Computer Assignment 1

CPE 261456 (Introduction to Computational Intelligence)

โดย

นายพีรณัฐ ธารทะเลทอง

รหัสนักศึกษา 550610530

เสนอ

ผศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่

**วิธีการทำงานของโปรแกรม**

โปรแกรมเขียนด้วยภาษา python โดยรับ parameter ผ่าน command line รายละเอียดดังนี้

Parameter -N arg : arg คือ ชื่อโครงสร้าง Neural network เช่น “2-4-1”

parameter -n arg : arg คือ learning rate

parameter -m arg : arg คือ momentum

parameter -e arg : arg คือ จำนวน epoch

parameter -c : ทำ 10% cross validation

parameter -t arg : arg คือ training set file

ตัวอย่างการเรียกใช้โปรแกรม

python ComputerAssignment1.py -N 4-4-1 -n 0.2 -m 0.3 -e 100 -t iris.pat

เริ่มต้นการทำงาน โปรแกรมจะอ่าน input และ desire-output จากไฟล์มาเก็บใน array จากนั้นนำมา shuffle ก่อนนำเข้า neural network เมื่อเริ่มเข้า neural network จะทำการ random init weight ซึ่งค่าอยู่ระหว่าง และ จากนั้นเข้าสู่กระบวนการ train โดยเริ่มจากการทำ feedforward networks โดย output แต่ละ layer ได้จากการ dot product ระหว่าง matrix output ของ layer ก่อนหน้า กับ matrix ของ weight ทุก weight ที่เข้า layer นั้น

จากนั้น ทำ back propagation โดยเริ่มจาก หา error จากสูตร

เพื่อนำค่า error ไปหาค่า gradients ที่ output layer จากสูตร

จากนั้น หา gradients ใน hidden layer จากสูตร

เพื่อนำ gradients ที่ได้ของทุก node นำไปปรับ weight จากสูตร

จากนั้นกลับไปเริ่มต้นใหม่ให้ครบรอบจำนวน epoch ที่รับเข้ามา เพื่อปรับ weight ให้ดีขึ้นเรื่อยๆ

**ทดลองกับปัญหา XOR เพื่อทดสอบความถูกต้องของโปรแกรม**

ผลการทดลองครั้งที่ 1 (50 epochs)

C:\Users>python ComputerAssignment1.py -N 2-4-1 -n 0.2 -m 0.3 -e 50

------------Variable------------

Neural name 2-4-1

Activation func tanh

Learning rate 0.2

Momentum 0.3

Epoch 50

TrainingFile -

------------Training------------

==================================100%

------------Testing-------------

Features Output Desired class

0 0 [ 0.2285700218] 0

0 1 [ 0.6795426040] 1

1 0 [ 0.5946130258] 1

1 1 [ 0.3343745376] 0

Error AV 0.0538852660

Accuracy 100.0000%

ผลการทดลองครั้งที่ 2 (200 epochs)

C:\Users>python ComputerAssignment1.py -N 2-4-1 -n 0.2 -m 0.3 -e 200

------------Variable------------

Neural name 2-4-1

Activation func tanh

Learning rate 0.2

Momentum 0.3

Epoch 200

TrainingFile -

------------Training------------

============================100%

------------Testing-------------

Features Output Desired class

0 0 [ 0.0065803017] 0

0 1 [ 0.9336247061] 1

1 0 [ 0.9295828521] 1

1 1 [-0.0011971559] 0

Error AV 0.0011761235

Accuracy 100.0000%

ผลการทดลองครั้งที่ 3 (400 epochs)

C:\Users>python ComputerAssignment1.py -N 2-4-1 -n 0.2 -m 0.3 -e 400

------------Variable------------

Neural name 2-4-1

Activation func tanh

Learning rate 0.2

Momentum 0.3

Epoch 400

TrainingFile -

------------Training------------

============================100%

------------Testing-------------

Features Output Desired class

0 0 [ 0.0063966836] 0

0 1 [ 0.9681230407] 1

1 0 [ 0.9625096198] 1

1 1 [ 0.0105083843] 0

Error AV 0.0003216266

Accuracy 100.0000%

ผลการทดลองครั้งที่ 4 (1000 epochs)

C:\Users>python ComputerAssignment1.py -N 2-4-1 -n 0.2 -m 0.3 -e 1000

------------Variable------------

Neural name 2-4-1

Activation func tanh

Learning rate 0.2

Momentum 0.3

Epoch 1000

TrainingFile -

------------Training------------

============================100%

------------Testing-------------

Features Output Desired class

0 0 [ 0.0005324577] 0

0 1 [ 0.9834712507] 1

1 0 [ 0.9806382148] 1

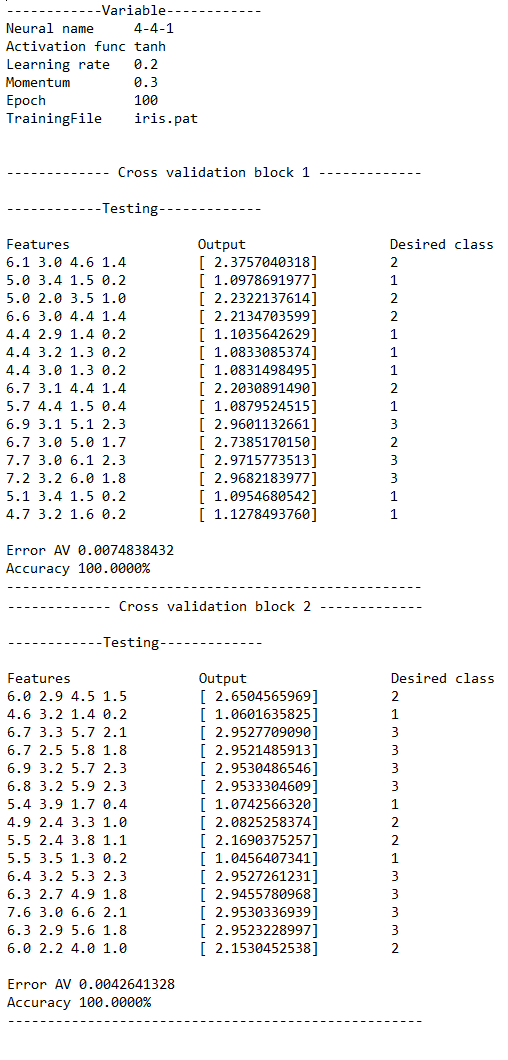
1 1 [-0.0105726275] 0

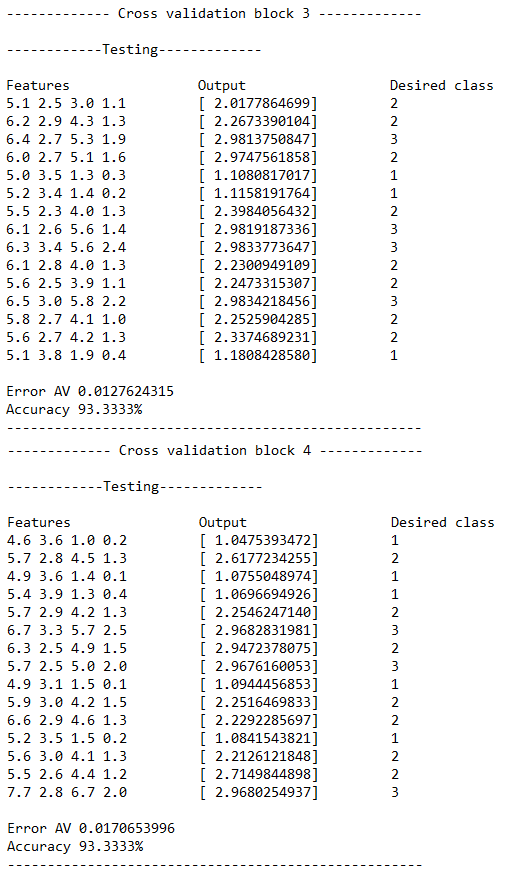
Error AV 0.0000950178

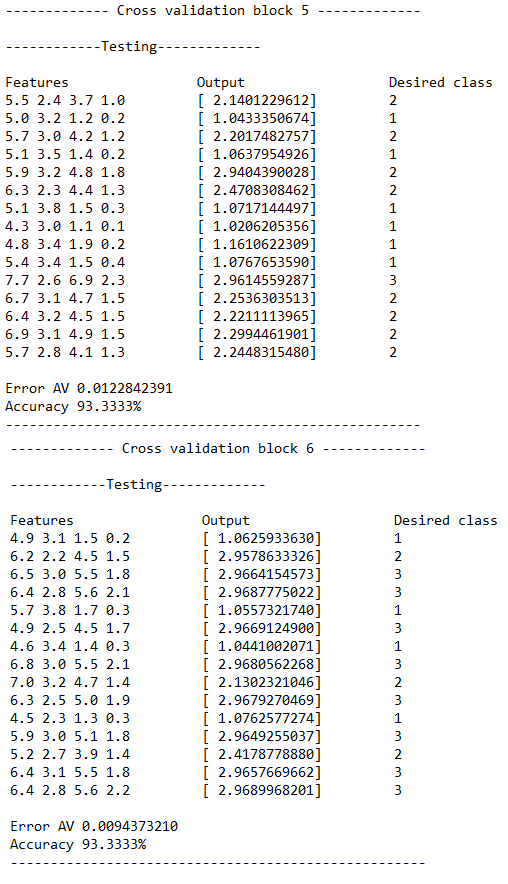
Accuracy 100.0000%

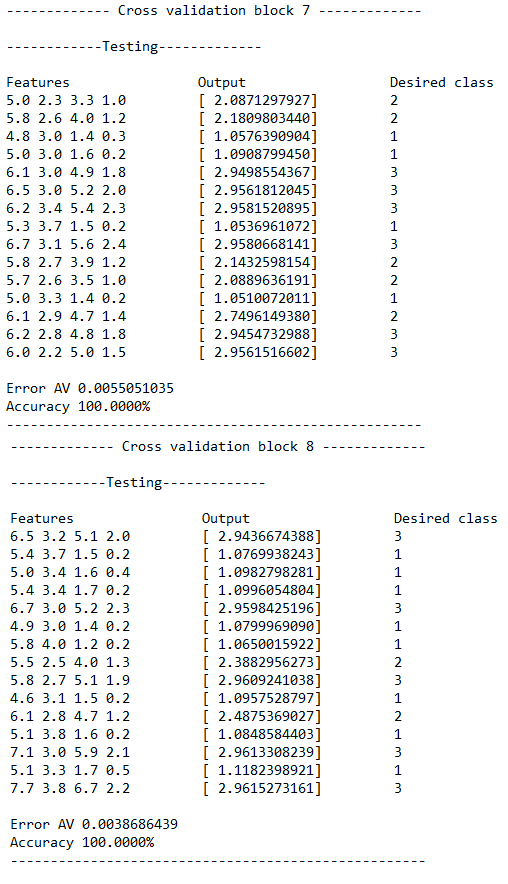
จะเห็นได้ว่า เมื่อจำนวน epoch เพิ่มขึ้นจะทำให้ Error เฉลี่ยลดลงไปด้วย จึงสรุปได้ว่า โปรแกรมสามารถแก้ปัญหา XOR ได้ถูกต้อง

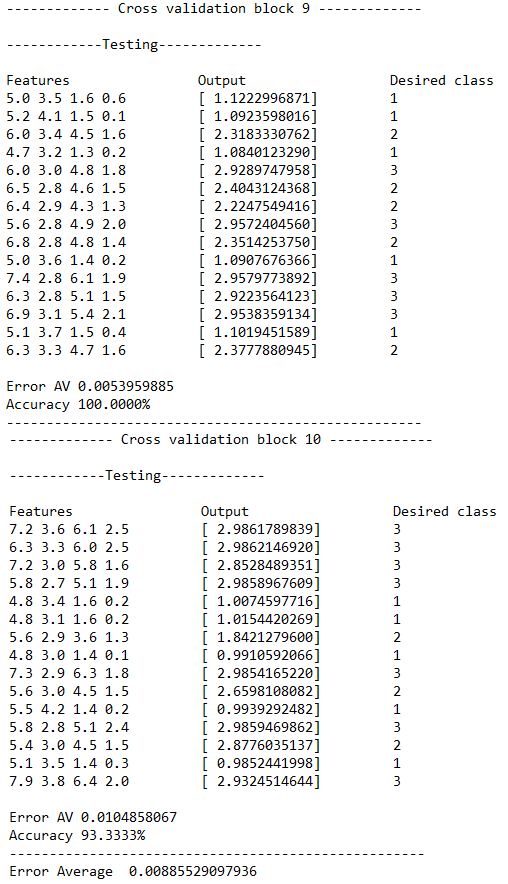
**การทดลองโดยใช้ 10% cross validation กับ Training set** [**iris.pat**](http://sansanee.cpe.eng.cmu.ac.th/IntroCI/dataset/iris.pat)

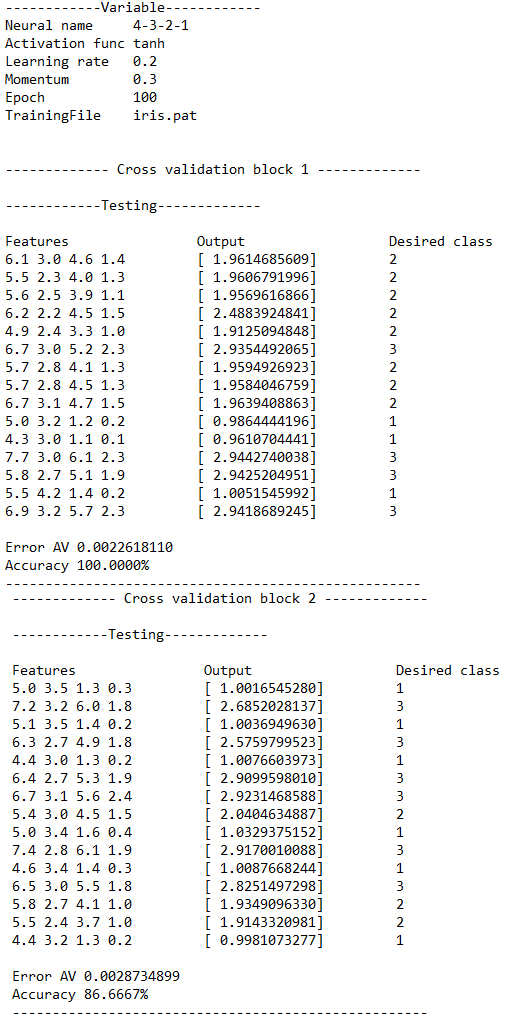
Neural network 4-4-1

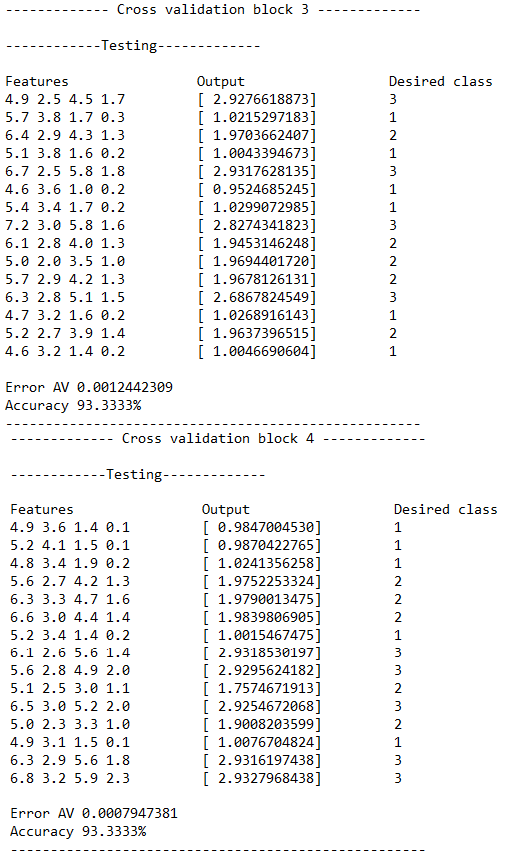


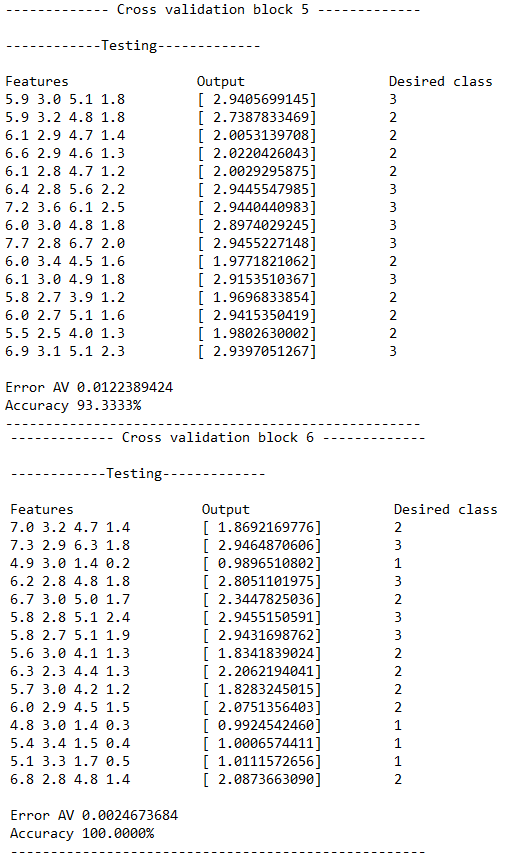


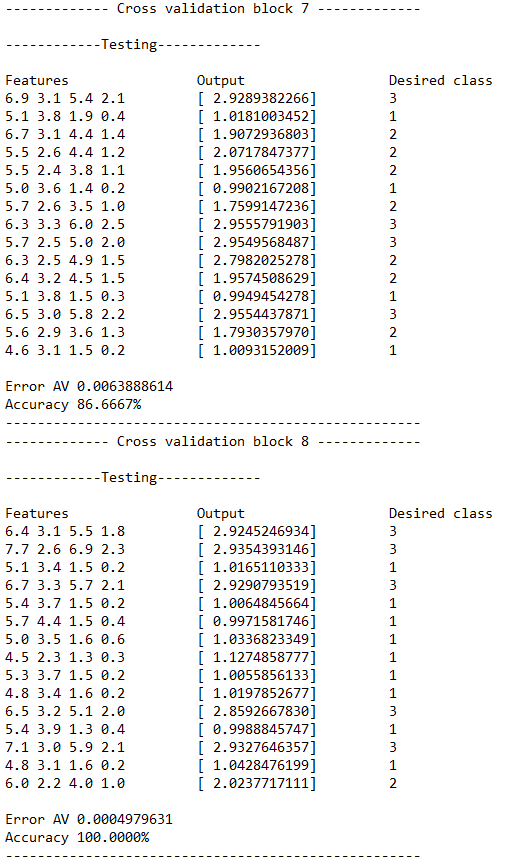


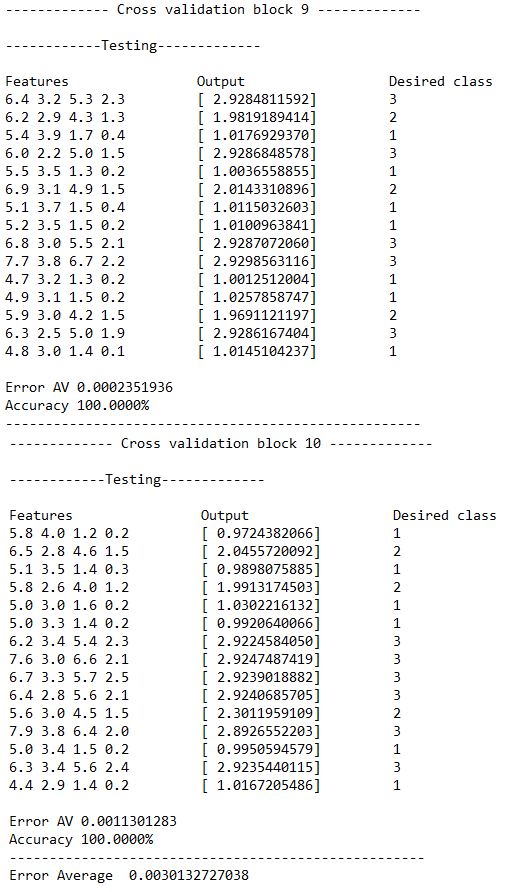


Neural network 4-3-2-1







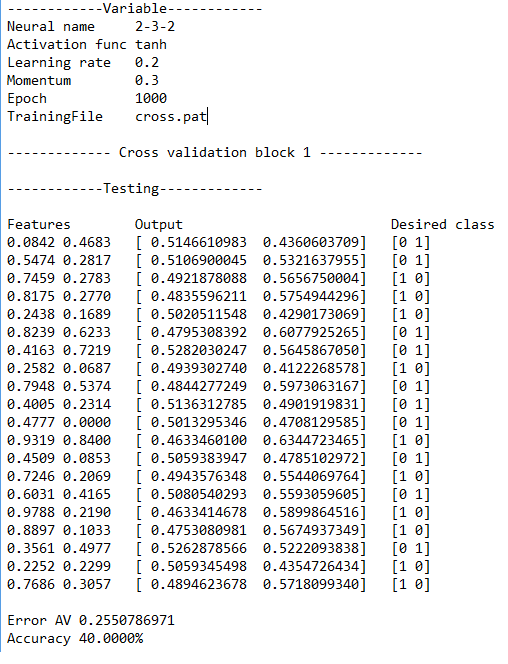


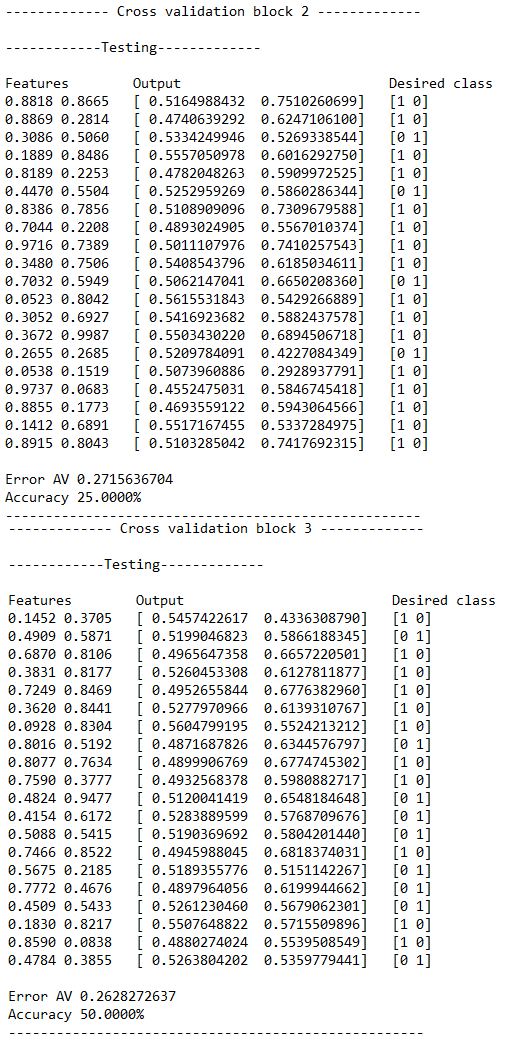
จะเห็นได้ว่า เมื่อเปลี่ยนจำนวน hidden layer จาก 4-4-1 เป็น 4-3-2-1 ทำให้ค่า error เฉลี่ยลดลง

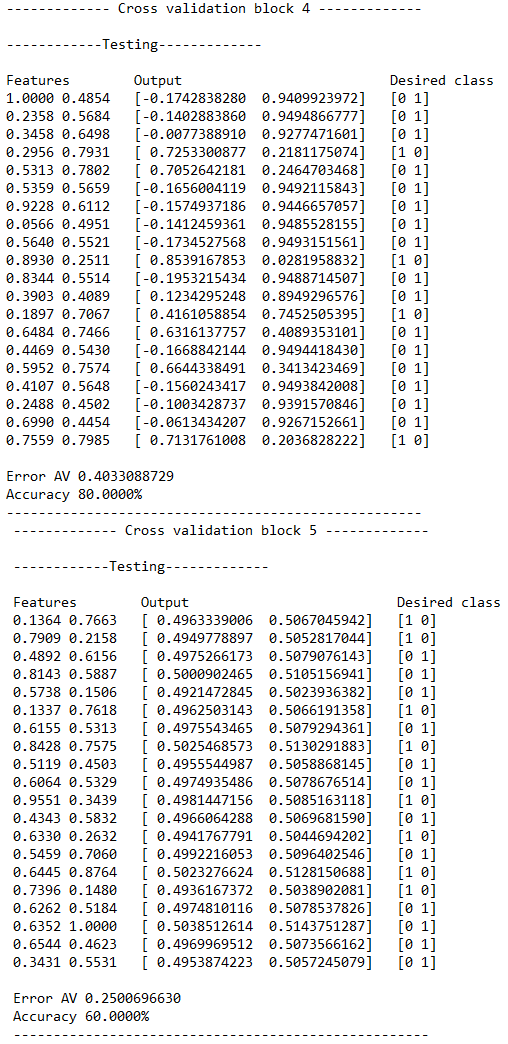
จึงสรุปได้ว่าโครงค่ายที่มีจำนวน hidden layer และ node มากกว่าจะให้ความถูกต้องมากกว่า

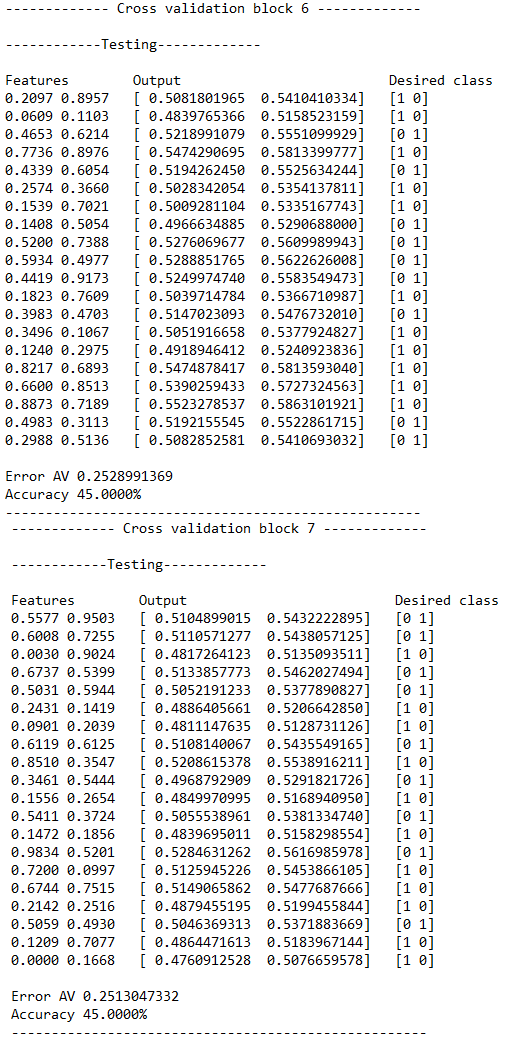
**การทดลองโดยใช้ 10% cross validation กับ Training set** [**cross.pat**](http://sansanee.cpe.eng.cmu.ac.th/IntroCI/dataset/iris.pat)

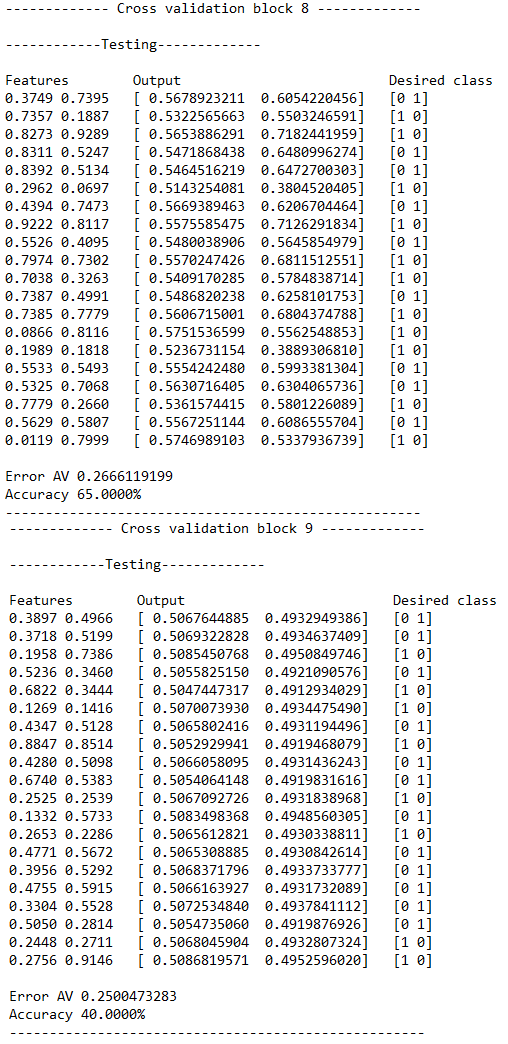
Neural network 2-3-2

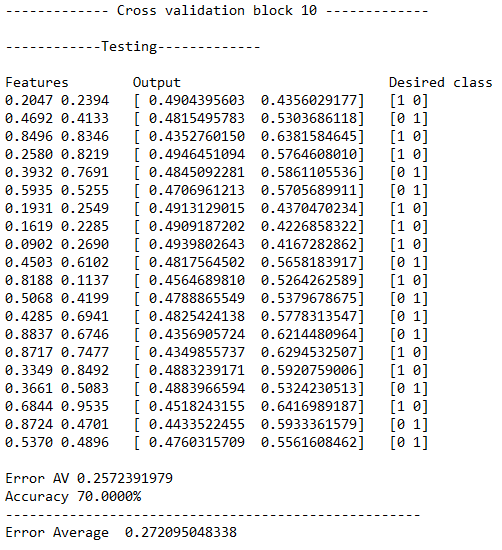




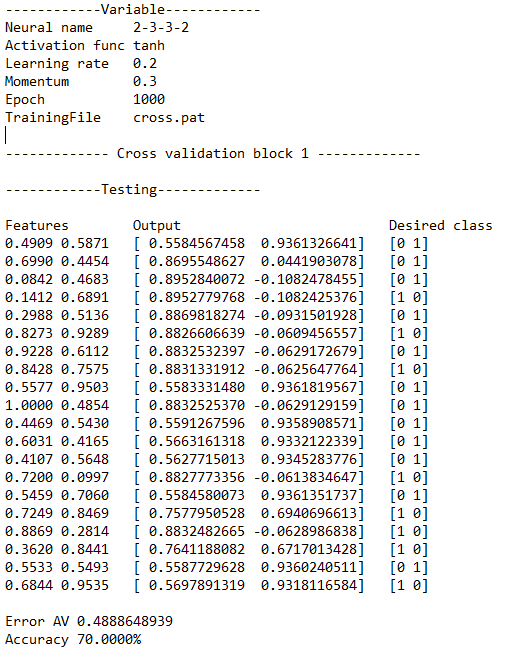


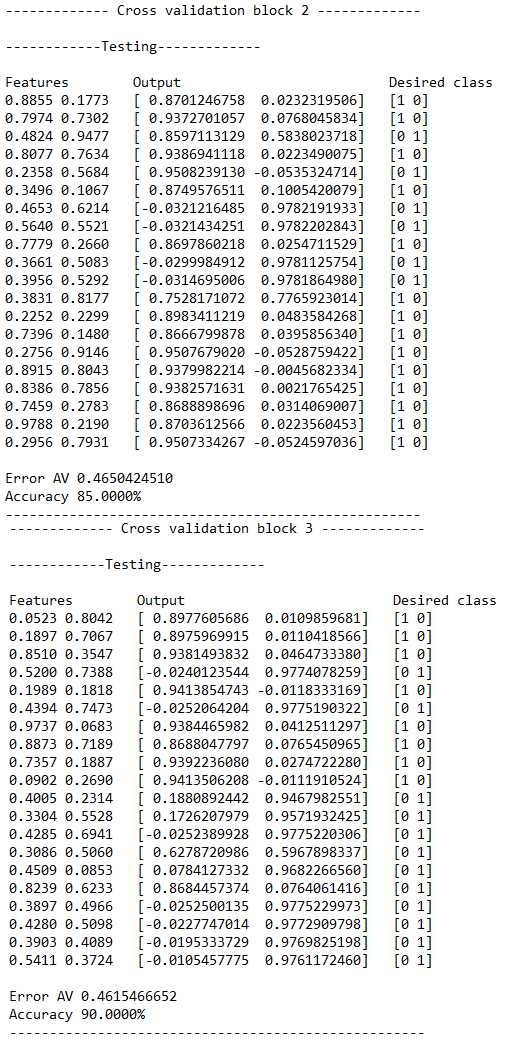


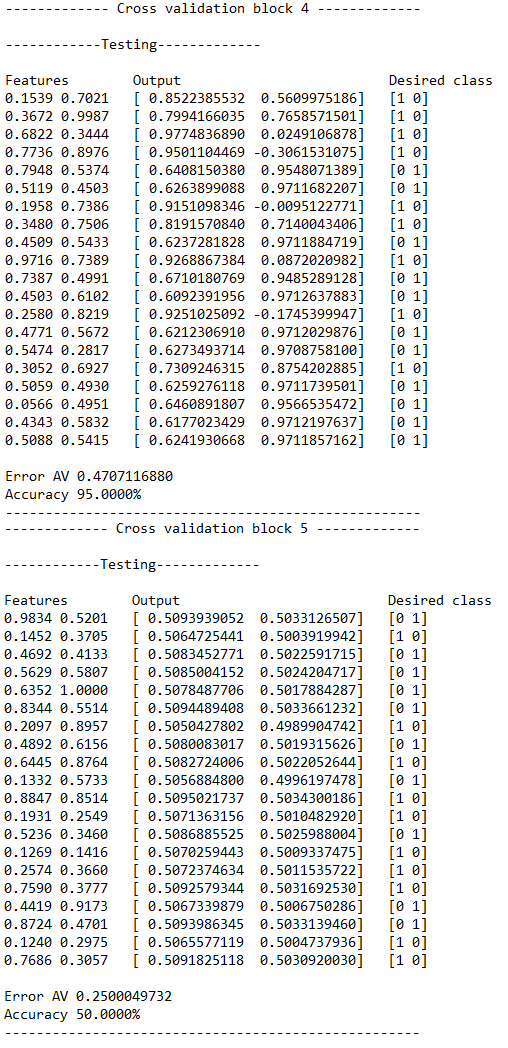


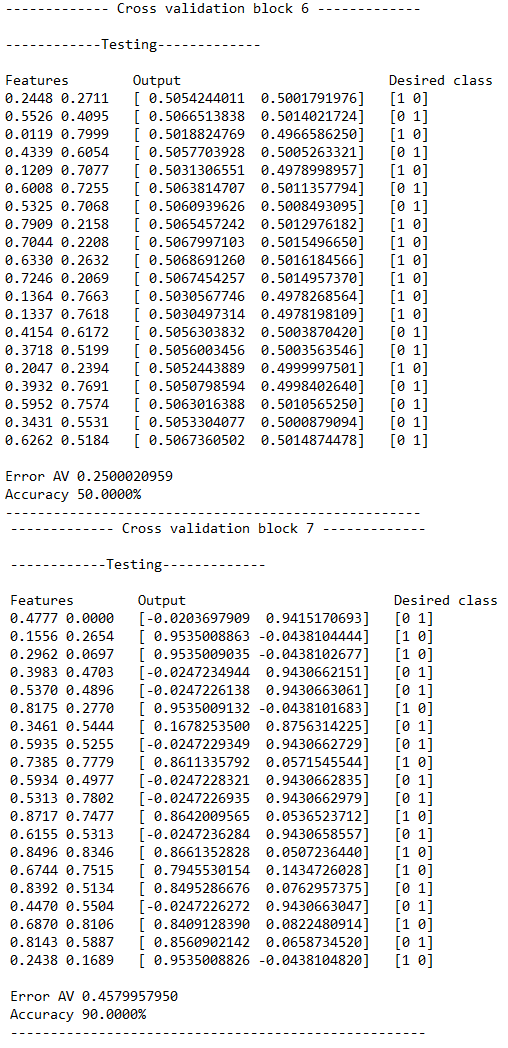


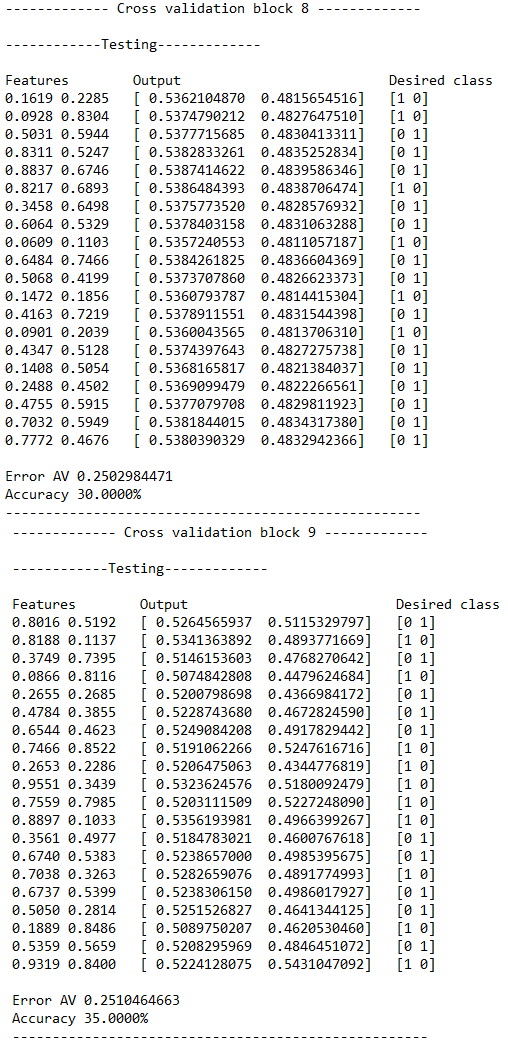
Neural network 2-3-3-2

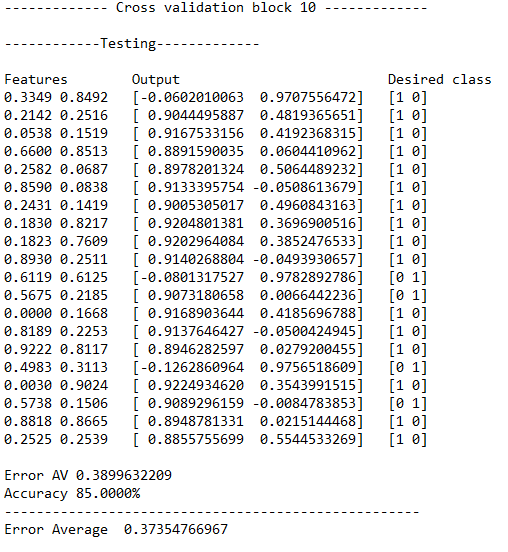












จะเห็นได้ว่า เมื่อเปลี่ยนจำนวน hidden layer จาก 2-3-2 เป็น 2-3-3-2 ทำให้ค่า error เฉลี่ยเพิ่มขึ้น

ซึ่งขัดกับผลการทดลองก่อนหน้า จึงยังสรุปไม่ได้ว่าโครงค่ายที่มีจำนวน hidden layer และ node มากกว่าจะให้ความถูกต้องมากกว่า ซึ่งอาจต้องปรับ learning rate และ momentum ให้ดีด้วย

**การทดลองปรับค่า learning rate กับ Training set** [**iris.pat**](http://sansanee.cpe.eng.cmu.ac.th/IntroCI/dataset/iris.pat)

กำหนด โครงข่าย Neural 4-4-1, Momentum = 0.3, Epoch = 100

ซึ่งทดลองปรับ learning rate เป็น 0.01, 0.05, 0.1, 0.3, 0.5 ตามลำดับ

ได้ผลการทดลองดังนี้

จึงสรุปได้ว่าถ้าปรับค่า learning rate น้อยไปหรือมากไป จะทำให้ Error Av เพิ่มขึ้น

**การทดลองปรับค่า Momentum** **กับ Training set** [**iris.pat**](http://sansanee.cpe.eng.cmu.ac.th/IntroCI/dataset/iris.pat)

กำหนด โครงข่าย Neural 4-4-1, learning rate = 0.1, Epoch = 100

ซึ่งทดลองปรับ momentum เป็น 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9 ตามลำดับ

ได้ผลการทดลองดังนี้

จึงสรุปได้ว่าถ้าปรับค่า momentum มากไป จะทำให้ Error Av เพิ่มขึ้น

**Code** (<https://github.com/porpeeranut/Computational_Intelligence_Assignment1> )

# ComputerAssignment1.py

import numpy as np

import math, sys, getopt, os, fileinput, copy, random

# Neural Networks Backpropagation

def usage():

fileName = os.path.basename(sys.argv[0])

print "\nusage: ", fileName,

print "[option]"

print " -N arg : arg is neural name"

print " -a arg : arg is activation function \"tanh\" or \"sigmoid\""

print " -n arg : arg is learning rate"

print " -m arg : arg is momentum"

print " -e arg : arg is number of epoch to exit"

print " -c : to test 10%s cross validation" % ('%')

#print " -E arg : arg is min average error to exit"

print " -t arg : arg is training set file"

print "\nex."

print fileName, "-N 2-4-1 -a tanh -n 0.2 -m 0.1 -e 10000 -t train.pat"

class NeuralNetwork:

def \_\_init\_\_(self, layers, activFunct, learning\_rate, momentum, epoch, error):

self.learning\_rate = float(learning\_rate)

self.momentum = float(momentum)

self.epoch = epoch

self.error = float(error)

if activFunct == 'sigmoid':

self.activation = sigmoid

self.activation\_derive = sigmoid\_derivertive

else:

self.activation = tanh

self.activation\_derive = tanh\_derivertive

# Init weight

self.init\_weights = []

for i in range(1, len(layers) - 1):

fanin = layers[i-1]

weightInit = 1/math.sqrt(fanin)

cellPrev = layers[i-1]

cellCurr = layers[i]

r = np.random.uniform(-1\*weightInit, weightInit, [cellPrev + 1, cellCurr + 1])

self.init\_weights.append(r)

fanin = layers[i-1]

weightInit = 1/math.sqrt(fanin)

cellOutput = layers[i+1]

cellPreOutput = layers[i]

r = np.random.uniform(-1\*weightInit, weightInit, [cellPreOutput + 1, cellOutput])

self.init\_weights.append(r)

#print "\nWeights:", self.init\_weights

def train(self, x, y):

#print "\n------------Training------------"

# add bias 1 to input layer

bias = np.atleast\_2d(np.ones(x.shape[0]))

x = np.concatenate((bias.T, x), axis=1)

self.weights = copy.deepcopy(self.init\_weights)

self.old\_weights = copy.deepcopy(self.weights)

for e in range(int(self.epoch) + 1):

# if e % int(int(self.epoch)/13) == 0:

# #print " Epoch", e, "--", e\*10 / int(int(self.epoch)/10), '%'

# sys.stdout.write('==')

# if e == int(self.epoch):

# print "%d%s" % (e\*10 / int(int(self.epoch)/10), '%')

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

y\_all = [x[i]]

v\_all = [[]]

# feedforward networks

for l in range(len(self.weights)):

v\_layer = np.dot(y\_all[l], self.weights[l])

y\_layer = self.activation(v\_layer)

v\_all.append(v\_layer)

y\_all.append(y\_layer)

# gradients at output layer

error = y[i] - y\_all[-1]

gradients = [error \* self.activation\_derive(v\_all[-1])]

# gradients at hidden layer

for l in range(len(y\_all)-2, 0, -1):

gradients.append(self.activation\_derive(v\_all[l])\*gradients[-1].dot(self.weights[l].T))

gradients.reverse()

# set new weight for back propagation

self.tmp\_old\_weights = copy.deepcopy(self.weights)

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

layer = np.atleast\_2d(y\_all[i])

gradient = np.atleast\_2d(gradients[i])

delta\_weight = self.weights[i] - self.old\_weights[i]

# print delta\_weight

# print

#self.weights[i] += self.learning\_rate \* layer.T.dot(gradient)

self.weights[i] += self.momentum \* delta\_weight + self.learning\_rate \* layer.T.dot(gradient)

self.old\_weights = copy.deepcopy(self.tmp\_old\_weights)

def test(self, listX, listY, trainingFile):

print "\n------------Testing-------------"

print "\nFeatures",

if trainingFile == "cross.pat":

print "\tOutput\t\t\t\tDesired class"

else:

print "\t\tOutput\t\t\tDesired class"

EsumSqr = 0

correct = 0

i = 0

np.set\_printoptions(formatter={'float': '{: 0.10f}'.format})

for x in listX:

if trainingFile == "iris.pat":

print " ".join('%0.1f' % f for f in x), "\t",

elif trainingFile == "cross.pat":

print " ".join('%0.4f' % f for f in x), "\t",

else:

print " ".join('%d' % f for f in x), "\t\t\t",

# add bias 1 to input layer

x = np.concatenate((np.ones(1), np.array(x)))

for l in range(0, len(self.weights)):

v = np.dot(x, self.weights[l])

x = self.activation(v)

error = listY[i] - x[-1]

Esum = 0;

if isinstance(error, np.float64):

Esum += error\*\*2

else:

for e in error:

Esum += e\*\*2

EsumSqr += Esum/2

#print "error",error

desireY = listY[i]

if trainingFile == "iris.pat":

out = x\*2+1

desireY = listY[i]\*2+1

if (out < 1.8 and desireY == 1.0) or (out >= 1.8 and out < 2.8 and desireY == 2.0) or (out >= 2.8 and desireY == 3.0):

correct = correct+1

elif trainingFile == "cross.pat":

if x[0] > x[1]:

out = np.array([1, 0])

else:

out = np.array([0, 1])

if (out == listY[i]).all():

correct = correct+1

out = x

else:

if (x < 0.5 and listY[i] == 0) or (x >= 0.5 and listY[i] == 1):

correct = correct+1

out = x

print out, "\t",

if trainingFile == "iris.pat":

print "%d" % (desireY)

else:

print desireY

i = i+1

Eav = EsumSqr/len(listX)

print "\nError AV %.10f" % (Eav)

print "Accuracy %.4f%s" % (correct/(len(listY)\*1.0)\*100.0, '%')

return Eav

def sigmoid(x):

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

def sigmoid\_derivertive(x):

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

def tanh(x):

return np.tanh(x)

def tanh\_derivertive(x):

return (1.0/np.cosh(x))\*\*2

def main(argv):

NNnameList = []

learning\_rate = 0.2

momentum = 0.1

epoch = 1000

error = 0.001

isCrossValid = 0

activFunct = 'tanh'

trainingFile = '-'

try:

opts, args = getopt.getopt(argv,"chN:a:n:m:e:E:t:")

if len(sys.argv) == 1:

usage()

sys.exit(2)

except getopt.GetoptError:

usage()

sys.exit(2)

for opt, arg in opts:

if opt == '-h':

usage()

sys.exit()

elif opt in ("-N"):

NNname = arg

NNnameList = arg.split('-')

NNnameList = map(int, NNnameList)

elif opt in ("-a"):

activFunct = arg

elif opt in ("-n"):

learning\_rate = arg

elif opt in ("-m"):

momentum = arg

elif opt in ("-e"):

epoch = arg

elif opt in ("-E"):

error = arg

elif opt in ("-c"):

isCrossValid = 1

elif opt in ("-t"):

trainingFile = arg

print "\n------------Variable------------"

print 'Neural name\t', NNname

print 'Activation func\t', activFunct

print 'Learning rate\t', learning\_rate

print 'Momentum\t', momentum

print 'Epoch\t\t', epoch

#print 'Min error\t', error

print 'TrainingFile\t', trainingFile

nn = NeuralNetwork(NNnameList, activFunct, learning\_rate, momentum, epoch, error)

listX = []

listY = []

shuffleX = []

shuffleY = []

if trainingFile == "cross.pat":

i = 1

with open(trainingFile) as f:

#with open("testcrs.pat") as f:

for line in f:

if i % 3 == 2: # features

tmp = line.split()

tmp = map(float, tmp)

listX.append(tmp)

if i % 3 == 0: # classes

tmp = line.split()

tmp = map(int, tmp)

listY.append(tmp)

i = i+1

rdIndex = random.sample(range(len(listX)), len(listX))

for i in rdIndex:

shuffleX.append(listX[i])

shuffleY.append(listY[i])

inputX = np.array(shuffleX)

outputY = np.array(shuffleY)

elif trainingFile == "iris.pat":

i = 1

with open(trainingFile) as f:

#with open("testiris.pat") as f:

for line in f:

if i != 1:

tmp = line.split()

# if int(tmp[4]) == 1:

# listY.append([1, 0, 0])

# elif int(tmp[4]) == 2:

# listY.append([0, 1, 0])

# elif int(tmp[4]) == 3:

# listY.append([0, 0, 1])

# set range y to (0,1)

listY.append((int(tmp[4])-1)/2.0)

tmp.pop()

tmp = map(float, tmp)

listX.append(tmp)

i = i+1

rdIndex = random.sample(range(len(listX)), len(listX))

for i in rdIndex:

shuffleX.append(listX[i])

shuffleY.append(listY[i])

inputX = np.array(shuffleX)

outputY = np.array(shuffleY)

else:

inputX = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

outputY = np.array([0, 1, 1, 0])

# print "\nInput X"

# print inputX

# print "\nDesire output"

# print outputY

if isCrossValid == 1:

# 10% cross validation

errorAV = 0.0

for p in range(0, 10):

print "\n\n------------- Cross validation block", p+1, "-------------"

block = int(round(len(listX)/10.0, 0))

end = (p\*block+block)-1

if p == 9:

end = len(listX)-1

tmpTestListX = []

tmpTestListY = []

trainListX = copy.deepcopy(shuffleX)

trainListY = copy.deepcopy(shuffleY)

for i in range(end, p\*block-1, -1):

tmpTestListX.append(trainListX[i])

tmpTestListY.append(trainListY[i])

trainListX.pop(i)

trainListY.pop(i)

testDataX = np.array(tmpTestListX)

testDataY = np.array(tmpTestListY)

trainDataX = np.array(trainListX)

trainDataY = np.array(trainListY)

nn.train(trainDataX, trainDataY)

errorAV = errorAV + nn.test(testDataX, testDataY, trainingFile)

print "----------------------------------------------------"

print "Error Average ", errorAV/10

else:

nn.train(inputX, outputY)

nn.test(inputX, outputY, trainingFile)

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1:])