实验报告——深度学习实验一

7203610508 杨一正

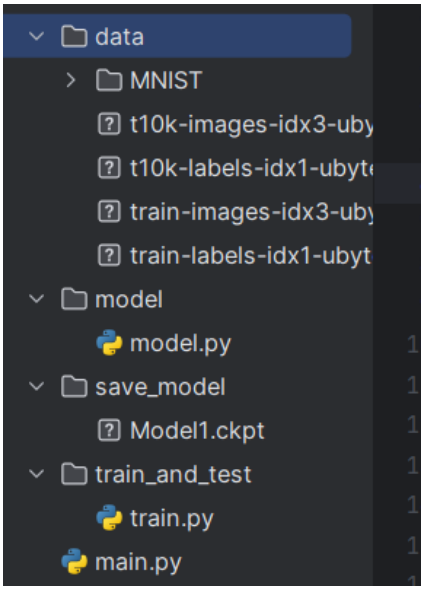
一、 实验环境

操作系统：windows11
Python 版本：conda 管理下的 python3.8
ptorch 及 cuda 版本：pytorch2.0.0 以及 cuda1.7.1
使用 IDE：pycharm

二、 实验过程

1. 简要的代码结构

代码结构如下图所示：



其中，data 目录下存放了所需要的数据集，model 目录下存放的则是对于神经网络模型结构的定义。save_model 目录下则是保存的训练好之后的模型文件。train_and_test 目录下是定义的训练过程以及测试过程以及一些结果可视化的部分。在根目录下有一份代码 main.py，负责调度这些任务。

2. 模型定义

实验中我定义了两个模型，一个是输入层-单隐藏层-输出层模型，另一个是输入层-双隐藏层-输出层模型。其中，为了减弱过拟合的印象，在输入层到隐藏层的部分使用了 p=0.3 的 dropout，意味着有 0.3 的概率神经元不会被激活。同时，激活函数使用了 LeakyReLu，较之 ReLu 函数能够预防神经元死亡的现象。

模型定义的代码如下（仅给出三层网络）：

```
1.  class Model1(nn.Module):
2.
3.      def __init__(self, input_size, hidden_size, num_classes):
4.          super(Model1, self).__init__()
5.          self.input2hidden = nn.Sequential(
```

```
            nn.Linear(input_size, hidden_size),
            nn.Dropout(p=0.3),  # 预防过拟合，采用 p=0.3 的概率不激活
            nn.LeakyReLU()  # 较 ReLu 更好
        )
        self.hidden2output = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = self.input2hidden(x)
        x = self.hidden2output(x)
        return x

    # 方便打印结果
    def __str__(self):
        return "Model1"
```

3. 训练以及测试过程

数据加载使用了 pytorch 自带的 MNIST 数据集方法，代码如下：

```
train_dataset = torchvision.datasets.MNIST(
            root="./data/",
            train=True,
            transform=transforms.ToTensor(),
            download=True
        )
self.train_loader = torch.utils.data.DataLoader(
            train_dataset,
            batch_size=batch_size,
            # 多线程载入，不让 CPU 卡脖子
            num_workers=4,
            shuffle=True
        )
```

损失函数我使用的是交叉熵损失函数，优化器使用的则是 Adam 优化器。在每一个 batch 的训练中，首先取出当前的数据和标签，并根据他们计算出当前的预测值和 loss。之后使用优化器反向传播梯度。最后，可以将这一个 batch 的训练结果统计。

其二者的定义代码如下：

```
# 定义损失函数、优化器
        self.criterion = nn.CrossEntropyLoss().to(self.device)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=learni
ng_rate)
```

测试过程则较为简单，首先使用 no_grad 方法禁用自动求导，提高我们

测试的效率，之后我们直接读取测试数据，根据模型的输出判断正确的个数，最终使用正确个数除以总个数即可得到最终的精确率。

4. 结果可视化部分

结果可视化使用 matplotlib.pyplot 实现。将 epoch 作为横坐标，在当前训练集上的 loss 函数值和准确率作为纵坐标。

5. 设置参数

在这个实验中，我主要想要分析的是学习率以及网络层数对结果的影响，所以我设置了两个网络结构，分别是三层和四层。输入层大小为 28*28，隐藏层大小为 500，如果有第二层隐藏层，那么同样也是 500。学习率拟分别置为 0.001、0.01、0.05、0.1 进行实验。epoch 置为 30，batch_size 置为 100。

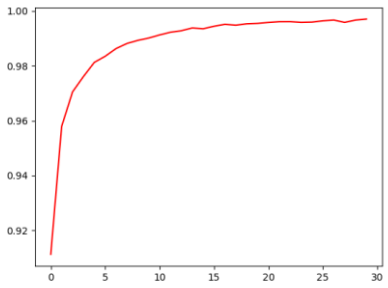
三、 实验结果与分析

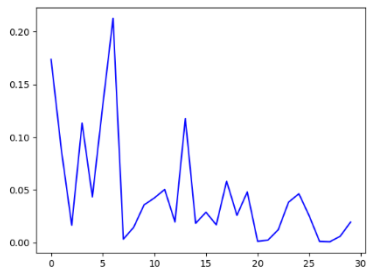
1. 运行时间相关

首次成功使用 GPU 运行代码时，我发现无论是 CPU 还是 GPU 占用率都是很低的，并且运行速度相较于 CPU 运行的代码似乎并没有太大区别。进行搜索之后发现，其实很大部分的效率都被 CPU 的数据读取速度卡住了，于是在 DataLoader 的参数中增加了一行 num_workers=4，意味着同时创建 4 个工作进程进行数据的读取工作，这样一来读取效率大大增加，训练时间也迅速缩短了，从原来的 10s 一个 epoch 变成了 4-6s。

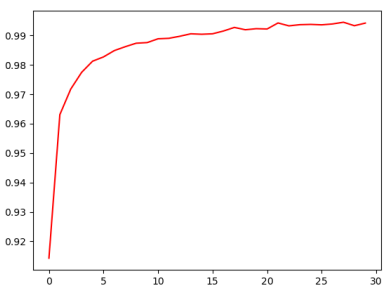
2. 精确度与损失值

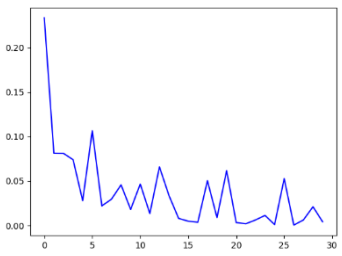
当学习率为 0.001 时，三层网络模型的精确率和 loss 函数值随迭代次数的变化如下图，其测试集准确率为 97.93%：



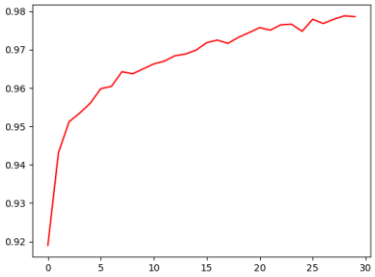
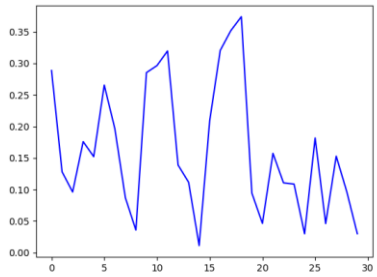


学习率为 0.001，四层网络模型的变化如下图，其测试集上精确率为 97.99%：

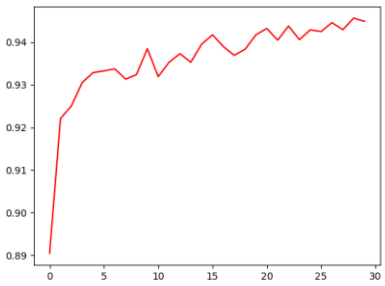
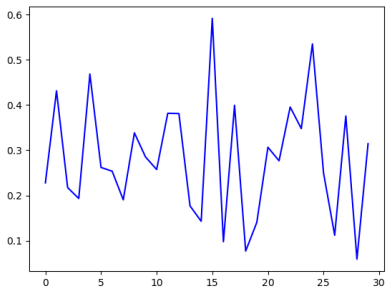




学习率为 0.01，三层网络模型的精确率和 loss 函数值随迭代次数的变化如下图，其测试集准确率为 95.75%：

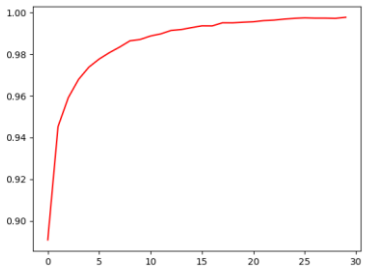
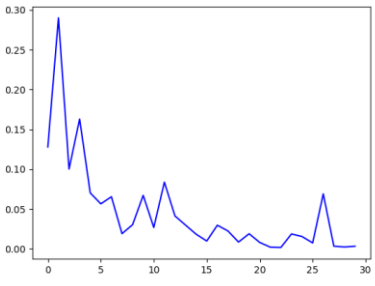



学习率为 0.001，四层网络模型的变化如下图，其测试集上精确率为 93.93%：

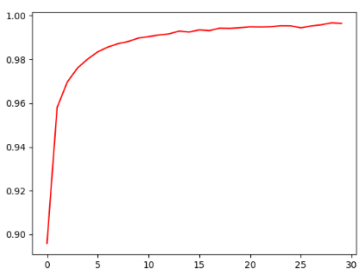



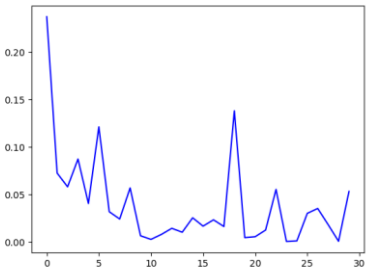
在这时我们发现随着训练两个模型的精确率随着学习次数明显发生了抖动，这也意味着我们没有必要再将学习率增大实验，这时的学习率已经是过高的一个值了。所以接下来我们降低学习率，设置为 0.0005。

学习率为 0.0005，三层网络模型的精确率和 loss 函数值随迭代次数的变化如下图，其测试集准确率为 98.06%：

学习率为 0.0005，四层网络模型的变化如下图，其测试集上精确率为 97.84%：





由上可知，我们的实验条件下找出的最佳模型是学习率 0.0005 的三层网络模型。为什么四层网络结构效果还要更差一些？我认为可能还是产生了过拟合导致的结果。