261456 Introduction to Computer Intelligence

**Computer Assignment 4**

นาย ดรันภพ เป็งคำตา 580610642

รายงานเป็นรายงานเกี่ยวกับการทดลองเรื่อง **Particle Swarm Optimization (PSO)** ด้วยการทำการ implement โปรแกรมตามประบวนการทำงานของ PSO เพื่อที่จะทำการหาค่าน้ำหนักที่เหมาะสมสำหรับโครงสร้างของ Multi layers perceptron (MLP) เพื่อใช้ทำนายค่าความเข้มข้นของเบนซิลในอากาศต่อชั่วโมง ของเวลาใน 5 วัน และ 10 วันถัดไป

**รูปแบบการทดลอง**

จะเป็นทดลองผลกระทบของ connectivity ของ node ต่อผลลัพธ์ของ PSO เนื่องจาก PSO มีการกำหนด neighbor ของแต่ละ node เพื่อใช้ในการกำหนดขอบเขตของ Local solution โดยค่า connectivity ที่จะใช้ทดสอบคือ **1** ตัว **5** ตัว และ **30** ตัว จากประชากรทั้งหมด **30** ตัว

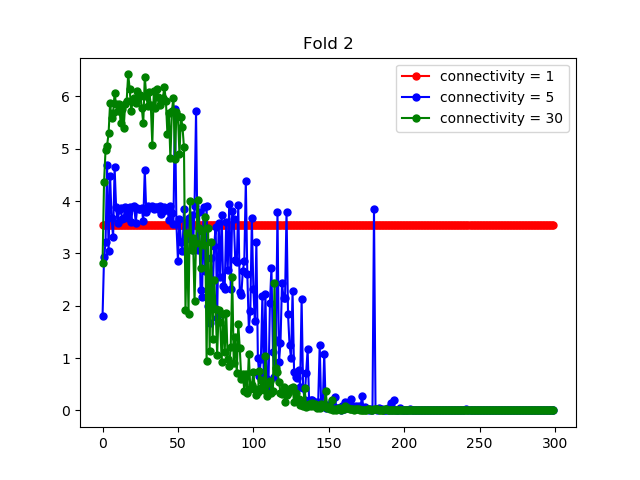
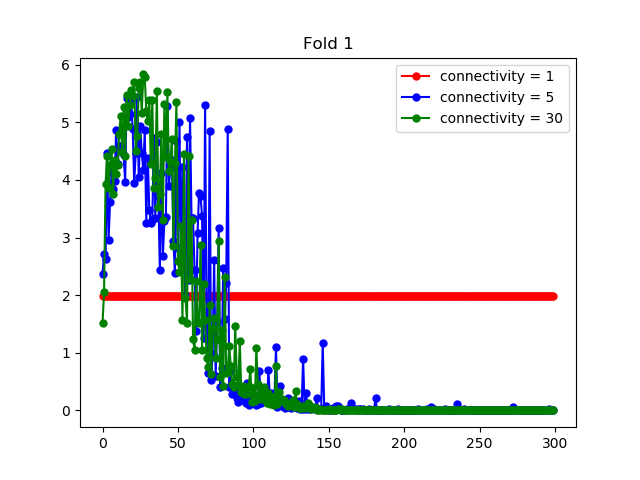
ใช้โครงสร้างของ MLP เป็น **8 – 3 – 2** โดยที่ชั้น Hidden layer ใช้ฟังชั่น **sigmoid** เป็น activation function และที่ที่ชั้น output ใช้ **Rectify linear** เป็น activation function ใช้จำนวนรอบ **300** รอบ วัดความคลาดเคลื่อนด้วย Mean Absolute Error (MAE) และใช้ **1 / MAE** เป็นค่า fitness ของตำแหน่ง(น้ำหนัก)นั้นๆ

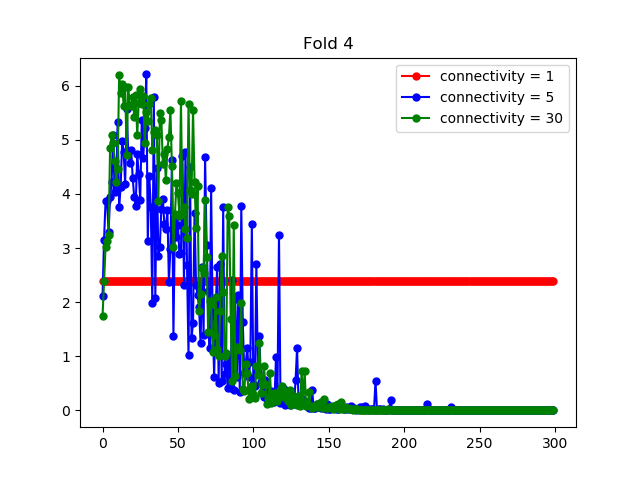
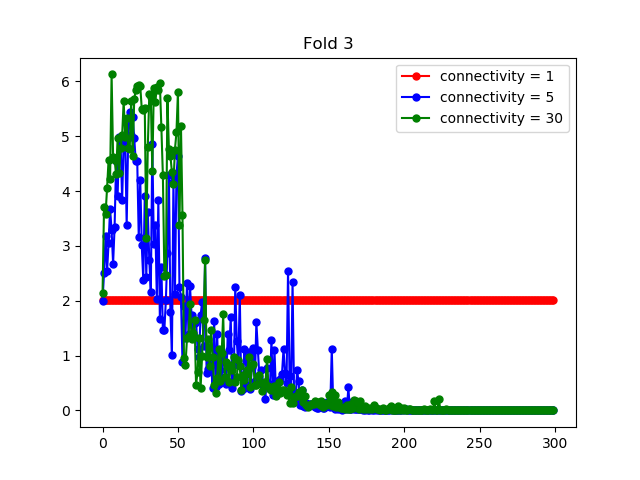
**รูปแบบของข้อมูล**

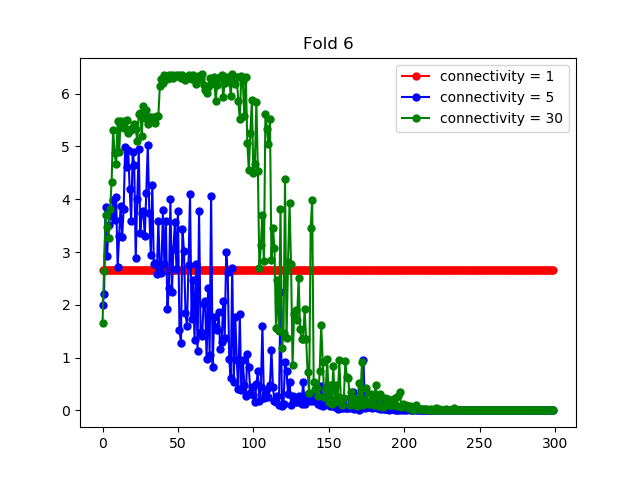
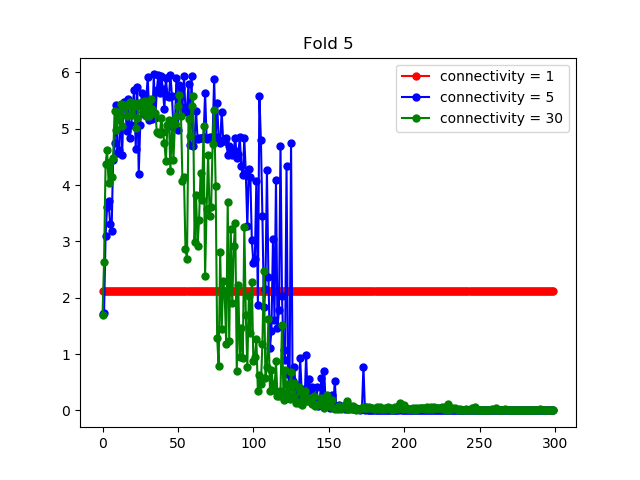
ข้อมูลที่ใช้ จะใช้ attribute ทั้งหมด 8 ตัวสำหรับการเป็น input ของระบบ โดย desired output นั้น จะใช้ค่า attribute ที่กำหนดไว้เป็น output ของข้อมูลที่มีระยะเวลาถัดไป 5 และ 10 วันมาเป็น desired output

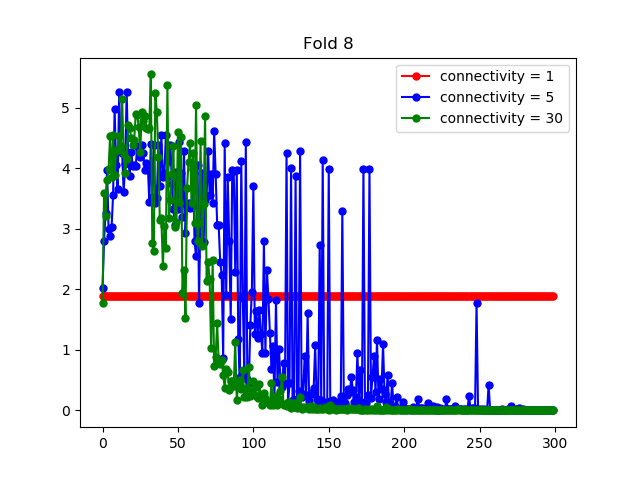
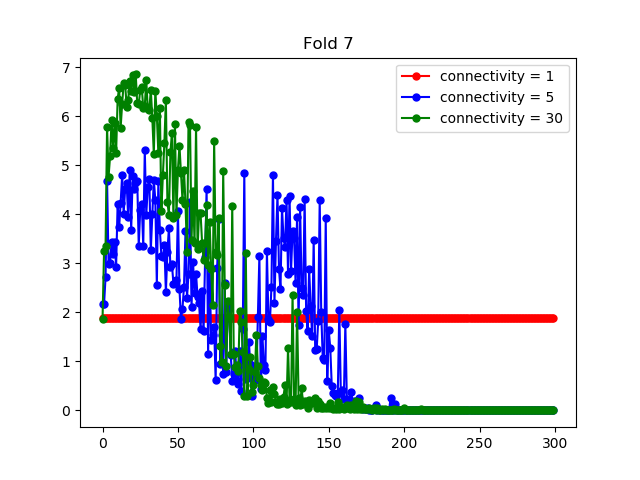
ลักษณะของข้อมูลที่ได้นั้น เป็นค่าจำนวนทศนิยม ที่มีช่วงค่าที่แตกต่างกันไป ดังนั้นจึงทำการ normalized ด้วย z-score และสำหรับoutput เลือกที่จะใช่ การ scale down แทนการใช้ z-score เนื่องจากมีความง่ายในการ denormalized เพื่อวัดผล อีกทั้งยังข้อมูลนี้มี missing value อยู่มาก จึงทำการจัดการโดยการตัดข้อมูลชุดที่มี missing value นั้นออกไป เพราะข้อมูลที่เหลืออยู่จัดได้ว่ามีปริมาณมากพอ

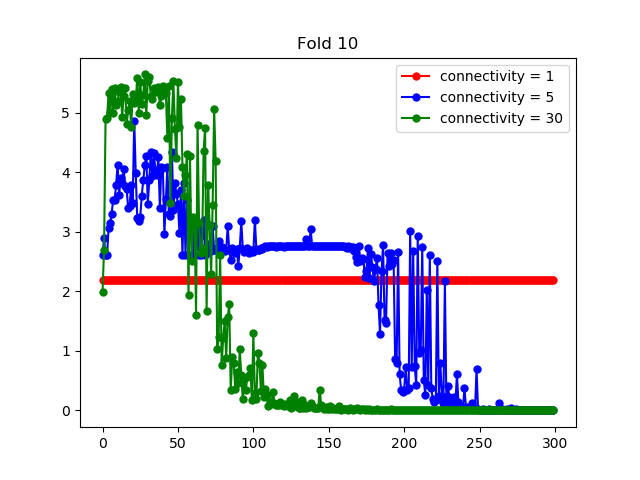
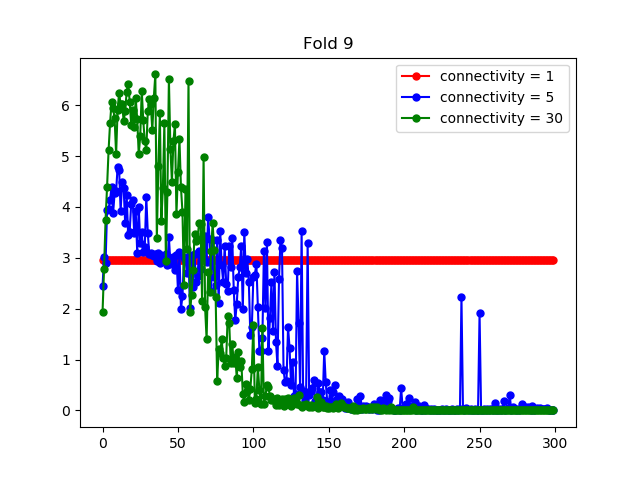
**ผลการทำงาน**

****

****

****

****

****

**กราฟแสดงค่า Fitness แบบ Global Best ของแต่ละ iteration**

**ตารางแสดงค่า MAE Global Best ของทุก Iteration**

|  |  |  |  |
| --- | --- | --- | --- |
| Fold | 1 | 5 | 30 |
| 1 | 50.97983173823 | 18.216095268897 | 16.762491802866 |
| 2 | 28.34579749397 | 17.59499306154785 | 15.388464800724 |
| 3 | 46.91021515221 | 18.611449038769 | 16.315556178037 |
| 4 | 42.47768350390 | 16.316473051974 | 16.236763933470 |
| 5 | 45.760330615478 | 17.174678536300 | 17.781366473709 |
| 6 | 36.828672056067 | 20.16649019147 | 15.967704918230 |
| 7 | 53.36607662555 | 18.506855308151 | 14.42124558580 |
| 8 | 55.21737596956 | 19.257325031302 | 18.161327734651 |
| 9 | 32.09304174758 | 21.045775732704 | 14.968416485332 |
| 10 | 44.6344172974 | 20.24452931350 | 17.463067293633 |
| **Average** | **43.66134422** | **18.71346645** | **16.34664052** |

**สรุปผลการทดลอง**

จากตารางผลการทดลองแสดงให้เห็นว่าที่ connectivity 30 ตัว ได้ผลค่าเฉลี่ยเออเร่อสัมบูรณ์น้อยที่สุด โดยโรงสร้างแบบนี้จะเป็นโครงสร้างที่ทุกตัวเชื่อมเข้าหากันทั้งหมด หรือแบบ Global best นั่นเอง และที่ค่าเออเร่อถัดมาจะเป็นแบบ 5 ตัวที่เป็นเพื่อบ้าน ซึงค่าเออเร่อร์ไม่นับว่าห่างจากกันมากนัก รูปแบบนี้เป็นรูปแบบที่เชื่อมกันบางส่วน จึงเป็นทำงานแบบ Local best และในรูปแบบที่ให้ผลลัพธ์แย่ที่สุดคือ 1 ตัวเดี่ยวๆ เป็นรูปแบบที่ไม่มีเพื่อนบ้าน จึงเป็นแบบ personal best

จากกราฟแสดงค่า fitness ที่ดีที่สุดในแต่ละ iteration จะเห็นรูปแบบหนึ่งที่มีความน่าสนใจ คือเส้นกราฟของ 1ตัว ค่าที่ได้จะเป็นค่าคงที่ในทุก iteration ผลแบบนี้คาดว่าเกิดจากการที่ ความเร็วเริ่มต้นของ node เป็น 0 และค่า fitness แรกเป็นค่าที่ดีที่สุด และเนื่องจากการมีเพียงเดียว จึงไม่มีผลจากส่วน Social component มากระตุ้นการเคลื่อนที่ ดังนั้นจึงควรปรับโค้ดให้มีการกระตุ้นในจังหวะแรกเล็กน้อยสำหรับกรณีนี้

**ภาคผนวค**

**Main.py**

if \_\_name\_\_ == '\_\_main\_\_':

model = '8x-3s-2l'

act = MLP.modelInit(model)[2]

k = 10

epoch = 300

populationSize = 30

data = pp.preprocess(pp.input('processedData.txt'))

trainSet, testSet = pp.kFolds(data, k)

for i in range(k):

if i > -1:

pso1 = PSO.ParticleSwarm((MLP.getError, MLP.feedForward), MLP.modelInit, (trainSet[i][:,:-2],trainSet[i][:,-2:]), model, populationSize)

pso2 = PSO.ParticleSwarm((MLP.getError, MLP.feedForward), MLP.modelInit, (trainSet[i][:,:-2],trainSet[i][:,-2:]), model, populationSize)

pso3 = PSO.ParticleSwarm((MLP.getError, MLP.feedForward), MLP.modelInit, (trainSet[i][:,:-2],trainSet[i][:,-2:]), model, populationSize)

pso1.defineTopology([i for i in range(0)])

pso2.defineTopology([i for i in range(2)])

pso3.defineTopology([i for i in range(15)])

agw1 = pso1.run(epoch)[0]

agw2 = pso2.run(epoch)[0]

agw3 = pso3.run(epoch)[0]

oa1 = MLP.feedForward(testSet[i][:,:-2], agw1, 0, act)

oa2 = MLP.feedForward(testSet[i][:,:-2], agw2, 0, act)

oa3 = MLP.feedForward(testSet[i][:,:-2], agw3, 0, act)

aac1 = MLP.getError(oa1[-1], testSet[i][:,-2:])

aac2 = MLP.getError(oa2[-1], testSet[i][:,-2:])

aac3 = MLP.getError(oa3[-1], testSet[i][:,-2:])

fig = plt.figure(i+1)

plt.title('Fold '+str(i+1))

plt.plot(np.arange(epoch), pso1.it, 'ro-', label='connectivity = 1', ms=5)

plt.plot(np.arange(epoch), pso2.it, 'bo-', label='connectivity = 5', ms=5)

plt.plot(np.arange(epoch), pso3.it, 'go-', label='connectivity = 30', ms=5)

plt.legend(loc='best')

fig.savefig('exp1,'+str((i+1))+'.png')

def preprocessData(file='AirQualityUCI.xlsx'):

data = pd.read\_excel(file).values

print(data.shape)

data = data[:,[3,6,8,10,11,12,13,14,5]]

print(data.shape)

res = []

for i in range(data.shape[0]):

if -200.0 in data[i]:

res.append(int(i))

data = np.delete(data, res, 0)

print(data.shape)

pData = np.asarray([np.concatenate([data[l][:-1],[data[l+120][-1]],[data[l+240][-1]]]) for l in range(len(data)-240)])

np.savetxt('processedData.txt', pData, fmt='%.13f')

def writeFile(fname, content):

f = open(fname, 'w')

f.write(content)

f.close()

**PSO.py**

class ParticleSwarm:

class Node:

def \_\_init\_\_(self, \_id, \_w, \_fitFunc, \_act):

self.neighbor = [self]

self.w = \_w

self.speed = 0

self.id = \_id

self.pBest = [0,0]

self.fitFunc = \_fitFunc

self.act = \_act

self.fitness = 0

self.timeStamp = -1

self.v = []

for i in range(len(\_w)):

self.v.append(np.zeros(\_w[i].shape))

def pairNeighbor(self, tNode):

if tNode not in self.neighbor and tNode is not self:

self.neighbor.append(tNode)

tNode.neighbor.append(self)

def updateFitness(self, data, \_timeStamp):

if self.timeStamp < \_timeStamp:

o = self.fitFunc[1](data[0],self.w,0,self.act)

fitness = 1.0 / self.fitFunc[0](o[-1],data[1])

if fitness > self.pBest[0]:

self.pBest = [fitness, self.w]

self.fitness = fitness

self.timeStamp = \_timeStamp

def getWDiff(self, \_w):

res = []

for i in range(len(self.w)):

tmp = \_w[i] - self.w[i]

res.append(tmp)

return res

def getLBest(self):

lbestf = 0

lbestn = []

for i in self.neighbor:

if i.fitness > lbestf:

lbestf = i.fitness

lbestn = i.w

return [lbestf, lbestn]

def updateVelocity(self, c1=2, c2=0.5):

r1 = np.random.rand()

r2 = np.random.rand()

cognitive = [(r1\*c1)\*l for l in self.getWDiff(self.pBest[1])]

social = [(r2\*c2)\*(l) for l in self.getWDiff(self.getLBest()[1])]

self.v = [self.v[i] + cognitive[i] + social[i] for i in range(len(self.v))]

def updatePosition(self):

newW = [self.w[i] + self.v[i] for i in range(len(self.w))]

self.w = newW

def \_\_init\_\_(self, fitnessFunc, initFunc, testData, seed, nPopulation):

self.population = []

self.fitnessFunc = fitnessFunc

self.initFunc = initFunc

self.testData = testData

self.model = seed

self.nPopulation = nPopulation

self.gBest = [0,0]

self.it = []

for i in range(nPopulation):

w,\_,act = MLP.modelInit(self.model)

n = self.Node(i,w,fitnessFunc,act)

self.population.append(n)

def defineTopology(self, connections):

for i in range(self.nPopulation):

for j in connections:

target = (i + j) % self.nPopulation

self.population[i].pairNeighbor(self.population[target])

def run(self, maxIt):

iGbest = [0,0]

for i in range(maxIt):

self.updateFitness(i)

iGbest = self.getGBest()

if self.gBest[0] < iGbest[0]:

self.gBest = iGbest

self.it.append(iGbest[0])

self.updateVelocity()

self.updatePosition()

print(i, self.gBest[0], iGbest[0])

return self.gBest[1], iGbest[1]

def updateFitness(self, \_it):

for n in self.population:

n.updateFitness(self.testData, \_it)

def updateVelocity(self):

for n in self.population:

n.updateVelocity()

def updatePosition(self):

for n in self.population:

n.updatePosition()

def getGBest(self, allTime=True):

gbestv = 0

gbestw = []

for n in self.population:

if n.fitness > gbestv:

gbestv = n.fitness

gbestw = n.w

return [gbestv, gbestw]

**MLP.py**

def modelInit(model):

layerSize = [int(n[:-1]) for n in model.split('-')]

activationLayer = [n[-1:] for n in model.split('-')]

nHidden = len(layerSize) - 1

weight = []

for i in range(nHidden):

weight.append(np.random.randn(layerSize[i+1], layerSize[i]))

bias = []

for i in range(nHidden):

bias.append(np.random.randn(layerSize[i+1]))

return weight, bias, activationLayer

def feedForward(input, weigth, bias, activation):

res = []

tmp = input

for i in range(len(activation)-1):

tmp = np.dot(tmp, np.transpose(weigth[i]))

tmp = act.activate(np.copy(tmp), activation[i+1])

res.append(np.asarray(tmp))

return res

def classInterprete(y):

return (y[:,0] <= y[:,1]).astype(int)

def getError(y, d):

diff = abs(d - y)

return np.average(np.sum(diff,axis=1))

**Preprocess.py**

def input(input):

res = []

with open(input, 'r') as inputFile:

res = np.asarray([list(map(float, line.rstrip().split(' '))) for line in inputFile])

return res

def preprocess(data):

feature = stats.zscore(data[:,:-2],axis=0)

output = data[:,-2:] / 100.0

return np.concatenate((feature,output), axis=1)

def kFolds(data, k=1):

trainSet = [[] for i in range(k)]

testSet = [[] for i in range(k)]

dataSize = len(data)

binSize = int(dataSize / k)

remainSize = dataSize % k

np.random.shuffle(data)

for i in range(k):

trainSet[i].extend(data[0:(i)\*binSize])

trainSet[i].extend(data[(i+1)\*binSize:dataSize-remainSize])

testSet[i].extend(data[i\*binSize:(i+1)\*binSize])

if remainSize != 0:

trainSet[i].extend(data[-(dataSize % k):])

return np.asarray(trainSet), np.asarray(testSet)

activatation.py

def sigmoidO(x):

return 1.0 / (1 + np.exp(0.0-x))

def activate(x, func, div=False):

res = x

if func == 's':

if div:

res = inverseSigmoid(x)

else:

res = sigmoid(x)

if func == 't':

if div:

res = 1.0 - np.tanh(x)\*\*2

else:

res = np.tanh(x)

if func == 'l':

if div:

res = 1. \* (x > 0)

else:

res = np.maximum(x, 0)

return res