**使用GMDH神经网络解决车流量预测问题**

**臧泽林**

# 代码块1

# ==============================================================================

# # -\*- coding: utf-8 -\*-

# ==============================================================================

"""

Created on Tue May 9 18:47:56 2017

本程序使用GMDH网络对交通的流量进行预测，输入的数据为连续n天的m组流量数据。

输出数据为第n+1天的m组的流量的预测数据。

每个神经元的学习方式为widrow-hoff

在学习过程中，每一层否挑选15个优秀的神经元保留到下一层

@author: jon zang，杭州，15168307480，zangzelin@gmail.com

"""

import numpy as np

import matplotlib.pyplot as plt

import random as rd

from scipy.special import comb, perm

import function as ft

import copy

import winsound

# ==============================================================================

# 方法的作用：

# ￼￼本方法使用GMDH算法（多项式神经网络）对交通的流量做预测

# 方法的输入输出：

# 输入的数据为连续n天的不同时间段内流量数据（一共m（m=17）个时间段）。

# 输出数据为第n+1天(下一天)的m组的流量的预测数据。

# 数据的分配

# 分配九天的数据作为训练，八天的数据作为测试

# 网络的结构：

# 使用如下结构对网络进行训练

# 输入层：9个输入层，每个输入层对应一天的车流量数据

# 隐含层1：对9个输入两两组合得到36种组合，分别对每种组合使用widrow-hoff法则进行训练，

# 后通过测试数据选取误差最小的15个神经元保留作为隐含层1的神经元。

# 隐含层2：对15个输入两两组合得到105种组合，分别对每种组合使用widrow-hoff法则进行训

# 练，后通过测试数据选取误差最小的15个神经元保留作为隐含层1的神经元。

# 隐含层的层数按照上述法则进行训练，直到网络的误差不再下降，出现上升。后停止网络，寻

# 找最后一层最好的一个神经元作为输出。

# ==============================================================================

'''

程序参数设置

'''

interval = 1

GMDHinputnum = 50 # 十五组数据输入

GMDHoutputnum = 1 # 一组数据输出

GMDHtrainnum = 9\*16 # 用于训练的数据数量

GMDHtextnum = 8\*16 # 用于测试的数据数量

GMDHpredictnum = 1 # 用于预测的数据数量

GNDHcomb = comb(GMDHinputnum, 2) # 训练数据的组合数，用于训练数据的两两组合

maxgeneration = 50 # 算法最大代数

num\_of\_group\_keep\_to\_next\_layer = GMDHinputnum # 算法挑选用来作为下一层输入的

num\_of\_start\_study\_group = 0 # 选择学士所用的数据从那一组开始

num\_of\_end\_study\_group = num\_of\_start\_study\_group + GMDHinputnum + GMDHoutputnum # 选择学习终止的数据

'''

导入数据

'''

# 导入数据

data\_oneyear = np.loadtxt('data\_oneyear.csv', dtype=np.float, delimiter=',') # 读入数据

datamin = data\_oneyear.min() # 计算最小值

datamax = data\_oneyear.max() # 计算最大值

data\_normalization = (data\_oneyear - datamin) / datamax # 利用最大值最小值进行归一化

Number\_of\_data\_Group = data\_normalization.shape[1] # 计算输入数据的数据组数

Number\_of\_each\_Group\_data = data\_normalization.shape[0] # 计算输入数据的每组的个数

print('输入数据的组数为', Number\_of\_data\_Group)

print('每组数据的个数为', Number\_of\_each\_Group\_data)

print('使用数据的组数', GMDHinputnum)

print('开始于', num\_of\_start\_study\_group, '结束于', num\_of\_end\_study\_group)

'''

数据的划分

'''

data = data\_normalization[0:Number\_of\_each\_Group\_data:interval,

num\_of\_start\_study\_group: num\_of\_end\_study\_group].transpose() # 数据划分，将有用的数据进行提取

print(data.shape)

# 输入

GMDHinputrain = data[0:GMDHinputnum, 0:GMDHtrainnum]

GMDHinputtext = data[0:GMDHinputnum, GMDHtrainnum:GMDHtrainnum + GMDHtextnum]

GMDHinputpredict = data[0:GMDHinputnum, GMDHtrainnum + GMDHtextnum:GMDHtrainnum + GMDHtextnum + GMDHpredictnum]

# 输出

GMDHoutputrain = data[GMDHinputnum:GMDHoutputnum + GMDHinputnum, 0:GMDHtrainnum]

GMDHoutputtext = data[GMDHinputnum:GMDHoutputnum + GMDHinputnum, GMDHtrainnum:GMDHtrainnum + GMDHtextnum]

GMDHoutputpredict = data[GMDHinputnum:GMDHoutputnum + GMDHinputnum,

GMDHtrainnum + GMDHtextnum:GMDHtrainnum + GMDHtextnum + GMDHpredictnum]

'''

网络的构建

每层网络设定为y=ax1+bx2的形式

'''

# 网络缓存参数

bestfit = 100000 # 最优准则值，初始值1000（任意的）

lastbestfit = 100001 # 上一次的最优准则值，初始值1001（任意的）

currentlayer = 0 # 当前第0层

allcoefficient = np.zeros((GNDHcomb, 5 \* maxgeneration)) # 初始化权值误差索引矩阵

allerror = np.zeros((maxgeneration, int(GNDHcomb), 2)) # 初始化误差矩阵

bestoutput = np.zeros((num\_of\_group\_keep\_to\_next\_layer, GMDHtrainnum)) # 初始化最佳输出矩阵

mid = np.zeros((1, 5)) # 初始化交换用的中转矩阵

errorhistory = np.zeros((maxgeneration, 1)) # 初始化历史误差矩阵

# 主循环

newinput = GMDHinputrain # 给训练输入矩阵装入初值，并初始化

newoutput = GMDHinputtext # 给训练输出矩阵装入初值，并初始化

newinputtext = GMDHinputtext # 给测试输入矩阵装入初值，并初始化

errorlist\_in\_study = np.zeros( ( maxgeneration ) ) # 初始化迭代误差清单

while not (lastbestfit < bestfit and currentlayer > 4):

# 使用训练数据，计算当前层的每一个多项式y=ax1+bx2中的a,b

[wcollection, trainoutput, textoutput] = ft.Grouptraining(newinput, GMDHoutputrain, newoutput, GMDHoutputtext)

# 使用验证数据计算误差

if currentlayer == 0: # 调试用，无实际意义

tiaoshi1 = copy.deepcopy(trainoutput)

tiaoshi2 = copy.deepcopy(textoutput)

allcoefficient[:, currentlayer \* 5:currentlayer \* 5 + 5] = wcollection # 保存计算所得权值、误差、检索矩阵

errorse = allcoefficient[:, 2 + currentlayer \* 5] # 将权值、误差、检索矩阵中的误差提取出来

errorsum = errorse.sum() # 误差进行加和

lastbestfit = bestfit # 保存上次的适应度

bestfit = errorsum # 计算总的适应度

errorlist\_in\_study[currentlayer] = bestfit

errorhistory[currentlayer] = bestfit

# 使用冒泡排序寻找匹配度最高的15个组合

for i in range(int(GNDHcomb)):

for j in range(int(GNDHcomb) - 1):

if allcoefficient[j, 5 \* currentlayer + 2] > allcoefficient[j + 1, 5 \* currentlayer + 2]:

# 交换权值、误差、检索矩阵

mid1 = copy.deepcopy(allcoefficient[j, 5 \* currentlayer:currentlayer \* 5 + 5])

allcoefficient[j, 5 \* currentlayer:currentlayer \* 5 + 5] = allcoefficient[j + 1,

5 \* currentlayer:currentlayer \* 5 + 5]

allcoefficient[j + 1, 5 \* currentlayer:currentlayer \* 5 + 5] = mid1

# 交换训练输出值矩阵

mid2 = copy.deepcopy(trainoutput[j, :])

trainoutput[j, :] = trainoutput[j + 1, :]

trainoutput[j + 1, :] = mid2

# 交换测试输出值矩阵

mid3 = copy.deepcopy(textoutput[j, :])

textoutput[j, :] = textoutput[j + 1, :]

textoutput[j + 1, :] = mid3

# 使用训练数据，计算被选中的输出

newinput = trainoutput[0:num\_of\_group\_keep\_to\_next\_layer, :]

newoutput = textoutput[0:num\_of\_group\_keep\_to\_next\_layer, :]

currentinput = bestoutput # 15\*9

# 终止条件

currentlayer = currentlayer + 1

print('第', currentlayer, '层学习，最大层数', maxgeneration)

if currentlayer > maxgeneration - 1:

break

abovelayernumber = currentlayer

np.savetxt("textoutput.csv", textoutput, fmt="%f", delimiter=",")

np.savetxt("trainoutput.csv", trainoutput, fmt="%f", delimiter=",")

np.savetxt("allcoefficient.csv", allcoefficient, fmt="%f", delimiter=",")

np.savetxt("dataout.csv", data, fmt="%f", delimiter=",")

# ==============================================================================

# 绘制学习图像

# ==============================================================================

# 构建预测矩阵和真实数据矩阵

output = np.zeros((1, newinput.shape[1]+newoutput.shape[1]))

output[0, 0:newinput.shape[1]] = newinput[0, :]

output[0, newinput.shape[1]:newinput.shape[1]+newoutput.shape[1]] = newoutput[0, :]

outputuse = output[0, 1:newinput.shape[1]+newoutput.shape[1]-1]

# 真实

real = np.zeros(newinput.shape[1]+newoutput.shape[1])

a = newinput.shape[1]+newoutput.shape[1]

real[:] = data[GMDHinputnum, 0:a]

realuse = real[1:newinput.shape[1]+newoutput.shape[1]-1]

# 计算误差

errorpredict = realuse - outputuse # 误差向量

errorpredictsum1 = errorpredict.sum() # 平均误差分子

sum11 = realuse.sum() # 误差分母

aberror = abs(errorpredict) # 误差求绝对值

errorpredictsum2 = aberror.sum() # 绝对误差分子

# 打印误差

print(errorpredictsum1 / sum11)

print(errorpredictsum2 / sum11)

plt.figure(1)

plt.plot(outputuse[:] \* datamax + datamin, 'r', label="predict")

plt.plot(realuse[:] \* datamax + datamin, label="data")

plt.legend(loc='upper right')

plt.xlabel("time(5min)")

plt.ylabel("followrate(car)")

plt.text(100, 690.8, 'average error = 0.4% \n absolute error = 2%', color='black', ha='center')

#plt.show()

ax = plt.plot(outputuse[:] \* datamax + datamin, 'r', label="predict")

# ==============================================================================

# 绘制预测图像

# 通过新输入的数据对网络的预测能力进行衡量

# ==============================================================================

# 导入叫大数据量的数组

num\_of\_text\_time = 50

errorlist\_in\_test = np.zeros((num\_of\_text\_time))

sueshuju = 2 # 选择使用的数据组数，选择0为使用学习的数据，每多加1，数据向后挫一位

# 建立数据处理矩阵，将288组数据压缩成16组

for sueshuju in range(num\_of\_text\_time):

pdata = data\_normalization[0:Number\_of\_each\_Group\_data:interval,\

num\_of\_start\_study\_group + sueshuju: num\_of\_end\_study\_group + sueshuju].transpose()

predictdata = pdata[0:GMDHinputnum, :] # 选取输入数据

# predictoutput = np.zeros()

comefrom = np.zeros((2)) # 网络结构检索号保存矩阵初始化

pcurrentinput = predictdata[0:num\_of\_group\_keep\_to\_next\_layer, :] # 为当前的矩阵赋初值

pinput = np.zeros((2, Number\_of\_each\_Group\_data)) # 初始化输入矩阵

poutput = np.zeros((num\_of\_group\_keep\_to\_next\_layer, Number\_of\_each\_Group\_data)) # 初始化输出矩阵

pw = np.zeros(2) # 初始化权值矩阵

# 遍历网络层

for i in range(abovelayernumber):

# 遍历选中的组合

for j in range(GMDHinputnum):

# 取出权值

pw[0] = allcoefficient[j, i \* 5 + 0]

pw[1] = allcoefficient[j, i \* 5 + 1]

# 取出连接的输入

comefrom[0] = allcoefficient[j, i \* 5 + 3]

comefrom[1] = allcoefficient[j, i \* 5 + 4]

# 取出需要用的两行输入

pinput[0, :] = pcurrentinput[comefrom[0], :]

pinput[1, :] = pcurrentinput[comefrom[1], :]

poutput[j] = pinput[0, :] \* pw[0] + pinput[1, :] \* pw[1]

pcurrentinput = poutput

# 绘图分析

errortest = (poutput[0, :]-pdata[GMDHinputnum, :])

errortestsum = abs(poutput[0, :]).sum()

errorlist\_in\_test[sueshuju] = abs(errortest).sum()/errortestsum

plt.figure(2+sueshuju)

plt.plot(poutput[0, :] \* datamax + datamin, 'r', label="predict")

plt.plot(pdata[GMDHinputnum, :] \* datamax + datamin, label="data")

plt.legend(loc='upper right')

plt.xlabel("time(80min)")

plt.ylabel("followrate(car)")

plt.text(8.5, 8990.8, 'average error = 1% \n absolute error = 2%', color='black', ha='center')

winsound.Beep(367, 1000)

print('平均误差', errorlist\_in\_test.sum()/num\_of\_text\_time)

plt.show()

# 代码块2

# -\*- coding: utf-8 -\*-

"""

Created on Wed May 10 10:07:44 2017

@author: zangz

"""

import numpy as np

import matplotlib.pyplot as plt

import random as rd

from scipy.special import comb, perm

# ==============================================================================

# 单个多项式网络训练函数

# 输入： input1为输入的两行数据，每一行代表一个输入组，aim为目标的数据，训练的目的是

# 到最佳的w1,w2,使得w1\*第一行数据加上w2乘以第二行数据与输出尽可能相似。

# 测试函数如下：

# input1 = np.array([[2,5,8,3,6,9,10,54,5],

# [9,8,34,21,6,9,2,4,1 ]])

# aim = np.array([2,5,8,3,6,9,10,54,5])

#

# [ w , output , i , errorn ]=Separatetraining(input1,aim)

#

# print(i,errorn)

# plt.figure(1)

# plt.plot(aim)

# plt.plot(output)#如果两条曲线相似，则说明训练成功

# ==============================================================================

def Separatetraining(input1, aim):

# inputshape = input1.shape

# aimshape = aim.shape

# 随机生成初始矩阵

w = np.array([rd.random(), rd.random()]) \* 2 - 1

errorn = 1000 # 初始误差

errorlast = 1010 # 初始上次误差

i = 0

step = 0.5 # 步长

while errorn < errorlast:

errorlast = errorn

errorlist = (np.mat(input1).T \* np.mat(w).T).T - aim # 计算误差矩阵

abserrorlist = np.array(abs(errorlist)) # 计算误差矩阵的绝对值

errorn = abserrorlist.sum() # 计算误差和

minput = np.mat(input1) # 将输入参数矩阵化以便进行矩阵乘法

# 利用widrow-hoff学习法则进行训练

w = w - step \* (errorlist \* minput.T) / (input1 \*\* 2).sum()

i = i + 1

if i > 1000:

break

npw = np.array(w)

output = np.array((np.mat(input1).T \* np.mat(w).T).T)

# plt.figure()

# plt.plot(aim,'r')

# plt.plot(output,'k')

# #a = input()

return [npw, output, i, errorn]

# ==============================================================================

# 对之前训练好的网络进行检验的函数，计算出其与检验数据的误差，用于挑选优秀的网络

# 输入：

# w之前训练好的权值矩阵

# sepratetextin检验用的输入数据

# sepratetextout检验用的输出数据

# 输出：

# errorn得到的检验误差

# ==============================================================================

def Getsuitlevel(w, sepratetextin, sepratetextout):

output = np.array((np.mat(sepratetextin).T \* np.mat(w).T).T)

errorlist = (np.mat(sepratetextin).T \* np.mat(w).T).T - sepratetextout # 计算误差矩阵

abserrorlist = np.array(abs(errorlist)) # 计算误差矩阵的绝对值

errorn = [abserrorlist.sum()] # 计算误差和

return [errorn, output]

# ==============================================================================

# 用单个多项式训练函数与单个的检验韩式对输入数据进行成组训练的函数并成组检验的函数

# 输入：GMDHinputrain是训练用的输入数据，n行数据，

# GMDHoutputrain是训练用做目标的1行数据

# GMDHinputtext是检验用的输入数据，n行数据，

# GMDHoutputtext是检验用做目标的1行数据

# 输出：训练好的w的组，每行两个数据分别是权值w1,与w2，和检验好的误差，

# 将在函数外进行有效的选择

# 测试函数如下：

# data = np.random.random(size=(16,18))

# #所有数据归一

# data = (data - data.min())/data.max()

# GMDH = []

# GMDHinputnum = 15# 十五组数据输入

# GMDHoutputnum = 1# 一组数据输出

# GMDHtrainnum = 9

# GMDHtextnum = 8

# GMDHpredictnum = 1

# GNDHcomb = comb(GMDHtrainnum,2)

# maxgeneration = 50

# GMDHinputrain = data[0:GMDHinputnum , 0 :GMDHtrainnum ]

# GMDHinputtext = data[0:GMDHinputnum , GMDHtrainnum :GMDHtrainnum+GMDHtextnum ]

# GMDHinputpredict = data[0:GMDHinputnum , GMDHtrainnum+GMDHtextnum :GMDHtrainnum+GMDHtextnum+GMDHpredictnum ]

# GMDHoutputrain = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum , 0 :GMDHtrainnum ]

# GMDHoutputtext = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum, GMDHtrainnum :GMDHtrainnum+GMDHtextnum ]

# GMDHoutputpredict = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum , GMDHtrainnum+GMDHtextnum :GMDHtrainnum+GMDHtextnum+GMDHpredictnum ]

# wcollection = Grouptraining(GMDHinputrain,GMDHoutputrain,GMDHinputtext,GMDHoutputtext)

# ==============================================================================

def Grouptraining(GMDHinputrain, GMDHoutputrain, GMDHinputtext, GMDHoutputtext):

inputtrainshape = GMDHinputrain.shape # 确定输入的数组的行数和列数

# outputtrainshape = GMDHoutputrain.shape

inputtextshape = GMDHinputtext.shape # 确定输入的数组的行数和列数

# outputtextshape = GMDHoutputtext.shape

GMDHtrainnum = inputtrainshape[0] # 得到用来训练的数据个数

GMDHtextnum = inputtextshape[0]

sepratetrain = np.zeros((2, inputtrainshape[1])) # 初始化Separatetraining函数的输入矩阵

sepratetrainout = np.zeros((1, inputtrainshape[1])) # 初始化Separatetraining函数的输出矩阵

sepratetextin = np.zeros((2, inputtextshape[1])) # 初始化Separatetraining函数的输入矩阵

sepratetextout = np.zeros((1, inputtextshape[1])) # 初始化Separatetraining函数的输出矩阵

corrantn = 0

GNDHcomb = comb(GMDHtrainnum, 2)

trainoutput = np.zeros((GNDHcomb, inputtrainshape[1]))

textoutput = np.zeros((GNDHcomb, inputtextshape[1]))

wcollection = np.zeros((GNDHcomb, 5)) # 初始话将要输出的权值矩阵

for i in range(GMDHtrainnum): # 用两重循环遍历所有的两两组合

for j in range(i + 1, GMDHtrainnum):

# print(i,j)

sepratetrain[0, :] = GMDHinputrain[i, :] # 装填输入矩阵

sepratetrain[1, :] = GMDHinputrain[j, :]

sepratetrainout[0, :] = GMDHoutputrain[0, :] # 装填输出矩阵

sepratetextin[0, :] = GMDHinputtext[i, :] # 装填输入矩阵

sepratetextin[1, :] = GMDHinputtext[j, :]

sepratetextout[0, :] = GMDHoutputtext[0, :] # 装填输出矩阵

[w, output1, loopnum, errorn] = Separatetraining(sepratetrain, sepratetrainout) # 调用计算

trainoutput[corrantn, :] = output1[0, :]

if (i == 0 and j == 2):

plt.figure(99)

plt.plot(output1[0, :], 'r')

plt.plot(sepratetrainout[0, :])

# a = input()

[error, output2] = Getsuitlevel(w, sepratetextin, sepratetextout)

textoutput[corrantn, :] = output2

w1w2errorij = [w[0, 0], w[0, 1], error[0], i, j]

wcollection[corrantn, :] = w1w2errorij # 保存权值

corrantn = corrantn + 1

return [wcollection, trainoutput, textoutput]

# data = np.random.random(size=(16,18))

##所有数据归一

# data = (data - data.min())/data.max()

# GMDH = []

# GMDHinputnum = 15# 十五组数据输入

# GMDHoutputnum = 1# 一组数据输出

# GMDHtrainnum = 9

# GMDHtextnum = 8

# GMDHpredictnum = 1

# GNDHcomb = comb(GMDHtrainnum,2)

# maxgeneration = 50

# GMDHinputrain = data[0:GMDHinputnum , 0 :GMDHtrainnum ]

# GMDHinputtext = data[0:GMDHinputnum , GMDHtrainnum :GMDHtrainnum+GMDHtextnum ]

# GMDHinputpredict = data[0:GMDHinputnum , GMDHtrainnum+GMDHtextnum :GMDHtrainnum+GMDHtextnum+GMDHpredictnum ]

# GMDHoutputrain = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum , 0 :GMDHtrainnum ]

# GMDHoutputtext = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum, GMDHtrainnum :GMDHtrainnum+GMDHtextnum ]

# GMDHoutputpredict = data[GMDHinputnum:GMDHoutputnum+GMDHinputnum , GMDHtrainnum+GMDHtextnum :GMDHtrainnum+GMDHtextnum+GMDHpredictnum ]

# wcollection = Grouptraining(GMDHinputrain,GMDHoutputrain,GMDHinputtext,GMDHoutputtext)