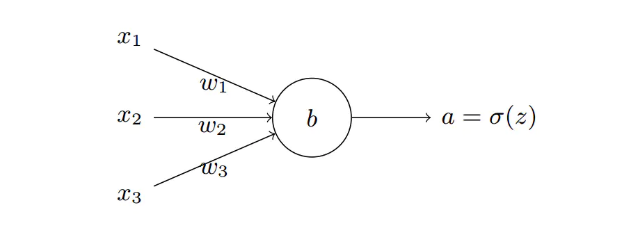
Softmax回归模型（多元分类）

初期

一开始主要在模型的构建上花费时间。了解了一些神经网络的概念后，我决定使用只有输入层和输出层的简单的神经网络结构来构建模型，其输出层激活用softmax，共有784个输入层结点，10个输出层结点。 如图所示，简单的神经网络：



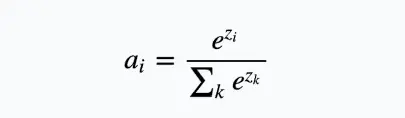
在这个过程中，我首先面对的第一个难点是矩阵的运算。一开始我只是简单地把权重W初始为（10，784）的矩阵，然后把单个样本的特征值X化为一个（1，784）的矩阵进行

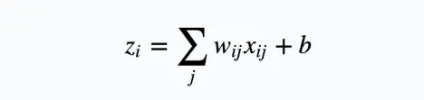
W\*X.T+b

公式运算，但是在处理样本量比较大的情况下用for循环遍历全部样本就使得效率十分缓慢。后来想到了直接将全部样本数据化成一个（2w，784）的矩阵直接进行W\*X.T+b运算,既提高了效率，又简化了代码量。

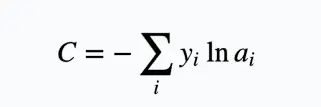
我面对的第二个难点则在用梯度下降更新权重时求偏导的问题上。查阅了网上很多博客，花了较多的时间去理解这一过程，才大致搞懂其中奥妙。

首先，softmax函数定义如下：

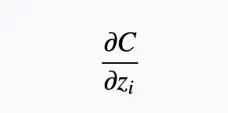


 其中：

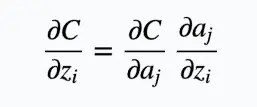
模型使用的损失函数是交叉熵函数：

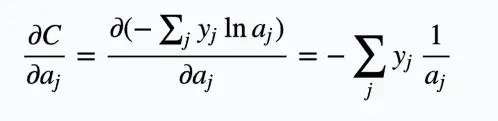


所求的梯度为：

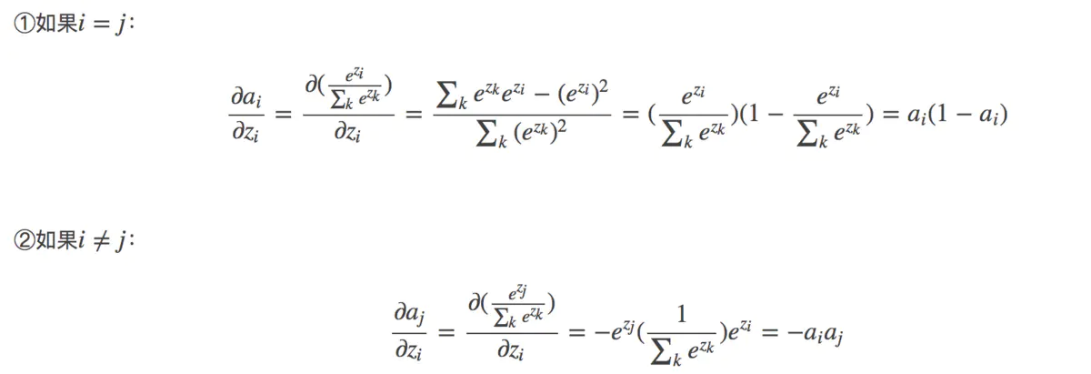


根据复合函数求导法则有：



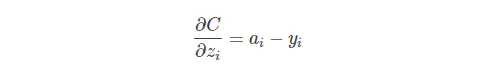
其中

由于softmax函数的形式，它的分母包含了每一个z，所以存在两种情况



i,j表示输出层的结点

两种情况相加合并得到



最后得到损失值对权重W偏导公式：

(p代表第几个输入结点)

在使用梯度下降的过程中，我学习了不少的东西。比如使用交叉熵作为模型的损失函数时，我了解了一些关于信息熵，条件熵和信息增益的相关概念和公式，这使得我进一步去了解了决策树和随机森林的一些大致原理。这些东西虽然还没来得及用代码去实现使用，但是这种学习过程让我印象深刻。

中期

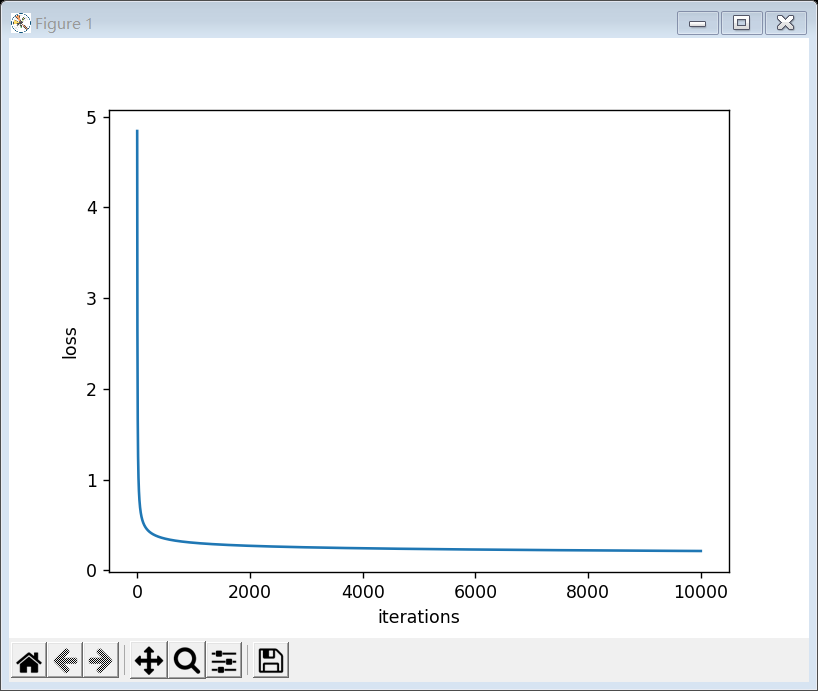
在模型构建完载入测试集进行数据预测后，我发现分数不太理想，于是在这阶段中我主要花时间在对数据样本的初始化处理，参数的调整以及思考探索可以用什么方法提高模型预测的准确度。

一、数据样本的预处理

我先在2w训练样本中找出值全为0的特征进行删除，删除了大概九十多个特征值，并在测试集上也把这九十多个特征值进行删除，然而结果并没有变化多少。而且后来通过查找索引发现，这九十多个特征有一些在测试集上它的值并不全为0，可见如此简单粗暴的处理方法并没有成效，便放弃了这种删除特征值的方法。

二、参数的调整

我先设置梯度下降的迭代次数一万次，然后不断调整学习率来获得较少的损失值，发现了学习率在0.3时，损失值普遍较少。而我在中期阶段最高的分数也是在没有对数据进行预处理的情况下使用这个学习率得到的。

梯度下降迭代的次数上，通过matplotlib库画出的损失值与迭代次数的关系图上可以清晰看到，6000次的迭代，损失值下滑的曲线较陡峭，6000次以后，损失值下降的曲线平滑，值的变化很轻微

三、其他的尝试

由于测试集只给出了特征量的数据，难以知道模型预测出来的结果与实际结果相差多大。在样本量有两万的情况下我便决定将15000样本划分为训练集，5000样本作为测试集，并将模型预测出来的结果与测试集的实际结果进行损失值的计算。结果发现，测试集的损失值有0.3几，而训练集的损失值只有0.1几。这让我思考是不是模型存在过拟合的问题，我便在模型里添加了正则化处理，对除偏置外的所有权重进行惩罚，在尝试了一些正则化参数lamda值后发现模型的分数并未提高，甚至有所下降，便放弃了这个处理。

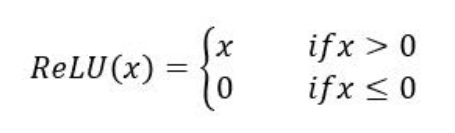
总的来说，这个过程我不断进行模型优化、评估，来提高模型预测的准确度，但是效果都不太理想。其中，我了解到的一些诸如PCA的特征处理方法由于时间原因尚未进行实际运用。

后期

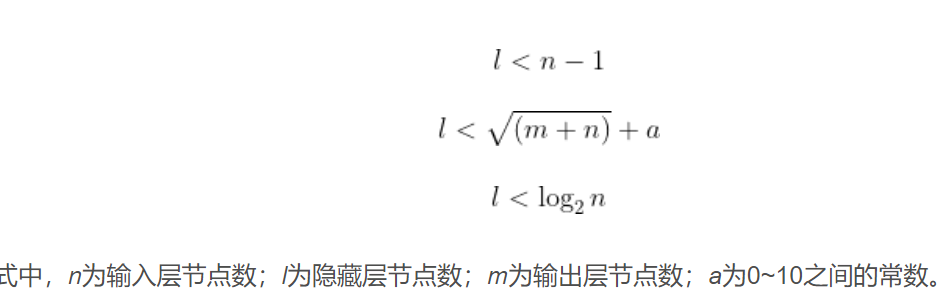
由于一系列模型优化的尝试都没有获得较大的提高，我便开始着重于提高模型的复杂度以此进行这类非线性的分类问题。

一、增加Relu隐藏层

在加深了对神经网络知识了解后，我开始考虑增加一层隐藏层。查阅博客发现Relu激活函数在大多数的情况下比sigmoid函数表现要好，我便使用Relu作为隐藏层的激活函数。



紧接着，隐藏层里结点个数的选择又是一个问题，我在查阅博客时发现了三个关于结点个数确定的公式：

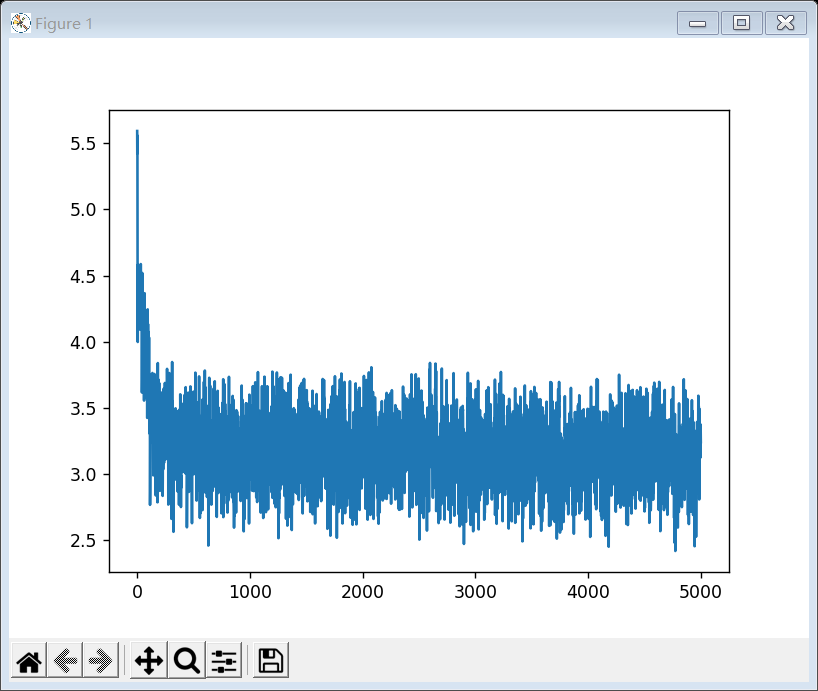


我便以此确定隐藏层的结点数为5个。增加一个隐藏层在模型原有的基础上改动并不是很大，只是添加了一个Relu函数和重新设置了从输入层到隐藏层，隐藏层到输出层的两组权重参数

多了一个隐藏层的关键难点在于权重的更新。由于Relu函数的形式，在对第一组权重梯度下降求偏导时，当输入值大于0时，Reluctant函数的导数为1，意味着隐藏层的梯度在经过Relu函数传播到输入层时并没有发生变化；当输入值小于等于0时，梯度则为0。

按照上面的原理，我进行了代码的更改。但遗憾的是，该模型在训练集上出现了loss曲线震荡的情况，在调整了学习率后也是如此。并且损失值不收敛，处在比较高的值，预测出来的分数更低。（后来发现，是我计算错了隐藏层的损失梯度，没有将输出层的梯度乘上隐藏层到输出层的权重，但由于进行其他模型尝试，并未来得及修改）

Loss曲线震荡



二、卷积神经网络

在听了分享会榜一同学的分享后，我才知道到训练集给出的784个特征代表的是长宽28\*28的图片。我便开始了解深度学习方面的卷积神经网络，并尝试python实现。

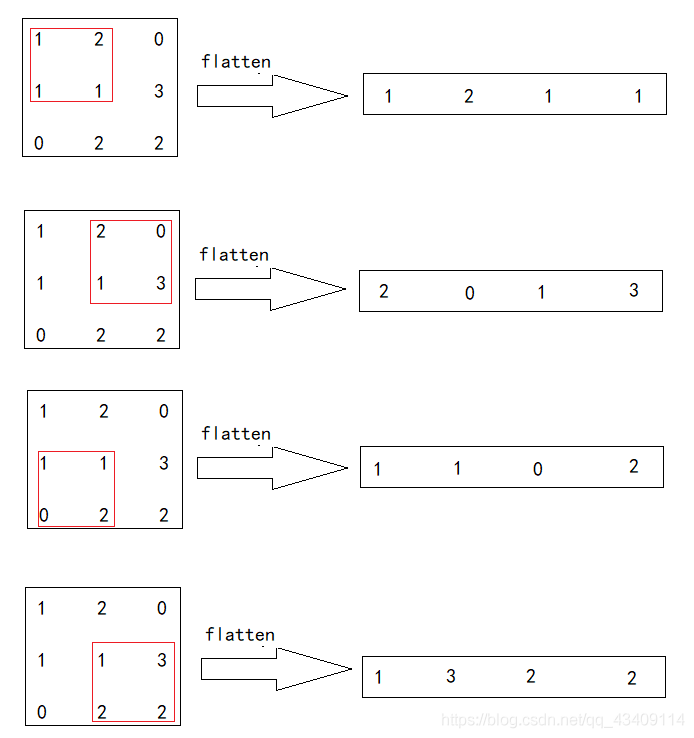
前向传播：

1.卷积层：

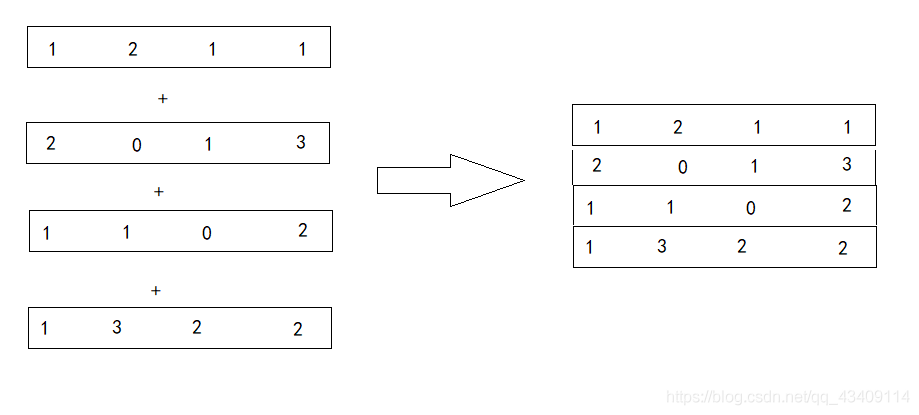
为便于理解，我只考虑了单通道的情况。在这里我使用3\*3卷积核去与28\*28图片做卷积运算，并使用了img2col算法进行运算

img2col算法：该算法的原理，就是将卷积运算，转换为矩阵乘法运算。

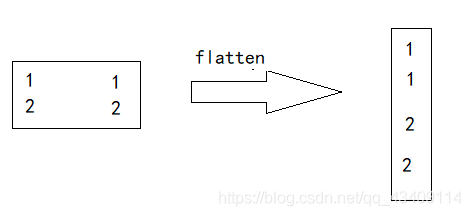
如：用2\*2卷积核与3\*3图片做卷积运算时，先提取图片每次做卷积处理的矩阵并化为一个行向量



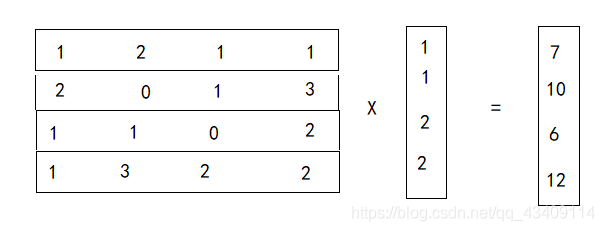
然后将这四个向量形成一个新的4\*4矩阵



将2\*2卷积核化成一个列向量



然后将新矩阵与卷积核做矩阵乘法运算，就得到了卷积值



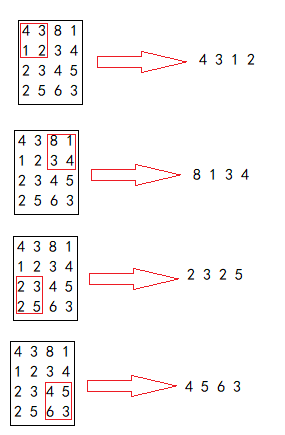
最后可将该卷积值reshape成2\*2的矩阵

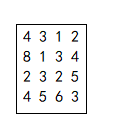
通过img2col算法我在卷积层输出了26\*26的特征矩阵

2.池化层

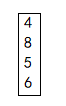
该层采用最大池化来删除卷积层的一些特征。在这里使用了类似img2col的处理

如：对于4\*4的图片特征矩阵，池化大小为2\*2，step为2，可得到4个向量

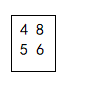


堆叠一起得

使用np.sum函数对每行求最大值得



reshape得到池化结果



3.全连接层

该层使用relu激活函数，对池化层输出的数据进行矩阵乘法

4.输出层

使用softmax函数

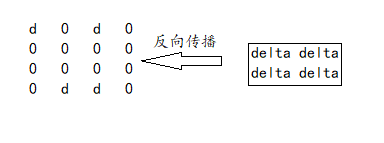
反向传播：

写出这个卷积神经网络的最主要难点就在反向传播上。输出层和全连接层的反向传播与普通的神经网络没有太大区别，不同的在卷积层和池化层的反向传播上。

1.池化层

这里采用的是上采样的做法，即将前向传播时池化层输出的特征值得到的损失放回到原来所在的位置

如：



这需要在前向传播时获得输出特征原来的索引值，然后在反向传播时根据索引放回原有位置。

2.卷积层

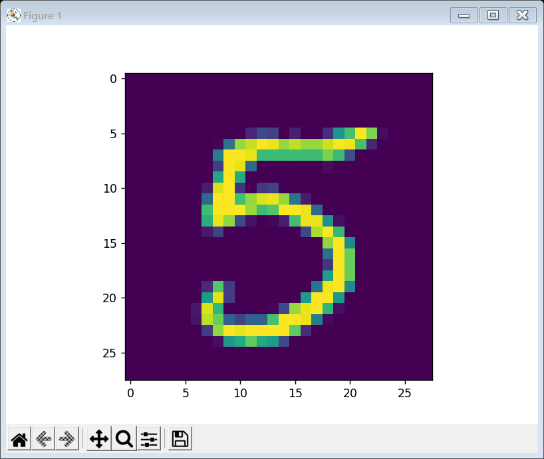
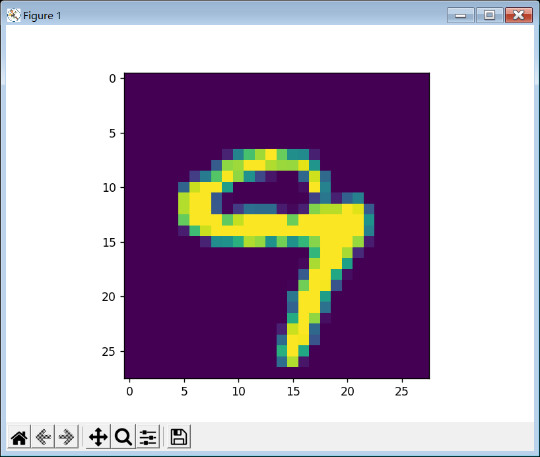
这一部分较为复杂，查阅了很久博客后，我大概的理解是：卷积层输入层与卷积层的损失值做卷积运算

在上交截止的最后一周大部分时间我都在实现着这个卷积神经网络的代码，但出来的结果不尽人意。首先，由于每个层我都只是用for循环之类简单语法来实现每个层的功能，大大降低了代码的运行速度；其次，网络最后输出层的损失值一直都在十点几，并没有下降，这个卷积神经网络并没有实现预期的功能。虽然如此，但我在这个过程加深对卷积神经网络的算法理解。

三、删去冗杂特征

在所写的卷积神经网络没有获得预期效果后，由于时间紧迫，也不能仔细地对该网络进行优化，于是我开始重新思考图片特征的含义。

用matplotlib库的imshow函数画出其中几个样本



发现图片数字部分的像素值高，而其他部分的像素值低，而且数字基本都存在图片的中间位置。那么可以认为，数字的识别主要靠特征值比较大的点进行。于是我将经过归一化处理后，样本平均值小于0.1的特征删除，删去了187个特征，再把新的特征代回到一开始简单的神经网络模型，分数居然提高到了0.91分。

最后由于时间原因，并未做其他的后续处理，最终完成得出效果最好的就是这个模型。

总结

这次这个模型的实现，虽然踩了很多坑，遇到了很多困难，但是通过这个过程我加深了对所学的回归模型、神经网络相关知识的理解运用，同时也拓展学习了深度学习里卷积神经网络的知识，让我收获很多。这次的代码实现让我深深意识到了构建模型前数据预处理的重要性。只有深刻了解数据特征的意义，才能根据相关样本选择不同的算法来构建更有效率的模型。

博客参考：

<https://www.jianshu.com/p/c02a1fbffad6>

<https://blog.csdn.net/qq_43409114/article/details/105747991>

<https://blog.csdn.net/qq_43409114/article/details/105426806>

<https://www.cnblogs.com/pinard/p/6494810.html>