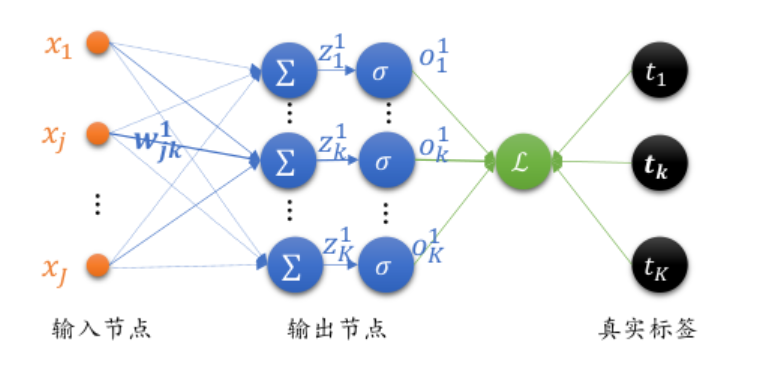
实验报告

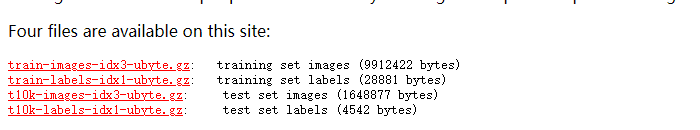
Bpnet:



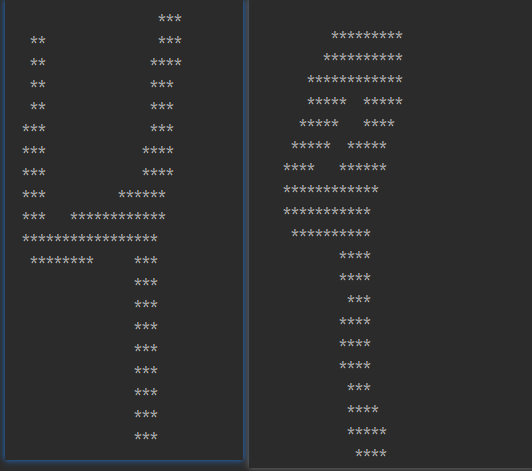
C++ BP网络实现mnist手写字符识别。

使用的数据集<http://yann.lecun.com/exdb/mnist/>

训练集60000张，测试集10000张



调用main.cpp里的printNumber()可以打印数据集中的字符。如图：



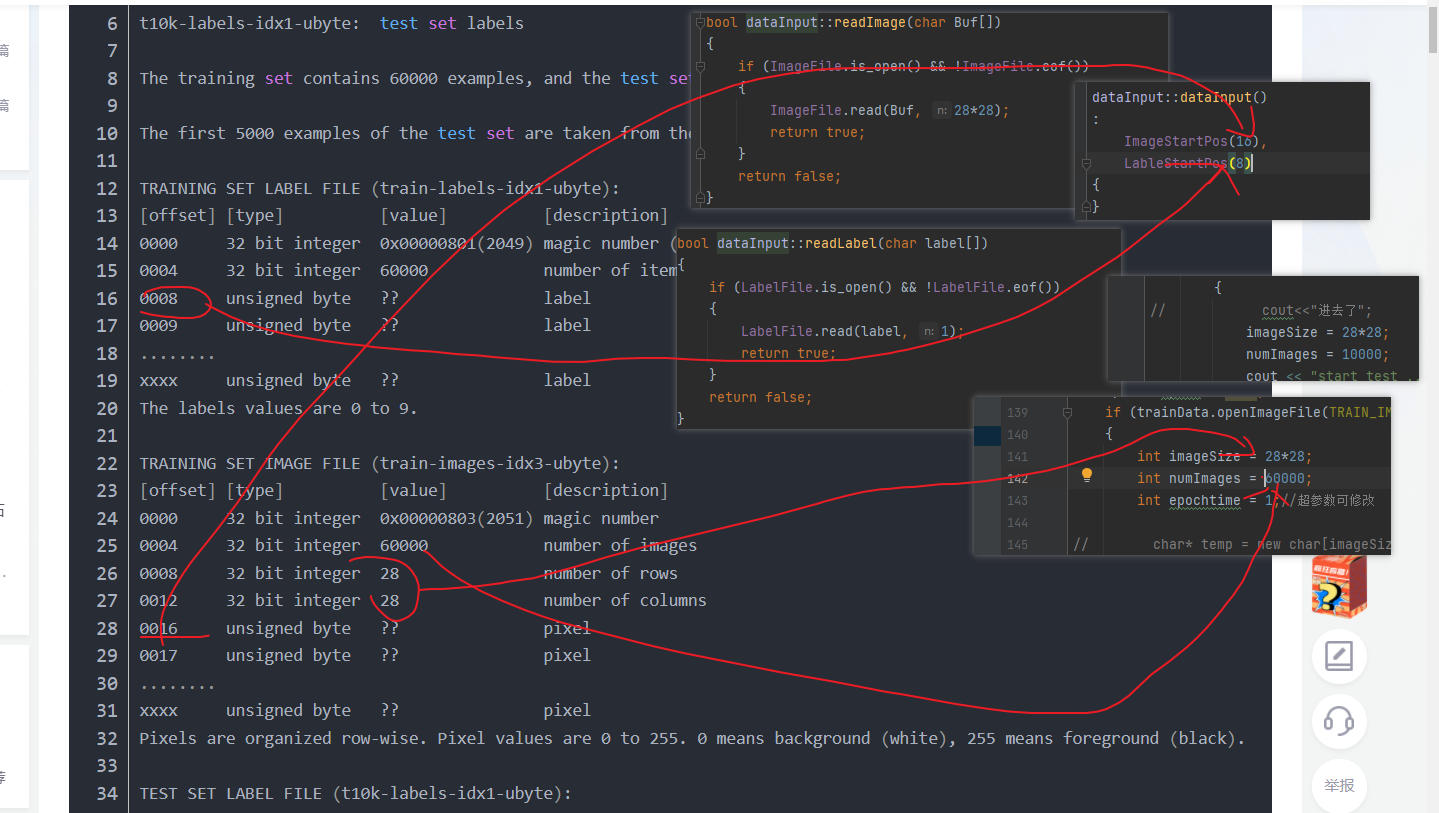
整体结构介绍 ：

class bpNet，位于bpnet.h,网络主体类，包括初始化网络，添加隐藏层，训练，前向反向传播等，有成员vector<Layer\*> layers;

class Layer,位于layer.h,定义了每层的节点数量，weight,bias,以及他们的初始化等。

其中weight二维数组，第一维度为这一层节点数目，第二维度为每个结点的输入数量，对于本网络而言，使用全连接，也即是上一层网络的节点数目。Bias一维数组，为这一层节点数目。

class dataInput,位于data\_input.h,处理数据，其中我根据[(40条消息) MNIST数据集的格式以及读取方式\_DarrenXf的专栏-CSDN博客\_mnist数据集的格式](https://blog.csdn.net/DarrenXf/article/details/85232255) 一文中说的数据集格式，确定了一些固定的参数：



main.cpp:进行网络的构建和训练，测试。

重要函数介绍：

位于bpnet.cpp中。

addLayer：添加隐藏层。

Sigmoid：激活函数。

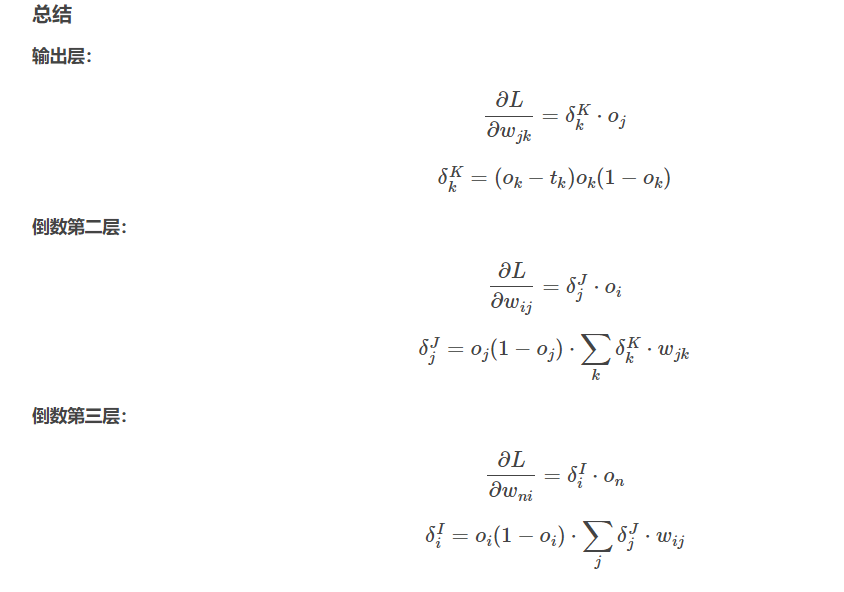
forward\_oneLayer:网络数据向前传播一层。

forward\_all\_layers：递归调用forward\_oneLayer，产生网络输出，并不计算loss,用于测试阶段。

forward: 递归调用forward\_oneLayer，产生网络输出,并计算loss,便于反向传播loss使用简单的相减形式计算。

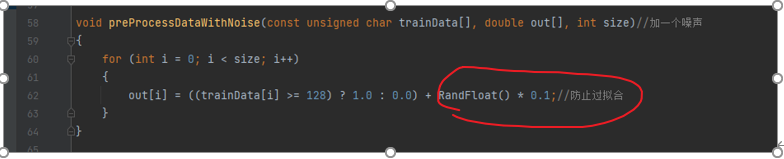


backward\_onelayer：网络误差反向传播一层。其中具体公式的编写参考[神经网络之反向传播算法（BP）公式推导（超详细） - jsfantasy - 博客园 (cnblogs.com)](https://www.cnblogs.com/jsfantasy/p/12177275.html)的末尾：



backward：递归调用backward\_onelayer，实现反向传播完整功能。

训练的时候对输入数据进行了添加noise的操作，



好处如下：

添加噪声意味着网络无法记住训练样本，因为它们一直在变化，从而导致网络权重更小且网络更健壮，泛化误差更低。

噪声意味着从已知样本附近的域中抽取新样本，从而平滑了输入空间的结构。这种平滑可能意味着映射功能更易于网络学习，从而导致更好，更快的学习。

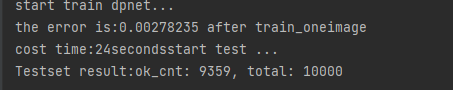
超参数：NET\_LEARNING\_RATE 学习率；epochtime ，epoch次数；NUM\_NODES\_PER\_LAYER，隐藏层节点的数量；隐藏层层数，num\_hiddenlayers ,通过bpnet->addLayer(NUM\_HIDDEN);添加；

Sigmod的SIGMOID\_PARAMETER参数等。

Default config:

NET\_LEARNING\_RATE:0.4; epochtime 1; NUM\_NODES\_PER\_LAYER:100; num\_hiddenlayers 1; SIGMOID\_PARAMETER: 0.7

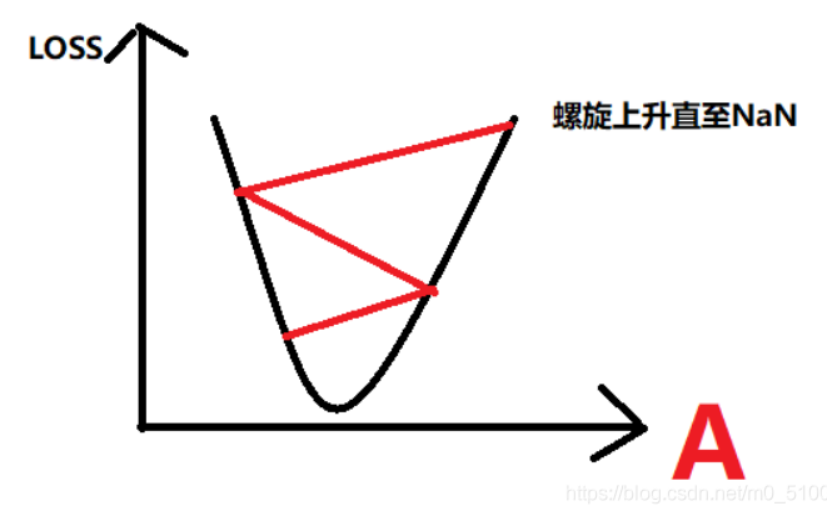
Result:



可见精度不错。

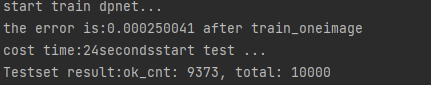
在此基础上，尝试通过修改超参数并用python画图：

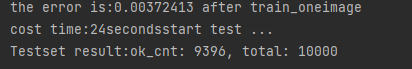
学习率的影响：太大，loss跳跃：

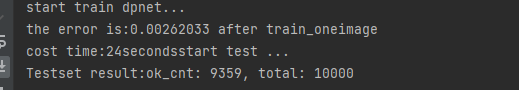


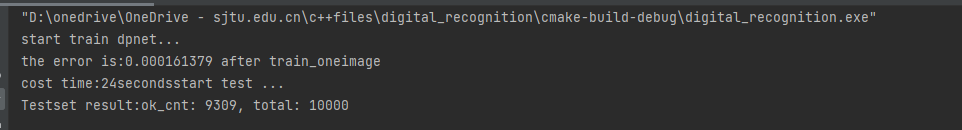
太小，学的太慢。

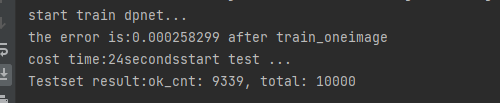
修改NET\_LEARNING\_RATE：

0.5 

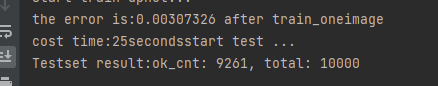
0.6 

0.7 

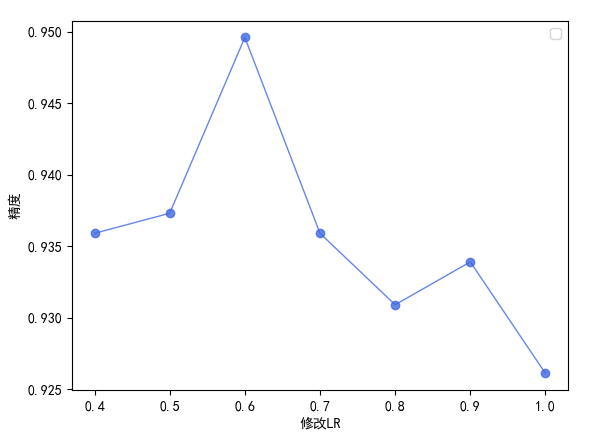
0.8 

0.9 

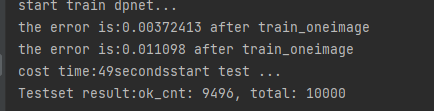
1



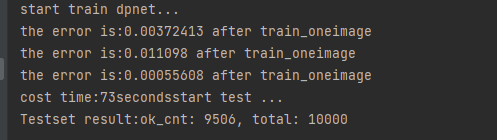
因此对于1层的结构，lr 0.6最好，但是影响很小



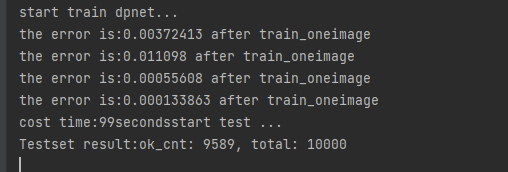
在1层，lr=0.6基础上增大epochtime，为2：



为3：

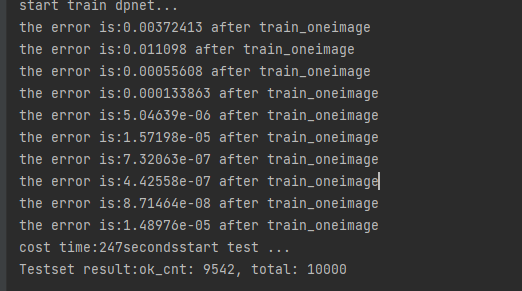


为4：

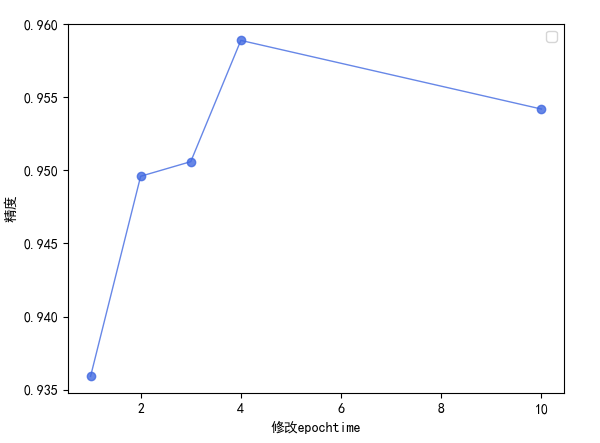


可见，迭代次数增加，进度上升， epoch1后没有学好，增加次数又可以继续学习，并且猜测由于都是手写数字集，过拟合对结果的影响很小导致。

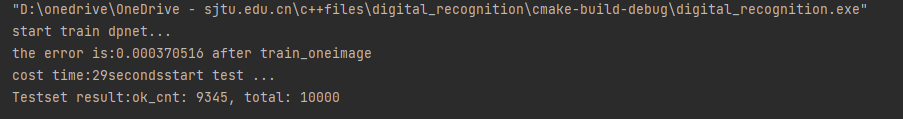
为10



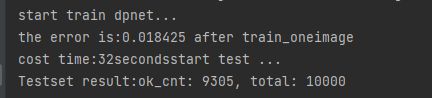
可见精度比4小，说明epoch4基本已经学好了，猜测由于学习率的设置并没有随时间减小，因此会出现上面的图中loss跳变的情况。



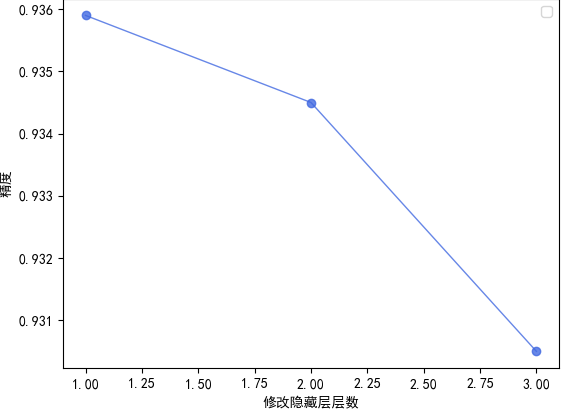
在默认config下，修改隐藏层层数为2



为3

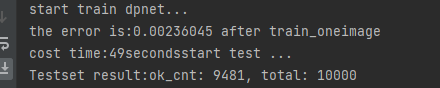


精度下降，猜测手写数据集任务简单，多层网络学到的高维特征反而不利于目标的分类，并且会有梯度消失问题。

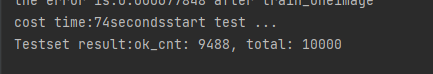


最后修改每层节点数量：

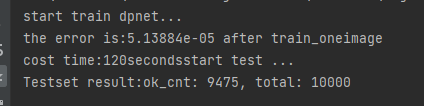
NUM\_NODES\_PER\_LAYER：200



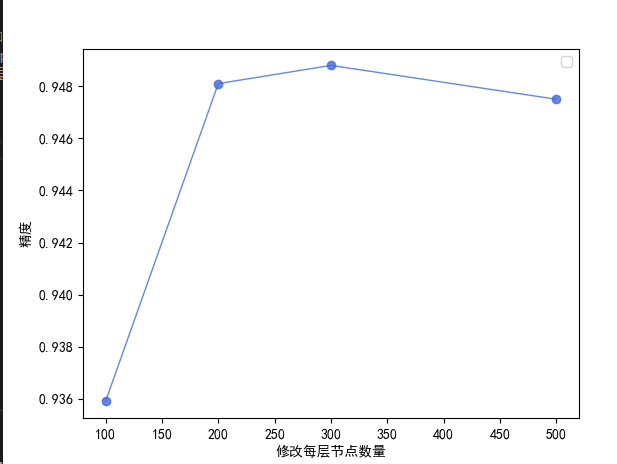
300



500



可见300节点最好，100-300上升原因是欠拟合，需要更多的神经元去学习数据，这和上面的增大epochtime的方式很类似，500下降的原因猜测是过拟合。



最后注意到每次参数一致，loss一致，这应该是rand.seed一样导致的，weight bias初始化值相同。

注意，助教需要运行的话需要改变数据集的目录，使用的根目录调用的：

