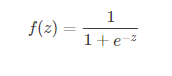
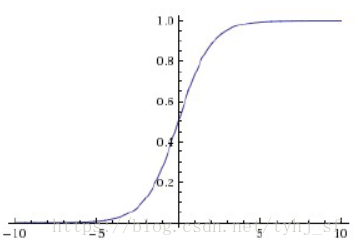
**实验名称：**神经网络

**实验目的:**用python实现神经网络

1. **Sigmoid函数**

Sigmoid 是常用的非线性的激活函数，它的数学形式如下

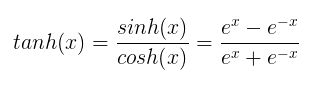
Sigmoid的几何图像如下：

特点：   
 它能够把输入的连续实值变换为0和1之间的输出，特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1.

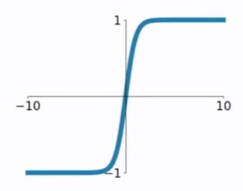
缺点：

当输入稍微远离了坐标原点，函数的梯度就变得很小了，几乎为零。在神经网络反向传播的过程中，我们都是通过微分的链式法则来计算各个权重w的微分的。当反向传播经过了sigmod函数，这个链条上的微分就很小很小了，况且还可能经过很多个sigmod函数，最后会导致权重w对损失函数几乎没影响，这样不利于权重的优化，这个问题叫做梯度饱和，也可以叫梯度弥散。

1. **Tanh函数**

Tanh函数公式和曲线如下**：**

Tanh的几何图像如下：

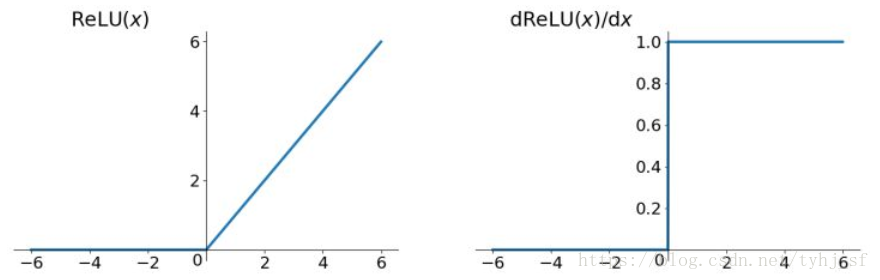
****

特点：   
 tanh是双曲正切函数，tanh函数和sigmod函数的曲线是比较相近的，咱们来比较一下看看。首先相同的是，这两个函数在输入很大或是很小的时候，输出都几乎平滑，梯度很小，不利于权重更新；不同的是输出区间，tanh的输出区间是在(-1,1)之间，而且整个函数是以0为中心的，这个特点比sigmod的好。

1. **Relu函数**

Relu函数的解析式：

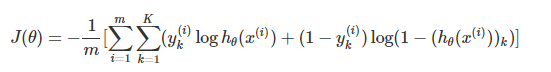
Relu函数及其导数的图像如下图所示：



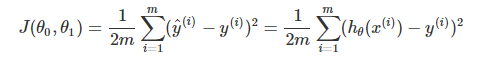
当X>0时，函数的输出值为X；当X<=0时，输出值为0。使用ReLU函数的最主要的好处是对于大于0的所有输入来说，它都有一个不变的导数值。常数导数值有助于网络训练进行得更快。

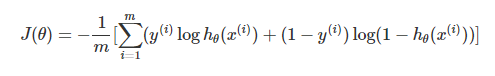
缺点：当输入是负数的时候，ReLU是完全不被激活的，这就表明一旦输入到了负数，ReLU就会死掉。这样在前向传播过程中，还不算什么问题，有的区域是敏感的，有的是不敏感的。但是到了反向传播过程中，输入负数，梯度就会完全到0

1. **代价函数**



1. **均方误差**

**6.交叉熵**



实验代码：

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import random

import math

import numpy as np

def MSE(y\_true,y\_pred,

sample\_weight=None,

multioutput='uniform\_average'):

sum = 0

length = len(y\_true)

for i in range(length):

sum += (y\_true[i]-y\_pred[i])\*\*2

sum = sum / length

return sum

dataset = pd.read\_csv('G:\BP-master\watermelon\_3.csv', delimiter=" ")

dataset=np.array(dataset)

m,n=np.shape(dataset)

for i in range(m):

for j in range(n):

dataset[i,j]=round(dataset[i,j],3) #定义数据格式，dataset[i,j]保留小数点后3位

trueY=dataset[:,n-1] #取类别标签

X=dataset[:,:n-1] #取属性部分

m,n=np.shape(X) #行列值

#according to P101, init the parameters

# v = d\*q .输入层到输出层权重

v=[[random.random() for i in range(n+1)] for j in range(n)]

# w = q\*l .隐藏层到输出层权重

w=[[random.random() for i in range(1)] for j in range(n+1)]

#激活函数

def sigmoid(iX,dimension):#iX一维矩阵

if dimension==1:

for i in range(len(iX)):

iX[i] = 1 / (1 + math.exp(-iX[i]))

else:

for i in range(len(iX)):

iX[i] = sigmoid(iX[i],dimension-1)

return iX

'''

累积误差逆传播算法：

累积BP算法直接针对累积误差最小化，他在读取整个训练集D一遍后才对

参数进行更新，其参数更新的频率低得多

'''

eta=0.2 #训练速率

d=n #输入向量的维度

l=1 #输出向量的维度

q=d+1 #隐层神经元的数目

theta=[random.random() for i in range(l)] #输出层神经元的阈值

gamma=[random.random() for i in range(q)] #隐层神经元阈值

trueY=trueY.reshape((m,l))

maxIter=5000 #最大训练时间

while(maxIter>0):

maxIter-=1

sumE=0

alpha = np.dot(X, v)#p101 line 2 from bottom, shape=m\*q

b = sigmoid(alpha - gamma,2) # b=f(alpha-gamma), shape=m\*q

beta = np.dot(b, w) # shape=(m\*q)\*(q\*l)=m\*l

predictY = sigmoid(beta - theta,2) # shape=m\*l ,p102--5.3

E = sum(sum((predictY - trueY) \* (predictY - trueY))) / 2 # 5.4 均方误差

g = predictY \* (1 - predictY) \* (trueY - predictY) # shape=m\*l p103--5.10

e = b \* (1 - b) \* ((np.dot(w, g.T)).T) # shape=m\*q , p104--5.15

w += eta \* np.dot(b.T, g) #shape (q\*l)=(q\*m) \* (m\*l) 隐层到输出层连接权

theta -= eta \* g # 5.12 阈值

v += eta \* np.dot(X.T, e) # 5.13 (d,q)=(d,m)\*(m,q) 输入层到隐层连接权

gamma -= eta \* e # 5.14 阈值

def predict(iX):

'''

beta： 输出层神经元接收到的输入

theta：输出层神经元的阈值

'''

alpha = np.dot(iX, v) #从输入层到隐层

b=sigmoid(alpha-gamma,2) # 隐层输出

beta = np.dot(b, w) #从隐层到输出层

predictY=sigmoid(beta - theta,2) #输出层输出

return predictY

def plotBestFit(dataArr,labelMat1,labelMat2):

'''

分类效果展示

@:param weights 回归系数

@:param path 数据文件路径

@:return null

'''

n = len(dataArr) #取行数

xcord1 = []; ycord1 = []

xcord2 = []; ycord2 = []

xcord3 = []; ycord3 = []

xcord4 = []; ycord4 = []

for i in range(n): #将训练前的数据分类存储

if int(labelMat1[i])== 1:#

xcord1.append(dataArr[i][0]); ycord1.append(dataArr[i][1])

else:

xcord2.append(dataArr[i][0]); ycord2.append(dataArr[i][1])

for i in range(n): #将训练后的数据分类存储

if int(labelMat2[i])== 1:

xcord3.append(dataArr[i][0]); ycord3.append(dataArr[i][1])

else:

xcord4.append(dataArr[i][0]); ycord4.append(dataArr[i][1])

'''

神经网络预测结果

'''

plt.figure("BPML1")

plt.title('Original')

plt.scatter(xcord1, ycord1, s=30, c='red', marker='s')

plt.scatter(xcord2, ycord2, s=30, c='green')

plt.xlabel('X1');plt.ylabel('X2')

fig = plt.figure("BPML2") #新建一个画图窗口

ax = fig.add\_subplot(111) #添加一个子窗口

ax.set\_title('Forecast')

ax.scatter(xcord3, ycord3, s=30, c='red', marker='s')

ax.scatter(xcord4, ycord4, s=30, c='green')

plt.xlabel('X1'); plt.ylabel('X2')

plt.show()

def main():

result = predict(X)

h = []

for i in range(len(result)):

if result[i] > 0.5:

h.append(1)

else:

h.append(0)

plotBestFit(dataset,trueY,h)

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

import numpy as np

import xlrd

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

def open\_excel(file):

"""

打开excel文件获取数据

:param file: 文件所在的位置

:return: 文件数据

"""

try:

data = xlrd.open\_workbook(file)

return data

except Exception as e:

print(str(e))

def split\_feature(row):

"""

将该行特征处理后放入列表中

:param row:一行特征数据

:return: 返回数据列表

"""

app = []

for i in range(16):

app = app + [row[i]]

return app

def loadDataSet(path, training\_sample, colnameindex=0, by\_name=u'sheet1'):

"""

加载数据

:param path: 数据文件存放路径

:param training\_sample: 数据文件名

:param colnameindex: 文件列名下标

:param by\_name: 表名

:return: 数据集和类别标签

"""

dataMat = [] # 定义数据列表

labelMat = [] # 定义标签列表

filename = path + training\_sample # 形成特征数据的完整路径

data = open\_excel(filename) # 打开文件获取数据

table = data.sheet\_by\_name(by\_name) # 获得数据表

nrows = table.nrows # 得到表数据总行数

colnames = table.row\_values(colnameindex) # 某一行数据 ['user\_id', 'age\_range', 'gender', 'merchant\_id','label']

for rownum in range(1, nrows): # 也就是从Excel第二行开始，第一行表头不算

row = table.row\_values(rownum) # 取一行数据

'''

判断2,3,6列数据是否为空，若为空则丢弃该行数据

'''

if row[1] == '' or row[2] == '' or row[5] == '':

continue

if row:

app = split\_feature(row) # 将特征值转化为列表

dataMat.append(app)

labelMat.append(float(row[16])) # 获取类别标签

return dataMat, labelMat

def show\_accuracy(a, b, tip):

"""

计算准确率

:param a: 真实类别

:param b: 预测标签

:param tip: 描述

:return: 准确率

"""

acc = a.ravel() == b.ravel()

print("%s Accuracy:%.3f" % (tip, np.mean(acc)))

def main():

"""

主函数

:return: null

"""

path = "E:\\"

training\_sample = 'featuredata.xls' # 特征数据文件

trainingSet, trainingLabels = loadDataSet(path, training\_sample) # 取特征数据和标签数据

x = np.array(trainingSet) # 将数据部分列表（list）格式转化为数组(array)格式

y = np.array(trainingLabels) # 将标签部分的列表（list）格式转化为数组格式（array）

'''

将数据分为训练数据和测试数据两部分

x\_train 训练数据

x\_test 测试数据

y\_train 训练数据标签

y\_test 测试数据标签

'''

train\_data, test\_data, train\_label, test\_label = train\_test\_split(x, y, random\_state=1, test\_size=0.3)

# 定义多层感知机分类算法

clf = MLPClassifier(activation='relu', solver='adam', alpha=0.0001)

clf.fit(train\_data, train\_label) # 利用训练数据训练模型

hat\_test\_label = clf.predict(test\_data)

print(classification\_report(test\_label, hat\_test\_label))

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

"""

程序入口

"""

main()