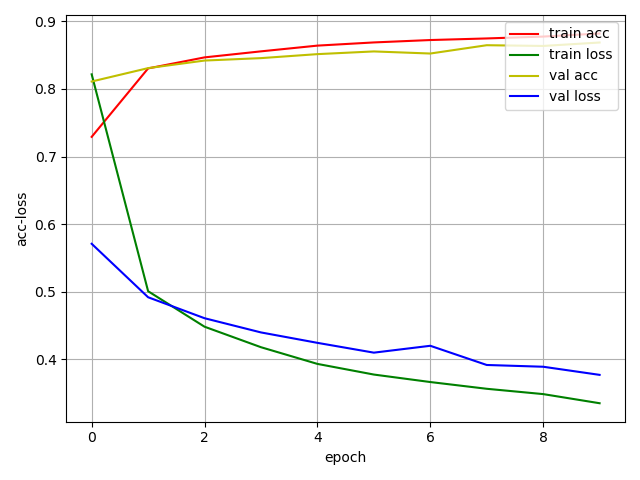
数字系统设计实验一

1. **TensorFlow实验**
2. (1) 该教程用的是CNN（卷积神经网络）结构，包括1个输入层和2个全连接层；损失函数是sparse categorical cross entropy（交叉熵损失）函数；优化方法是Adam。

(2) 常用的损失函数还有：均方误差损失、SVM合页损失、smooth L1损失，等等；

(3) 常用的优化方法还有：梯度下降法（GD，其中常用的是随机梯度下降法SGD）、共轭梯度法（CG）、牛顿法和拟牛顿法、Momentum、AdaGrad、Adadelta、RMSprop、Adam和Adamax、Nadam，等等。

1. 通过回调函数让程序自动绘制loss曲线，结果如下：



其中acc代表accuracy；train代表训练集，val代表测试集。

1. 卷积神经网络，对照组的结构为：

第1层，卷积层，输入28×28×1，卷积核大小5×5，通道数32；

第2层，池化层，池化核大小2×2；

第3层，卷积层，卷积核大小5×5，通道数64；

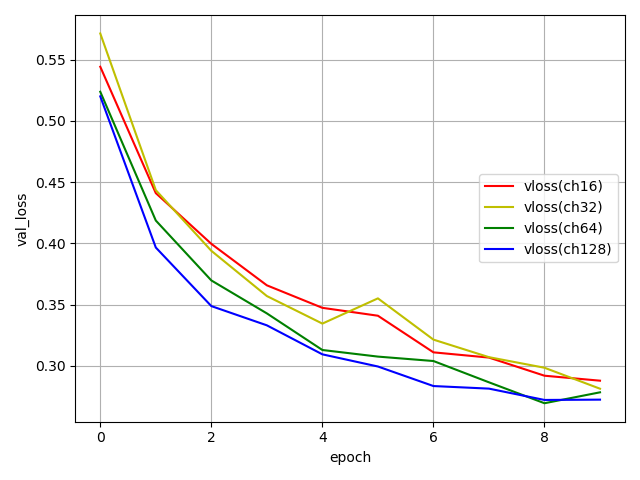
第4层，池化层，池化核大小2×2；

第5层，全连接层，输出通道数为1000；

第6层，全连接层，输出通道数为10。

1. 增减卷积层通道数，实践中发现识别准确率很大程度上取决于第5层全连接层的参数数目，故而设计增减卷积层通道数的网络结构时应尽量保持全连接层的参数数目一致。

因此保持第3层通道数不变，将第1层的通道数分别改成16, 64, 128，与对照组的loss曲线作比较，结果如下：



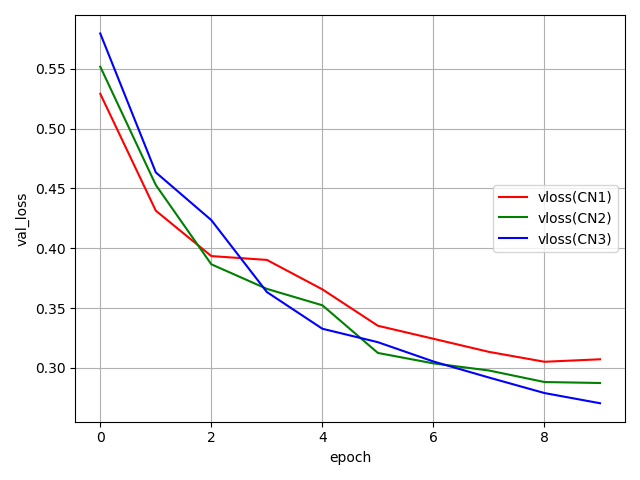
可见增加卷积通道数，loss曲线下降更多，但程度不大；当通道数较多时会出现loss随训练迭代次数反而增大的现象，推测可能与过拟合有关。

分析参数数目、运算量大小和测试集max准确率：

|  |  |  |  |
| --- | --- | --- | --- |
| 卷积层通道数 | 参数数目 | 运算量大小 | 测试集max准确率 |
| 16 | 1 061 090 | 30s | 89.66% |
| **32** | **1 087 106** | **44s** | **90.12%** |
| 64 | 1 139 138 | 73s | 90.17% |
| 128 | 1 243 202 | 155s | 90.46% |

可见随着卷积层通道数的增加，测试集上的最高准确率增加缓慢，运算量却几乎指数上升。

1. 增减卷积层数，确保第5层的参数数目一致；分别将第3、4层去掉，将第1层改成64通道，第二层改成6×6池化，得到一层卷积层的网络；将第4层池化层改成64通道、5×5卷积核的卷积层，得到三层卷积层的网络。与对照组的loss曲线作比较，结果如下：



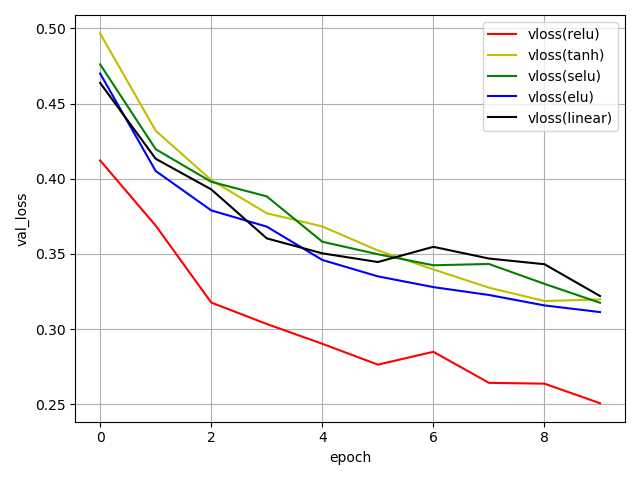
可见增加卷积层数，loss曲线下降更多。

分析参数数目、运算量大小和测试集max准确率：

|  |  |  |  |
| --- | --- | --- | --- |
| 卷积层数 | 参数数目 | 运算量大小 | 测试集max准确率 |
| 1 (645, p6) | 1 036 674 | 30s | 89.00% |
| **2 (325, p2, 645, p2)** | **1 087 106** | **42s** | **89.71%** |
| 3 (325, p2, 645, 645) | 1 189 570 | 50s | 90.60% |

可见随着卷积层数的增加，测试集上的最高准确率的增加较缓慢，运算量的增加也不太过分。

1. 采用不同的激活函数，这里采用简化的卷积神经网络结构，第1层是16通道、3×3卷积核的卷积层，第2层是2×2的池化层，第3层是1000输出通道的全连接层，第4层是10输出通道的全连接层。激活函数分别选取relu, tanh, selu, elu和linear，loss曲线如下：



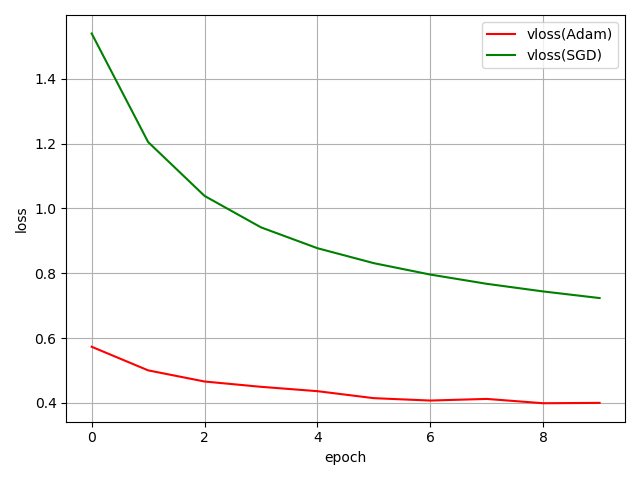
可见relu函数优势明显，其次稍有优势的是elu函数。

分析参数数目、运算量大小和测试集max准确率：

|  |  |  |  |
| --- | --- | --- | --- |
| 激活函数 | 参数数目 | 运算量大小 | 测试集max准确率 |
| relu | 2 715 170 | 23s | 91.00% |
| tanh | 2 715 170 | 23s | 88.68% |
| selu | 2 715 170 | 33s | 88.69% |
| elu | 2 715 170 | 24s | 88.81% |
| linear | 2 715 170 | 21s | 88.53% |

可见准确率上relu优势明显，其他几个函数相差不大；运算量上selu最大，linear最小，但小得不多。

1. 采用教程中的网络结构，分别用Adam和SGD优化方法：



可见Adam的收敛速度比SGD快许多。

1. **Jetson Nano实验**
2. 搭建TensorFlow环境：

Step1: 连接Jetson Nano到Internet: 将Jetson Nano开发板用USB连接线连接到计算机，然后在计算机-网络适配器属性中将计算机的无线网络共享到USB以太网，并配置以太网卡的IP获取方式、DNS服务器和DHCP服务器为自动。原因是，Jetson Nano开发板的IP地址固定为192.168.55.1，如果DHCP服务器没有配置为自动，计算机就会自己设置一个IP地址并且给Jetson Nano分配另一个IP地址，这样就不能用192.168.55.1:8888访问Jupyter了。

Step2: 在Jetson Nano上利用virtualenv创建虚拟环境。

Step3: 在Jetson Nano上安装Tensorflow-gpu：参照官网上的教程，联网后运行：

**$** sudo pip3 install --pre --extra-index-url https://developer.download.nvidia.com/compute/redist/jp/v42 tensorflow-gpu

这样pip3就可以自动从Nvidia官网下载tensorflow-gpu并且安装到Nano上。

Step4: 在计算机上搭建相同的TensorFlow环境，并进行兼容性测试。

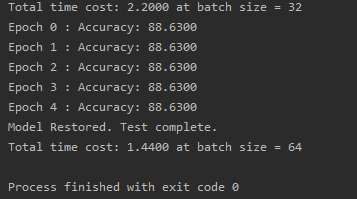
1. 分别在计算机上和Jetson Nano上运行图像分类代码，并记录程序在不同的处理器（CPU、GPU）上和不同的batch-size下的运行性能。

Step1: 在计算机上编写针对Fashion Mnist数据集的神经网络（这里测试了两种网络结构：纯全连接，与先卷积后全连接）。利用计算机的GPU在数据集上进行模型训练并导出为.h5格式。



图：两种模型的导出文件

Step2: 在计算机上编写Fashion Mnist数据集上的测试python程序，加载已经训练好的模型并对10000张测试集图片进行分类。利用with tf.device()语句控制计算机CPU和GPU的使用。



图：计算机读入导出的模型文件并在不同的batch-size下进行前向推断

Step3: 将python代码用Jupyter上载到Jetson Nano上，然后用python3命令运行代码。测试结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试平台 | 全连接网络运行时间 | 卷积网络运行时间 | 测试平均正确率 |
| 计算机CPU | 6.79s | 53.64s | ~90% |
| 计算机GPU | 13.16s | 18.81s | ~90% |
| Jetson Nano CPU | 40.19s | 354.5s | ~90% |
| Jetson Nano GPU | 66.44s | 110.2s | ~90% |

注：表中的计算机CPU和计算机GPU是指在计算机的Intel Core i7-8750H CPU和Nvidia GeForce-GTX 1060 GPU上运行的结果。

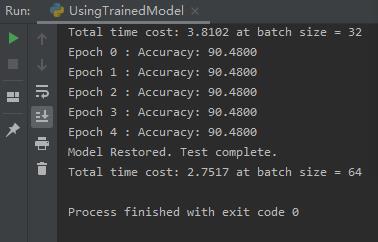
从表中可以得出结论：如果一个模型仅含有全连接神经元，那么用CPU来执行这个网络效率较高；如果一个模型中有大量卷积运算，那么用GPU来执行效率更高。初步推测：卷积运算由于卷积核的尺寸较小，所以具有比较大的并行计算的可能性。

Step4: 分别在计算机和Jetson Nano上运行不同batch-size和不同处理器条件下的处理器开销。

计算10000张测试集图片的分类问题，同样是所有数据遍历5遍的条件下，不同batch size的分类计算速度对比如下表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试网络，测试平台 | batch=2 | batch=5 | batch=10 | batch=32 | batch=64 | 正确率 |
| 全连接网络，计算机GPU | 48.99s | 19.38s | 10.16s | 3.35s | 2.14s | 88.78% |
| 全连接网络，计算机CPU | 28.40s | 12.40s | 6.39s | 2.20s | 1.44s | 88.78% |
|  | | | | | | |
| 卷积网络，计算机GPU | 56.54s | 21.33s | 10.84s | 3.81s | 2.58s | 90.15% |
| 卷积网络，计算机CPU | 63.29s | 27.20s | 14.92s | 7.42s | 6.76s | 90.15% |
|  | | | | | | |
| 全连接网络，Nano GPU | 203.86s | 83.97s | 42.54s | 14.04s | 9.02s | 88.78% |
| 全连接网络，Nano CPU | 168.79s | 69.10s | 36.28s | 12.55s | 8.16s | 88.78% |
|  | | | | | | |
| 卷积网络，Nano GPU | 427.05s | 282.50s | 121.90s | 37.82s | 30.92s | 90.15% |
| 卷积网络，Nano CPU | 373.36s | 171.92s | 78.50 | 51.78 | 47.45s | 90.15% |

从上表中可以得出结论，拥有卷积层的神经网络其图像分类效果略好于纯的全连接网络；卷积运算适合于在GPU上实现，而CPU适合计算全连接层。总体来说，Jetson Nano的GPU性能约为Nvidia GeForce GTX1060的1/4~1/8左右，其CPU性能约为i7-8750H的1/3左右。由于网络实际模型相同，在不同的平台上运行相同的神经网络，其识别正确率是相同的。



图：计算机GPU在不同的batch size下计算图片分类问题。

1. **实验中遇到的问题**

**朱结奥**

1. 环境配置：我在配置python的conda环境时，遇到了conda与pip不兼容的问题：conda安装的包不能用pip卸载，pip安装的包不能用conda卸载。而且如果一个包被conda卸载以后，pip里竟然还赫然在列！如果再用conda安装回原来卸载的包，很可能安装会出错，因为之前根本没有卸干净。因此我每次卸载包以后，都找到包所在的目录site-packages去“斩草除根”，这样就不怕包安装不全或者卸不干净的问题了。
2. Tensorflow-gpu版本问题：TensorFlow有两种：普通TensorFlow与TensorFlow-gpu，不同版本的TensorFlow生成的h5文件很可能不兼容，因此必须保证计算机上的tensorflow-gpu版本和Jetson Nano上的版本相同，都为TensorFlow-gpu的某个兼容的版本。起初我按照教程在电脑上安装了TensorFlow的CPU版本，然后生成的h5文件在Nano上无论如何都运行出错，报的错是Nano上的tensorflow-gpu版本和cuDNN库版本不符。在助教的指导下，我降低了Nano上的tensorflow-gpu版本，然而h5模型还是无法运行。直到我又把计算机上的tensorflow-gpu版本改成相同后重新训练模型，模型才得以成功运行。
3. PyCharm IDE的设置：PyCharm的IDE必须要手动配置Python解释器才能和conda兼容；否则PyCharm会自动调用virtualenv自己为工程新建一个虚拟环境，导致新的虚拟环境里没有TensorFlow。
4. Jetson Nano的USB联网。我首先想用Internet桥接来让Nano联网，发现不可行，究其原因是因为Nano上无法登录我的清华网。于是我打算自建局域网（使用Windows的网络共享功能），结果登陆不上Nano的Jupyter了。因此我请教了同寝室的刘长昊同学，他选修了计算机网络。最终在他的帮助下，我们定位到了问题所在：Windows自带的网络共享功能会自己建立一个DHCP服务器，会擅自给Nano分配一个IP地址，因此原先的192.168.55.1:8888就不再能访问了。解决这个问题的方法是，禁用本地的DHCP服务，把IP地址设定为自动获取。