Lab1-Part2 实验文档

19302010074 张诗涵

一、 代码结构设计

CNNModel：继承了 nn.Module 类，初始化时需要调用父类的初始化函数，使用 nn.Sequential()加入具体的卷积或池化、全连接层。大致结构为一层卷积+一层池化，最后全连接层中使用了两层间隔的线性+激活，最终映射到 12 个参数上。

其中每一层的 inchannel、outchannel、kernel size 等具体参数直接见图：

```
self.layer1 = torch.nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=25, kernel_size=3),
    nn.BatchNorm2d(25),
    nn.SELU()
)
self.layer3 = nn.Sequential(
    nn.MaxPool2d(kernel_size=2, stride=2),
)
self.layer4 = nn.Sequential(
    nn.Conv2d(25, 50, kernel_size=3),
    nn.BatchNorm2d(50),
    nn.SELU(),
)
self.layer5 = nn.Sequential(
    nn.MaxPool2d(kernel_size=2, stride=2),
)
self.fc = nn.Sequential(
    nn.Linear(50 * 5 * 5, 1024),
    nn.ReLU(inplace=True),
    nn.Linear(1024, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 12)
)
```

MyDataSet：继承了 data 包中的 datasets（这是为了便利之后使用 pytorch 框架进行训练，可以直接使用其提供的接口进行 batch、计算 loss 等等），复写其 getItem 与 len 方法，在 getItem 中将 index 指引下的 Picture 实例根据其 path 打开图片，转换为 numpy 对象，label 即此对象的 type 属性值，返回时 pytorch 会自动将 numpy 对象转为 tensor 对象。

Main 函数：实际进行模型学习的部分，首先用 part1 一样的方法区分训练集和测试集为 test_set 与 train_set，将其作为参数构建对应的 MyDataset 类的实例，最终将其初始化为可迭代的 dataLoader 对象，设置训练集上的 batch size 为 25，而验证集上为 1。接着进行训练。

二、 对比实验

基本模型：两层卷积层+两层池化层+一层全连接层

1. 使用 relu 作为卷积层激活函数：
   准确率在 94.2%左右，据大佬同学说 relu 的激活性能在这里并不好使我猜测可能是因为 relu 容易出现不激活的情况，而如果激活了值比较大容易震荡?
2. 使用 leakyRelu 作为卷积层的激活函数：
   准确率在 99.1%左右，效果最佳，个人猜测原因是它避免了 relu 不激活的问题，因

此效果更好，。其突出的特点是训练速度特别快，第一个 epoch 跑完正确率已然高达 95%（相比较而言，sigmoid 在第一个 epoch 跑完后正确率只有不到 50%），它的过拟合问题不如 selu 严重。

3. 使用 selu 作为卷积层激活函数：
   准确率在 97.5%左右。其训练速度也较快，但是也因此过拟合问题尤为突出，在大约 40 次 epoch（甚至更少）之后就不升反降，甚至掉到了 88%。
4. 使用 sigmoid 作为卷积层激活函数：
   准确率在 97.15%左右，没啥特点，效果不好不坏。
5. 使用 tanh 作为卷积层激活函数：
   准确率在 98.5%左右。我认为可能是其相比于 sigmoid 能更接近于-1、1 两个极值（中间更陡、导数更大），因此在激活效果上会优于 sigmoid 的效果。

三、 避免过拟合的尝试

1) Weight decay：
   Weight decay 是课堂上教的一个防止过拟合的方法，根本上是为了防止权重系数变得过大，由于发生过拟合时往往是过于考虑训练集中每个样本的特征，此时会出现的情况便是权重极大、模型十分波折、泛化性能差，因此减小权重的绝对值有助于降低过拟合的影响。
2) Batch Normalization：
   不仅可以提高训练速度、减少震荡（因为本身提供的数据不算多，本次实验 batch 统一设置成了 25），而且由于是多个样本 loss 的平均，消除了某个具体样本的过多影响，使得模型不会太偏重于某一个特征，因此尽管它的主要目的不是防止过拟合、也会有一定的效果，并且在使用 BN 之后就可以不用使用 dropout 的策略、lr 也可以适当增大。
3) Dropout：
   Dropout 是我在网上查到的另一个防止过拟合的方式，它会以一定概率将一些神经元暂时从网络中丢弃，相当于每个 mini-batch 在训练不一样的网络，最终再结合在一起，它要求每个神经元都与随机的伙伴共同合作从而避免了神经元间的联合适用性，提高了泛化性能。
   然而此方法存在的问题是它会大大延长训练时间，并且它在大数据集上效果优秀而在量较小的数据集上性能较差。
4) 动量（Momentum）：
   此方法会综合考虑本次调整方向与上次调整方向，使得调整的角度不那么尖锐、抖动与震荡不明显；同时由于设置惯性，它也拥有在遇到局部最小值时冲出局部最小找到更好的全局最小值的能力。
5) Early drop：
   这个比较好理解，因为在训练集上调整网络参数，除了震荡的情况外、正确率总是在上升，而在验证集就不会如此，如果验证集上 acc 连续几次下降的话即可认为模型已经出现了过度向训练集数据倾斜的过拟合问题，通过 if 判断直接跳出训练即可。

最终考虑到本次实验的特点，比如数据量较小，使用 dropout 的效果较差。最终使用的方法主要为剩下三种，由于 Adam 的 optimizer 自带了 momentum 参数所以自己无需进行实现，使用了 nn 自带的 BatchNorm2d()函数进行 BN，参数为 25（即一开始设定的 batch_size）；而对于 early drop，我猜用了它的理念，一开始会进行统一的 100 个 epoch 训练时会观察一定间隔后在验证集上的正确率，对于那些 100 个 epoch 后仍持续升高的加大训练次数，而

对于在后几十个 epoch 上就出现过拟合问题的模型(如前文提到的 SELU)就减少训练次数、加大验收频率做出更细微的调整。