数据分析第三次作业作业报告

一、使用模型进行分类训练

1.1 数据读取以及预处理

本次作业中实现了一个HWDataset类来实现数据的读取。使用助教提供的LoadData函数读取并划分测试与训练集合，默认划分的方式是每一个字符中15张图片作为训练集，5张图片作为测试集。接着使用该划分好的数据集初始化Dataset类并使用pytorch的官方DataLoader作为dataloader，训练中采用shuffle，并且batchsize设置为8。

```
class HWDataset(Dataset):
    def __init__(self, data, label):
        self.data = data
        self.label = label

    def __getitem__(self, index):
        data = torch.FloatTensor(self.data[index].reshape([1,28,28]))

        return data, int(self.label[index])

    def __len__(self):
        return len(self.data)

train_image, train_label, test_image, test_label = LoadData(args.num_classes,
args.num_samples_train,

 args.num_samples_test, args.seed)
train_loader = torch.utils.data.DataLoader(HWDataset(train_image, train_label),
shuffle=True, batch_size=8)
test_loader = torch.utils.data.DataLoader(HWDataset(test_image, test_label),
shuffle=False, batch_size=1)
optimizer = torch.optim.SGD(model.parameters(),lr=0.01, momentum=0.5)
```

实验中主要采用SGD优化器，其中lr针对不同的模型设置有所区别，同时本次试验中均采用50类分类进行试验。下文中会有所介绍。

1.1 CNN模型

本次主要采用的是基于CNN模型的网络结构，网络结构如下所示。分别使用了卷积层，BN层，以及线性FC层。使用SGD优化器，lr=0.01， momentum=0.5的配置下，能够比较快速的收敛，测试集的准确率达到88.0%。

```
class ComplicateModule(torch.nn.Module):
```

```
    def __init__(self):
        super(ComplicateModule, self).__init__()
        self.conv1 = torch.nn.Conv2d(1, 20, 5, 1)
        self.bn = torch.nn.BatchNorm2d(20)
        self.conv2 = torch.nn.Conv2d(20, 50, 5, 1)
        self.fc1 = torch.nn.Linear(4 * 4 * 50, 500)
        self.relu1 = torch.nn.ReLU()
        self.relu2 = torch.nn.ReLU()
        self.relu3 = torch.nn.ReLU()

    def forward(self,x ):
        x = self.conv1(x)
        x = self.bn(x)
        x = self.relu1(x)
        x = F.max_pool2d(x, 2, 2)
        x = self.relu2(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4 * 4 * 50)
        x = self.relu3(self.fc1(x))
        return x

class HWModel(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.compact_model = ComplicateModule()
        self.fc2 = torch.nn.Linear(500, dim)

    def forward(self, x):
        x = self.compact_model(x)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1), x
```

1.2 全连接网络

接着我实现了一个全连接网络用于对比实验。使用上述的同样配置下我发现准确率训练时只有2%，并且始终保持不变，与随机猜的概率一样，非常不正确。分析后发现可能是学习率太大，导致模型不容易收敛。于是采用更小的学习率，使用lr=0.0001后，模型开始正常训练并收敛。在迭代了500个epoch之后，模型最终的准确率只有42%，相比较CNN还是差一大截。可以看到CNN模型中对于图像特征的捕捉能力的确更强一些。

```
class HWModel2(torch.nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.fc1 = torch.nn.Linear(28 * 28, 256)
        self.relu1 = torch.nn.ReLU()
```

```
        self.relu2 = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(256, 128)
        self.fc3 = torch.nn.Linear(128, dim)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1), x
```

二、参数影响对比

2.1 模型结构对比

我们从上文中可以发现，全连接网络更不易收敛，而且最终的准确率远低于CNN的模型。可以发现在50类的图像分类问题上，CNN的模型要优于全连接网络。

2.2 类别数量影响

我们上述的实验均采用50类作为训练参数。直观来看类别数越少对于模型的分类更加容易，模型也更容易取得高的准确率。比如我们以上述的全连接神经网络为例，当类别数为4的时候，全连接神经网络更快的完成了收敛并且最终取得了60%的准确率。而对于CNN网络的模型，取得了100%的准确率。

2.3 学习率的影响

实验中我们发现学习率对于收敛速度有着较大的影响。在CNN的实验中，如果我们使用lr=0.0001的学习率，其收敛速度较慢，我们可以从下面的测试集合上的准确率观察。

```
Loss: 3.8614266395568846  Accuracy: 3.6%)

Loss: 3.814951068878174  Accuracy: 4.4%)

Loss: 3.7583655643463136  Accuracy: 10.0%)

Loss: 3.6815319728851317  Accuracy: 14.8%)

Loss: 3.5631233530044555  Accuracy: 20.0%)

Loss: 3.392041548728943  Accuracy: 30.8%)

Loss: 3.186803535938263  Accuracy: 38.4%)

Loss: 2.8950999636650083  Accuracy: 42.4%)
```

```
Loss: 2.5784620113372805  Accuracy: 48.4%)

Loss: 2.2822532200813295  Accuracy: 52.0%)

Loss: 2.042517492234707  Accuracy: 56.0%)

Loss: 1.8635055210590363  Accuracy: 58.8%)

Loss: 1.7141475659310819  Accuracy: 61.6%)

Loss: 1.580579483151436  Accuracy: 62.4%)

Loss: 1.4692711887285113  Accuracy: 67.2%)

Loss: 1.3700069902017713  Accuracy: 67.2%)

Loss: 1.3005276415087283  Accuracy: 67.6%)

...
```

但是当我们设置lr=0.01时，其收敛速度明显快了

```
Loss: 3.5606781444549562  Accuracy: 10.4%)

Loss: 0.8053249050672412  Accuracy: 82.0%)

Loss: 0.7169053299594563  Accuracy: 86.0%)

Loss: 0.7521638039512734  Accuracy: 86.0%)

Loss: 0.7676123221919527  Accuracy: 86.0%)

Loss: 0.7983079949713309  Accuracy: 86.0%)

Loss: 0.7972716573972233  Accuracy: 86.0%)

Loss: 0.8311308328613143  Accuracy: 85.6%)
...
```

但是对于全连接神经网络而言，当学习率设置为0.01的时候没有办法收敛，只有当学习率设置为0.0001的时候，全连接神经网络才会开始收敛。

2.4 BatchNormalization层的影响

在实验中我发现BN层对于模型的训练有非常大的影响。首先BN层对于模型的训练加速，即模型的收敛速度有着较大的影响。我们在上述的全连接网络中加入BN层后，发现模型此时可以在lr=0.001的设置下正常训练收敛。不加BN层的时候，模型在lr=0.001的情况下无法收敛。

```
加入BN层之后，lr=0.001的设置下的训练log
Loss: 3.910389452934265  Accuracy: 2.4%)
Loss: 3.761312795639038  Accuracy: 13.6%)
Loss: 3.493898091316223  Accuracy: 22.8%)
Loss: 3.0813276352882384  Accuracy: 30.0%)
Loss: 2.6909398422241213  Accuracy: 40.0%)
Loss: 2.4141599695682525  Accuracy: 44.8%)
Loss: 2.171633493423462  Accuracy: 46.4%)
Loss: 2.0833971489965917  Accuracy: 47.2%)

不加BN层的时候lr=0.001的设置下的训练log
Loss: 3.9288706216812135  Accuracy: 2.0%)
Loss: 3.920674015045166  Accuracy: 2.0%)
Loss: 3.919844970703125  Accuracy: 2.0%)
Loss: 3.919600122451782  Accuracy: 2.0%)
Loss: 3.91944766998291  Accuracy: 2.0%)
Loss: 3.9193417501449583  Accuracy: 2.0%)
Loss: 3.919251706123352  Accuracy: 2.0%)
```

同样的，在原本的CNN中我们本身就采用了BN层，如果去掉着一层，仍然保持lr=0.01的原有设置，发现模型无法收敛。实验后发现BN层对于模型的快速收敛还是有着较大的帮助。

三、作业收获

本次作业完成了不同的模型对比和训练，从头写一个模型并开始训练并且取得一个比较好的结果还是比较开心的。全连接网络和CNN的精度对比让人印象比较深刻，比较直观的感受了CNN强大的特征提取和表达能力。同时发现BN层对于模型的训练加速有着比较好的效果。