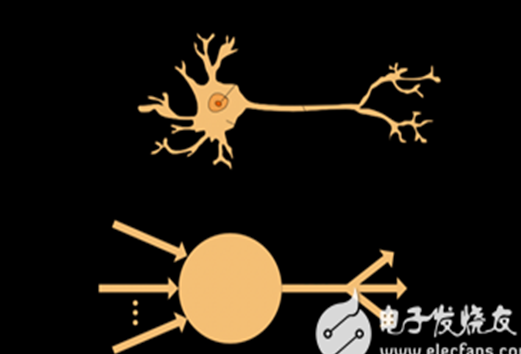
第六节、神经网络

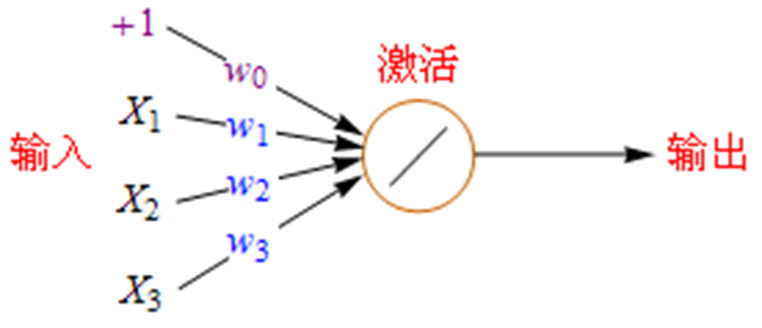
一、神经网络的基本概述

正如我们的人脑一样，在一个层次上和神经元网络中有数百万个神经元，这些神经元通过一种称之为synapses（突触）的结构彼此紧紧相连。它可以通过 Axons（轴突），将电信号从一个层传递到另一个层。这就是我们人类学习事物的方式。 每当我们看到、听到、感觉和思考时，一个突触（电脉冲）从层次结构中的一个神经元被发射到另一个神经元，这使我们能够从我们出生的那一天起，就开始学习、记住和回忆我们日常生活中的东西。

人工神经网络（Artificial Neural Network，即ANN ），是20世纪80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象， 建立某种简单模型，按不同的连接方式组成不同的网络，由大量的节点（或称神经元）之间相互联接构成。

1.神经元：就像形成我们大脑基本元素的神经元一样，神经元形成神经网络的基本结构。想象一下，当我们得到新信息时我们该怎么做。当我们获取信息时，我们一般会处理它，然后生成一个输出。类似地，在神经网络的情况下，神经元接收输入，处理它并产生输出，而这个输出被发送到其他神经元用于进一步处理，或者作为最终输出进行输出。



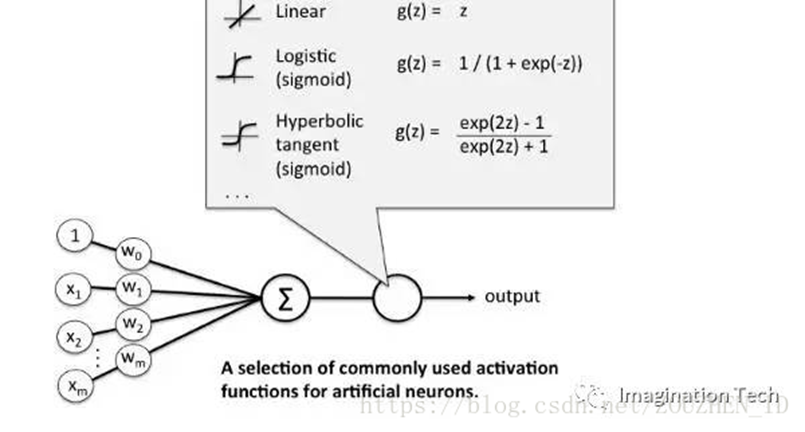


2.权重：当输入进入神经元时，它会乘以一个权重。例如，如果一个神经元有两个输入，则每个输入将具有分配给它的一个关联权重。我们随机初始化权重，并在模型训练过程中更新这些权重。训练后的神经网络对其输入赋予较高的权重，这是它认为与不那么重要的输入相比更为重要的输入。为零的权重则表示特定的特征是微不足道的。

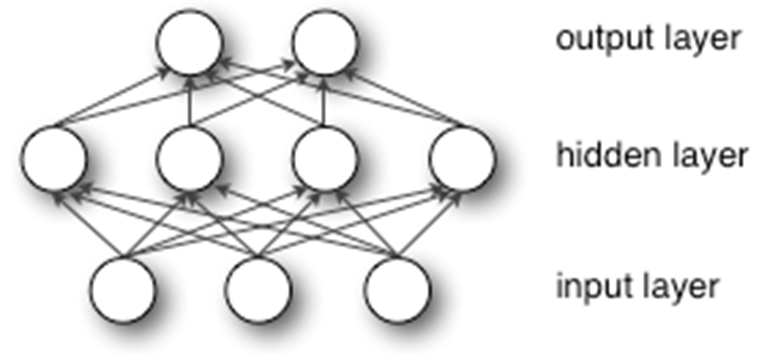
让我们假设输入为a，并且与其相关联的权重为W1，那么在通过节点之后，输入变为a \* W1。

3.偏差：除了权重之外，另一个被应用于输入的线性分量被称为偏差。它被加到权重与输入相乘的结果中。基本上添加偏差的目的是来改变权重与输入相乘所得结果的范围的。添加偏差后，结果将看起来像a\* W1 +偏差。这是输入变换的最终线性分量。

4.激活函数：一旦将线性分量应用于输入，将会需要应用一个非线性函数。这通过将激活函数应用于线性组合来完成。激活函数将输入信号转换为输出信号。应用激活函数后的输出看起来像f（a \* W1 + b），其中f（）就是激活函数。(后面将详细介绍)



5.输入/输出/隐藏层（Input / Output / Hidden Layer） ：正如它们名字所代表的那样，输入层是接收输入那一层，本质上是网络的第一层。而输出层是生成输出的那一层，也可以说是网络的最终层。处理层是网络中的隐藏层。这些隐藏层是对传入数据执行特定任务并将其生成的输出传递到下一层的那些层。输入和输出层是我们可见的，而中间层则是隐藏的。

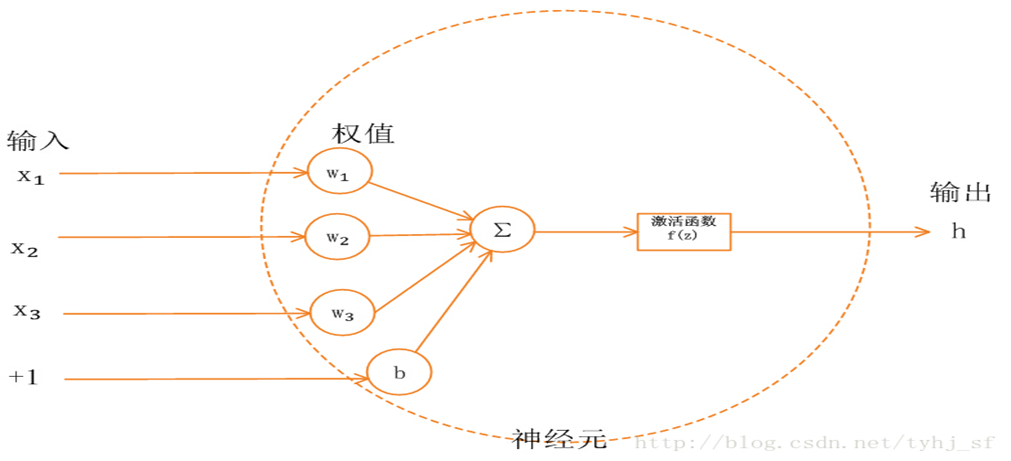


6.成本函数（Cost Function）：当我们建立一个网络时，网络试图将输出预测得尽可能靠近实际值。我们使用成本/损失函数来衡量网络的准确性。而成本或损失函数会在发生错误时尝试惩罚网络。

我们在运行网络时的目标是提高我们的预测精度并减少误差，从而最大限度地降低成本。最优化的输出是那些成本或损失函数值最小的输出。 (后面将详细介绍)。

2、Activation function

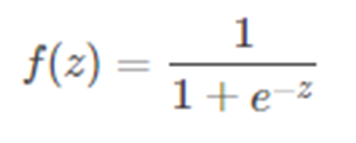
激活函数：对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。如图在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数。引入激活函数是为了增加神经网络模型的非线性。没有激活函数的每层都相当于矩阵相乘。就算你叠加了若干层之后，无非还是个矩阵相乘罢了。



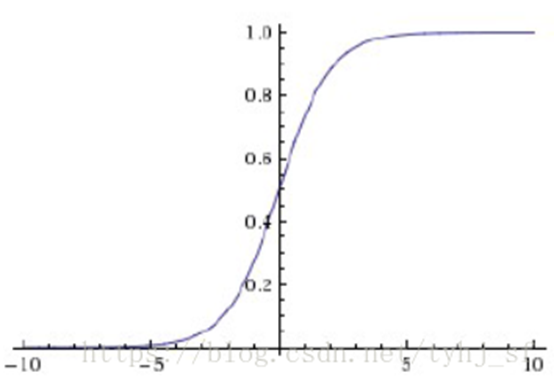
一个没有激活函数的神经网络将只不过是一个线性回归模型（Linear regression Model）罢了，它功率有限，并且大多数情况下执行得并不好。我们希望我们的神经网络不仅仅可以学习和计算线性函数，而且还要比这复杂得多。同样是因为没有激活函数，我们的神经网络将无法学习和模拟其他复杂类型的数据，例如图像、视频、音频、语音等。这就是为什么我们要使用人工神经网络技术，诸如深度学习（Deep learning），来理解一些复杂的事情，一些相互之间具有很多隐藏层的非线性问题，而这也可以帮助我们了解复杂的数据。

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：



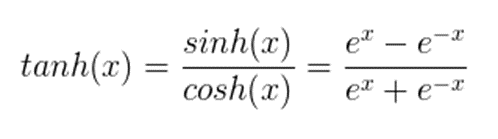
特点：   
 它能够把输入的连续实值变换为0和1之间的输出，特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1.

缺点：

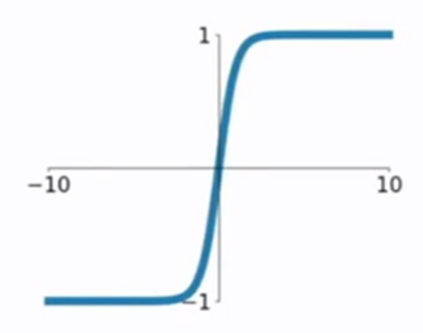
当输入稍微远离了坐标原点，函数的梯度就变得很小了，几乎为零。在神经网络反向传播的过程中，我们都是通过微分的链式法则来计算各个权重w的微分的。当反向传播经过了sigmod函数，这个链条上的微分就很小很小了，况且还可能经过很多个sigmod函数，最后会导致权重w对损失函数几乎没影响，这样不利于权重的优化，这个问题叫做梯度饱和，也可以叫梯度弥散。

Tanh函数

Tanh函数公式和曲线如下：



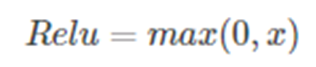
Tanh的几何图像如下：



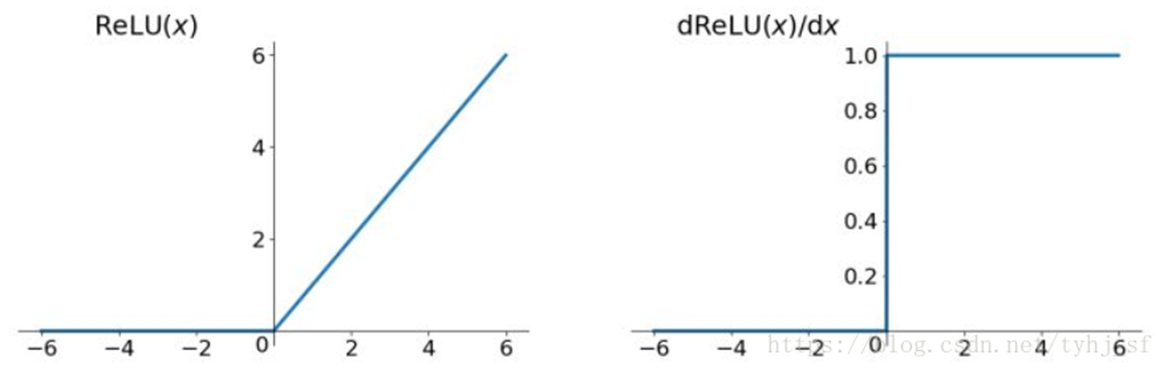
特点：   
 tanh是双曲正切函数，tanh函数和sigmod函数的曲线是比较相近的，咱们来比较一下看看。首先相同的是，这两个函数在输入很大或是很小的时候，输出都几乎平滑，梯度很小，不利于权重更新；不同的是输出区间，tanh的输出区间是在(-1,1)之间，而且整个函数是以0为中心的，这个特点比sigmod的好。

Relu函数

Relu函数的解析式：



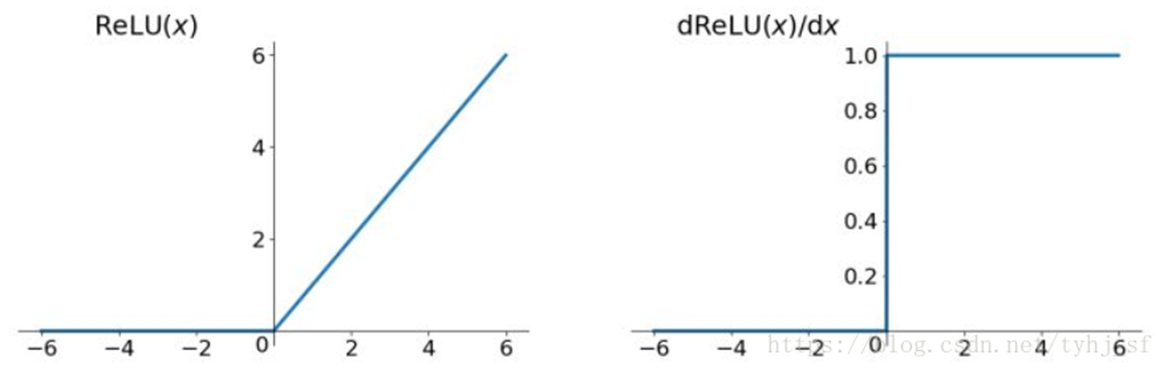
Relu函数及其导数的图像如下图所示：



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。

当然，缺点也是有的：

1. 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



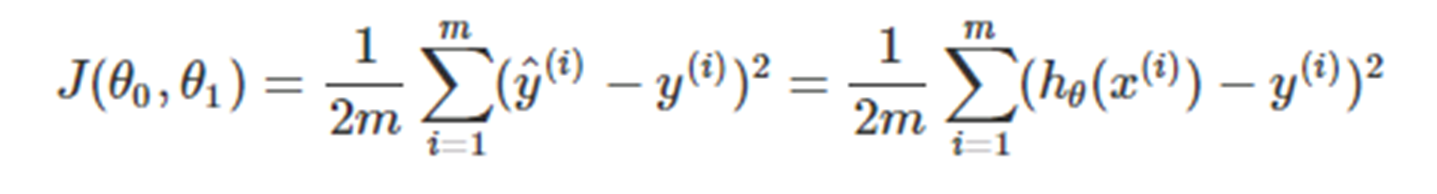
3、Cost function

代价（损失）函数

在数学优化，统计学，计量经济学，决策理论，机器学习和计算神经科学中，代价函数，又叫损失函数或成本函数，它是将一个或多个变量的事件阈值映射到直观地表示与该事件。 一个优化问题试图最小化损失函数。 目标函数是损失函数或其负值，在这种情况下它将被最大化。

均方误差

均方误差：在线性回归中，最常用的是均方误差(Mean squared error)，具体形式为：



m：训练样本的个数；

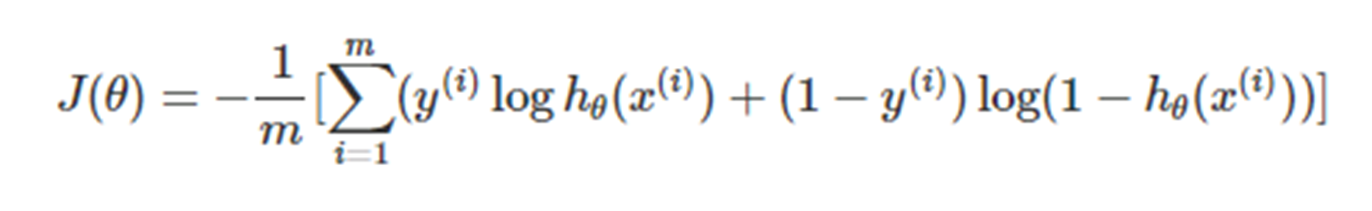
hθ(x)：用参数θ和x预测出来的y值；

y：原训练样本中的y值，也就是标准答案

上角标(i)：第i个样本

交叉熵

交叉熵：在逻辑回归中，最常用的是代价函数是交叉熵(Cross Entropy)，交叉熵是一个常见的代价函数，具体形式为：



m：训练样本的个数；

hθ(x)：用参数θ和x预测出来的y值；

y：原训练样本中的y值，也就是标准答案

上角标(i)：第i个样本

信息量

我们定义事件X=x0X=x0的信息量为： I(x0)=−log(p(x0))I(x0)=−log(p(x0))，可以理解为，一个事件发生的概率越大，则它所携带的信息量就越小，而当p(x0)=1p(x0)=1时，熵将等于0，也就是说该事件的发生不会导致任何信息量的增加。举个例子，小明平时不爱学习，考试经常不及格，而小王是个勤奋学习的好学生，经常得满分，所以我们可以做如下假设：

事件A：小明考试及格，对应的概率P(xA)=0.1，信息量为 I(xA)=−log(0.1)=3.3219

事件B：小王考试及格，对应的概率P(xB)=0.999，信息量为I(xB)=−log(0.999)=0.0014

熵

那么什么又是熵呢？还是通过上边的例子来说明，假设小明的考试结果是一个0-1分布XAXA只有两个取值{0：不及格，1：及格}，在某次考试结果公布前，小明的考试结果有多大的不确定度呢？你肯定会说：十有八九不及格！因为根据先验知识，小明及格的概率仅有0.1,90%的可能都是不及格的。怎么来度量这个不确定度？求期望！不错，我们对所有可能结果带来的额外信息量求取均值（期望），其结果不就能够衡量出小明考试成绩的不确定度了吗。 即

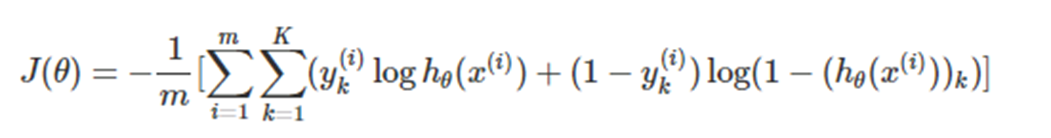
小明的熵： HA(x)=−[p(xA)log(p(xA))+(1−p(xA))log(1−p(xA))]=0.4690 对应小王的熵： HB(x)=−[p(xB)log(p(xB))+(1−p(xB))log(1−p(xB))]=0.0114

熵

我们再假设一个成绩相对普通的学生小东，他及格的概率是P(xC)=0.5,即及格与否的概率是一样的，对应的熵： HC(x)=−[p(xC)log(p(xC))+(1−p(xC))log(1−p(xC))]=1 其熵为1，他的不确定性比前边两位同学要高很多，在成绩公布之前，很难准确猜测出他的考试结果。 可以看出，熵其实是信息量的期望值，它是一个随机变量的确定性的度量。熵越大，变量的取值越不确定，反之就越确定。

神经网络中的代价函数

神经网络中的代价函数与逻辑回归中的代价函数非常相似：

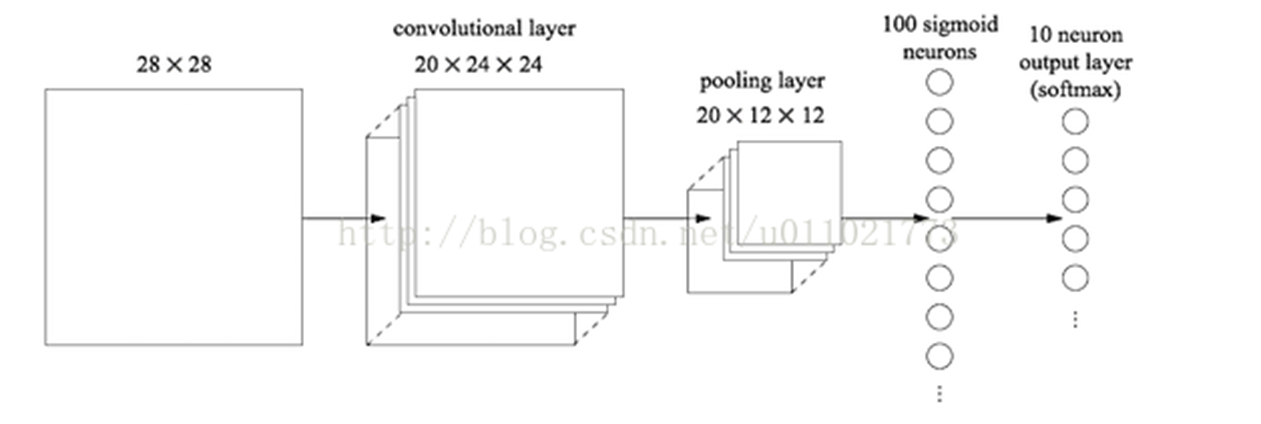


这里之所以多了一层求和项，是因为神经网络的输出一般都不是单一的值，K表示在多分类中的类型数。

4、Fully connected layer

全连接层

在卷积神经网络的最后，往往会出现一两层全连接层，全连接一般会把卷积输出的二维特征图转化成一维的一个向量



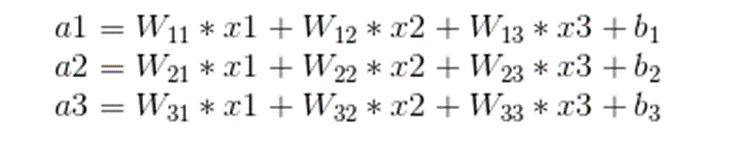
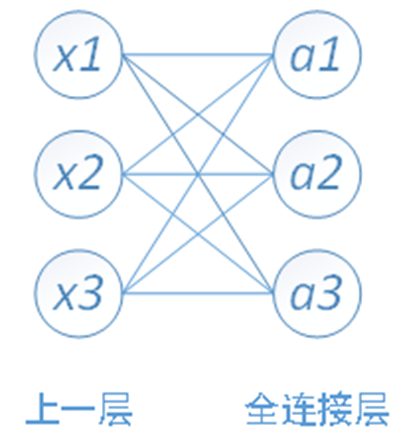
最后的两列小圆球就是两个全连接层，在最后一层卷积结束后，进行了最后一次池化，输出了20个12\*12的图像，然后通过了一个全连接层变成了1\*100的向量。

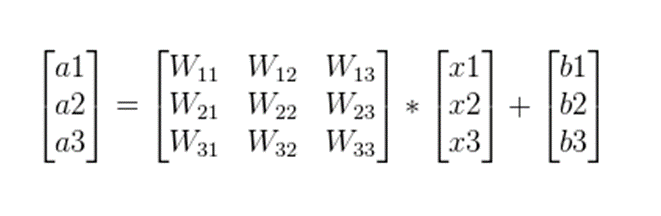
目的: 因为传统的网络我们的输出都是分类，也就是几个类别的概率甚至就是一个数--类别号，那么全连接层就是高度提纯的特征了，方便交给最后的分类器或者回归。

全连接层

全连接层的每一个输出都可以看成前一层的每一个结点乘以一个权重系数W，最后加上一个偏置值b得到

下面用一个简单的网络具体介绍一下推导过程，其中，x1、x2、x3为全连接层的输入，a1、a2、a3为输出，则





程序:

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

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()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import random

import math

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)

main()