# 神经网络实验报告

牛远卓

2022/05/16

## 数据集

数据集简介：Omniglot 数据集包含来自50 个不同字母的1623个不同手写字符，如下图所示。

数据规模：共200个类别，每个类别有20个样本，15个作为训练样本，另外5个作为测试样本。每个样本为28\*28。



## 实验过程

### 数据导入

实验先将（200，15，28，28）的训练数据和（200，5，28，28）的测试数据分别通过dataload.py转换成按照类成文件夹的png图片中。再将他们通过torchvision.datasets.ImageFolder加载进网络。

### 神经网络

我用的是Alexnet神经网络。我是对着Alexnet论文中的结构改的。网络结构如下：

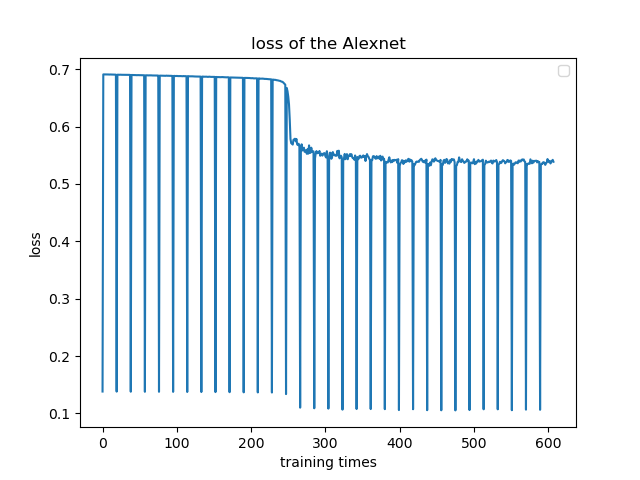
class AlexNet(nn.Module):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.features = nn.Sequential(  
 nn.Conv2d(3, 96, kernel\_size=11, stride=4, padding=0),  
 nn.LocalResponseNorm(size=5,k=2),  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2),  
 nn.Conv2d(96, 256, kernel\_size=5, padding=2),  
 nn.LocalResponseNorm(size=5,k=2),  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2),  
 nn.Conv2d(256, 384, kernel\_size=3, padding=1),  
 nn.ReLU(inplace=True),  
 nn.Conv2d(384, 384, kernel\_size=3, padding=1),  
 nn.ReLU(inplace=True),  
 nn.Conv2d(384, 256, kernel\_size=3, padding=1),  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2)  
 )  
 # 分类器（前面的卷积层已经全部写好，提取出特征了）  
 # AlexNet的卷积层比较简单，层数不深，就直接写在features函数里面了  
 # 特征层操作：（卷积、激活、池化）\*2、（卷积、激活）\*2、卷积、激活、池化  
 self.avgpool = nn.AdaptiveAvgPool2d(6)  
 # 分类器操作：(dropout、全连接、激活)\*2、全连接  
 self.classifier = nn.Sequential(  
 nn.Dropout(),  
 nn.Linear(256 \* 6 \* 6, 4096),  
 nn.ReLU(inplace=True),  
 nn.Dropout(),  
 nn.Linear(4096, 4096),  
 nn.ReLU(inplace=True),  
 nn.Linear(4096, 1000)  
 )  
  
 def forward(self, x):  
 x = self.features(x)  
 x = self.avgpool(x)  
 x = torch.flatten(x, 1)  
 x = self.classifier(x)  
 return x

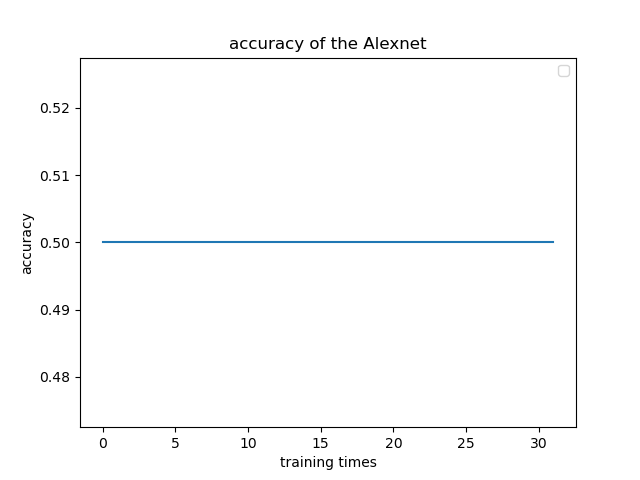
batchsize=32

epoch=32

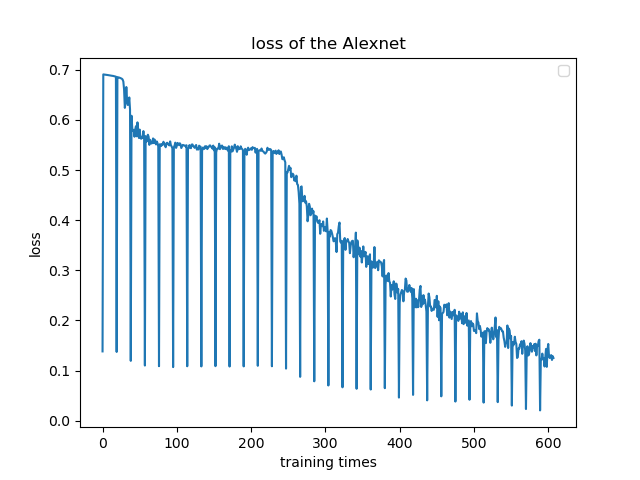
前20个epoch lr=0.001, momentum=0.9之后的 lr=0.0005， momentum=0.9 这里的optimizer用的是SGD。

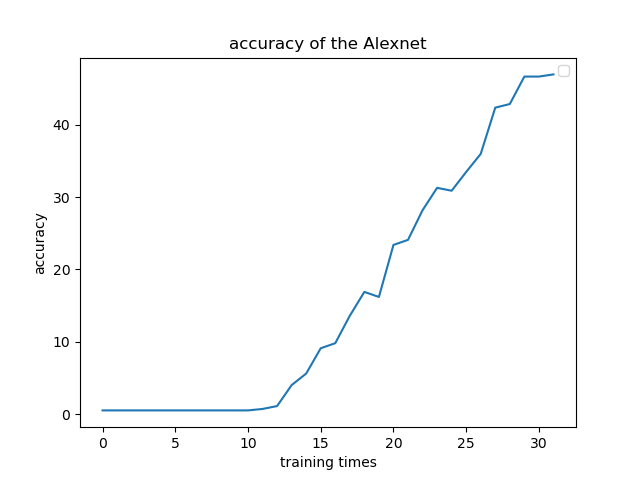
这样的效果很差



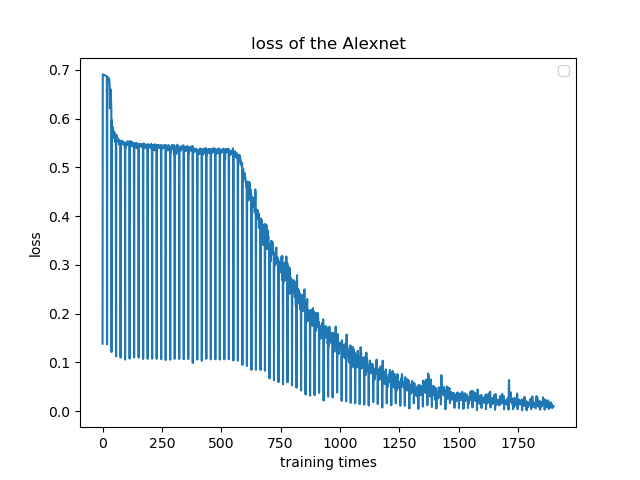


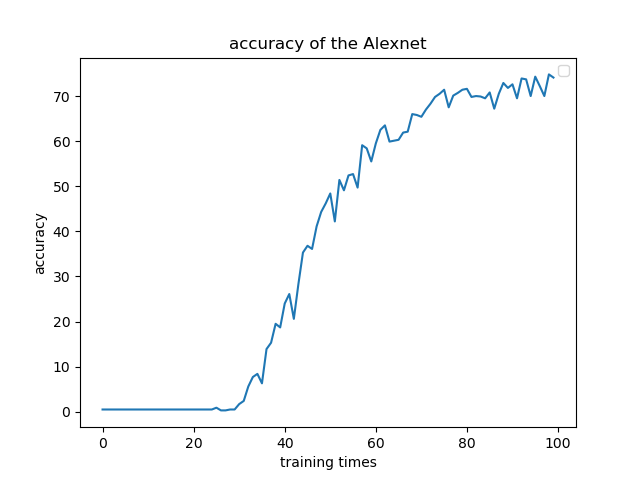
可以看到，正确率根本没变，而且只有0.005，分明就是在猜测。而loss基本不降，只降一点点。这说明至少不是没有反传，也就是不是代码的问题，而是learning rate太小的问题。于是，我将他调大一倍，效果显著。





虽然acc与loss变化幅度增大了，但是acc数值50%不到，还是太小。值得欣慰的是暂时没看出游过拟合的迹象，于是我调大了epoch至100.

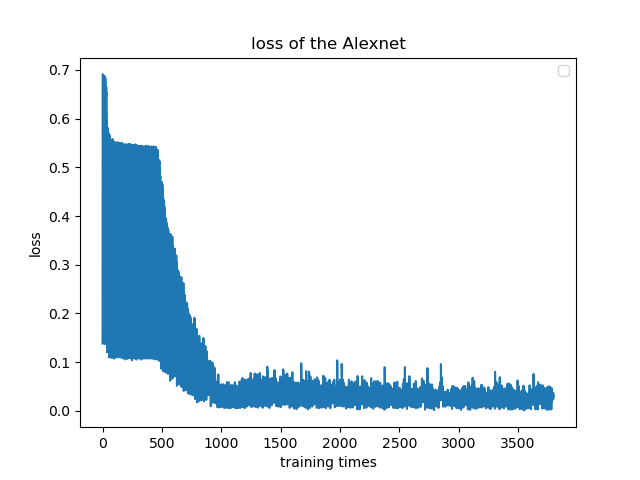


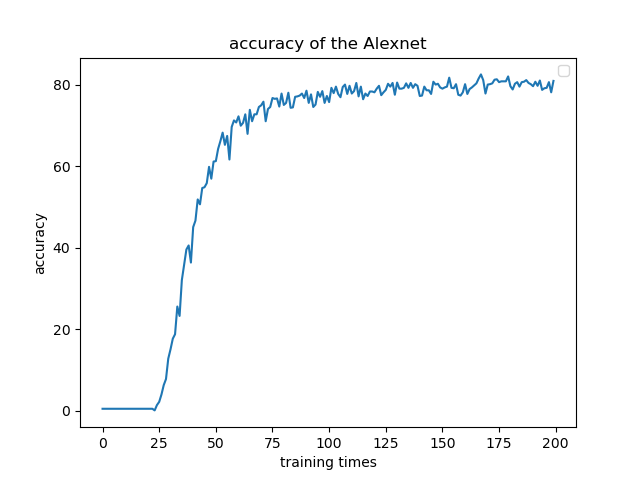


可以看到，结果好了很多，但是acc在70%左右开始浮动，逐渐出现过拟合的问题。于是，我使用了l2 norm和数据增强来降低generation gap。

前100个epoch的lr=0.01, momentum=0.9，之后的50个 lr=0.007, momentum=0.9,weight\_decay=0.0001（weight\_decay是防止过拟合用的），最后50个epoch lr=0.005, momentum=0.9,weight\_decay=0.0001。其中所有optimizer都用了SGD

数据增强用了随机裁剪与随机水平翻转（由于我看大多数据集水平翻转后不破坏图像语义特征，而竖直反转会，所以就没用竖直反转）





最高的时候我的acc达到了81%。

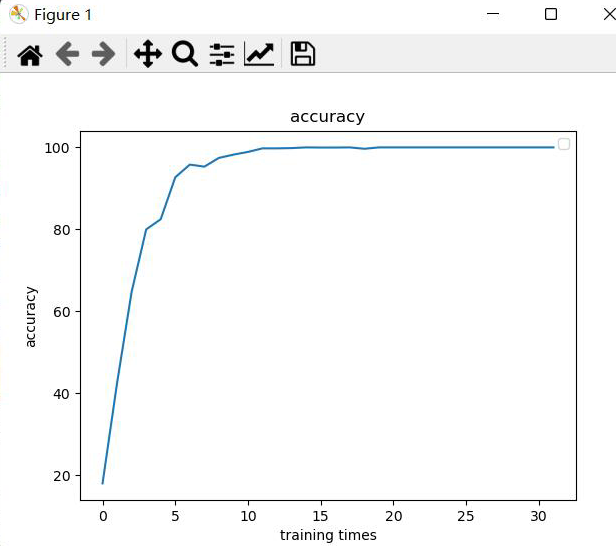
但是我听说有同学在很短时间内就能达到1，恐怖如斯。于是，我重新想了一下整个项目。发现我的数据导入存在巨大问题。原来的数据知识一个4位的数据：训练（200，15，28，28）；测试（200，5，28，28）。然而，由于我是将mat数据集先转成png图片，再通过ImageFolder导入进网络。所以，我的数据集每张图片变成彩色的RGB图像了，也就是（3，28，28）而不是（1，28，28）。这导致了我的数据集被我人工改动了，更具体的说是改复杂了三倍。所以，这对我的神经网络的训练压力加大了。

简而言之，我烦的问题是不应该先将mat数据集改成png图片，再导入网络，而是应该直接将mat数据集导入。所以，我需要重新写一下dataset类。

class MyDataset(Dataset):  
 def \_\_init\_\_(self,data,transform=None):  
 self.data=data  
 self.size=0  
 self.transform=transform  
 self.size=len(data)\*len(data[0])  
# print(len[data])  
# print(len(data[0]))  
 self.name\_list=[]  
 for i in range(0,len(data)):  
 for j in range(0,len(data[0])):  
 self.name\_list.append([i,j])  
   
 def \_\_len\_\_(self):  
 return self.size  
   
   
 def \_\_getitem\_\_(self,idx):  
   
 i=self.name\_list[idx][0]  
 j=self.name\_list[idx][1]  
 img=self.data[i][j]  
 label=i  
 sample={'image':img,'label':label}  
 if self.transform:  
 sample=self.transform(sample)  
 return sample

上面这段是根据dataset类的文档写的。

之后，我使用了resnet18重新训练，什么trick都没用（数据增强，动态调整learning rate等）效果非常amazng啊。



不到15个epoch,测试正确率就达到了100%！！！

## 实验收获