261456 Introduction to Computer Intelligence

**Computer Assignment 1**

นาย ดรันภพ เป็งคำตา 580610642

รายงานนี้เป็นรายงานเกี่ยวกับการทดลองเรื่อง MLP (Multi layers perceptron) ด้วยชุดคำสั่งที่ implement ขึ้นมาตามแนวคิดของ MLP โดยในการทดสอบการทำงานและประสิทธิภาพจะใช้การทดลองย่อย 2 แบบเพื่อเป็นการทดสอบ

**การทดลองย่อยที่ 1**

เป็นการทดลองในการทำนายระดับน้ำด้วย MLP โดยข้อมูลที่ใส่เข้าไปในระบบ เป็นระดับน้ำในช่วงเวลาก่อนหน้า และผลลัพธ์ที่ได้เป็นระดับน้ำที่ทำนาย

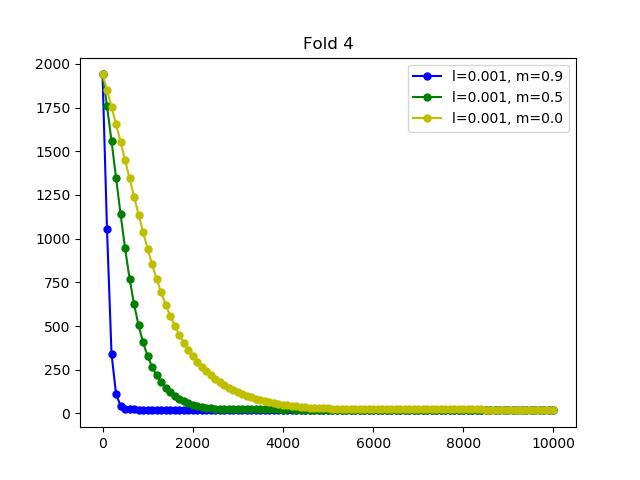
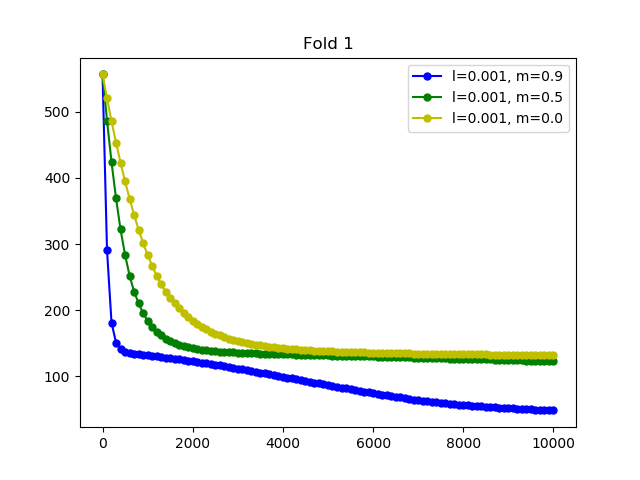
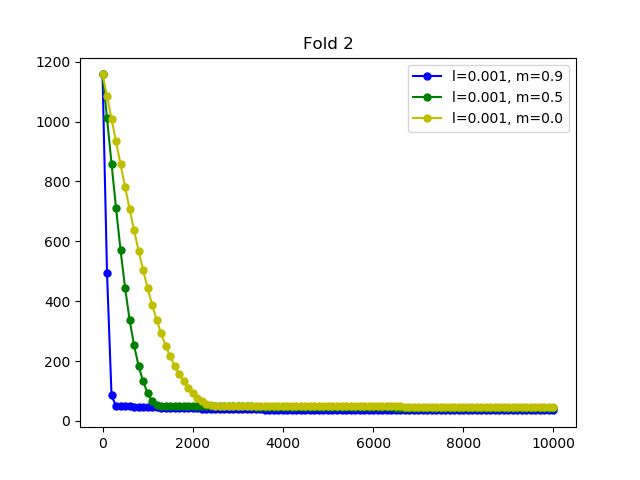
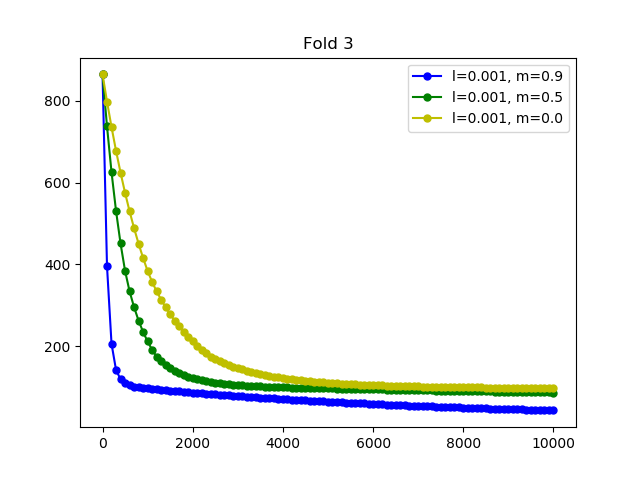
**รูปแบบการทดลอง**

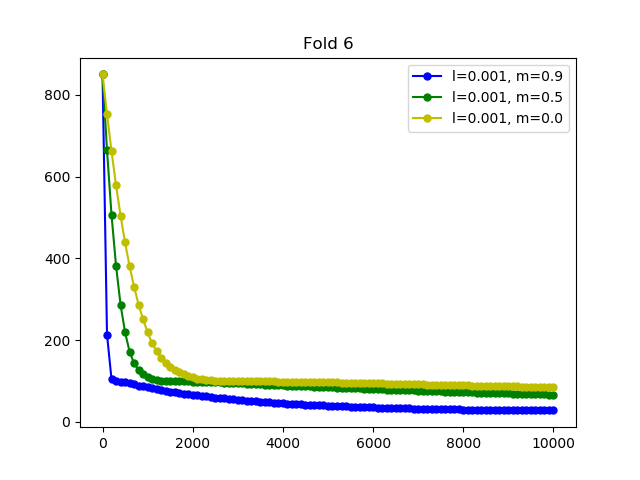
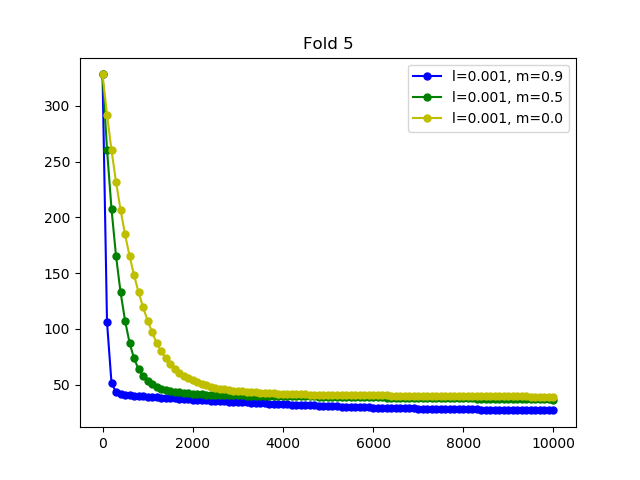
ทำการทดลองด้วยโครงสร้าง 8 – 5 – 1 จำนวน 10000 รอบ(epoch) โดยใช้ค่า learning rate 0.001 แต่จะใช่ต่า momentum rate ที่แตกต่างกัน 3 ค่าคือ 0.9, 0.5, 0.0 เพื่อทดสอบว่าผลลัพธ์ภายใต้ โครงสร้างเดียวกัน และ learning rate เดียวกันแต่ค่า momentum rate ที่แตกต่างกันจะให้ผลลัพธ์อย่างไร และวัดผลด้วยค่า MSE (Mean Square Error) และ k-folds cross validation ด้วยค่า k = 10

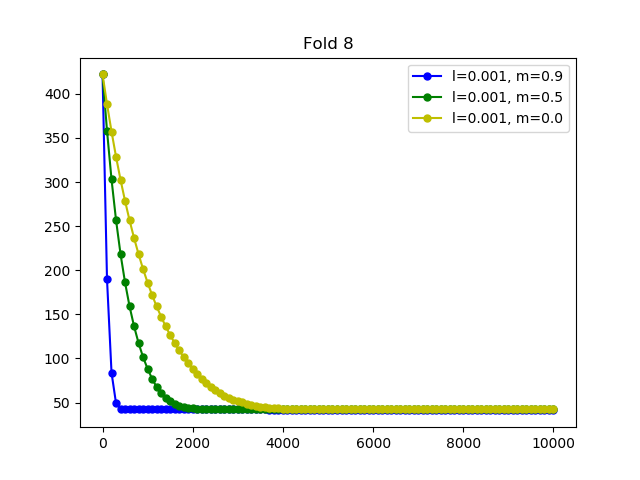
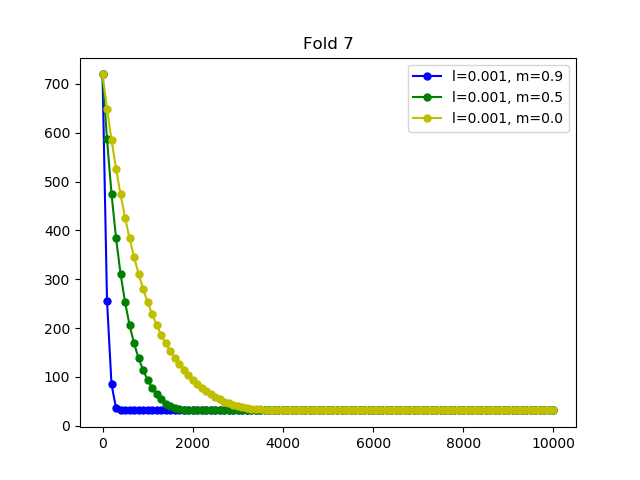
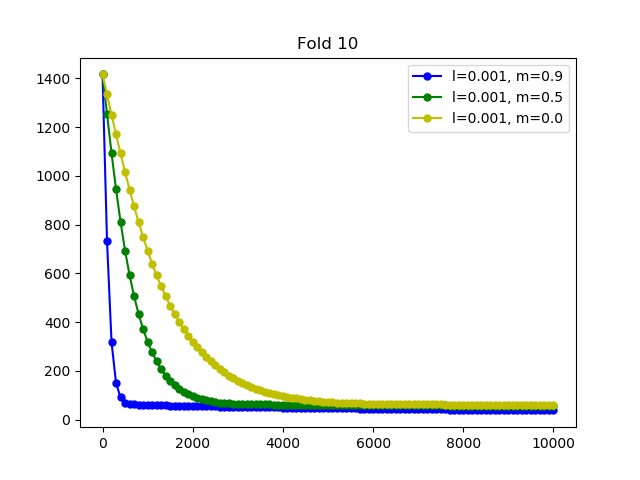
**รูปแบบของข้อมูล**

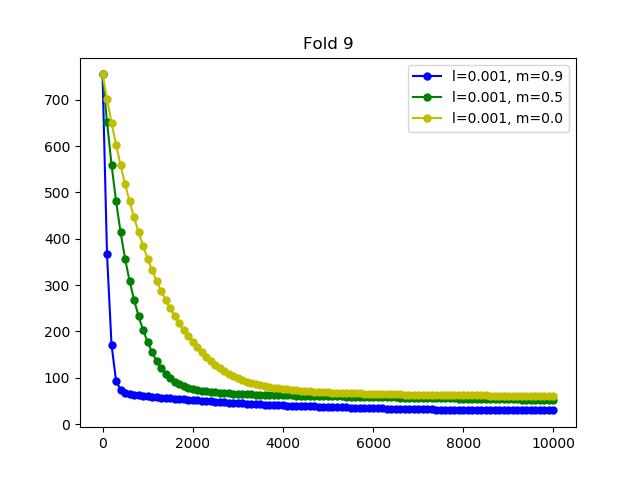
ข้อมูลเป็นจำนวนเต็มบวก โดยมีค่าต่ำสุดเป็น 95 และมากที่สุดเป็น 628 ดังนั้นในการเตรียมข้อมูล ข้อมูลเหล่านี้จะถูก normalize ให้อยู่ในช่วง 0 – 1 จากช่วง 0 – 700 โดยได้เผื่อค่าจากสูงสุดต่ำสุดของจริง เนื่องจากหากใช้ activation function ด้วย Sigmoid การลู่เข้า 1 หรือ 0 นั้นจะต้องเป็นค่าที่สุดขีดอย่างมาก ดังนั้นผลลัพธ์ที่ได้จึงจะอยู่ในช่วง 0 – 1 ด้วยเช่นกัน ผลลัพธ์ที่แท้จริงจึงต้องผ่านการ de-normalize กลับไปเป็นค่าในช่วงเดิม

**ผลการทดลอง**

****

****

****



**ตารางแสดงผลการทดลอง**

|  |  |  |  |
| --- | --- | --- | --- |
| Fold | m = 0.9 | m = 0.5 | m = 0.0 |
| 1 | 48.38709677 | 122.58064516 | 131.64516129 |
| 2 | 35.32258065 | 42.38709677 | 46.41935484 |
| 3 | 42.93548387 | 86.61290323 | 97.00000000 |
| 4 | 17.58064516 | 20.83870968 | 22.5483871 |
| 5 | 26.93548387 | 36.61290323 | 39.25806452 |
| 6 | 28.09677419 | 66.41935484 | 84.77419355 |
| 7 | 31.67741935 | 32.61290323 | 32.70967742 |
| 8 | 41.09677419 | 42.58064516 | 42.87096774 |
| 9 | 29.51612903 | 51.83870968 | 60.16129032 |
| 10 | 38.35483871 | 56.67741935 | 60.83870968 |
| **Average** | **33.99032258** | **55.91612903** | **61.82258065** |

**สรุปผลการทดลอง**

จากตารางผลการทดลอง จะเป็นได้ว่า ที่ค่า momentun rate 0.9 ให้ค่าเฉลี่ยความผิดพลาดน้อยที่สุด รองลงมาเป็น ที 0.5 และ 0.0 ตามลำดับ จากการทดลองจะเห็นได้ว่า ความเฉลี่ยความผิดพลาด แปลผกผันกับ ค่าmomentum rate ภายให้การเทรนที่ 10000 รอบ และจากกราฟแสดงผลการทดลองทั้ง 10 fold ทำให้มราบว่า ที่ค่า momentum สูงขึ้น ทำให้ผลลัพ converge ได้ไวขึ้น แต่ถึงกระนั้น เมื่อถึงจุดๆหนึ่ง ค่าจะไม่ได้converge ไว เหมือนขณะเริ่มต้น และยังค่อยๆ ลดลงไปอย่างช้า ซึ่งทำให้ยังไม่สามารถสรุปได้อย่างแน่นอนว่า ที่momentum rate มากๆ จะให้ผลลัพธ์ที่แม่นยำกว่าสำหรับ จำนวน epoch ใดๆ หรือ โครงสร้างใดๆ

**การทดลองย่อยที่ 2**

เป็นการทดลอง classify ของข้อมูลที่มี 2 class (0, 1) และ (1, 0) โดยมี 2 input และ 2 output

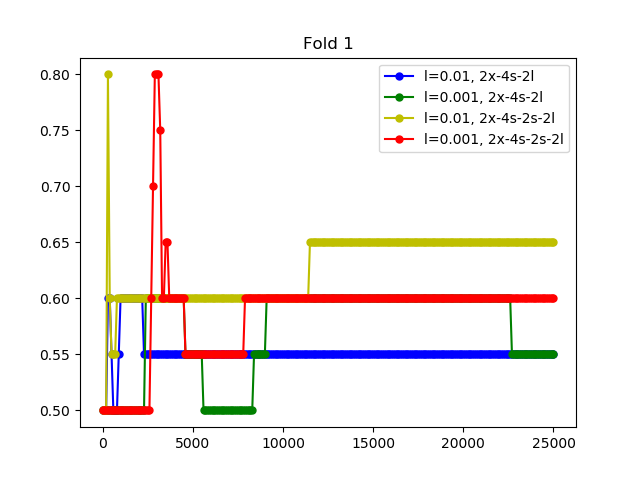
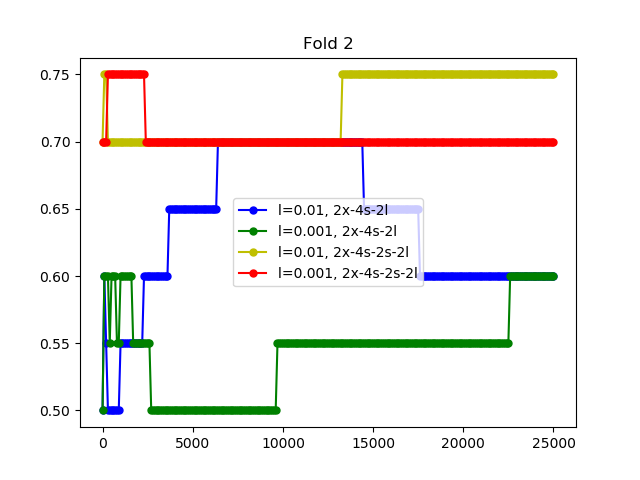
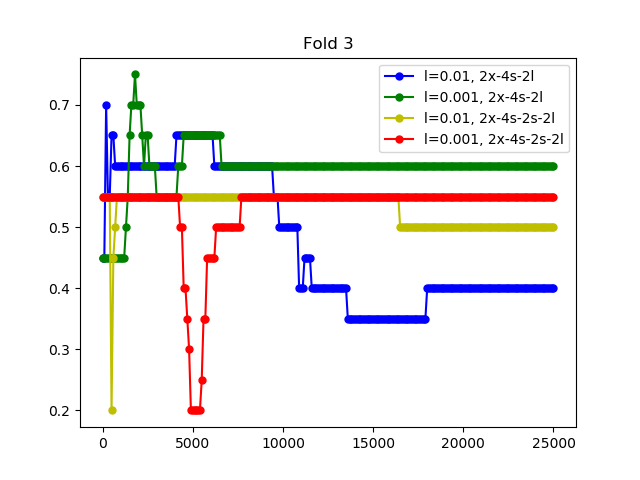
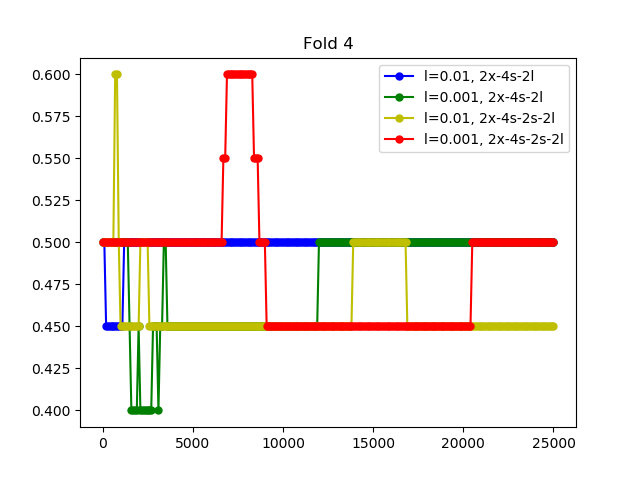
**รูปแบบการทดลอง**

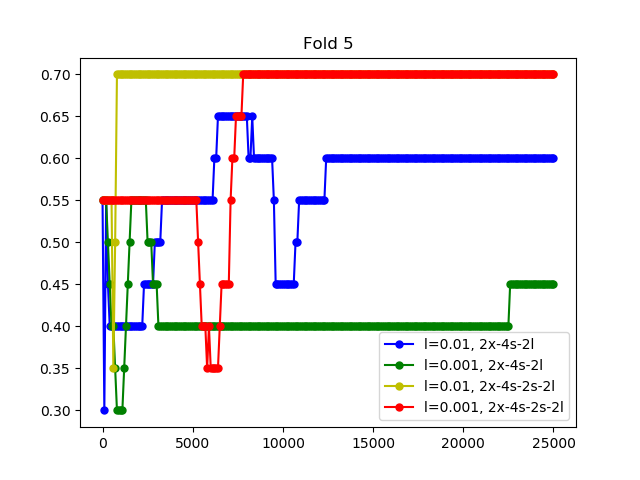
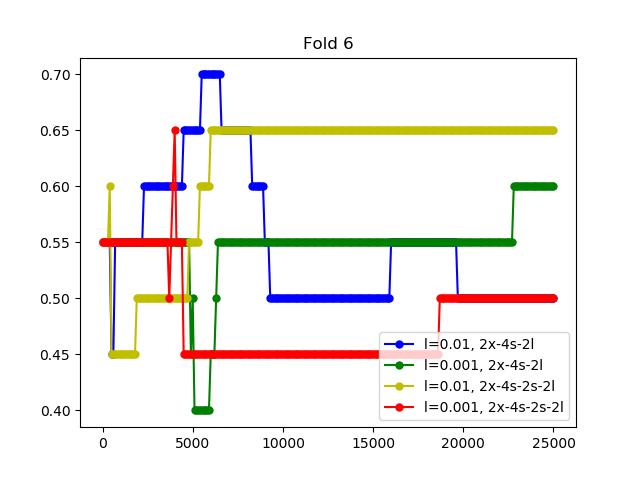
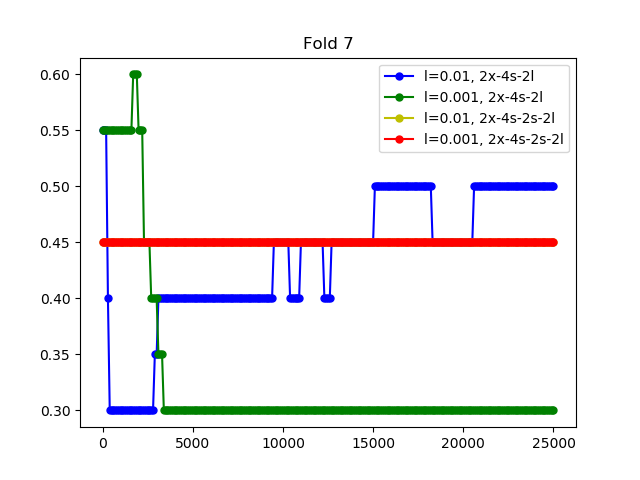
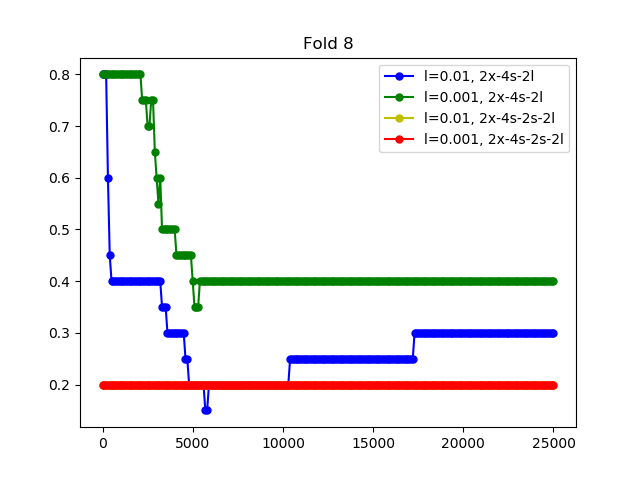
ในการทดลองนี้จะทำการทดลองด้วยโครงสร้างที่แตกต่างกับ 2 แบบ คือ 2 – 4 – 2 และ 2 – 4 – 2 – 2 ในแต่ละโครงสร้างจะทดลองด้วย learning rate ที่แตกต่างกัน 2 ค่า คือ 0.01 และ 0.001 จำนวน 25000 รอบ และวัดผลด้วย อัตราส่วน True positive + True negative ต่อจำนวนผลลัพธ์ทั้งหมด กอปรกับการใช้ k-fold cross validation ด้วยค่า k = 10

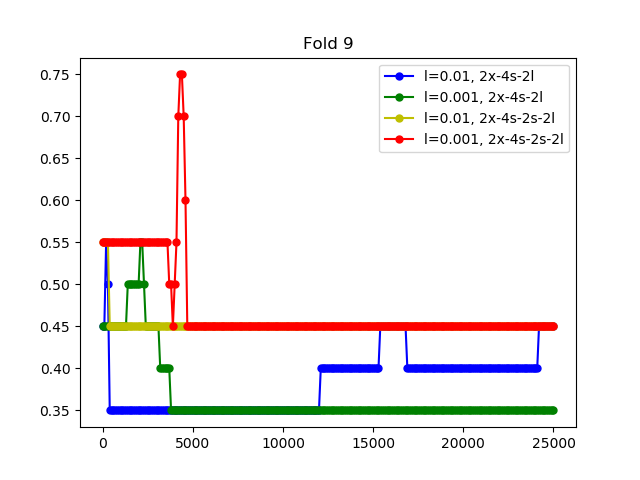
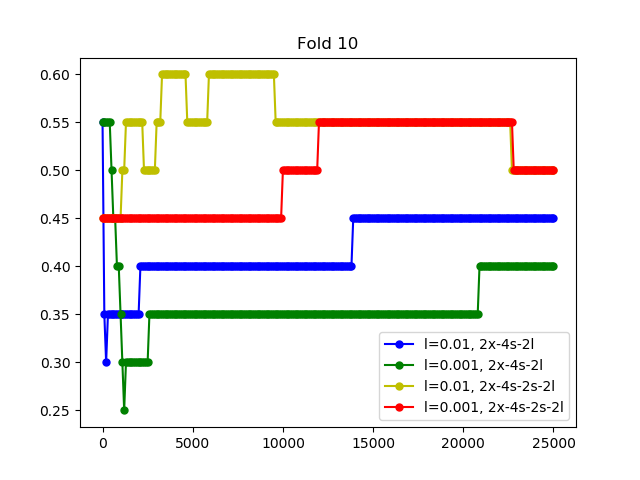
**รูปแบบของข้อมูล**

ในส่วนอินพุตมีทั้งหมด 2 ค่า ต่อ 1 ชุดข้อมูล เป็นทศนิยมโดยมีค่าอยู่ระหว่าง 0 – 1 ในส่วนของผลลัพธ์ มี 2 ค่าเท่ากับ อินพุต แต่เป็นค่าจำนวนเต็ม 2 ค่าคือ 0 และ 1 และรูปแบบของผลลัพธ์จากข้อมูลทั้งหมดเป็นได้ได้ 2 รูปแบบคือ (0, 1) และ (1, 0) ดังนั้นในการทดลองนี้จึงไม่จำเป็นต้องมีการ normalize เหมือนกับการทดลองย่อยที่1

**ผลการทดลอง**

**กราฟแสดงผลการทดลองในแต่ละ fold**

****

****

**ตารางแสดงผลการทดลอง**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| fold | | l = 0.01  2 – 4 - 2 | | | l = 0.001  2 – 4 - 2 | | | l = 0.01  2 – 4 – 2 - 2 | | | l = 0.001  2 – 4 – 2 – 2 | | |
| **1** | **0** | **%** | **1** | **0** | **%** | **1** | **0** | **%** | **1** | **0** | **%** |
| 1 | **1** | 9 | 8 | 55.0 | 9 | 8 | 55.0 | 8 | 5 | 65.0 | 7 | 5 | 60.0 |
| **0** | 1 | 2 | 1 | 2 | 2 | 5 | 3 | 5 |
| 2 | **1** | 7 | 7 | 60.0 | 5 | 5 | 60.0 | 8 | 5 | 75.0 | 7 | 5 | 70.0 |
| **0** | 1 | 5 | 3 | 7 | 0 | 7 | 1 | 7 |
| 3 | **1** | 5 | 8 | 40.0 | 4 | 3 | 60.0 | 7 | 8 | 50.0 | 8 | 8 | 55.0 |
| **0** | 4 | 3 | 5 | 8 | 2 | 3 | 1 | 3 |
| 4 | **1** | 5 | 5 | 50.0 | 5 | 5 | 50.0 | 4 | 5 | 45.0 | 5 | 5 | 50.0 |
| **0** | 5 | 5 | 5 | 5 | 6 | 5 | 5 | 5 |
| 5 | **1** | 9 | 8 | 60.0 | 4 | 6 | 45.0 | 9 | 6 | 65.0 | 9 | 6 | 65.0 |
| **0** | 0 | 3 | 5 | 5 | 0 | 5 | 0 | 5 |
| 6 | **1** | 7 | 8 | 50.0 | 8 | 7 | 60.0 | 9 | 7 | 65.0 | 9 | 10 | 50.0 |
| **0** | 2 | 3 | 1 | 4 | 0 | 4 | 0 | 1 |
| 7 | **1** | 6 | 5 | 50.0 | 4 | 7 | 30.0 | 0 | 0 | 45.0 | 0 | 0 | 45.0 |
| **0** | 5 | 4 | 7 | 2 | 11 | 9 | 11 | 9 |
| 8 | **1** | 2 | 0 | 30.0 | 5 | 1 | 40.0 | 0 | 0 | 20.0 | 0 | 0 | 20.0 |
| **0** | 14 | 4 | 11 | 3 | 16 | 4 | 16 | 4 |
| 9 | **1** | 5 | 7 | 45.0 | 3 | 7 | 35.0 | 9 | 11 | 45.0 | 9 | 11 | 45.0 |
| **0** | 4 | 4 | 6 | 4 | 0 | 0 | 0 | 0 |
| 10 | **1** | 4 | 6 | 45.5 | 3 | 6 | 40.0 | 5 | 6 | 50.0 | 8 | 9 | 50.0 |
| **0** | 5 | 5 | 6 | 5 | 4 | 5 | 1 | 2 |
| average | | **48.5** | | | **47.5** | | | **53.0** | | | **51.5** | | |

**สรุปผลการทดลอง**

จากตารางผลการทดลอง จะเห็นได้ว่า ที่ l = 0.01 โครงสร้าง 2 – 4 – 2 – 2 มีความแม่นยำมากที่สุดที่ 25000 รอบ และรองลงมาเป็น ที่โครงสร้างเดียวกัน แต่มี learning rate น้อยกว่า ในทำนองเดียวกับโครงสร้าง 2 – 4 – 2 ที่จำนวน learning rate มากกว่า มีความถูกต้องมากกว่า แต่จากกราฟ จะเห็นว่าค่านั้นยังแกว่งอยู่มาก จึงคิดว่าที่ 25000 ยังไม่ใช่จำนวนรอบที่model จะ converge ถึงที่สุด

**ภาคผนวค**

**assignment.py**

**โค้ดสำหรับการรันทั่วไป**

data = sys.argv[1]

trainSet, testSet = pp.kFolds(data,10)

w,b,a = mlp.modelInit(sys.argv[2])

chunk = trainSet[0]

inpu = np.delete(chunk, -1, 1)

inpu = pp.normalRange(inpu).tolist()

d = np.delete(chunk, range(len(chunk[0])-1), 1)

d = pp.normalRange(d)

for i in range(sys.argv[3]):

if i == 0:

dW = []

dB = []

o = mlp.feedForward(inpu, w, b, a)

if (i+1) % 100 == 0:

print(i+1, testing(w,b,a,testSet[0]))

w,b,dW,dB = mlp.backpropagate(inpu,o,w,dW,dB,b,a,d,0.001,0.5)

สามารถสั่งทำงานได้ด้วยคำสั่ง

python assignment.py ‘dataset\_file’ ‘model’ ‘epoch’

**dataset\_file** ชื่อไฟล์ของ dataset

**mode**l โครงสร้างที่จะใช้ มีลักษณะการใช้งานดังนี้คือ

**no.input – no.hidden1 activation\_type – no.hidden2 activation\_type - …. – no.output**

activation\_type มี ดังนี้ x = ไม่ทำอะไร , s = sigmoid

,t = hyperbolic tangent , l = relu

ยกตัวอย่างเช่น ‘3x-8s-4t-1l’

**epoch** จำนวนรอบในการเทรน

**preset ของการทดลองย่อยที่ 1**

inputFile = 'Flood\_dataset.txt'

model = '8x-5s-1x'

epoch = 10000

k = 10

data = pp.input(inputFile)

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

res = []

for i in range(k):

w,b,a = mlp.modelInit(model)

d1=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w),copy.deepcopy(b),a,0.001,0.9)

d2=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w),copy.deepcopy(b),a,0.001,0.5)

d3=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w),copy.deepcopy(b),a,0.001,0.0)

fig = plt.figure(i+1)

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

plt.plot(d1[:,0], d1[:,1], 'bo-', label='l=0.001, m=0.9', ms=5)

plt.plot(d2[:,0], d2[:,1], 'go-', label='l=0.001, m=0.5', ms=5)

plt.plot(d3[:,0], d3[:,1], 'yo-', label='l=0.001, m=0.0', ms=5)

plt.legend(loc='best')

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

res.append([d1[:,1][-1],d2[:,1][-1],d3[:,1][-1]])

with open("exp1\_result.txt", "w") as outFile:

outFile.write(str(np.asarray(res)))

with open("exp1\_result.txt", "a") as outFile:

outFile.write('\n' + str(np.average(np.asarray(res), axis=0)))

**preset ของการทดลองย่อยที่ 2**

inputFile = 'cross.pat'

model1 = '2x-4s-2x'

model2 = '2x-4s-2s-2x'

epoch = 25000

k = 10

data = pp.input(inputFile,clean=True)

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

res = []

for i in range(k):

w1,b1,a1 = mlp.modelInit(model1)

w2,b2,a2 = mlp.modelInit(model2)

d1=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w1),copy.deepcopy(b1),a1,0.01,0.5,resAttr=2,confusion=True)

d2=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w1),copy.deepcopy(b1),a1,0.001,0.5,resAttr=2,confusion=True)

d3=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w2),copy.deepcopy(b2),a2,0.01,0.5,resAttr=2,confusion=True)

d4=training(trainSet[i],testSet[i],epoch,copy.deepcopy(w2),copy.deepcopy(b2),a2,0.001,0.5,resAttr=2,confusion=True)

y1 = (d1[:,0,0] + d1[:,1,1])/20.0

y2 = (d2[:,0,0] + d2[:,1,1])/20.0

y3 = (d3[:,0,0] + d3[:,1,1])/20.0

y4 = (d4[:,0,0] + d4[:,1,1])/20.0

fig = plt.figure(i+1)

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

xaxis = np.concatenate( (np.arange(1,2),np.arange(100,epoch+1,100)) )

print(xaxis)

plt.plot(xaxis, y1, 'bo-', label='l=0.01, 2x-4s-2l', ms=5)

plt.plot(xaxis, y2, 'go-', label='l=0.001, 2x-4s-2l', ms=5)

plt.plot(xaxis, y3, 'yo-', label='l=0.01, 2x-4s-2s-2l', ms=5)

plt.plot(xaxis, y4, 'ro-', label='l=0.001, 2x-4s-2s-2l', ms=5)

plt.legend(loc='best')

fig.savefig('exp2,'+str((i+1))+'.png')

res.append([d1[-1],d2[-1],d3[-1],d4[-1]])

res = np.asarray(res)

**function training**

def training(trainingSet, testSet, epoch, w, b, a, learnRate, momentum, resAttr=1, confusion=False):

res = []

inpu = np.delete(trainingSet, range(len(trainingSet[0]))[-resAttr:], 1)

d = np.delete(trainingSet, range(len(trainingSet[0])-resAttr), 1)

if confusion:

inpu = pp.normalRange(inpu).tolist()

d = pp.normalRange(d)

for i in range(epoch):

if i == 0:

dW = []

dB = []

o = mlp.feedForward(inpu, w, b, a)

if (i+1) % 100 == 0 or i == 0:

print(i+1,'\n', testing(w,b,a,testSet,resAttr,confusion))

if confusion:

res.append(testing(w,b,a,testSet,resAttr,confusion))

else:

res.append((i+1, testing(w,b,a,testSet,resAttr,confusion)))

w,b,dW,dB = mlp.backpropagate(inpu,o,w,dW,dB,b,a,d,learnRate,momentum)

try:

return np.asarray(res)

except:

return res

**function testing**

def testing(w, b, a, data, resAttr=1, confusion=False):

inpu = np.delete(data, range(len(data[0]))[-resAttr:], 1)

d = np.delete(data, range(len(data[0])-resAttr), 1)

o = mlp.feedForward(confusion and pp.normalRange(inpu).tolist() or inpu.tolist(), w, b, a)[-1]

if confusion:

return mlp.confusion(o,d)

else:

o = np.round(pp.normalize(o, denorm=True))

return mlp.mse(o,d)

**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.rand(layerSize[i+1]))

return weight, bias, activationLayer

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

res = []

tmp = []

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

tmp = np.dot(len(tmp)==0 and input or tmp, np.transpose(weigth[i]))

for j in range(tmp.shape[0]):

for k in range(tmp.shape[1]):

tmp[j][k] += bias[i][k]

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

res.append(np.asarray(tmp))

return res

def mse(y, d):

res = 0.0

dCp = np.asarray(d)

yCp = y

return np.average(np.abs(dCp - yCp))

def backpropagate(input,y, weight, dWo, dBo, bias, activation, d, learnRate, momentum):

nB = bias.copy()

if len(dWo) == 0:

dWo = [np.zeros(weight[i].shape) for i in range(len(weight))]

if len(dBo) == 0:

dBo = [np.zeros(bias[i].shape) for i in range(len(bias))]

dW = [np.zeros(weight[i].shape) for i in range(len(weight))]

dB = [np.zeros(bias[i].shape) for i in range(len(bias))]

localGradient = [[] for i in range(len(y))]

for i in reversed(range(len(y))):

if i+1 == len(y):

error = d - y[i]

localGradient[i] = (act.activate(y[i], activation[i], True) \* error).T

else:

localGradient[i] = (act.activate(y[i], activation[-1], True) \* (np.dot(localGradient[i+1].T, weight[i+1]))).T

dW[i] = (momentum\*dWo[i]) + (learnRate \* np.dot(localGradient[i], i==0 and input or y[i-1]) / len(input))

dB[i] = (momentum\*dBo[i]) + np.average(learnRate \* localGradient[i], axis=1)

nW[i] += dW[i]

nB[i] += dB[i]

return nW, nB, dW, dB

def confusion(y, d):

res = np.zeros((2,2),dtype=int)

for i in range(len(d)):

a = y[i][0] > y[i][1] and 1 or 0

b = d[i][0] > d[i][1] and 1 or 0

res[a][b] += 1

return res

**preprocess.py**

def input(input, clean=False):

res = []

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

if clean:

lines = inputFile.readlines()

for i,\_ in enumerate(lines):

if (i+1) % 3 == 0:

res.append(list(map(float, str(lines[i-1].rstrip() +'\t'+ lines[i].rstrip()).split())))

else:

res = [list(map(int, line.rstrip().split('\t'))) for line in inputFile]

return res

**สำหรับจัดการการรับข้อมูล**

def normalize(x, denorm=False):

MAX = 700.0

MIN = 0.0

if denorm:

return (x \* (MAX - MIN) + MIN)

return (x - MIN) / (MAX - MIN)

สำหรับการ normalize ข้อมูล

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 trainSet, testSet

สำหรับการแบ่ง fold ของ cross validation

**activation.py**

def sigmoidO(x):

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

def inverseSigmoid(x):

return sigmoidO(x) \* (1.0 - sigmoidO(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