



**2020-3-10**

**南京大学**

**软件学院**

实验一神经网络报告

**江辉--171250506**

目录

[一、 实验目的 1](#_Toc34776007)

[二、 内容描述 1](#_Toc34776008)

[1、 环境及配置 1](#_Toc34776009)

[2、 项目目录描述 1](#_Toc34776010)

[三、 运行用例和截图 2](#_Toc34776011)

[1、 不同的keep\_prob值的对照试验 2](#_Toc34776012)

[2、 不同的网络层结构的对照试验 3](#_Toc34776013)

[3、 不同的卷积核大小的对照试验 4](#_Toc34776014)

[4、 不同的学习速率大小的对照试验 5](#_Toc34776015)

[四、 思路方法 6](#_Toc34776016)

[五、 假设 6](#_Toc34776017)

[六、 过程分析描述 7](#_Toc34776018)

[1、 数据预处理 7](#_Toc34776019)

[2、 数据分割 7](#_Toc34776020)

[3、 构造CNN网络层和训练 8](#_Toc34776021)

[4、 结果保存和分析 9](#_Toc34776022)

[七、 重要函数 10](#_Toc34776023)

[1、 卷积核、池化核、全连接参数的初始化 10](#_Toc34776024)

[2、 卷积计算 10](#_Toc34776025)

[3、 池化 10](#_Toc34776026)

[八、 核心参数 11](#_Toc34776027)

[1、 5\*5的卷积核 11](#_Toc34776028)

[2、 2\*2的池化核 11](#_Toc34776029)

[3、 2层的全连接网络 11](#_Toc34776030)

[4、 其余超参数 12](#_Toc34776031)

[九、 实验中出现的问题和相应的解决办法 12](#_Toc34776032)

[十、 对实验的评价和感觉 12](#_Toc34776033)

# 实验目的

本次实验使用python构建一个CNN神经网络完成MNIST“手写体识别”任务，

# 内容描述

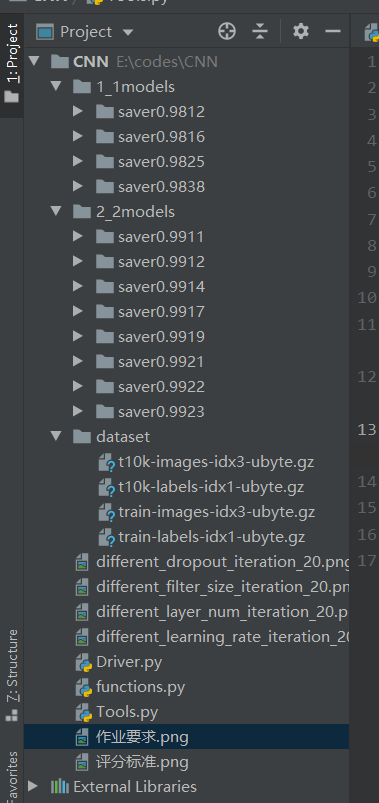
本程序用python编写，使用TensorFlow框架实现CNN神经网络。预先定义好各个层卷积层的卷积核大小、池化层函数和全连接层的网络大小，运用Adam优化算法来学习，使用dropout来避免过拟合。并且通过对比试验观察不同参数、结构的网络执行的效率和准确率等情况。

## 环境及配置

本项目推荐使用pycharm打开

最好配好TensorFlow-gpu+ cuDNN+CUDA，这样可以用gpu训练

我的gpu配置是GTX1050，大概是3-4s左右一次迭代，一次训练约1-2min，(4w训练集)，不同环境运行时可能不同，仅供参考



## 项目目录描述

**1\_1models**存放了使用1层卷积层 + 1层全连接层训练的结果

**2\_2models**存放了使用2层卷积层 + 2层全连接层训练的结果

saver后面的数字是对应存档的测试集准确率

dataset存放了MNIST的数据文件

然后下面是我用pyplot画出来的几个对比试验结果图

**Driver.py**里写了main函数和几个对照试验的参数，我把对应的对照试验都封装成函数了，可以直接在main函数里调用运行

**functions.py**里写了整个model的所有函数，所有模型相关的函数入口和参数都以注释的形式写在这个文件里了

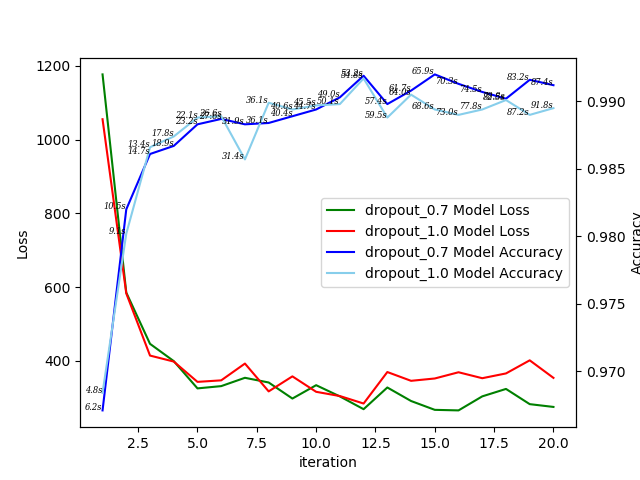
**Toos.py**里有我写的用来测试是否使用gpu的函数，和一个我写的用matplotlib.pyplot将结果可视化的函数，如果需要跑程序的话可以参考使用

**difference\_xxxxx.png**是我完成各个对照试验的结果可视化后的折线图

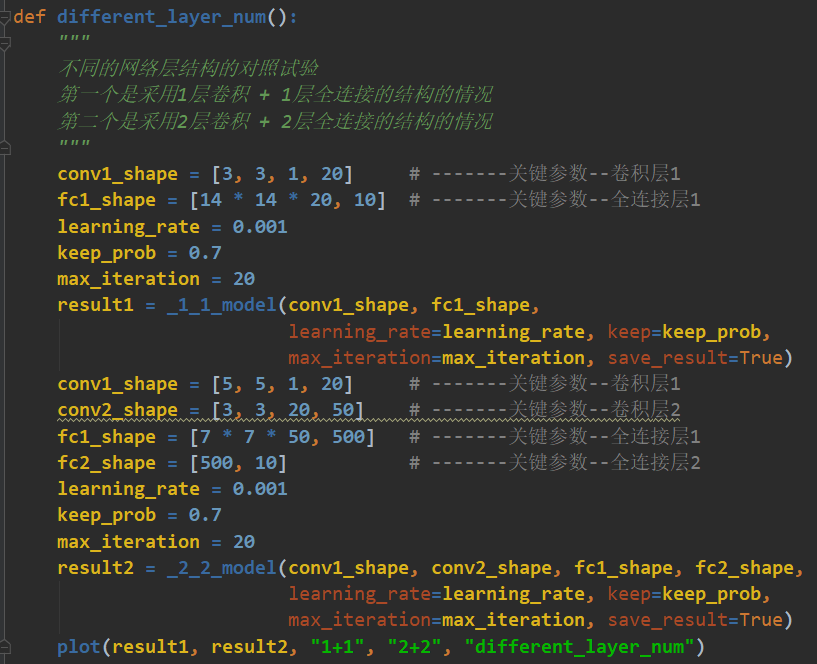
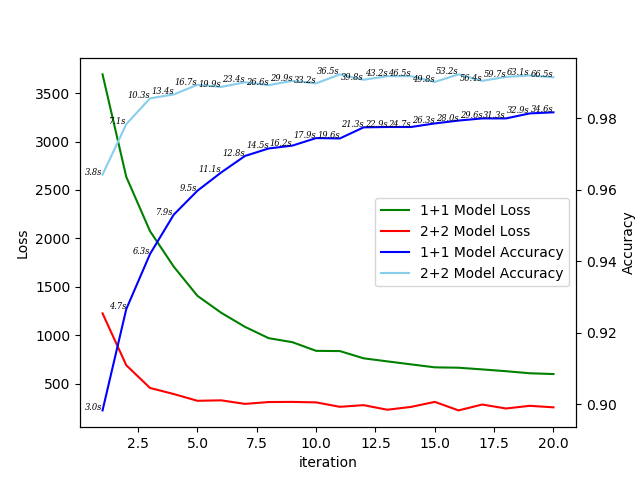
# 运行用例和截图

## 不同的keep\_prob值的对照试验

右图为输入的用例数据，下图为相应展示的结果折线图，折线图里点上的数字是运行的时间(s)，可以看到，**keep\_prob为0.7和为1.0的情况**都可以在测试集上达到**99%**以上的正确率，且运行时间上没有很大差别（一次迭代约5s，keep\_prob为1.0时平均运行时间稍长一些），但是当不使用dropout（即keep\_prob=1.0）时，训练的效果会有一定波动，且平均准确率略低于使用了dropout的情况(keep\_prob=0.7)，而且**不使用dropout的loss曲线**在迭代次数多了之后有一点向上升高的趋势，开始变得**过拟合**，而使用了dropout的则没有发现这个趋势



## 不同的网络层结构的对照试验

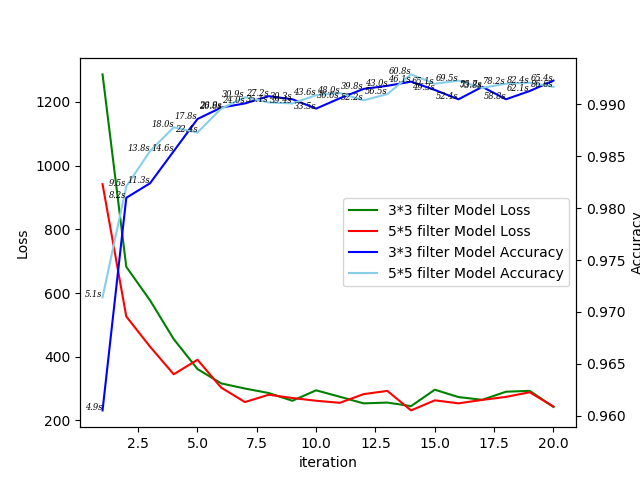
右图为输入的用例数据，下图为相应展示的结果折线图，折线图里点上的数字是运行的时间(s)，可以看到，当采用**1层卷积 + 1层全连接**的结构时结果缓慢上升，接近稳定在**98%**附近，而采用**2层卷积 + 2层全连接**的结构则很快就稳定在了**99%**的准确率。

**运行时间上**有较大差别，**1层卷积 + 1层全连接**20次迭代总时间34s，而**2层卷积 + 2层全连接**则花费了66s，是前者的2倍，因此准确率和时间需要作出权衡

## 不同的卷积核大小的对照试验

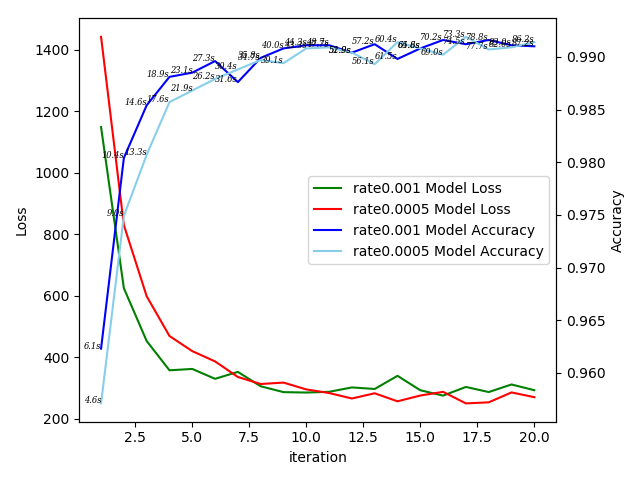
右图为输入的用例数据，下图为相应展示的结果折线图，折线图里点上的数字是运行的时间(s)，可以看到，采用**3\*3**的卷积核在刚开始时效率会稍微比用**5\*5**的卷积核差一些，但是在几次迭代后两者就没有什么差别了，最终都达到了99%的准确率，不过3\*3的收敛稳定性稍差一些。

在**运行时间上**，**3\*3**的卷积核在20次迭代总用时**65s**，而**5\*5**的卷积核用时**86s**，因此在这个对比条件下效率和准确率也是负相关的关系。



## 不同的学习速率大小的对照试验

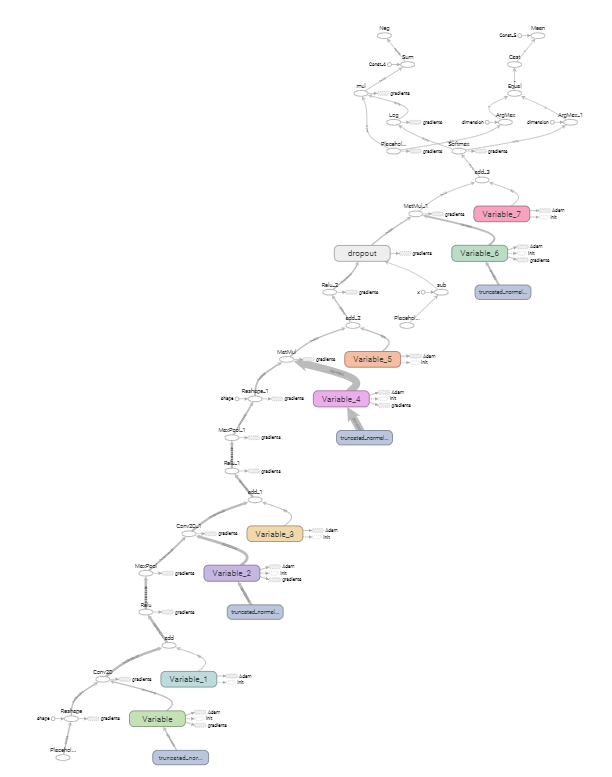
右图为输入的用例数据，下图为相应展示的结果折线图，折线图里点上的数字是运行的时间(s)，可以看到，采用**0.0005的学习速率**和**采用0.001的学习速率**在刚开始的收敛速度上有一点区别，但在经过几次迭代后两者几乎就没有区别了，即使在运行时间上两者也没有区别。我试过0.01的学习率，报错了，在某处计算时出现了nan，这也说明在学习率合适的情况下，小学习率只是影响刚开始的速率，这使得学习效率下降



# 思路方法

详细实现在六、过程分析描述中，也可以在Driver代码中执行custom()方法，然后进入terminal输入tensorboard --logdir=log，进入TensorBoard查看流程图

1. 首先通过TensorFlow提供的数据集读取函数读取数据
2. 对数据集进行batch=400的分割
3. 构建LeNet-5结构的CNN网络，即2层(卷积层+池化层) + 2层全连接层
4. 对中间层采用relu激活，对输出层采用softmax激活
5. 使用adam优化学习算法进行后向传播学习参数

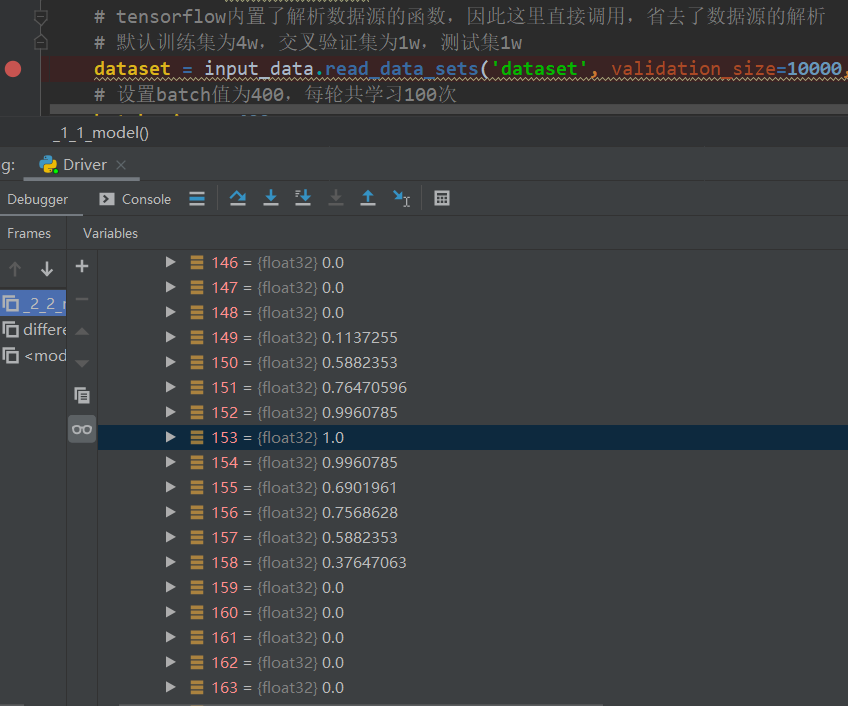


# 假设

假设输入的数据已经去噪和归一化，这也是我在载入MNIST数据集后简单观测到的结果

# 过程分析描述

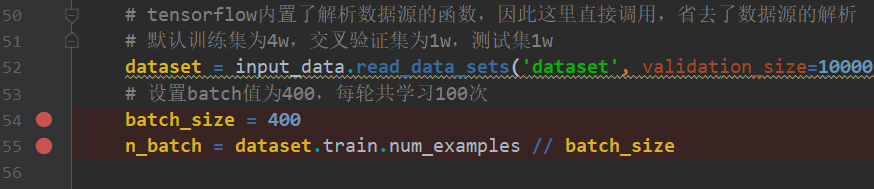
## 数据预处理

TensorFlow已经提供了读取MNIST数据的方法，因此我直接调用了它的方法。我本来想对数据进行去噪、归一化等操作，但是发现里面数据已经是归一化的数据，且看了几张图的数据之后我没有找到噪声点，所以就省去了这两个步骤。图中可以看到数组里的片段中数据∈[0, 1.0]，且非常集中，周围没有噪声

## 数据分割

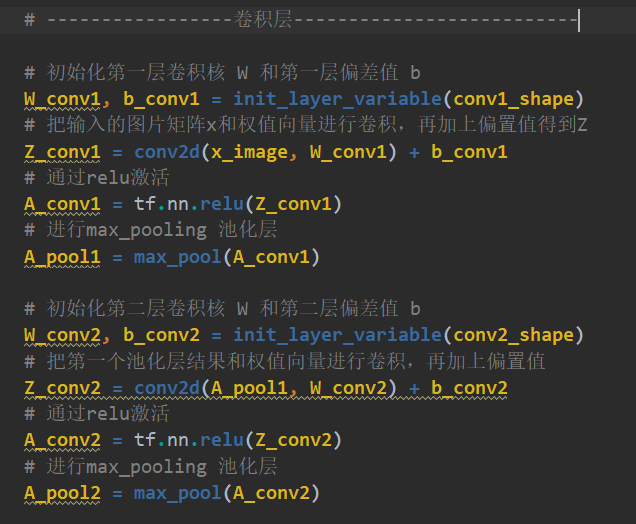
MNIST数据集里有6w数据，我选择训练集为4w，交叉验证集1w，测试集1w

并以batch=400将其分割为100个小数据集，每次迭代将学习100次



## 构造CNN网络层和训练

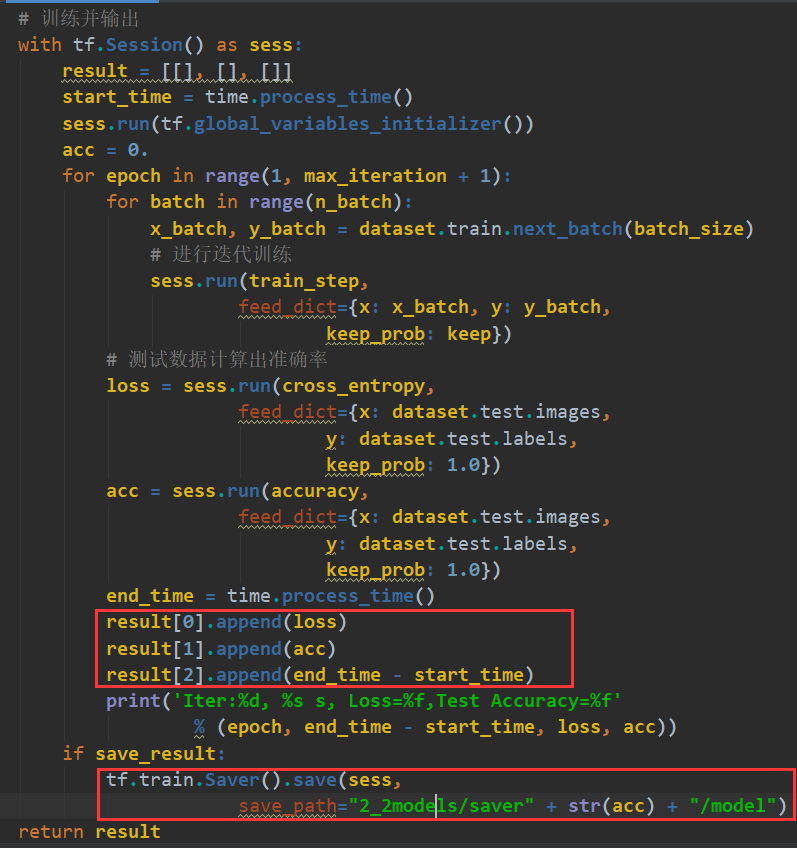
我主要采用的是LeNet-5结构，即2层(卷积层+池化层) + 2层全连接层的CNN神经网络

使用5 \* 5的卷积核，中间层使用rule激活，且每次卷积后进行2 \* 2池化(步长为2)，即缩小一半，相关的shape参数和初始化各层参数的方法会在后面说明

经过两次卷积后的激活值进行2次全连接和relu激活，最后经过softmax函数输出10个预测值，并计算代价函数(交叉熵)，学习时采用adam优化算法，其实也可以用momentum或RMSprop，不过adam已经是二者结合的方式，所以就不做多的对比测试了

## 结果保存和分析

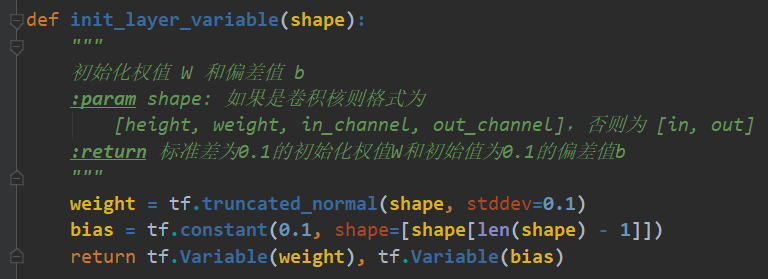
每次迭代我都会保存数据，用以在后面通过图片打印出来方便查看和分析，主要保存的数据有**代价值、准确率、运行时间**，前面所有的分析和展示都是基于这3个数据指标



# 重要函数

## 卷积核、池化核、全连接参数的初始化

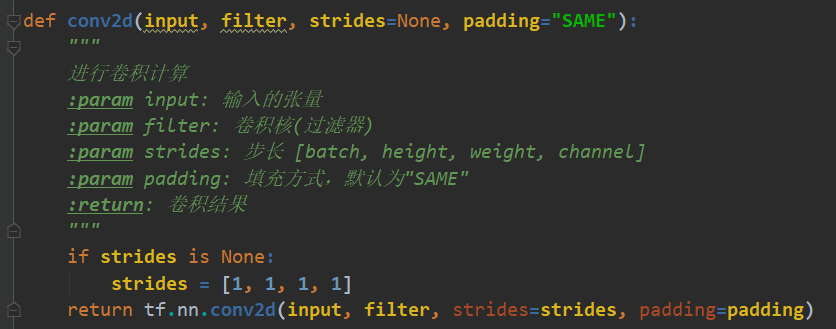
我将参数初始化封装了一个方法，只需要传入对应的shape即可获得随机的参数张量



## 卷积计算

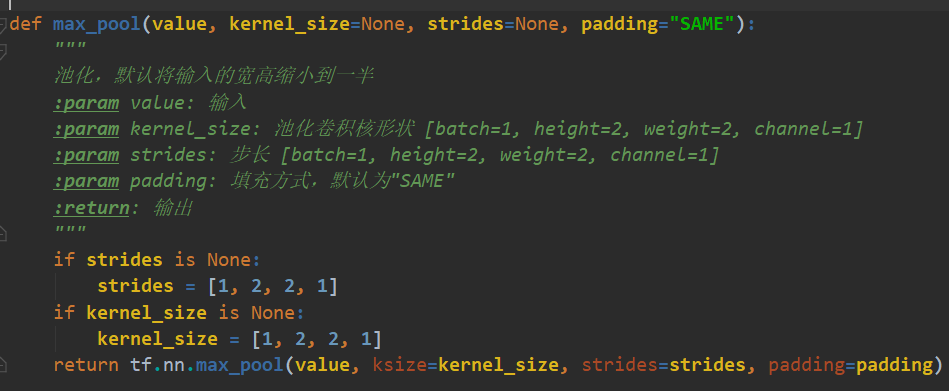
我默认使用步长为1，1的卷积方式

填充时采用了保留边界的SAME填充



## 池化

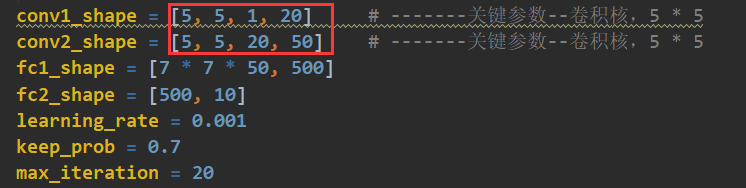
我采用了 2 \* 2 的池化核，2为步长，即将卷积输出缩小为原来的一半



# 核心参数

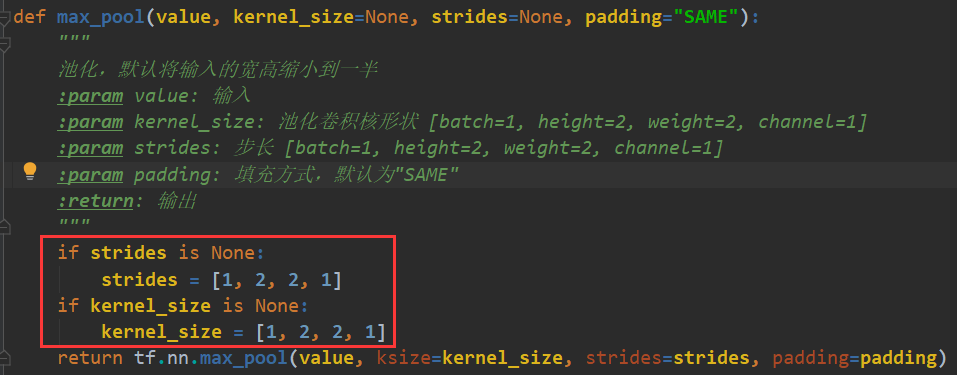
## 5\*5的卷积核

我在所有对照实验里主要采用 5 \* 5 的卷积核，第一层输出20个通道，第二层输出50个通道



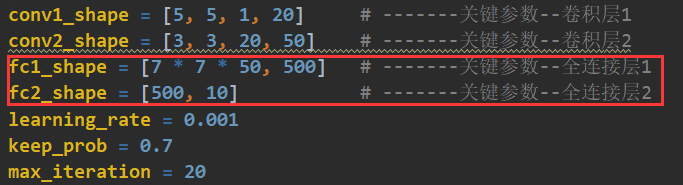
## 2\*2的池化核

所有对照试验中的池化核都是采用默认的2 \* 2 池化



## 2层的全连接网络

我在所有对照实验里主要采用卷积层输出7\*7\*50个节点后进行全连接到50个节点，再输出10个预测值的全连接层

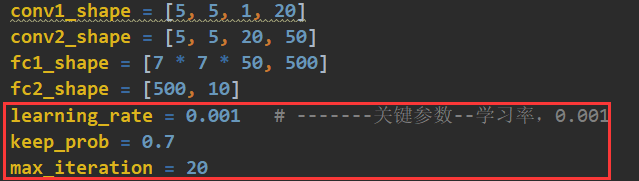


## 其余超参数

为避免折磨我的gpu，我选择了每组模型最多跑20次迭代，实际上这也跑出了不错的结果

学习率主要使用0.001，我尝试了0.01但是报错，某处计算出了nan

训练中的保留率主要设置为0.7



# 实验中出现的问题和相应的解决办法

1. 刚开始时想解析MNIST提供的源文件，但是后在搜索资料的时候发现TensorFlow已经提供了解析数据源的方法，所以就直接调用了
2. 刚开始时想通过excel绘制表格来将结果进行可视化展示，后来发现每次都要操作很麻烦，而且展示的效果也不是很好，就去学习了pyplot并写了展示并保存图片的方法
3. 刚开始时看到安装TensorFlow要使用gpu的话要安装些麻烦的包，图方便就直接使用cpu跑程序时，发现实在太慢，即使只使用很小的网络速度也十分感人，最后还是不得已去装了TensorFlow-gpu+ cuDNN+CUDA，最后效果很满意

# 对实验的评价和感觉

这次实验前我正在自学CNN，正好有这个机会上手实践代码，体会到了从输入到卷积到池化到全连接再到输出的全过程，也体会到了gpu对机器学习计算的超强能力。这也是我第一次接触TensorFlow框架，以前学自学机器学习的时候都是手撸代码，实在难写，这次也算学了一个工具，本来也是想手写实现的，因为学期太紧，还有其他课程，如果时间不是那么紧张的话还是希望尝试一下手动实现CNN网络的。不过可惜的是写这次实验代码中因为直接使用了TensorFlow中对数据源的读取，没有体验到解析数据源的步骤，以后有时间的话还是得再回头写写。最后就是关于tensorboard的使用，刚开始做可视化时按照以前的习惯用了pyplot。后来才知道有更方便的tensorboard可以用，后续的学习中一定要去学会用这些方便的工具