实验三 BP神经网络非线性分类实验

姓名：唐川淇 学号：1131190111

# 问题重述

给定的数据分为两类（前50个为一类，后50个为一类），编程构建BP神经网络，以80%的数据为训练数据，20%的数据为测试数据，验证分类结果。并分析：

1. 选择不同的激励函数时，分类效果比较
2. 选择不同的梯度学习算法，分类效果比较

# 问题求解

首先加载nump和sklearn库中的MLPClassifier：

from sklearn.neural\_network import MLPClassifier  
import numpy as np

从文本中读取数据：

data = np.loadtxt('data.txt')

选取第11到90行为测试数据，1到10行、91到100行为测试数据，并为数据添加标签：

X = data[10:90, :]

Y = np.zeros(80) # 添加标签  
for i in range(80):  
 if i < 40:  
 Y[i] = -1  
 else:  
 Y[i] = 1

建立神经网络，激活函数选为，训练的方法选用拟牛顿法。神经网络包含两个隐含层，每层10个节点，建立网络。根据导入的数据训练网络：

clf = MLPClassifier(solver='lbfgs', activation='identity', alpha=1e-5, hidden\_layer\_sizes=(10, 10), random\_state=1)  
clf.fit(X, Y)

利用训练好的网络对验证数据集的数据进行预测：

for i in data[0:10, :]:  
 ans = clf.predict([i])  
 print(ans)  
for i in data[90:100, :]:  
 ans = clf.predict([i])  
 print(ans)

得到结果：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据 | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 类别 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 数据 | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** | **100** |
| 类别 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

发现，训练的神经网络在对数据进行分类时，将1-10分到了第一类，91-100分到了第二类，得到的结果和真实情况完全相同，准确率达到100%

## 不同的激励函数分类效果比较

在神经网络中，隐层和输出层节点的输入和输出之间具有函数关系，这个函数称为激励函数。使用恰当的激励函数可以更逼近目标，因此需要对不同的激励函数进行比较分析。

### identity

线性激活函数，表达式为：



### logistic

Sigmoid函数，表达式为：



它能够把输入的连续实值变换为0和1之间的输出，但是在深度神经网络中梯度反向传递时导致梯度爆炸和梯度消失。

### tanh

tanh函数，表达式为：



tanh也是一种非常常见的激活函数，与sigmoid相比，它的输出均值为0，这使得它的收敛速度要比sigmoid快，减少了迭代更新的次数。

然而tanh和sigmoid一样具有饱和性，会造成梯度消失。

### ReLU

ReLU函数，表达式为：



ReLU是针对sigmoid和tanh的饱和性二提出的新的激活函数。

### 四种激励函数比较

首先使用包含两个隐含层，每层节点数为10的神经网络，选用不同的激励函数训练网络。得到对测试集的准确率：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 激励函数 | identity | Sigmoid | tanh | ReLU |
| 准确率 | 100% | 100% | 100% | 100% |

发现使用每一个激励函数的准确率都达到了100%，为了区分不同的激励函数，将隐含层节点分别降低为7和7，训练网络，得到对测试集的准确率：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 激励函数 | identity | Sigmoid | tanh | ReLU |
| 准确率 | 100% | 100% | 85% | 50% |

发现在使用ReLU为激励函数时，神经网络将所有的样本都分为了一类，最终的准确率也只有50%，效果非常不好。使用tanh为激活函数时，有3个样本被错误的划分了类别，准确率只有85%。而使用线性激励函数和Sigmoid函数的准确率都达到了100%，为了区分这两种激励函数，继续减少隐含层节点数至5和5，训练网络，得到对测试集的准确率：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 激励函数 | identity | Sigmoid | tanh | ReLU |
| 准确率 | 100% | 100% | 100% | 50% |

发现使用tanh为激励函数的神经网络的准确率又达到了100%，虽然节点数目变少了但是准确率增大了。考虑可能是由于这些原因导致的：（1）样本比较少，由于只有20个测试样本，结果可能具有一定的偶然性。（2）并不是更多的节点就能更逼近目标函数，只有使用合适数量的节点才能准确的预测结果。为了再次区分前三个激励函数的优劣，再次调整节点数目为3，3两种激励函数仍未100%，而tanh与ReLU都为50%。再次调整节点数目为2，1，发现前两种激励函数仍然同为100%，说明在该问题下，两种激励函数具有同样好的效果。

## 不同的学习算法分类效果比较

优化算法就是一种能够帮我们最小化或者最大化目标函数的一类算法。

### lbfgs

拟牛顿法，拟牛顿法是求解非线性优化问题最有效的方法之一。

### sgd

随机并行梯度下降算法，简称SPGD算法。作为一种无模型优化算法，比较适用于控制变量较多，受控系统比较复杂，无法建立准确数学模型的最优化控制过程。

### adam

交叉熵梯度优化。在分类算法中，经常会用到交叉熵损失函数。表达形式如下：

loss = -ylogx -(1-y)log(1-x)

特性如下：

1）当预估的值，接近0或1，且与样本一致时，loss和0比较接近

2）借助sigmod函数的导数特性，交叉熵的损失函数对于w的梯度可以推出，所以，loss越大，梯度就越大。loss和梯度成正比。

### 三种学习算法比较

首先使用包含两个隐含层，每层节点数为10的神经网络，选用不同的学习算法。得到对测试集的准确率：

|  |  |  |  |
| --- | --- | --- | --- |
| 学习算法 | lbfgs | sgd | adam |
| 准确率 | 100% | 50% | 100% |

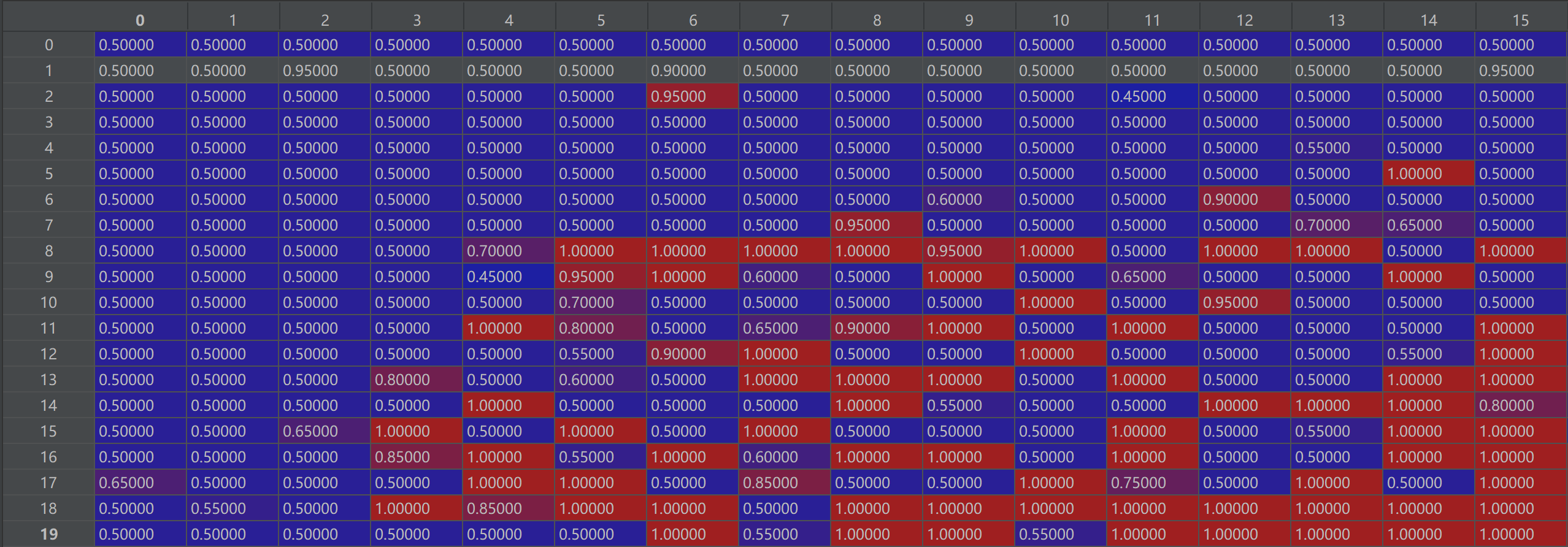
注意到使用随机梯度下降的方法的准确率只有50%，效果很差。而拟牛顿法和交叉熵优化方法的准确率都达到了100%。调整神经网络节点数为2，2，得到如下结果：

|  |  |  |  |
| --- | --- | --- | --- |
| 学习算法 | lbfgs | sgd | adam |
| 准确率 | 50% | 50% | 100% |

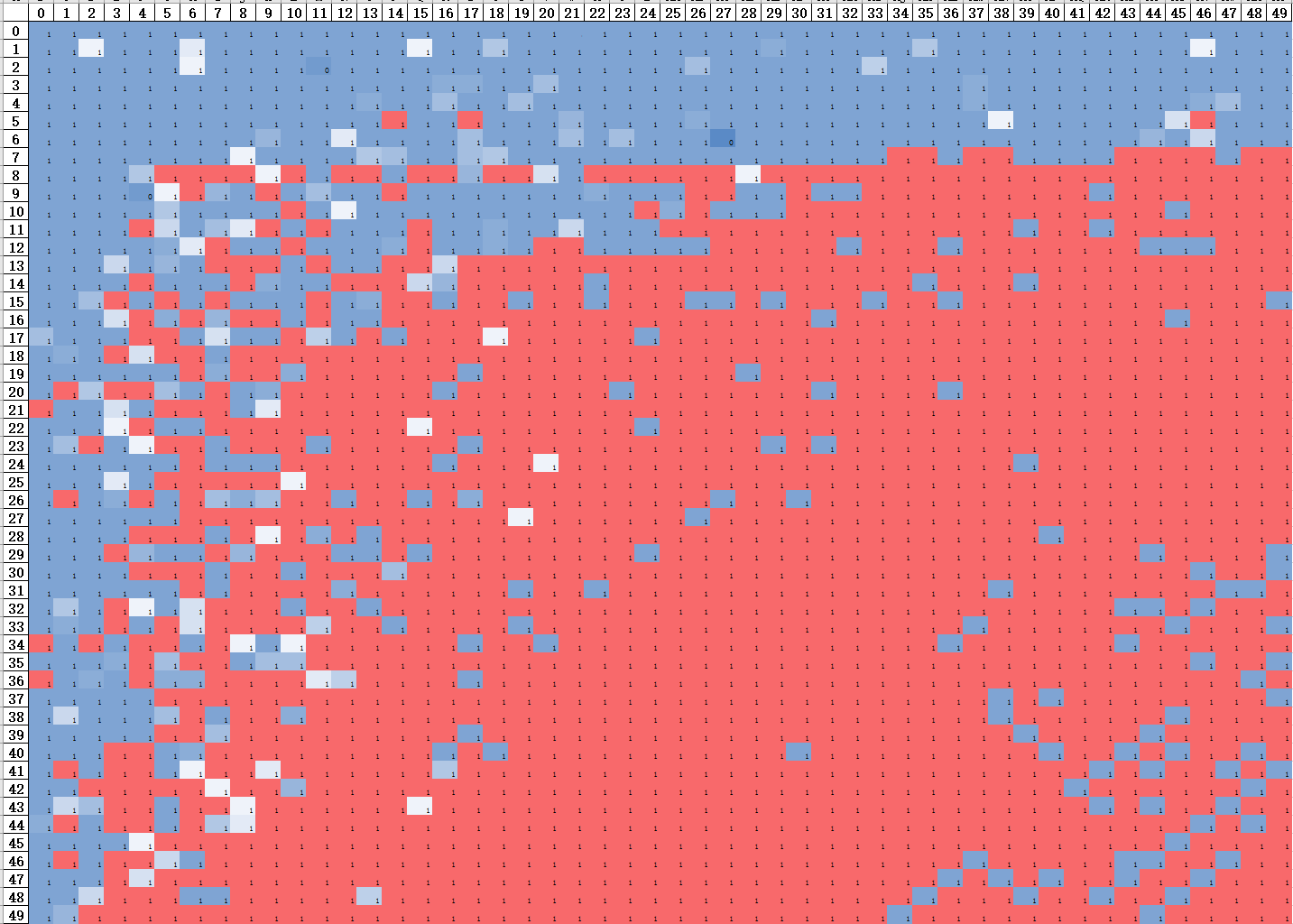
发现在该问题下，使用交叉熵优化方法有较好的结果。

## 其他影响因素讨论

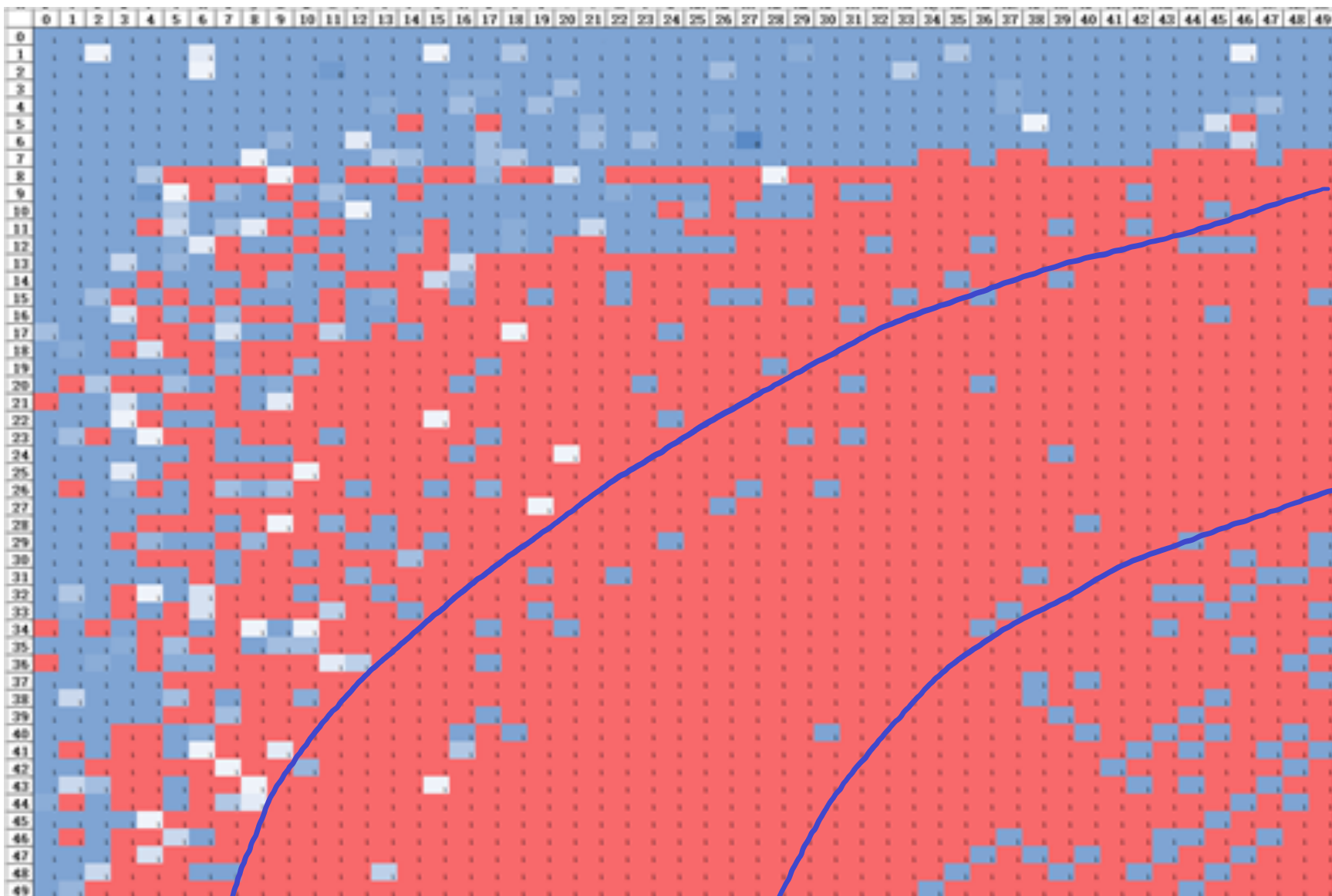
在实验中发现，准确率和神经网络节点数没有明显的线性关系，因此考虑设置不同数量的节点，研究神经网络的准确率。在实验中统一使用Sigmoid+交叉熵优化的方法。第一层节点数分别设置为1~20，第二层节点数设置为1~16，计算出不同节点数搭配的误差：



在图中，纵坐标为第一层结点的数量，横坐标为第二层节点的数量，表格中的数值代表准确度。准确度越高方格约偏向于红色。注意到，随着节点数量设置的增多，神经网络的准确率也逐渐增加，但是其中变化的趋势并没有完全呈现出线性的关系。考虑更多的节点训练网络，得到如下的准确率图：



在图中可以发现大致形成一条带状，在带中的准确率较高：



因此选择带中的节点具有较高的准确率。

|  |
| --- |
| 代码 |
| from sklearn.neural\_network import MLPClassifier  import numpy as np  import pandas as pd  data = np.loadtxt('data.txt')  X = data[10:90, :] # 获取其中11-91行作为训练数据  Y = np.zeros(80) # 添加标签  for i in range(80):  if i < 40:  Y[i] = -1  else:  Y[i] = 1  aa = 80  bb = 80  Result\_s = np.zeros((aa, bb))  for ii in range(aa):  for jj in range(bb):  clf = MLPClassifier(solver='adam', activation='logistic', alpha=1e-5, hidden\_layer\_sizes=(ii+1, jj+1), random\_state=1)  clf.fit(X, Y)  ActAns = np.zeros(20);  for i in range(20):  ActAns[i] = 1  if i < 10:  ActAns[i] = -1  datatest1 = data[0:10, :]  datatest2 = data[90:100, :]  data\_t=np.r\_[datatest1, datatest2]  ans = clf.predict(data\_t)  num=0  for i in range(20):  if ActAns[i] == ans[i]:  num = num+1  Result\_s[ii,jj]=num/20;  w\_in = pd.DataFrame(Result\_s)  writer = pd.ExcelWriter('A.xlsx') # 写入Excel文件  w\_in.to\_excel(writer, 'page\_1', float\_format='%.5f')  writer.save()  writer.close() |