bp神经网络文档

一、代码基本架构

1、代码包及文件结构

bp 代码包
|-network.py 用于拟合sin(x)的bp神经网络
|-sin_fitting.py 对sin(x)进行拟合的程序
|-network_img.py 用于识别汉字图片的bp神经网络
|-img_recog.py 进行汉子图片识别的程序
|-img_para_script.py 测试参数脚本
|-img_para_script_2.py 测试参数脚本
|-sin_para_script.py 测试参数脚本
|-sin_para_script_2.py 测试参数脚本

2、神经网络代码架构(network.py)

该源代码文件中定义构建Network类

class Network
attribute: -learning_rate 学习率 -hidden_layer_num 隐含层数量 -inputs 输入向量 -expec_outputs 期待输出向量 -outputs 实际输出向量 -outputs_bias 输出层偏置 -outputs_weight 输出层权重 -input_layers 隐含层的输入值 -output_layers 隐含层的输出值 -biases 隐含层的偏置 -weights 隐含层的权重 -thf_deris 隐含层激活函数的导数值 -out_deris 隐含层输出值的导数值 -weight_deris 隐含层权重的导数值 -thf_out_products 隐含层激活函数导数值与输出值导数值的乘积，方便后续计算 -bias_deris 隐含层偏置的导数值 -outputs_deri 输出层的导数值 -outputs_weight_deri 输出层权重的导数值 -outputs_bias_deri 输出层偏置的导数值

class Network
method: -__forward() 正向计算 -__back() 反向传播并更新权重及偏置 -__thres_fun(inputs) 激活函数 -__thres_fun_deri(inputs) 激活函数导数 -__sigmiodal(inputs) sigmiodal函数 -__sigmiodal_deri(inputs) sigmiodal函数导数 -__linear_deri(inputs) 线性函数导数 -__linear(inputs) 线性函数 -__loss() 损失函数 -train(inputs, expec_outputs, learning_rate) 根据输入值，期待输出值，学习率对神经网络进行训练 -compute(inputs) 根据输入值计算输出结果

补充说明：

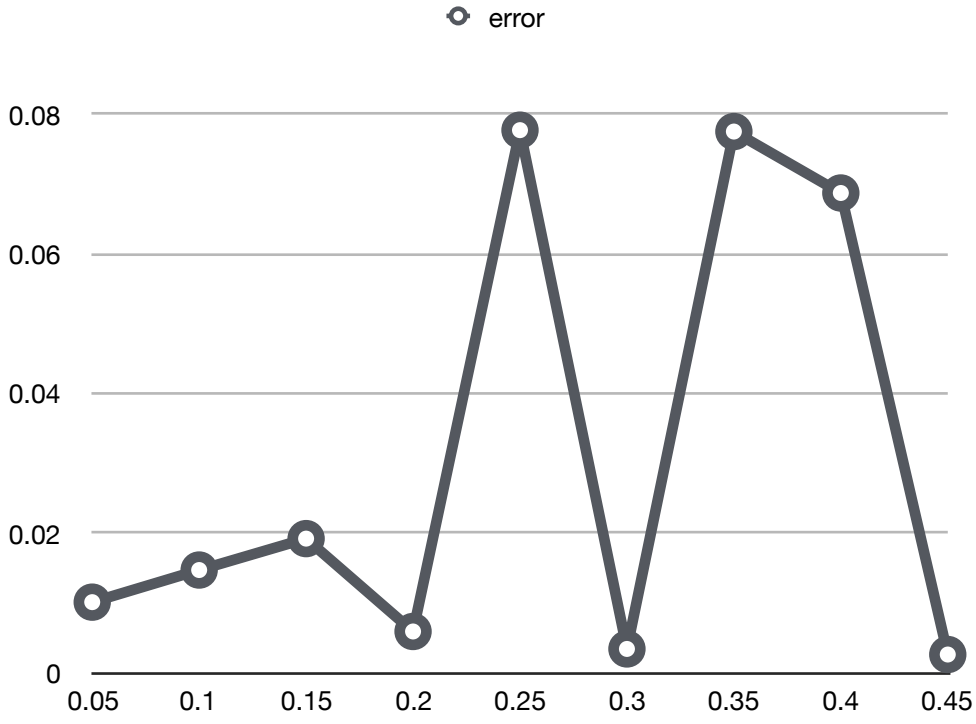
（1）Network中的正向与反向计算均调用numpy库中的矩阵运算实现

（2）神经网络中权重值及其导数值存储在二维矩阵中，其余数据如输入输出值等存储在一维向量中

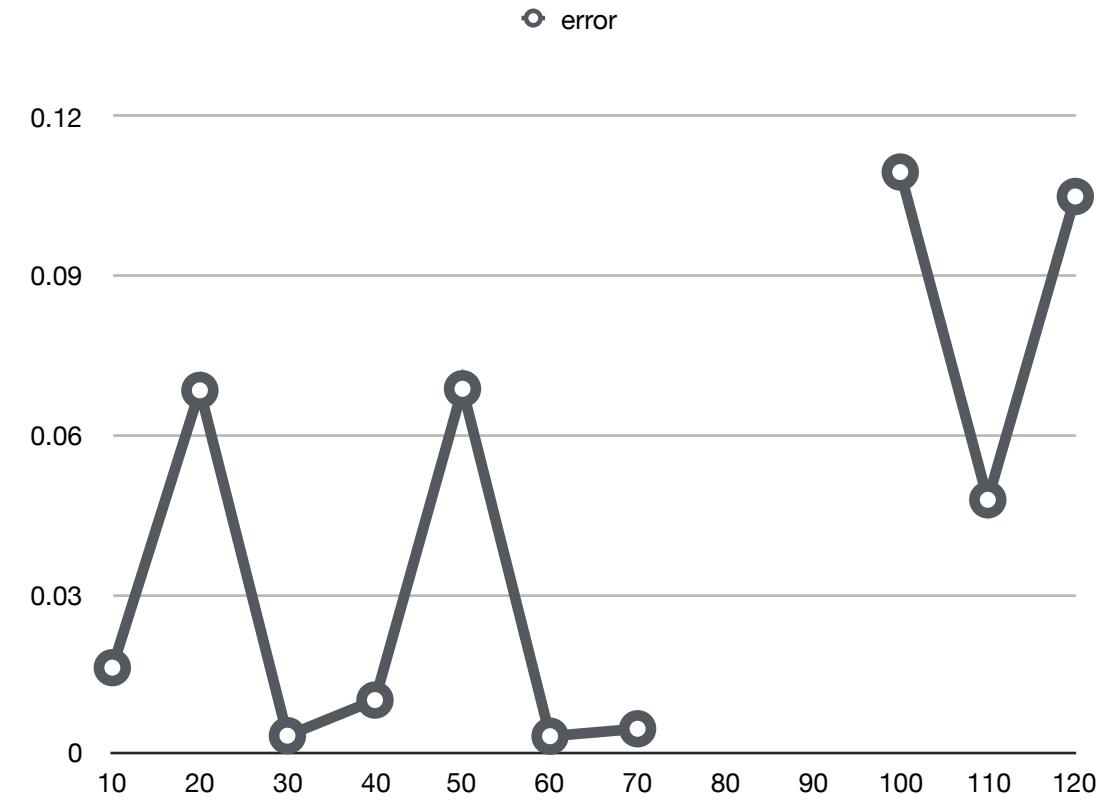
（3）network_img.py为用于识别汉字图片的bp神经网络，整体架构与network.py基本一致，差别在于最后输出层，network_img.py在最后的输出层上增加了sigmiodal函数最为最后输出(反向传播时新增了对新增函数的处理)，将输出值限定在(0,1)范围内

二、不同网络结构及参数比较

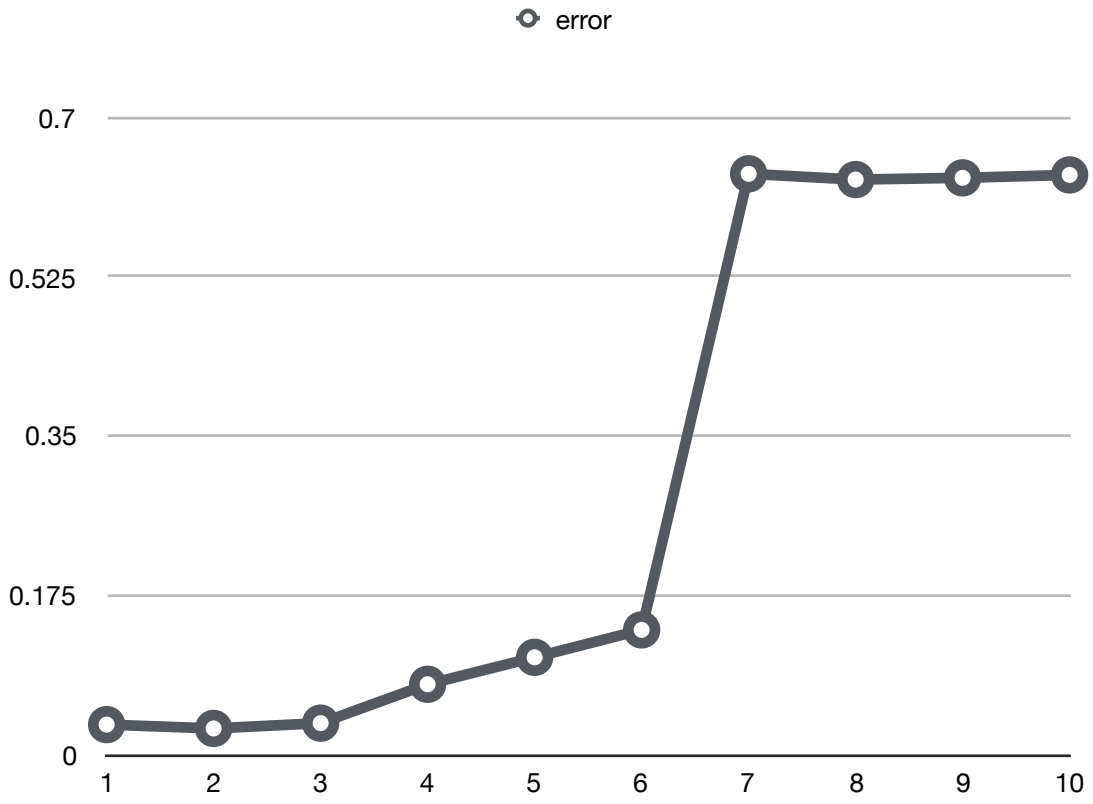
1、拟合sin(x)



单层隐含层神经元数量30，学习率对拟合误差的影响

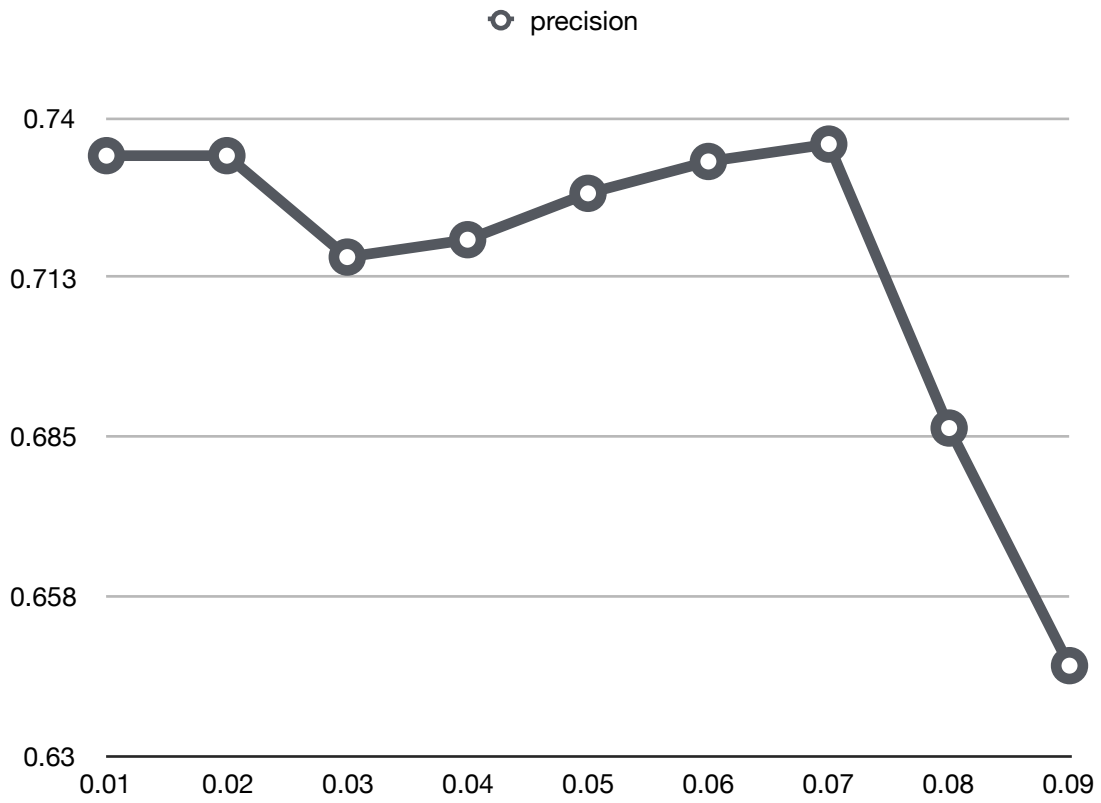


学习率0.3，单隐含层神经元数量对拟合误差的影响(缺点处为训练失败)

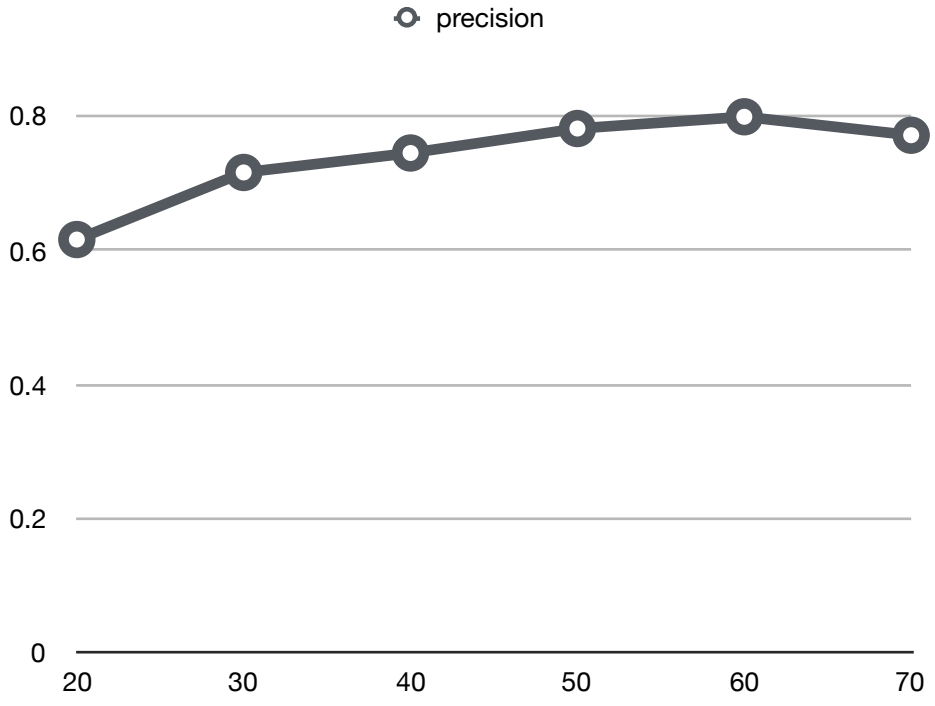


学习率0.05，每层10个神经元，隐含层数量对拟合误差的影响

2、图片汉字识别



单层隐含层神经元数量30，学习率对识别率的影响



学习率0.03，单层隐含层神经元数量对识别率的影响

3、参数影响总结

（1）总体来讲，各个参数与训练理想结果并**非线性关系**，而是对于**不同的训练集存在不同的最优值。**

（2）当**神经网络结构一定时，学习率并非越小越好，而是存在局部的最优值。**比如上述实验中对sin(x)的拟合，当学习率为0.45时可以取到最小误差，图像分类中则是0.03为最优值。通过实验发现，对于简单的网络结构，学习率的最优值一般稍大一些，而对于复杂的网络结构，学习率则应适当取小一些，否则神经网络在训练过程中很有可能过度发散导致最终训练不出来。

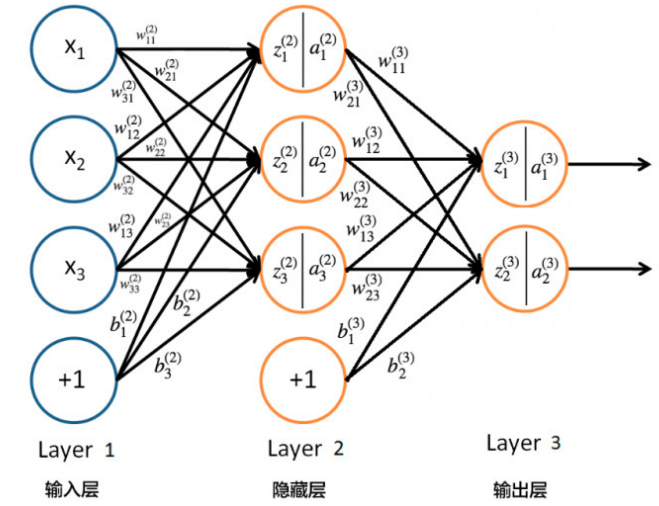
（3）当**学习率一定时，单层的神经网络的神经元数量并非越多越好，而是存在局部的最优值。**上述实验中，对于sin(x)的拟合在单层30或60个神经元时取得较好的训练结果，对于图像分类则是在单层60个神经元时取得了较好的训练结果。通过实验发现，对于单层神经元数量的选择应该遵循一个从简原则，即选择能够符合当前训练集要求的最少数量的神经元。因为神经元数量过少时，拟合能力不够，无法较好的完成训练任务，而当神经元数量过多时，则较容易出现过拟合的现象，导致训练结果变差。

（4）**对神经网络层数，应该首选单层隐含层的神经网络。**从上述实验中看到，对于sin(x)的拟合误差随层数增加而增加。究其原因应该在于训练任务较简单（每次训练的取样点均相同），而隐含层数量增加时容易出现过拟合。另外，对于相同的训练任务多层神经网络会增加计算资源的开销，并不划算。（再者，在本次图像分类中通过实践发现，多层网络的识别准确率跟随便蒙一个的准确率差不多，和傻子一样。）

（5）对于**初始权重的选择尽量选择一个较小的随机数。**考虑过为权重赋一个既定初值，但考虑到训练集的随机性，所以随机权重初值更好一些。初始权重值过大则容易导致网络越训练越发散，觉得原因在于反向传播算法的调整能力有限，而如果初始权重过大超过算法调整能力则不好训练。

三、反向传播算法

1、个人认为，反向传播算法通俗来说就是一种**监督学习**，相当于有一个老师站在旁边先给小朋友一道题让你自己算，想怎么算就怎么算，然后算出来结果之后，老师根据结果告诉小朋友这题算的哪里有问题让小朋友改正，改正之后再算，老师再根据算出来的结果再教，小朋友再改，直到最后小朋友把题目算对或者接近争取为止



2、严谨一些的解释，反向传播算法通过构建一个包含输入层，隐含层，输出层的神经网络，经过多次训练构建起一种我们想要的**训练集与期待输出之间的映射。**反向传播算法包括**信号的正向传播，与误差的反向传播。**训练的过程首先是根据输入进行正向的计

算，计算后得到实际输出，将实际输出与预计输出进行对比，再由误差反向的对网络的权重和偏置进行更新，更新的原理是对误差进行求导，使得更新后的网络走向误差变小的方向。

四、实验过程中遇到的问题

1、bug！！bug是这次实验中最大的问题，不同于以往代码中的bug，这次实验中出现的bug大多都是等级和难度更好的玄学bug。解决措施就是，流着泪也要把自己写的bug都de完。

2、参数调整问题，在本次实验当中，寻找较优的参数使得误差在要求范围内也是一个问题。为了尽量多的找到更好的参数，尝试了很多次，后来发现手动测试实在太蠢，于是写了几个脚本，得到了许多参数的训练结果。

3、训练时间过长。对于拟合sin(x)的训练用时很短，就能够得到理想的结果，然而对于图片分类的用时却很长。起初以为是图片分类的输入向量过大拖慢了运行速度，后来发觉这个应该不是主要的原因，再次回顾代码才发现了自己的智障个，原来在每次对一张图片训练的时候我都进行了一次本地图片文件的读取，过多的IO时间拖慢了整体程序的运行。然后对训练部分代码进行重构，将训练集中所有图片读入内存中等待调用，通过牺牲部分内存的办法换取了大量的运行时间。

4、图像分类准确率提升慢。为了降低图像分类训练时间，减少训练次数也是一个重要途径。为了保证训练质量，就要加快识别准确率的提升。为了加快提升，改变了训练方式，由原来的一个字一个字的训练改为一句话一训练，其实就是修改了一下循环方式，从而大大的优化了训练速度。