对卷积神经网络并行处理超参数选取的实验

丁明朔 1800013099

2018/12/9

一、概述

在代亚非老师硬件系统的课上，对多核处理器与神经网络进行了简要的介绍。

1. 多核处理器

多核处理器可以并行处理多个任务，使得总时长成倍减少。一个任务可以横向拓展处理器的功能，通过划分出并行的子任务，达到充分利用多个可执行内核，进而在特定的时间内执行更多的任务的目的。

当出现多个任务的时候，多核 CPU 会调用多个核心并行计算，每个核心负责一部分数据。每个核心都有独立的一级缓存和二级缓存，而三级缓存被所有核心共享，用于核心之间的同步与通信。

而 GPU 的多核处理方式却并不完全相同。GPU 的核数非常多：

显卡型号	CUDA 核心数[1]
GTX 1080 Ti	3584
GTX 1080	2560
GTX 1060	1280

表格 1 常见显卡的 CUDA 核心数

而且 GPU 的控制器比较弱，所以 GPU 选择将线程分组控制。令 32 个线程为一个 warp，一个 warp 的线程执行的是同一条指令，但是指令的数据不同，一般通过 CPU 向 GPU 发送指令。

容易想到，一个 warp 包含的线程数过多，就可能会导致内核的浪费，而一个 warp 包含的线程数过少，就会导致并行性不好，所以 32 这个数字是权衡后的结果。可以预见到在计算机硬件高速迭代的今日，32 并非是一成不变的数字。

2. 线程与进程

前面的讨论引入了线程和进程的概念。**线程**是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。而**线程**是进程中的一部分，一个进程可以包含多个线程在运行。

需要注意的是，新建一个进程往往耗费的代价更大，而新建一个线程在操作系统中的代价要小的多。因此我们一般会尽可能使用多线程而不是多进程。

同时，同一个进程的不同线程之间可以直接通过三级缓存进行通信，这使得信息交换变得非常容易且迅速。而进程之间通信则非常麻烦，可能会使用 pipeline 通信或直接在磁盘上进行读写，这也是我们使用多线程的一个重要原因。

3. 神经网络

神经网络是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。

神经网络由于它特有的一些性质，可以在某些程度上通过并行计算来帮助它的拟合。

在神经网络上的并行方案一般分为两种：

1. **模型并行**：不同的内核训练模型的不同部分，比较适合需训练参数比较多的计算。
2. **数据并行**：不同的内核训练不

[1] 数据来自：https://www.nvidia.com/

同的数据，比较适合计算数据量比较大的计算。

现代神经网络一般具有三种常见的连接方式，我们用**层**来称呼它们：

1. **卷积层**：使用卷积操作对数据进行处理，让模型能够具有更大更生动的感受野。具有较少的参数和极大的数据量。
2. **池化层**：对当前数据进行降采样，使模型只关注数据比较突出显著的部分。无需训练。
3. **全连接层**：将上一层的所有神经元与下一层的所有神经元全部连接起来，能够据此获取非常丰富的信息。具有极大的参数和较少的数据量。

现代神经网络模型常常使用这三类层组合，来实现对图像的分析与处理。很多神经网路会将卷积层和池化层按某种方案交错排列作为一个组，然后将这个组重复若干次，并在最后将所有神经元摊开，加入若干全连接层进行深入的数据加工与处理。

与其功能相对应，卷积层适合数据的并行处理，而全连接层适合模型的并行处理。在常用的**随机梯度下降算法**中，数据的并行处理可以利用 GPU 的性质非常好地进行处理，而模型的并行的一种简单的策略是将各个内核所计算的梯度取一个平均，并将其作为总的梯度进行反向传播。

综上，在并行调整神经网络时，同时并行处理的数据组数是一个非常重要的超参数，我们一般记其为 batch_size。

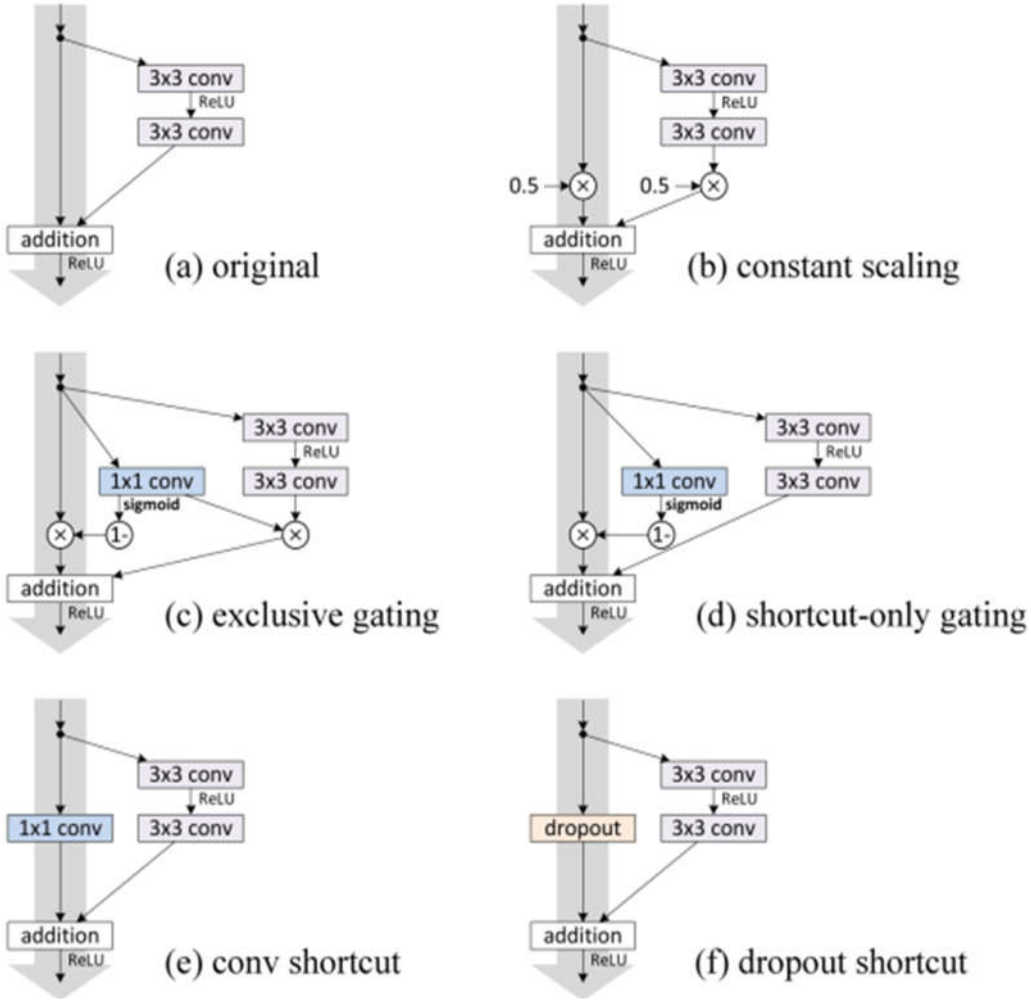


图 1　RESNET 的几种连接方法

4. 随机梯度下降算法

随机梯度下降算法又称 SGD，是计算机科学领域寻找一个方案的最优解的常用算法之一。其算法非常简单，在这里并不做推导。

需要注意的是 SGD 也有一个非常重要的超参数α，我们一般称其为学习率，在本文中记为 learning_rate。

当 SGD 算法不能再有效地对模型在训练集上的准确率进行优化时，我们称模型**收敛**。易发现模型收敛的效率与上述参数的选取息息相关，因此本文设计实验，试图研究这两个超参数对模型收敛的影响。

二、实验

本文利用控制变量法，设计实验讨论使用 SGD 算法时，batch_size 及 learning_rate 的选取对神经网络收敛效率的影响。需要注意的是，本文的是在并行处理这一大背景下，对神经网络模型的收敛情况进行探讨，并不对模型在验证集上的准确度及模型本身的正确性进行讨论，同时也不考虑过拟合等情况。

出于时间成本和经济成本的考虑，本文选取 CIFAR-10 数据集和 LeNet 模型进行实验。

本文所做实验的源代码、模型及 tensor board 数据均已开源到 https://github.com/harmoniikaa/CNN-Comparison。

1. CIFAR-10[2]数据集简介

CIFAR-10 数据集由若干 32*32 的 RGB 图像构成。其有 50000 张训练图和 10000 张测试图，它们被分为十类，每类有 5000/1000 张图片。

由于数据集的图片足够小，因而比较方便并行实验的设计。

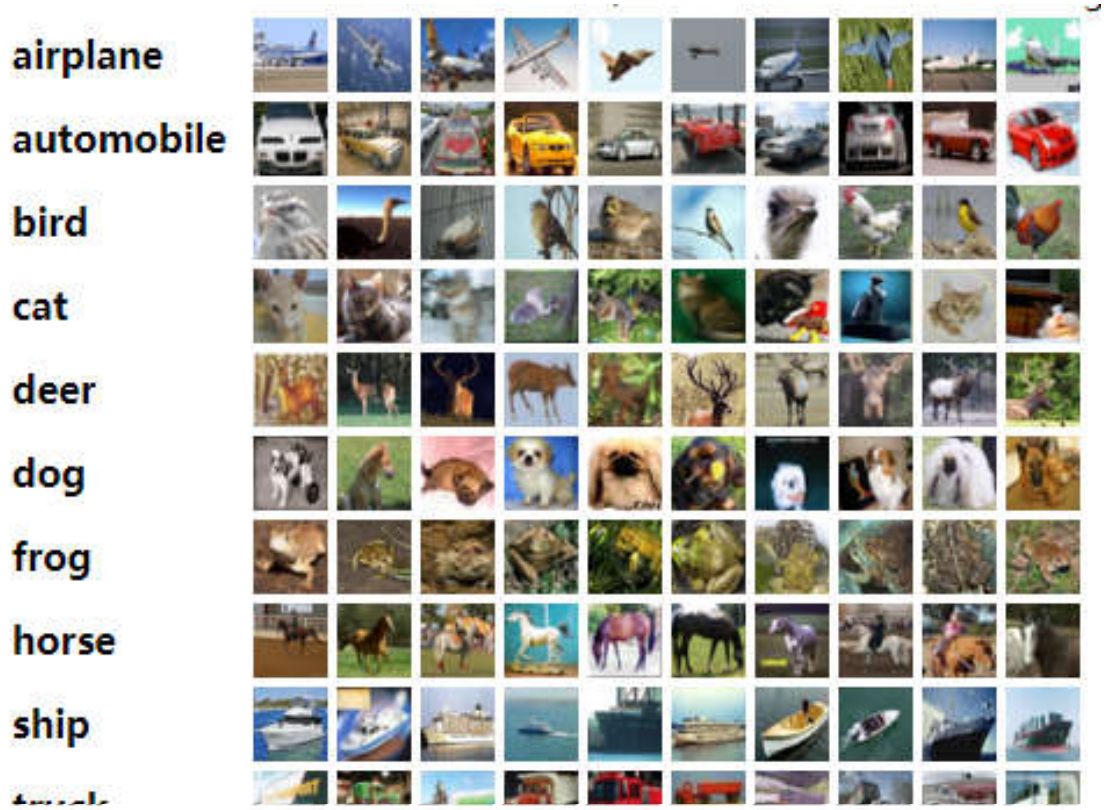


图 2　CIFAR-10 样例数据

[2] 数据集来源：
http://www.cs.toronto.edu/~kriz/cifar.html

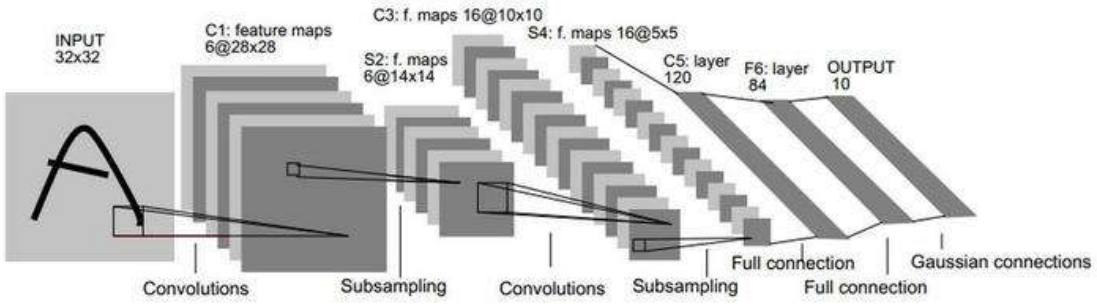


图 3　LeNet 模型图示

2. LeNet 模型简介

LeNet 是一个用来识别手写数字的最经典的卷积神经网络，是 Yann LeCun 在 1998 年设计并提出的。LeNet 的网络结构规模较小，但包含了卷积层、池化层、全连接层，非常适合我们的实验。

要注意的是，在原版本的 LeNet 中，输入数据的维数是(32,32,1)的，然而本文的实验输入数据维数为(32,32,3)。这会影响到模型的准确性，但并不会影响到模型的收敛的情况。

并且为了避免过拟合，本文在 F6 与 OUTPUT 之间的全连接层加入了 50%的 dropout，这是一个有趣的尝试。

3. 实验的无关变量

本文的实验 CPU 使用 Intel Core i7-8750H，内存 16G，GPU 使用单块 GTX 1050 Ti，显存 4G。

神经网络框架选择 keras 2.2.0，其后端选用 tensorflow 1.9.0，编程语言为 Python 3.6.5，CUDA 版本为 9.0。

4. batch_size 相关的实验设计

固定 learning_rate 为 0.01，选取不同大小的 batch_size，运行 500 个 epoch，将 LeNet 模型使用 SGD 对 CIFAR-10 的 50000 张训练图片进行拟合，并观察结果。

本文选取如下的 batch_size：

batch_size					
8	16	32	64	128	256

表格 2　batch_size 的选取

5. learning_rate 相关的实验设计

固定 batch_size 为 128，选取不同大小的 learning_rate，运行 500 个 epoch，将 LeNet 模型使用 SGD 对 CIFAR-10 的 50000 张训练图片进行拟合，并观察结果。

本文选取如下的 learning_rate：

learning_rate			
0.5	0.1	0.05	0.01
0.005	0.001	0.0005	0.0001
0.00005		0.00001	

表格 3　learning_rate 的选取

三、结论

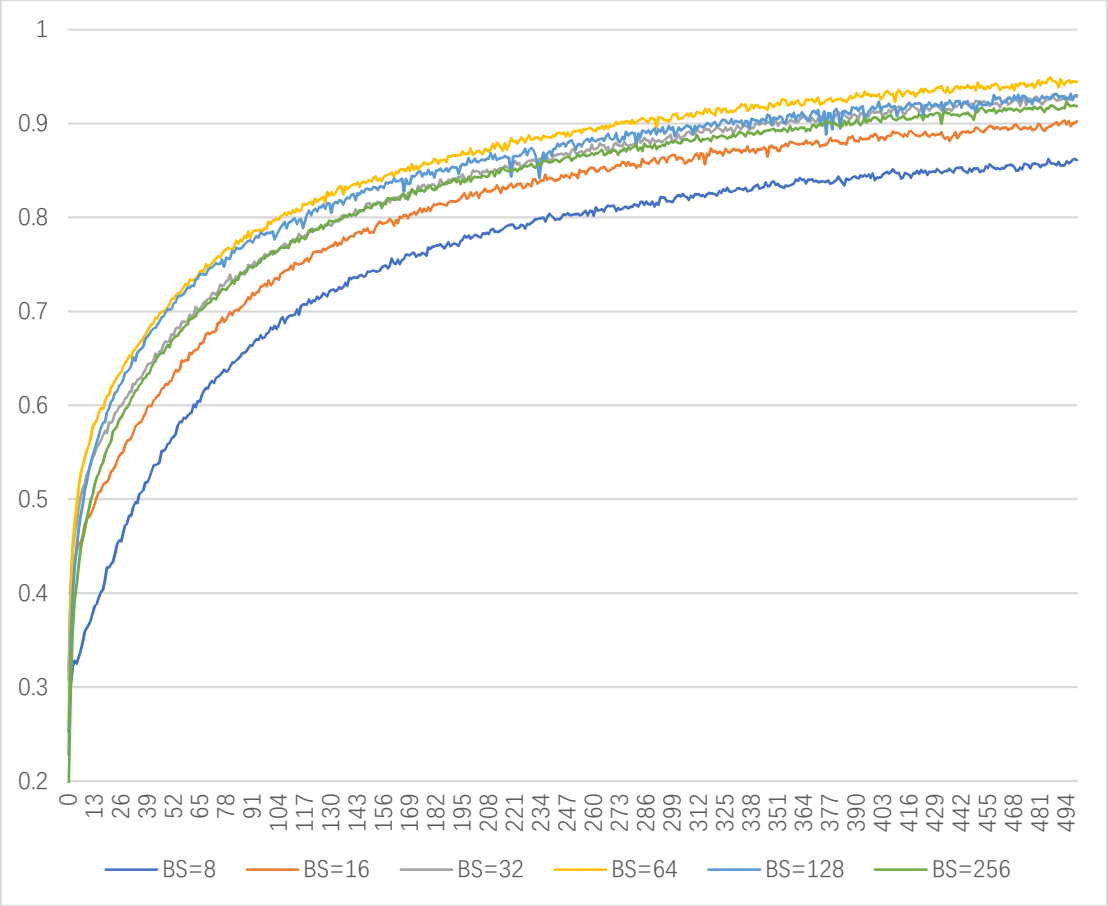
1. 与 batch_size 相关的实验结论

对于不同 batch_size 的模型收敛情况如图表 1 所示。其中纵轴为训练集在该模型下的准确率。

容易发现如下两点结论：

1. 随 batch_size 增大，训练相同的 epoch，训练集在该模型下的准确率呈先上升再下降的趋势。
2. 随 batch_size 增大，模型在前期收敛的速率呈先上升再下降的趋势。

这两点结论非常有趣，可以认为 batch_size 的选取不宜过大也不宜过小。在 LeNet 中我们可以认为选择 64 为宜，而在其他的模型中还需要进一步的实验去进行验证。



图表 1　各 batch_size 的收敛情况

需要注意的是，batch_size 的选取同样影响训练所需时间。batch_size 越大，训练一个 epoch 所需时间就越短。在作者的电脑上，batch_size 取 8 时，一个 epoch 平均需要 31s，而 batch_size 取到 16 时，一个 epoch 只需要 16s，这是成倍的减少，令时间成本大幅度减少，这也是在下一个实验中 batch_size 取 128 而不取 64 的原因。

2. 与 learning_rate 相关的实验结论

对于不同 learning_rate 的模型收敛情况如图表 2 所示。

同样的，也能发现两个结论：

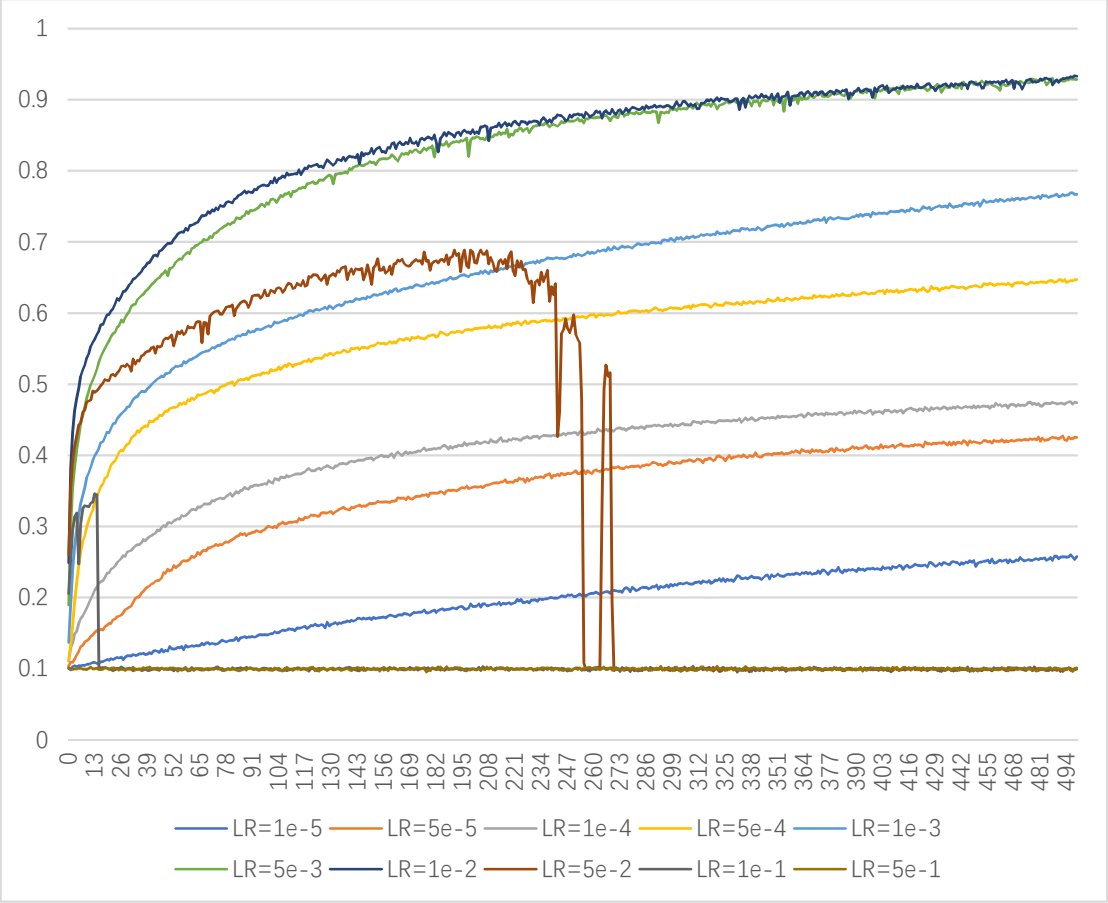
1. 随 learning_rate 增大，训练相同的 epoch，训练集在该模型下的准确率呈先上升再下降的趋势。
2. 随 learning_rate 增大，模型在前期收敛的速率呈先上升再下降的趋势。

然而一个非常有意思的现象发生了，当 learning_rate 过大时，模型非但不存在上述两个结论，甚至不会收敛。故在时间效率与精确程度之间权衡的同时，也需要慎重考虑学习率过大的情况。

四、讨论

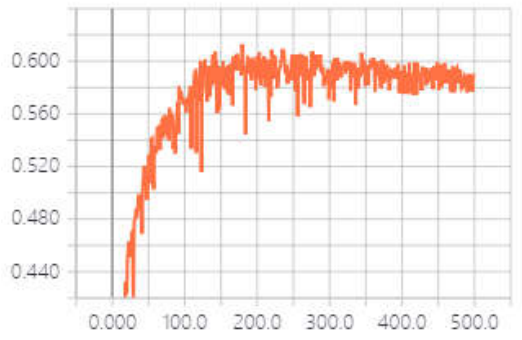
本文的实验时基于 LeNet 模型，在 CIFAR-10 数据集上进行的实验，实验所得数据及结论仅对该模型有效。但是实验方法却具有一定程度的普适性，在使用神经网络进行调参的时候可以参照本实验的方法进行若干预实验，以确定一个较好的参数。

需要注意的是，本文将准确率定义为在训练集上的准确率，衡量的是模型收敛的效率问题，而并不是在测试



图表 2　各 learning_rate 收敛情况

集上的准确率。测试集上的准确率也是一个非常有趣的问题：



图表 3　LR=1e-3 时测试集的准确率

碍于时间成本，本文无法继续深入讨论这一问题。

五、后记

本文为计概听课报告，是作者在听完五节计概课后有感而发，做了一系列相关实验。若是实验操作不够科学，或是结论的分析不够完备，敬请谅解。

由于贫穷的缘故，作者只能选取一个超级小的网络做超级简单的实验。同时由于拖延症的缘故，作者无法完成本来设想的一些实验：

- 把卷积核换成空洞卷积或 depthwise 卷积
- 探究层数对准确率的影响
- 研究测试集上的准确率
- ……

不过作者真的认真做了一系列实验，并从中总结出了一些有用的信息。

希望可爱的助教老师能够多给作者几分，希望胡老师能够发出“我感觉，这位同学是真的懂了”的感慨。

谢谢！