# 神经网络

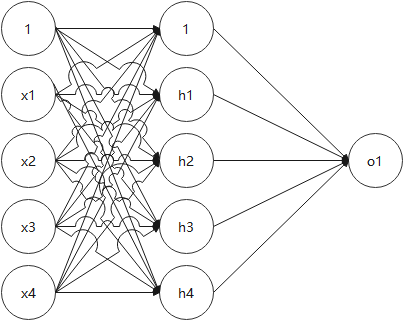
1. 模型概述
2. 概念描述

生物神经网络主要是指人脑的神经网络，它是人工神经网络的技术原型。人脑是人类思维的物质基础，思维的功能定位在大脑皮层，后者含有大约10^11个神经元，每个神经元又通过神经突触与大约103个其它神经元相连，形成一个高度复杂高度灵活的动态网络。作为一门学科，生物神经网络主要研究人脑神经网络的结构、功能及其工作机制，意在探索人脑思维和智能活动的规律。

人工神经网络是生物神经网络在某种简化意义下的技术复现，作为一门学科，它的主要任务是根据生物神经网络的原理和实际应用的需要建造实用的人工神经网络模型，设计相应的学习算法，模拟人脑的某种智能活动，然后在技术上实现出来用以解决实际问题。因此，生物神经网络主要研究智能的机理；人工神经网络主要研究智能机理的实现，两者相辅相成。

1. 公式描述

网络图示:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **数据集** | x= | y= |  | |  |
| **激活函数** |  | | | | |
| **损失函数** | L()= | | | | |
| **前向传播** |  | | |  | |
| **反向传播** | = -  = - | | =(输出层->隐层)  = | | |
| **反向传播导数详解** | =- = =  *=*===  *=*=\* =- \*  可得到链式求导法则 | | | | |

1. 重要代码详解

激活函数和激活函数的梯度：分别返回

def sigmoid(z):

    return 1/(1+np.exp(-z))

def grad\_sigmoid(z):

    f=sigmoid(z)

    return f\*(1-f)

损失函数：返回L()=

def loss(y\_true,y\_pred):

    m = y\_pred.shape[0]

    logprobs = np.multiply(np.log(y\_pred), y\_true) + np.multiply((1 - y\_true), np.log(1 - y\_pred))

    cost = - np.sum(logprobs) / m

return cost

前向传播：利用矩阵减少运算，最后返回即

def forward\_propagation(x,theta0,theta1):

    h=sigmoid(x\*theta0.T)

    h=np.insert(h,obj=1,values=0,axis=1)

    o=sigmoid(h\*theta1.T)

return o

反向传播：利用矩阵减少运算量，一层一层向下运算随后利用梯度下降法来不断更新参数，达到最小化损失函数的目的。

def back\_propagation(x,y,theta0,theta1,alpha,count):

    l=np.zeros(count)

    for i in range(count):

        for j in range(x.shape[0]):

            h0=1

            sum\_h1=float(x[j,:]\*theta0[0,:].T)

            sum\_h2=float(x[j,:]\*theta0[1,:].T)

            sum\_h3=float(x[j,:]\*theta0[2,:].T)

            sum\_h4=float(x[j,:]\*theta0[3,:].T)

            h1=sigmoid(sum\_h1)

            h2=sigmoid(sum\_h2)

            h3=sigmoid(sum\_h3)

            h4=sigmoid(sum\_h4)

            h=np.matrix([h0,h1,h2,h3,h4])

            sum\_h=np.matrix([h0,sum\_h1,sum\_h2,sum\_h3,sum\_h4])

            sum\_o1=float(h\*theta1.T)

            o1=sigmoid(float(sum\_o1))

            y\_pred=o1

            y\_true=y[j]

            delta0=y\_pred-y\_true

            delta1=delta0\*grad\_sigmoid(sum\_o1)

            delta2=float(delta1)\*np.multiply(theta1,grad\_sigmoid(sum\_h))

            theta1-=alpha\*delta1\*h

            theta0-=alpha\*np.multiply(x[j,:],delta2)

        y\_pred=forward\_propagation(x,theta0,theta1)

        l[i]=loss(y,y\_pred)

        print("第 %d 次的loss: %.5f"%(i,l[i]))

    return theta0,theta1,l

测试正确率：计算正确率为多少

def prediction(predict,y\_test):

    correct=0

    for i in range(len(predict)):

        if predict[i] == y\_test[i]:

            correct+=1

    return correct/len(y\_test)

三.结果分析

这里我只用了iris数据集中的前两类实现了二分类，每隔四个数据点取一个测试点，利用数据点得到最优参数为

=

[[ 1.45419275 2.81220621 -4.95160161 0.77928625 0.13940537]

[ 0.72830206 2.26845322 -5.07619814 0.69262857 0.85602132]

[ 0.81225712 2.5908466 -5.31967507 0.53653026 0.24317812]

[ 0.81220038 2.69488945 -4.69820539 1.24495679 0.58982374]]

=

[[-2.30632358 -0.87758612 3.94877995 3.02629543 -0.85696307]]

利用测试数据集测试 正确率为100% 可以看出该网络正确率很高。

我这里使用了一个隐藏层，隐藏层中有四个隐藏神经元（不包括偏置项）对于这样偏少的数据量有点复杂了，导致训练速度很慢。

四.源码

from sklearn.datasets import load\_iris

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

iris=load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

data['target'] = iris.target

data=data.iloc[:100,:]

#这里我进行二分类，只将0,1拿出来进行分类

#四个输入神经元 一个隐藏层 四个隐藏神经元 （这里不包括偏置项，实际计算包括偏置项） 一个输出神经元

#设置两个theta  每一个表示一层的权重  一个theta有四行五列 一个theta有一行五列 每一行表示后一个神经元的输入权重(随机初始化)

theta0=np.matrix(np.random.random((4,5)))

theta1=np.matrix(np.random.random((1,5)))

train=[]

test=[]

num=1

for item in np.array(data):

    if num==5:

        test.append(item)

        num=1

    else:

        train.append(item)

        num+=1

train=pd.DataFrame(train)

test=pd.DataFrame(test)

train.insert(0,'Bias',1)

test.insert(0,'Bias',1)

cols = train.shape[1]

x\_train=np.matrix(train.iloc[:,0:cols-1])

y\_train=np.matrix(train.iloc[:,cols-1:cols])

cols = test.shape[1]

x\_test=np.matrix(test.iloc[:,0:cols-1])

y\_test=np.matrix(test.iloc[:,cols-1:cols])

def sigmoid(z):

    return 1/(1+np.exp(-z))

def grad\_sigmoid(z):

    f=sigmoid(z)

    return np.multiply(f,(1-f))

def loss(y\_true,y\_pred):

    m = y\_pred.shape[0]

    logprobs = np.multiply(np.log(y\_pred), y\_true) + np.multiply((1 - y\_true), np.log(1 - y\_pred))

    cost = - np.sum(logprobs) / m

    return cost

def forward\_propagation(x,theta0,theta1):

    h=sigmoid(x\*theta0.T)

    h=np.insert(h,obj=1,values=0,axis=1)

    o=sigmoid(h\*theta1.T)

    return o

def back\_propagation(x,y,theta0,theta1,alpha,count):

    l=np.zeros(count)

    for i in range(count):

        for j in range(x.shape[0]):

            h0=1

            sum\_h1=float(x[j,:]\*theta0[0,:].T)

            sum\_h2=float(x[j,:]\*theta0[1,:].T)

            sum\_h3=float(x[j,:]\*theta0[2,:].T)

            sum\_h4=float(x[j,:]\*theta0[3,:].T)

            h1=sigmoid(sum\_h1)

            h2=sigmoid(sum\_h2)

            h3=sigmoid(sum\_h3)

            h4=sigmoid(sum\_h4)

            h=np.matrix([h0,h1,h2,h3,h4])

            sum\_h=np.matrix([h0,sum\_h1,sum\_h2,sum\_h3,sum\_h4])

            sum\_o1=float(h\*theta1.T)

            o1=sigmoid(float(sum\_o1))

            y\_pred=o1

            y\_true=y[j]

            delta0=y\_pred-y\_true

            delta1=delta0\*grad\_sigmoid(sum\_o1)

            delta2=float(delta1)\*np.multiply(theta1,grad\_sigmoid(sum\_h))

            theta1-=alpha\*delta1\*h

            theta0-=alpha\*np.multiply(x[j,:],delta2)

        y\_pred=forward\_propagation(x,theta0,theta1)

        l[i]=loss(y,y\_pred)

        print("第 %d 次的loss: %.5f"%(i,l[i]))

    return theta0,theta1,l

alpha=0.05

count=300

theta0\_bcak,theta1\_back,loss\_history=back\_propagation(x\_train,y\_train,theta0,theta1,alpha,count)

pred=forward\_propagation(x\_test,theta0\_bcak,theta1\_back)

for i in range(len(pred)):

    if pred[i]>=0.5:

        pred[i]=1

    else:

        pred[i]=0

def prediction(predict,y\_test):

    correct=0

    for i in range(len(predict)):

        if predict[i] == y\_test[i]:

            correct+=1

    return correct/len(y\_test)

pred\_rate=prediction(pred,y\_test)

print("鸢尾花前两种数据正确率:%.2f"%pred\_rate)

print(theta0\_bcak,theta1\_back)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(np.arange(count), loss\_history, 'r')

ax.set\_xlabel('count')

ax.set\_ylabel('loss')

ax.set\_title('loss\_graph')

plt.show()