HW4-Report

高易远 精 83 2018010650

1 连接服务器及配置

1.使用 VSCode 中 Remote-SSH 插件连接服务器；

2.终端输入 nvidia-smi 命令查看 GPU 型号及驱动：

NVIDIA-SMI 460.91.03　　Driver Version: 460.91.03　　CUDA Version: 11.2

3.下载 miniconda：

Miniconda3 will now be installed into this location:

/home/gaoyiyuan/miniconda3

4.更换下载源；

5.终端输入 source ~/.bashrc 命令激活 conda，创建虚拟环境，名称为 NMDA_gaoyy；

6.在虚拟环境下补充安装 pytorch 及其它 package。

7.将本地文件克隆到服务器：

scp -r -P 6722 D:_YiyuanData\大四\课程\神经建模与数据分析\hw4\Assignment4 gaoyiyuan@39.106.230.70:/home/gaoyiyuan/work/hw4

8.在服务器上进行训练和测试。

感谢老师和助教提供的计算资源！

2 CnnNet

按照上课讲的思路，搭建了一个卷积网络 CnnNet，结构是：

[(conv-bn-relu)*2-maxpool-dropout]*2-fc,

loss: CrossEntrophyLoss,

optimizer: Adadelta,

scheduler: CosineAnnealingLR

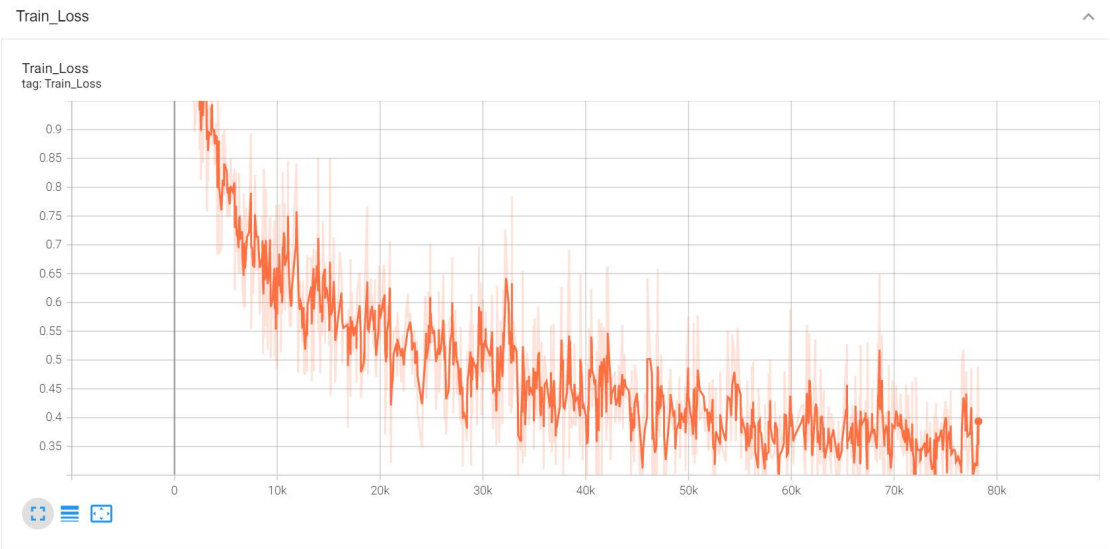
按照之前的经验设置超参数：

batch size: 训练集 128，测试集 100

lr: Adadelta 默认的 1.0

epoch: 200

训练过程中的 train loss 曲线，用 tensorboard 查看如下：



测试结果如下：

测试集上准确率 87.53%，还不错。

```
Test set: Average loss: 0.0037, Accuracy: 8753/10000 (88%)


Test set: Accuracy of airplane: 883.0/1000.0 (88%)


Test set: Accuracy of automobile: 929.0/1000.0 (93%)


Test set: Accuracy of bird : 790.0/1000.0 (79%)


Test set: Accuracy of cat  : 740.0/1000.0 (74%)


Test set: Accuracy of deer : 890.0/1000.0 (89%)


Test set: Accuracy of dog  : 827.0/1000.0 (83%)


Test set: Accuracy of frog : 946.0/1000.0 (95%)


Test set: Accuracy of horse: 887.0/1000.0 (89%)


Test set: Accuracy of ship : 933.0/1000.0 (93%)


Test set: Accuracy of truck: 928.0/1000.0 (93%)
```

3 ResNet-18

实现参考[1]：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

网络结构：前处理，四个 layer（layer1-4），average pool 到 1*1，fc；

注意：

1. 每个 layer 包含 2 个 BasicBlock，每个 BasicBlock 中有 2 次卷积。

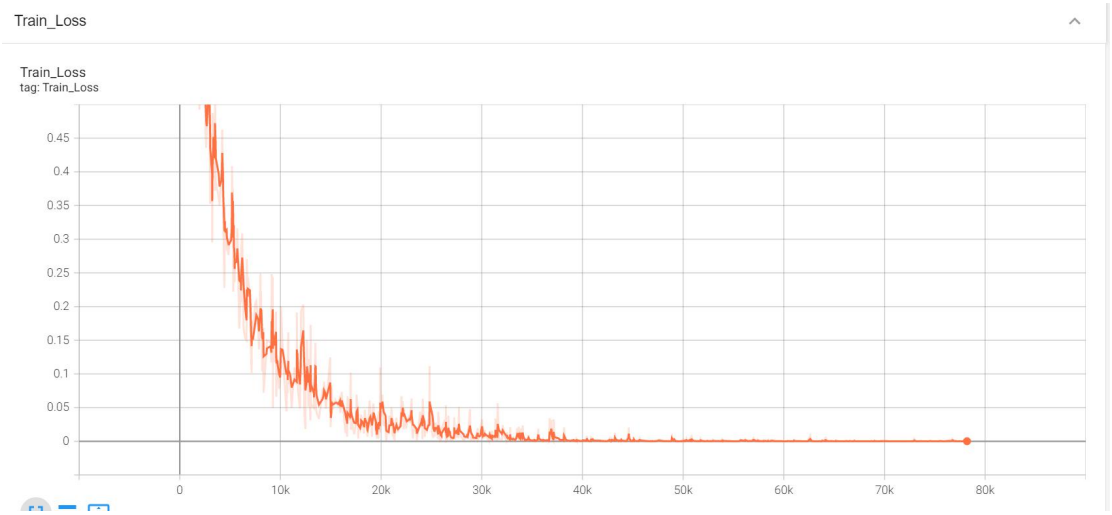
2. 只有 layer2, layer3, layer4 在第一个 BasicBlock 进行下采样（特征图尺寸减半），layer1 没有下采样。

3. 区别是，cifar-10 的图片 size 是 32*32，太小了，所以在这次对 cifar-10 前处理的时候没有按照原本的 ResNet 进行下采样。

超参数的设置与上一部分相同，同样训练 200 个 epoch，在服务器空闲的时候使用 nn.DataParallel 函数用 6 个 GPU 来加速训练。

[1] He, K. , Zhang, X. , Ren, S. , & Sun, J. . (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE.

train loss 曲线：



测试结果：

```
Test set: Average loss: 0.0043, Accuracy: 9383/10000 (94%)

Test set: Accuracy of airplane: 941.0/1000.0 (94%)

Test set: Accuracy of automobile: 974.0/1000.0 (97%)

Test set: Accuracy of bird : 927.0/1000.0 (93%)

Test set: Accuracy of cat  : 859.0/1000.0 (86%)

Test set: Accuracy of deer : 947.0/1000.0 (95%)

Test set: Accuracy of dog  : 894.0/1000.0 (89%)

Test set: Accuracy of frog : 957.0/1000.0 (96%)

Test set: Accuracy of horse: 960.0/1000.0 (96%)

Test set: Accuracy of ship : 964.0/1000.0 (96%)

Test set: Accuracy of truck: 960.0/1000.0 (96%)
```

4 总结

之前使用过 pytorch，所以在报告中没有涉及具体函数的使用，也没有遇到太多困难，本次实验的收获主要是进一步熟练了 pytorch 以及学习如何使用服务器进行训练和测试。

关于调参：根据之前经验，optimizer: Adadelta, scheduler: CosineAnnealingLR 是比较好的选择，batch size 的大小适中，影响也不会太大。学习率采用 Adadelta 默认的 1.0。同时，在准备数据阶段，做了如下的数据增强：padding 后随机裁剪，以及一半的概率水平翻转，可以有效防止过拟合现象。也就没有划分测试集。从 train loss 曲线上看，训练过程比较正常，也考虑到训练一次的时间比较长，就没有再调了。

从结果上看，ResNet-18 要比自己随便搭建的 CnnNet 效果好很多，ResNet 加入了 identity mapping，更有效；最终测试集上准确率达到了 93.83%，效果还不错。

完成后发现也有不足之处，没有保存模型；比较理想的框架是每个 epoch 中，训练一次，测试一次，如果测试准确率高于最高的准确率，更新最高准确率并保存模型，torch.save。

关于代码的其它文字说明写在了 ipynb 文件中。