครั้งที่ 2

Fold	Fitness
Fold 1	0.010234164561734326
Fold 2	0.0392490543667484
Fold 3	0.06558196379984695
Fold 4	0.026689723109169575
Fold 5	0.03490345632715396
Fold 6	0.023004671560783153
Fold 7	0.021112299037273317
Fold 8	0.0479491088793699
Fold 9	0.027763087890868184
Fold 10	0.02774983859562331

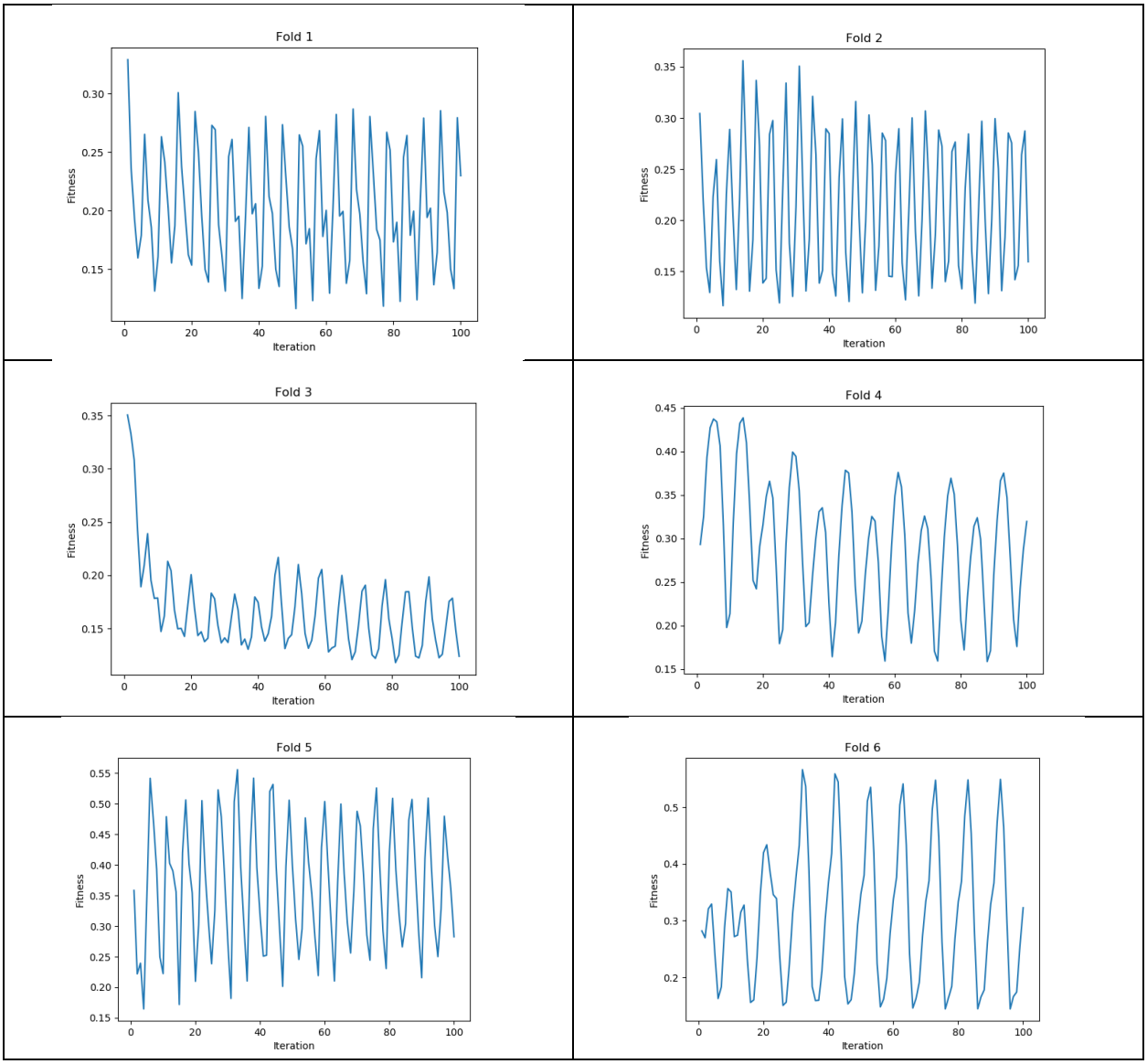
การทดลองครั้งที่ 3

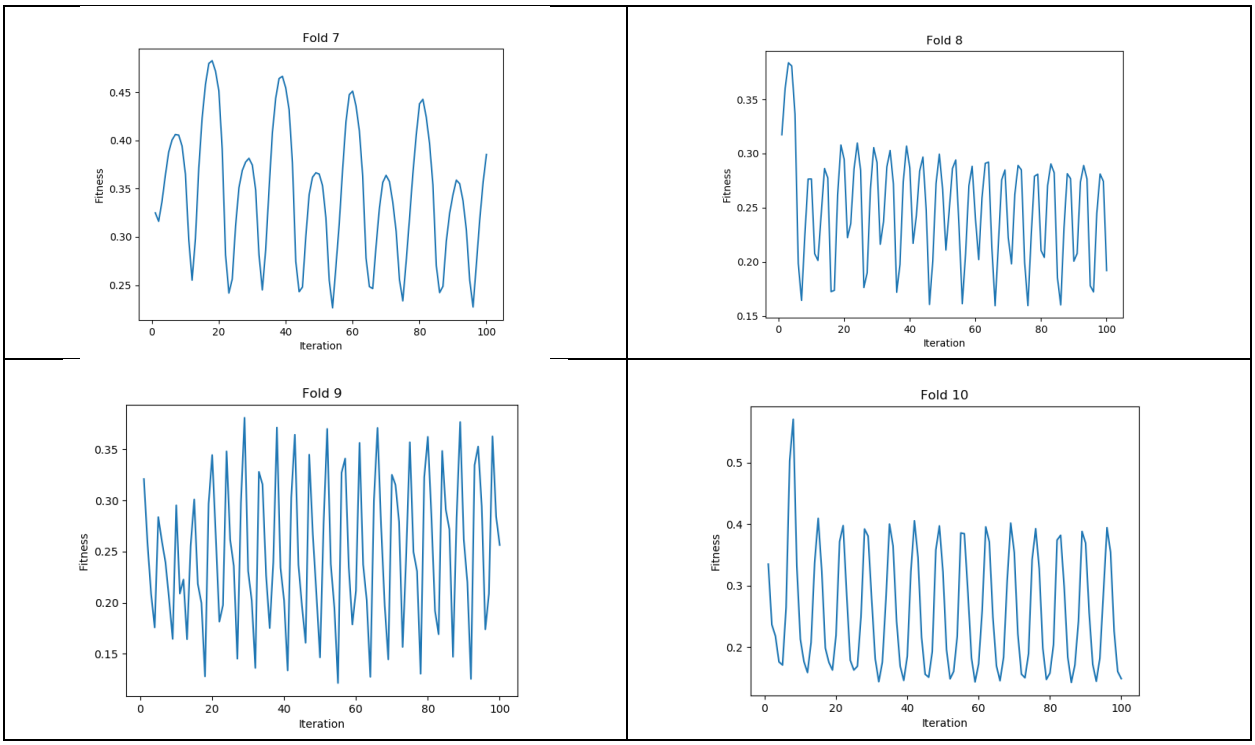
การทดลองครั้งที่ 3 จะทำการลดจำนวน Layer ลงมาดัง รูปที่ 7



รูปที่ 7 จำนวน Layer และ Node ในแต่ละ Layer

ตาราง 5 ค่า Fitness ของแต่ละ Fold ของการทดลองครั้งที่ 3





จาก ตาราง 5 นั้นจะเห็นได้ว่าเกือบทุก Fold นั้นไม่ได้ลู่เข้าเลยแต่ค่าในแต่ละ Iteration นั้นกระโดดไปมาอาจจะเพราะการ Random $\boldsymbol{\rho_1, \rho_2}$ แต่ละ Fold นั้นสูงทั้งหมดแต่นั้นอาจจะไม่ใช่เหตุผลเดียวโดยจาก Network ในการทดลองครั้งนี้เป็นไปตามรูปที่ 7 จะเห็นได้ว่าไม่มี Hidden Layer เลยและจำนวน Node นั้นก็ยังน้อยอีกด้วยทำให้ค่าในแต่ละ Fold นั้นไม่ได้ลู่เข้าแต่กลับกันจาก

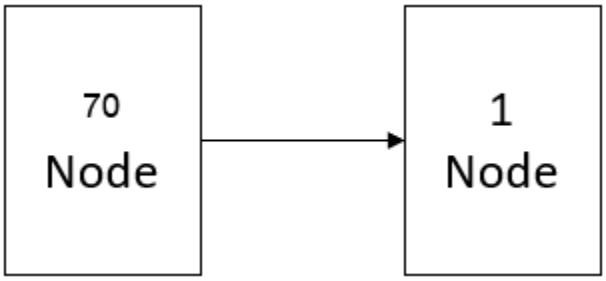
ตาราง *6* จะเห็นได้ว่าค่า Fitness กับ Test set แต่ละ Fold นั้นก็ยังได้ต่ำอยู่และ Fold ที่มีค่า Fitness ต่ำที่สุดคือ Fold 2

ตาราง 6 แสดงค่า Fitness ของ Test set แต่ละ Fold ของการทดลองครั้งที่ 3

Fold	Fitness
Fold 1	0.034780554758632384
Fold 2	0.01162929276929691
Fold 3	0.02030348051973223
Fold 4	0.08624744167831479
Fold 5	0.04403095896623724
Fold 6	0.10940734642801157
Fold 7	0.13114373633746823
Fold 8	0.05597359275446113
Fold 9	0.01946293575171072
Fold 10	0.03727316575478198

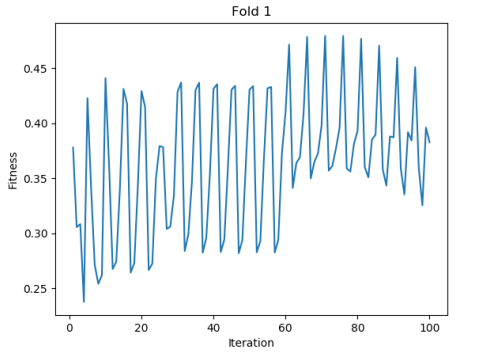
การทดลองครั้งที่ 4

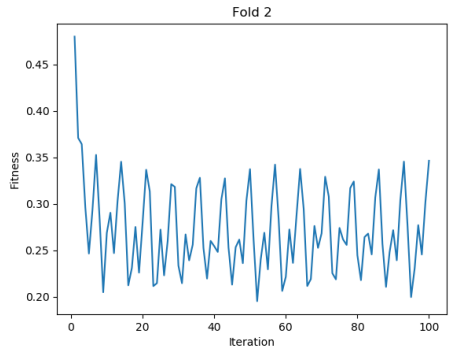
จากการทดลองครั้งที่ 3 จะทำการลดจำนวน Layer ลงมาดัง รูปที่ 8

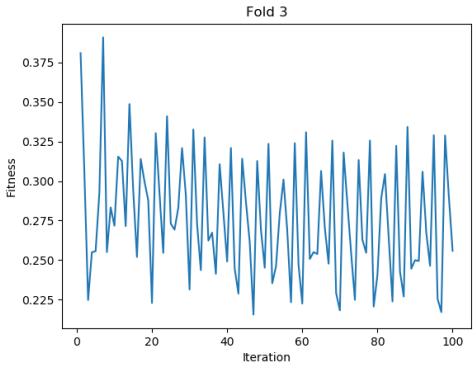


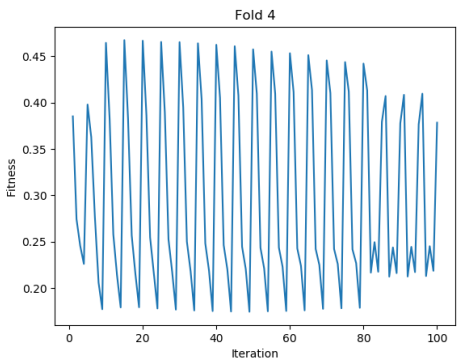
รูปที่ 8 จำนวน Layer และ Node ในแต่ละ Layer

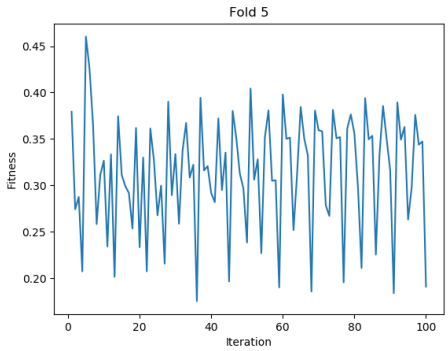
ตาราง 7 ค่า Fitness ของแต่ละ Fold ของการทดลองครั้งที่ 4

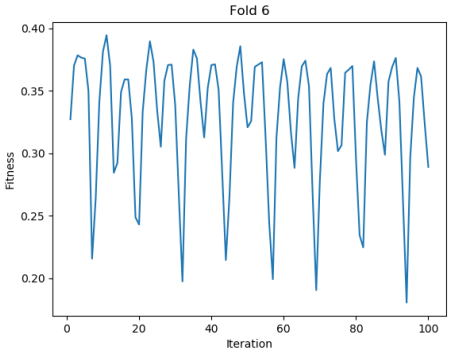


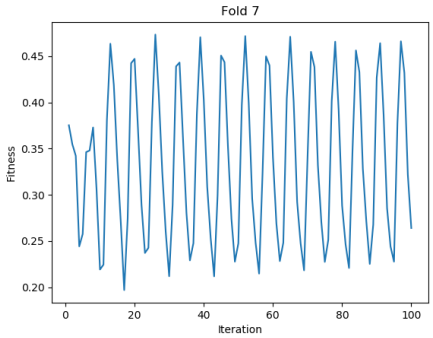


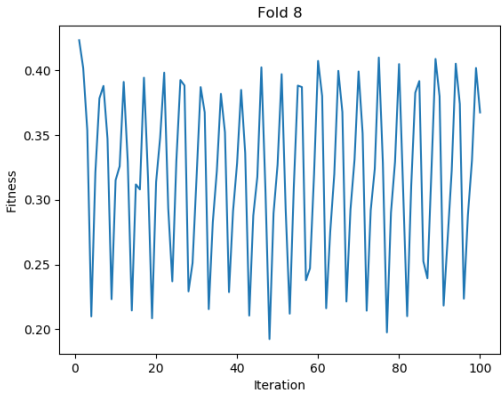

Fold 3
Fitness
Iteration

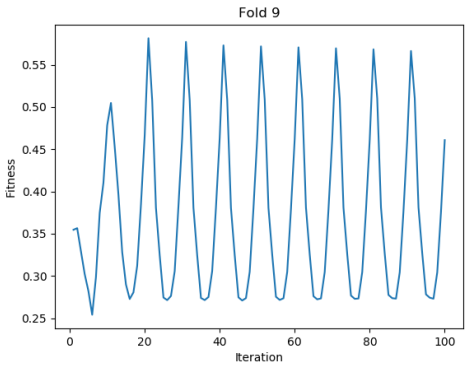

Fold 4
Fitness
Iteration

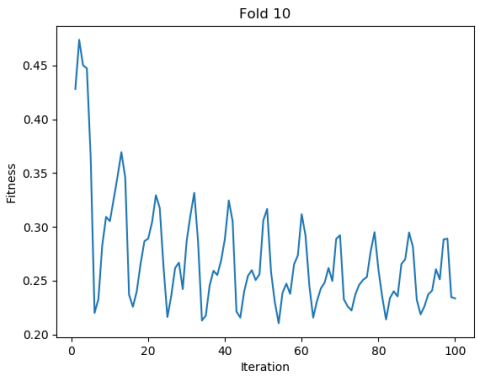

Fold 5
Fitness
Iteration


Fold 6
Fitness
Iteration


Fold 7
Fitness
Iteration


Fold 8
Fitness
Iteration


Fold 9
Fitness
Iteration


Fold 10
Fitness
Iteration

จากการทดลองที่ 3 นั้นได้ทำการเพิ่มจำนวน Node ในการทดลองที่ 4 นี้ดัง รูปที่ 8 เพื่อดูว่าถ้าเพิ่มจำนวน Node แล้วแต่ละ Fold นั้นจะลู่เข้าหรือเปล่า โดยที่ผลการทดลองดัง ตาราง 7 ซึ่งจะเห็นได้ว่าไม่ได้มีผลมากนักและจาก ตาราง 8 จะเห็นได้ว่าแต่ค่า Fitness ของแต่ละ Fold นั้นสูงที่สุดในการทดลองทั้ง 4 ครั้ง ค่า Fitness ที่น้อยที่สุดคือ Fold 7

ตาราง 8 แสดงค่า Fitness ของ Test set แต่ละ Fold ของการทดลองครั้งที่ 4

Fold	Fitness
Fold 1	0.13734909974084558
Fold 2	0.08268511117110684
Fold 3	0.054716382039849576
Fold 4	0.1244549756095977
Fold 5	0.10803180740706839
Fold 6	0.08731068734815368
Fold 7	0.04267949753843406
Fold 8	0.046782312134706905
Fold 9	0.18069372544965698
Fold 10	0.04582749685045916

สรุปผลการทดลอง

จากผลการทดลองทั้ง 4 ครั้งนั้นจะเห็นได้ว่าจากกราฟของแต่ละ Fold ทั้ง 10 Fold นั้นพอลดจำนวน Hidden Layer ลงนั้นจะเห็นได้ว่าค่า Fitness นั้นคิดมาจาก [2] คือค่า MAE นั้นของแต่ละ Fold Training set นั้นไม่ได้แย่ไปกว่าเดิมเลยเพราะเนื่องจากได้ทำการ Random ρ_1, ρ_2 ซึ่งสองค่านี้อาจจะได้สูงเลยทำให้หลายๆ Fold นั้นไม่สามารถที่จะลู่เข้าได้แต่จะเห็นได้ว่าการทดลองที่ 3, 4 นั้นถ้า Fitness นั้นแทบจะไม่ลู่เข้าเลยเนื่องจากทำการลด Hidden Layer ไปจึงทำให้ค่าในแต่ละ Fold ไม่ลู่เข้าแต่จากการทดลองทั้ง 4 ครั้งดัง ตาราง 9 นั้นค่า Fitness ถึงจะลด Hidden Layer ลงแต่ค่า Fitness ที่ดีที่สุดของแต่ละ Fold การทดลองนั้นแทบจะไม่แตกต่างกันเลยดังนั้นคิดว่าปัจจัยที่มีผลต่อการอัพเดทตำแหน่งที่สำคัญของแต่ละ Group_Population นั้นคือ ρ_1, ρ_2 เพราะจากการทดลองทั้ง 4 ครั้งนั้นได้ทำการ Random ครั้งเดียวต่อ Fold ทำให้ถ้าเกิดว่า Random ได้ค่ามากจะทำให้ไม่สามารถลู่เข้าได้

ตาราง 9 สรุปของการทดลอง Fold ไหนที่มีค่า Fitness มากที่สุด

การทดลอง	Fold	Fitness
1	1	0.0158610425710746
2	1	0.010234164561734326
3	2	0.01162929276929691
4	7	0.04267949753843406

ภาคผนวก

```
import numpy as np
import copy
import random
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

#### Attributes | 3, 6, 8, 10, 11, 12, 13, 14 to input
#### Attributes | 5 to output

def main():
    input_x, true_x = getData()

    input_x = transposeList(input_x)

    x_train, y_train, x_test, y_test= createFold(10, input_x, true_x)


    for i in range(len(x_train)): #### Fold
        print("Fold {}" .format(i+1))

        neural = NeuralNetwork(x_train[i], y_train[i])
        neural.addLayer(70)
        # neural.addLayer(10)
        # neural.addLayer(5)
        # neural.addLayer(10)
        neural.addLayer(1)

        neural.fit(50, 100)

        neural.plotFitness(i+1)
        neural.evaluate(x_test[i], y_test[i])

        print("---------------")

def getData(): ### return list
    path = 'AirQualityUCI.xlsx'
    listdata = []

    df = pd.read_excel(path)
    x = []
    cou = 0
```

```
    for atr in df.columns:
        if cou == 3 or cou == 6 or cou == 8 or cou >= 10 :
            x.append(normalizeList(df[atr]))
        elif cou == 5 :
            y = normalizeList(df[atr])
        cou += 1

    return x, y

def transposeList(datalist):
    return list(map(list, zip(*datalist)))

def normalizeList(x):
    # x = transposeList(x)
    nor_x = []

    minx = min(x)
    maxx = max(x)

    for i in range(len(x)):
        nor = (x[i] - minx)/(maxx - minx)
        nor_x.append(nor)
    return nor_x

class NeuralNetwork:
    def __init__(self, inpu, out):
        # np.random.seed(1)
        self.Fullinput = inpu
        self.FullTrueOutput = out
        self.input = np.asarray(copy.deepcopy(inpu))
        self.weight = []
        self.bias = []
        self.predict = 0
        self.lr = 0
        self.E = 0
        self.countLayer = 0
        self.Node = []
        self.output = []
        self.TrueOutput= None
        self.cou = 0
        self.deltaweight = []
        self.err = []
        self.deltabias = []
        self.gradient = [] # back to front
        self.momentum = 0
```

```
        self.Loss = []
        self.minimumLoss = 10000000
        self.rememberweightT_1 = []
        self.countsample = 0
        self.rememberweightT_2 = []
        self.npopulation = 0
        self.group_pop = []
        self.pbest = []
        self.gbest = [[100000, 0]]
        self.velocity = []

    def addLayer(self, node):
        self.countLayer+=1
        self.Node.append(node)

    def sigmoid(self, v):
        return 1/(1+np.exp(-v))

    def diffsigmoid(self, y):
        return y*(1-y)

    def createweight(self):
        for i in range(len(self.Node)):
            if ( i == 0 ):
                self.weight.append(2*np.random.rand(len(self.input), self.Node[i]
) - 1)
                self.bias.append(2*np.random.rand(self.Node[i]) - 1)
                # self.deltaweight.append(np.ones((len(self.input), self.Node[i])
))
                # self.deltabias.append(np.ones(self.Node[i]))
                # self.rememberweightT_1.append(np.zeros((len(self.input), self.N
ode[i])))
                # self.rememberweightT_2.append(np.zeros((len(self.input), self.N
ode[i])))
            else:
                self.weight.append(2*np.random.rand(self.Node[i-
1], self.Node[i]) - 1)
                self.bias.append(2*np.random.rand(self.Node[i]) - 1)
                # self.deltaweight.append(np.ones((self.Node[i-
1], self.Node[i])))
                # self.deltabias.append(np.ones(self.Node[i]))
                # self.rememberweightT_1.append(np.zeros((len(self.input), self.N
ode[i])))
                # self.rememberweightT_2.append(np.zeros((len(self.input), self.N
ode[i])))
```

```
    def FeedForward(self, chromosome): # each sample, each generation
        self.output = []
        self.output.append(self.input.T)
        out = np.array(self.output[0])
        for i in range(len(self.Node)): # feed in each layer
            v = np.dot(copy.deepcopy(chromosome[i].T), copy.deepcopy(self.output[
i]))
            out = self.sigmoid(v)
            self.output.append(out)

        return out

    def fit(self, npopulatation, iteration):
        self.npopulation = npopulatation
        self.fitnessplot = []
        for k in range(iteration) :
            fitne = []
            for i in range(npopulatation): # individual
                if k == 0 :
                    temp = self.population()
                    temp = np.asarray(copy.deepcopy(temp))
                    out = self.FeedForward(temp)

                else :
                    out = self.FeedForward(copy.deepcopy(self.group_pop[i][1]))

                self.Loss = []

                for j in range(len(self.Fullinput)): # sample
                    err = out[0][j] - copy.deepcopy(self.FullTrueOutput[j])
                    self.Loss.append(err)
                mae = np.asarray(copy.deepcopy(self.Loss))

                fitness = self.fitness(mae)
                fitne.append(fitness)

                #### compare find pbest and gbest
                if k == 0 :
                    self.group_pop.append([fitness, temp])
                    self.velocity.append(copy.deepcopy(self.population()))
                    self.pbest.append(copy.deepcopy(self.group_pop[i]))
                # else :
                if fitness < self.pbest[i][0] : ### best of self
                    self.pbest[i][0] = copy.deepcopy(fitness)
```

```
                    self.pbest[i][1] = copy.deepcopy(self.group_pop[i][1])

                if  fitness < self.gbest[0][0] : ### best of global
                    self.gbest[0][0] = copy.deepcopy(fitness)
                    self.gbest[0][1] = copy.deepcopy(self.group_pop[i][1])

                # update velocity
                if k + i == 0 :
                    rho1, rho2 = self.rho()

                self.velocity[i] = copy.deepcopy(self.velocity[i]) + rho1*(copy.d
eepcopy(self.pbest[i][1]) - copy.deepcopy(self.group_pop[i][1])) + rho2*(copy.dee
pcopy(self.gbest[0][1]) - copy.deepcopy(self.group_pop[i][1]))

                # update position
                self.group_pop[i][1] = copy.deepcopy(self.group_pop[i][1]) + copy
.deepcopy(self.velocity[i])

            self.fitnessplot.append(np.mean(np.asarray(copy.deepcopy(fitne))))

    def population(self): # return list
        weight = []
        for  i in range(len(self.Node)):
            if (i == 0):
                weight.append(np.random.uniform(-
1, 1, (len(self.Fullinput[0]), self.Node[i])))
            else :
                weight.append(np.random.uniform(-1, 1, (self.Node[i-
1], self.Node[i])))

        return weight

    def fitness(self, mae): #### Cal fitness from error | input Integer | how to
fitness maximum
        return np.mean(np.abs(mae))

    def evaluate(self, x_test, y_test):
        x_test = np.asarray(copy.deepcopy(x_test))
        y_test = np.asarray(copy.deepcopy(y_test))

        self.input = x_test
        fitne = []
        for i in range(len(self.group_pop)):

            predict = self.FeedForward(self.group_pop[i][1])
```

```
            Loss = []
            for j in range(len(y_test)):
                err = predict[0][j] - y_test[j]
                Loss.append(err)

            mse = np.asarray(copy.deepcopy(Loss))
            mse = np.power(mse, 2)/ 2
            mse = np.mean(mse) # scalar
            fitness = self.fitness(mse)
            fitne.append(fitness)

        fitne = np.asarray(copy.deepcopy(fitne))
        fitne = np.mean(fitne)
        print("Fitness = {}" .format(fitne))

    def plotFitness(self, fold):
        fitness = self.fitnessplot
        fig, ax = plt.subplots()
        ax.plot(range(1, len(fitness)+1), fitness)
        ax.set(xlabel='Iteration', ylabel='Fitness', title='Fold {}' .format(fold
))
        fig.savefig("Fold {}.png" .format(fold))

    def rho(self):
        r1 = np.random.uniform(0, 1)
        r2 = np.random.uniform(0, 1)

        c1 = np.random.uniform(0, 2)
        c2 = np.random.uniform(0, 2)

        rho1 = r1*c1
        rho2 =  r2*c2

        return rho1, rho2

def createFold(fold, x_data, y_data):
    crossvalidation = int(len(x_data)*fold/100)
    # print(len(x_data))
    x_train_testingset = []
    y_true_testingset = []

    x_train_trainingset = []
    y_true_trainingset = []
```

```
    for i in range(fold):
        if i == (fold-1) :
            x_train_testingset.append(x_data[0+i*crossvalidation:len(x_data)])
            y_true_testingset.append(y_data[0+i*crossvalidation:len(y_data)])

            x_train_trainingset.append(x_data[0:0+i*crossvalidation])
            y_true_trainingset.append(y_data[0:0+i*crossvalidation])
        else:
            x_train_testingset.append(x_data[0+i*crossvalidation:crossvalidation+
i*crossvalidation])
            y_true_testingset.append(y_data[0+i*crossvalidation:crossvalidation+i
*crossvalidation])

            x_train_trainingset1 = x_data[0:i*crossvalidation]
            x_train_trainingset2 = x_data[crossvalidation*(i+1):len(x_data)]

            x_train_trainingset.append(x_train_trainingset1 + x_train_trainingset
2)

            y_true_trainingset1 = y_data[0:i*crossvalidation]
            y_true_trainingset2 = y_data[crossvalidation*(i+1):len(x_data)]

            y_true_trainingset.append(y_true_trainingset1 + y_true_trainingset2)

    return x_train_trainingset, y_true_trainingset, x_train_testingset, y_true_te
stingset

if __name__ == "__main__":
    main()
```