深度学习 Lab1 实验报告

PB19151769　　马宇骁

2022 年 3 月 30 日

摘要：简单描述 fine tuning(dropout、normalization、learning rate decay、residual connection、网络深度等), 最后的 test 准确率，对比，以及 Loss 随着 epoc 改变的曲线。
关键词：深度学习，CNN，ResNet

1　实验目标

使用 pytorch 或者 tensorflow 实现卷积神经网络，在 ImageNet 数据集上进行图片分类。研究 dropout、normalization、learning rate decay、residual connection、网络深度等超参数对分类性能的影响。

2　网络模型代码结构

2.1　CNN

2.1.1　简述

卷积神经网络主要由这几类层构成：输入层、卷积层，ReLU 层、池化（Pooling）层和全连接层。通过将这些层叠加起来，就可以构建一个完整的卷积神经网络。

由于参数共享，每个滤波器包含 $F*F*D$ 个权重，卷积层一共有 $F*F*D_1*K$ 个权重和 K 个偏置。在输出数据体中，第 d 个深度切片（空间尺寸是 W_2*H_2 ），用第 d 个滤波器和输入数据进行有效卷积运算的结果（使用步长 S），最后在加上第 d 个偏差。

在连续的卷积层之间会周期性地插入一个池化层。它的作用是逐渐降低数据体的空间尺寸，这样的话就能减少网络中参数的数量，使得计算资源耗费变少，也能有效控制过拟合。

穿插使用归一化，ReLU，Dropout（在训练过程的前向传播中，让每个神经元以一定概率 p 处于不激活的状态。以达到减少过拟合的效果。），最后全连接完成整个 CNN。

- $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow CONV \rightarrow RELU \rightarrow POOL] * n_1 \rightarrow [FC \rightarrow RELU] * n_2 \rightarrow FC$

2.1.2　实现

根据书上 LeNet-5 和 AlexNet 的方法考虑借鉴其中的算法，使用 5*5 的卷积核，2*2 的池化层，发现一次 train 的准确率太低不足 0.5 个百分点；然后考虑使用 5*5 的卷积层，步长 2，零填充等类似后者的 5 层卷积层 2 层全连接层（AlexNet 为 3 层，但执行速度太慢），结果改进并不显著，只达到 0.7-0.9 的准确率百分点。于是进行多次尝试，例如:

PYTHON 程序显示如下：

```
######################## task 1.1 ##########################
self.relu0 = nn.ReLU()
self.mp = nn.MaxPool2d(2) # 池化之后的size变成(32*32*32)
  # O = （I - K + 2P）/ S +1

self.conv1 = nn.Sequential(nn.Conv2d(32, 64, 5,stride=1,padding=0),
                           nn.Dropout(0.5),
                           nn.BatchNorm2d(64),
                           nn.ReLU(),
                           nn.MaxPool2d(2)) # \rightarrow (14*14*64)
self.conv2 = nn.Sequential(nn.Conv2d(64, 512, 3,stride=1,padding=1),
                           nn.Dropout(0.5),
                           nn.BatchNorm2d(512),
                           nn.ReLU(),
                           nn.MaxPool2d(2)) #  -> (7*7*512)
self.conv3 = nn.Sequential(nn.Conv2d(512,512,2,stride=1,padding=0),
                           nn.Dropout(0.5),
                           nn.BatchNorm2d(512),
                           nn.ReLU(),
                           nn.MaxPool2d(2)) #  (3*3*512)
self.dense = nn.Sequential(nn.Linear(3*3*512, 3*3*512),
                           nn.ReLU(),
                           nn.Dropout(0.5),
                           nn.Linear(3*3*512,200))
#发现这样准确率还是太低了emmm
```

最终在查阅资料-尝试的反复循环中，使用如下 CNN 模型加上助教的第一层，共 6 层网络，其单次 train 的准确率可以到 2.3-8.9 个百分点。

PYTHON 程序显示如下：

```
self.conv1 = nn.Conv2d(32, 64, 2, stride = 1, padding=1)
self.bn1 = nn.BatchNorm2d(64)

self.conv2 = nn.Conv2d(64, 192, 2, stride=1, padding=1)
self.bn2 = nn.BatchNorm2d(192)

self.conv3 = nn.Conv2d(192, 256, 2, stride=1, padding=1)
self.bn3 = nn.BatchNorm2d(256)

self.conv4 = nn.Conv2d(256, 256, 3, stride=1, padding=1)
self.bn4 = nn.BatchNorm2d(256)

self.maxpool = nn.MaxPool2d(3, 2)

# self.fc1 = nn.Linear(192*3*3, 192*3*3)
self.fc2 = nn.Linear(256*3*3, 200)
```

2.2 ResNet

2.2.1 简述

ResNet 是解决了深度 CNN 模型难训练的问题。对于一个堆积层结构（几层堆积而成）当输入为 x 时其学习到的特征记为 $H(x)$，现在我们希望其可以学习到残差 $F(x) = H(x) - x$，这样其实原始的学习特征是 $F(x) + x$。之所以这样是因为残差学习相比原始特征直接学习更

容易。当残差为 0 时，此时堆积层仅仅做了恒等映射，至少网络性能不会下降，实际上残差不会为 0，这也会使得堆积层在输入特征基础上学习到新的特征，从而拥有更好的性能。

2.2.2 实现

由于残差网络学习的更好的特性，因此，在构建补全时，加上助教的一层使用 6 层网络，对于优化单次 train 的准确率就相比 CNN 容易，采用如下 5+1 层残差网络的第一次尝试就突破 2 个百分点大关，就不再继续修改折磨。

PYTHON 程序显示如下：

```
######################### task 2.2 ##########################
self.conv1 = nn.Sequential(nn.MaxPool2d(kernel_size=3,stride=2,padding=1),
                           ResNetLayer(64,64,False),
                           ResNetLayer(64,64,False),
                           ResNetLayer(64,64,False))
self.conv2 = nn.Sequential(ResNetLayer(64,128,True),
                           ResNetLayer(128,128,False),
                           ResNetLayer(128,128,False),
                           ResNetLayer(128,128,False))
self.conv3 = nn.Sequential(ResNetLayer(128,256,True),
                           ResNetLayer(256,256,False),
                           ResNetLayer(256,256,False),
                           ResNetLayer(256,256,False))
self.conv4 = nn.Sequential(ResNetLayer(256,512,True),
                           ResNetLayer(512,512,False),
                           ResNetLayer(512,512,False))
self.avg_pool = nn.AdaptiveAvgPool2d(1)
self.fc = nn.Linear(512, 200)
```

3 训练模型完成及分析

3.1 综述

训练模型要求通过读取 data 数据集里的 train 的 90000 张训练图片进行图片学习，最后预测 test 的 10000 张图片来测试预测的模型的准确率。

考虑到在 CNN 模型单次运行 train 函数在本人所使用的笔记本电脑（HUAWEI matebookX Pro2019，显卡 MX250）所需时间 15 分钟，第一次尝试预测运行了 7 个小时且因为模型优化不佳准确率偏低整个事件过于离谱，因此，采取用 59 元租用两块 gtx1060 的服务器一天来训练模型。

3.1.1 补全及改进

对于助教所提供的 train 的代码框架，发现只判断是否将模型放在 gpu 上训练，于是，考虑修改为使用两块 gpu 进行并行计算加速训练（实际发现并行计算确实能加速大约 180-190%），如下：

PYTHON 程序显示如下：

```
# model = CNN()
model = ResNet()
model.apply(init_weights)

# if gpu:
```

```
#     model.cuda()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)
model.to(device) #用两块1060加速
```

然后预想设置 maxepoch 为 40，学习速率 0.015. 此时，构建一个优化器对象 Optimizer，用来保存当前的状态，并能够根据计算得到的梯度来更新参数；之后再利用 ReduceLROnPlateau 来更新学习率，用 min 模式检测 metric 是否不再减小，patience 用 10 作为不再减小的累计次数，最小的允许 lr 设置为 1e-5，其他的参数均为默认值；最后设置使用交叉熵损失函数作为每次的交叉损失的计算方式。

对于 validate，test 函数，检测准确性，不用.train()，故采取与 train 类似的方式完成。

对于 fit 函数，需要判断是否需要提前退出，即 early_stop_counter 大于 patience 的时候，不再迭代，更新模型权重之后退出循环，返回相应值。期间保留准确性和损失，方差历史。

最后的模型测试之前需要先将模型储存，再继续将单 gpu 改为并行计算进行测试，保留历史。

最后将历史进行作图，得到曲线。

3.2 CNN

使用自己编写的 CNN 模型，将 fit 函数的调用参数中的 maxepoch 设置为 40，单开 GPU 时发现一个单次 trian 需要大约 1.5 分钟，双开并行计算时大约 48 秒。最终，在 40 次迭代之后，发现最后输出如下：

- 29.099999999999998

将模型保存，进行测试，结果如下：

- Test Loss: 0.0496 | Test Accuracy: 28.6700

大于 20 个百分点，可以认为效果不错。将 Loss 和 Accuracy 随着 epoc 改变的曲线以及方差曲线展示如下：

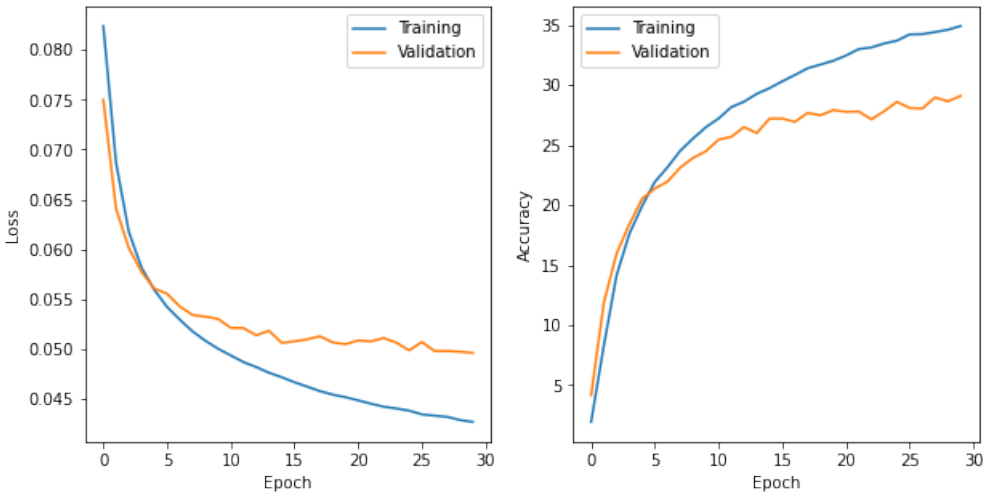

图 1: CNN

3.3 ResNet

由于发现在一次 train 单开 GPU 需要 6 分钟，并行计算也需要 3 分半，因此在 ResNet 网络的 fit 参数中的 maxepoch 设置为 15 减小运行总时间。最终，训练结果最后输出如下：

- 23.400000000000002

可以肯定，残差网络的训练预测模型确实比普通的卷积网络优秀，在训练次数明显小的情况下也能得到较为良好的模型训练结果。模型保存测试结果如下：

- Test Loss: 0.0587 | Test Accuracy: 22.8100

将 Loss 和 Accuracy 随着 epoc 改变的曲线以及方差曲线展示如下：

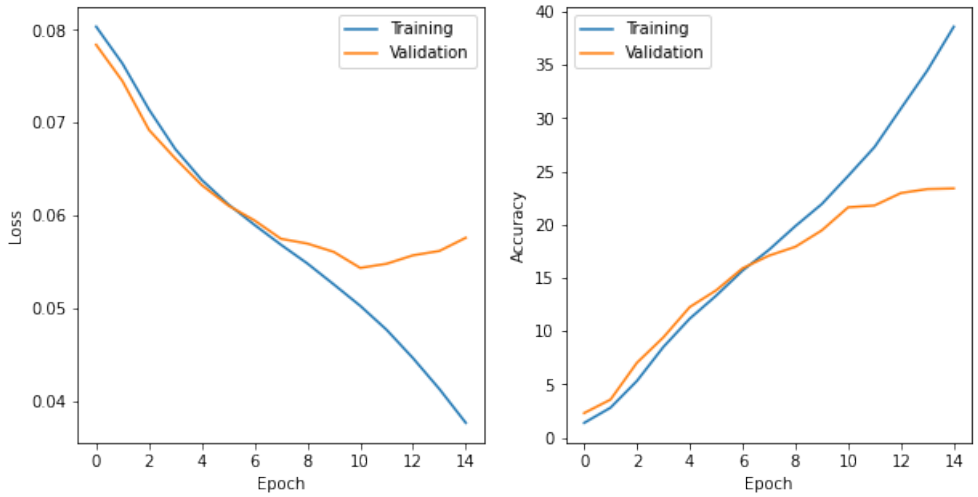

图 2: ResNet