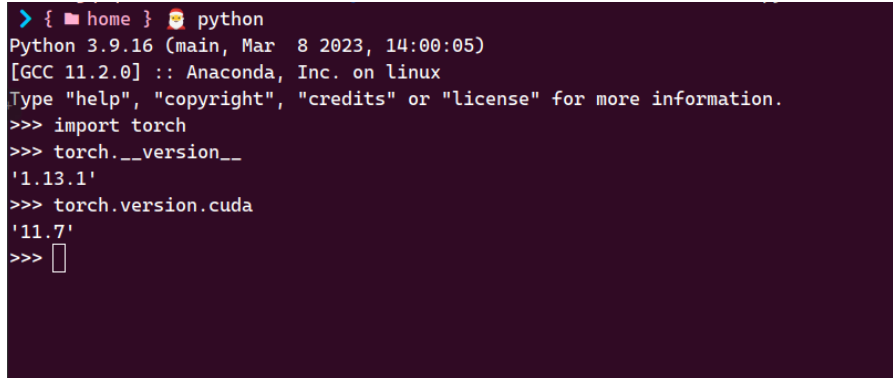


实验 1-深度学习框架熟悉

符兴 7203610316

1 实验环境

Ubuntu 20.04, Python 3.9, PyTorch1.13, Cuda11.7;



```
> { home } python
Python 3.9.16 (main, Mar 8 2023, 14:00:05)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1'
>>> torch.version.cuda
'11.7'
>>> 
```

图 1 Python 环境

2 实验过程

2.1 读取数据

```
# 加载数据
self.trainDataset = datasets.MNIST('./data', train=True, transform=transforms.ToTensor(), download=True)
self.testDataset = datasets.MNIST('./data', train=False, transform=transforms.ToTensor(), download=True)

self.trainDataLoader = DataLoader(
    dataset=self.trainDataset,
    batch_size=self.batchSize,
    shuffle=True,
    num_workers=8
)

self.testDataLoader = DataLoader(
    dataset=self.testDataset,
    batch_size=self.batchSize,
    shuffle=True,
    num_workers=8
)
```

图 2 加载数据

在本次实验中，所使用的是 MNIST 数据集，其中训练集 train 一共包含了 60000 张图像和标签，而测试集一共包含了 10000 张图像和标签。每张图片是一个 28*28 像素点的 0-9 的灰度手写数字图片；

在这里，使用 PyTorch 自带的工具包读取 MNIST 数据集并进行装载，batch_size 设置一个批次是多少张图片，shuffle=True 代表对数据集进行打乱，num_worker 代表使用多线程将数据加载进内存中；

2.2 构建模型

MLP 是多层感知机结构，在 PyTorch 中使用 `nn.Linear` 层进行构建；在这里使用两个 Linear 层，在每个 Linear 层输出上都是用 `LeakyReLU()` 激活函数。

```
class minstModel(nn.Module):
    def __init__(self, inputSize, labelCnt) -> None:
        super(minstModel, self).__init__()
        self.linear1 = nn.Sequential(
            nn.Linear(inputSize*inputSize, inputSize),
            nn.LeakyReLU()
        )
        self.linear2 = nn.Sequential(
            nn.Linear(inputSize, labelCnt),
            nn.LeakyReLU()
        )

    def forward(self, img):
        img = self.linear1(img)
        img = self.linear2(img)

        return img
```

图 3 网络结构

在本次使用中使用交叉熵作为损失函数，PyTorch 中的 `CrossEntropyLoss()` 函数中已经集成了 `softmax`，则在 `forward` 函数中不需要显式对结果做归一化处理；优化器选择 Adam 优化器，并且使用 `MultiStepLR` 对学习率根据 `epoch` 进行动态调整；

```
# 定义模型、优化器
self.model = minstModel(inputSize, labelCnt).to(self.device)
self.optim = torch.optim.Adam(self.model.parameters(), lr=self.lr)
self.criterion = nn.CrossEntropyLoss().to(self.device)
self.scheduler = MultiStepLR(self.optim, milestones=[5, 10, 15, 20, 25], gamma=0.8)
```

图 4 优化器和损失函数

2.3 训练过程

`forward()` 函数是模型进行前向传播的过程，其中 `img` 是图片数据；但是这里使用的是 Linear 层进行接收，需要对 `img` 数据扁平化，即将 `28*28` 的二维转为 `784` 的一维；同时，还需要将标签转化为对应的标签 Tensor，这是一个 `1*10` 的 Tensor，其中标签序号下的值为 `1`；然后通过模型获取图像标签数据，并使用优化器去优化网络中的参数。

```

for id, data in tqdm(enumerate(self.trainDataLoader)):
    # 处理数据
    imgs, labelsRaw = data
    imgs = torch.flatten(imgs,2,3)
    labels = []
    for j in labelsRaw:
        tmp = [0 for _ in range(self.labelCnt)]
        tmp[j.item()] = 1
        labels.append(tmp)
    labels = torch.tensor(labels, dtype=torch.float).to(self.device)

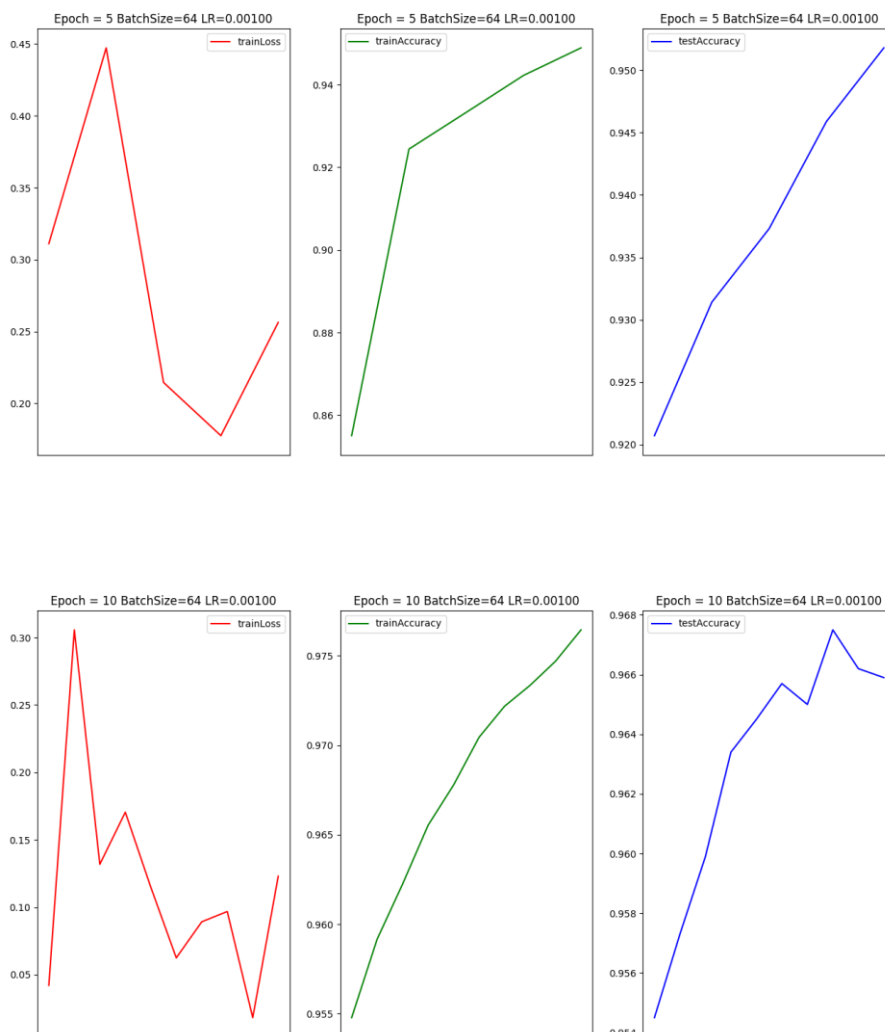
```

图 5 图像数据处理

3 实验结果

在实验过程中，具体分析了网络层数、学习率以及学习轮次带来的具体影响。

(1) 不同 Epoch 对模型学习带来的影响。



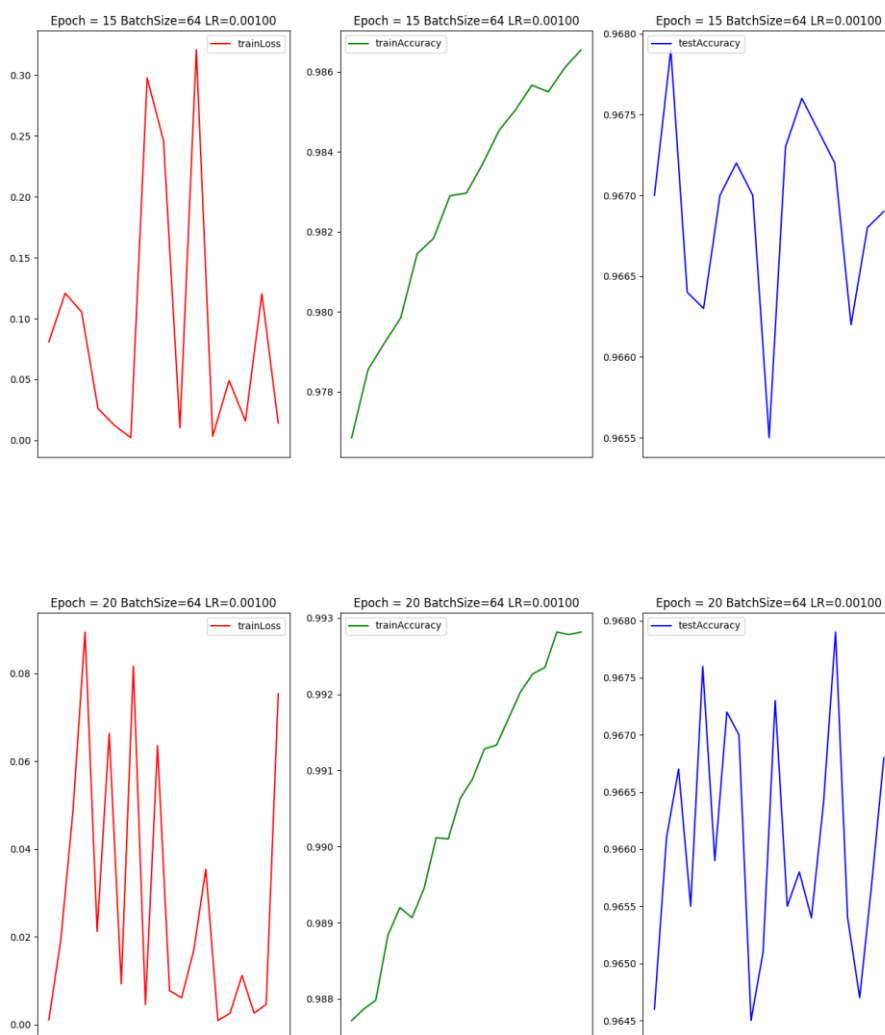


图 6 在网络层数为 2 时，不同 Epoch 的学习效果

从上述的图片中可以看出，随着 Epoch 的增加，训练集上的损失有减小的趋势，但是当到达一定轮次之后，模型的 loss 基本不再下降，是在一定范围内进行波动；同时，结合训练集上的正确率和测试集上的正确率曲线可以看出，虽然训练集的正确率在不断上升，甚至可以达到 0.997 左右，但是测试集上的正确率先升高后逐渐降低；这种情况说明，学习多个轮次，模型可以获得更多的数据，但是当学习轮次过多，模型出现过拟合的现象，就是训练集正确率在不断上升，而测试集的正确率出现了下降的趋势。

(2) 不同学习率对模型学习带来的影响。

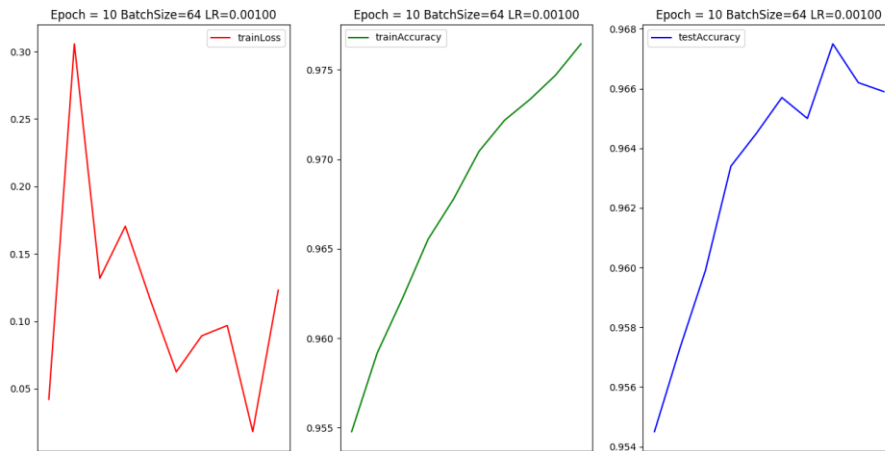


图 7 网络层数为 2，学习率为 0.001 的效果

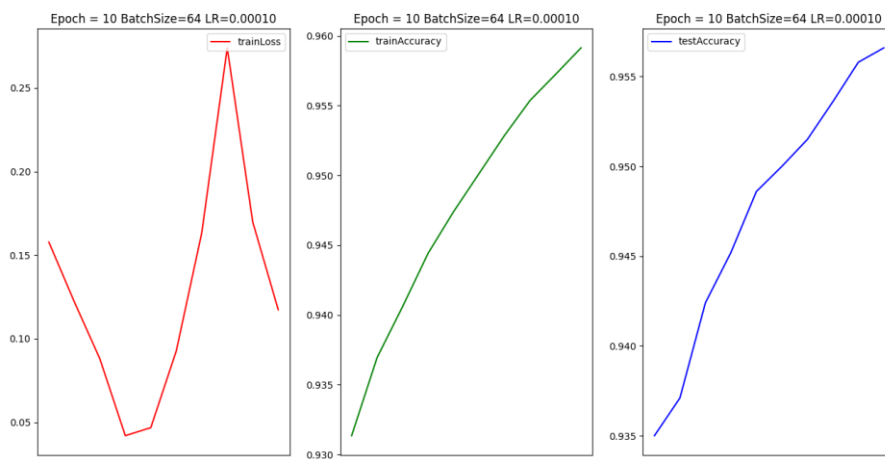


图 8 网络层数为 2，学习率为 0.0001 的效果

从上图的结果可以看到，当学习率过小，模型学习的速度会变慢；图 5 和图 6 同样经过了 10 个学习轮次，0.001 学习率的训练集正确率和测试集正确率比 0.0001 学习率要高；除此之外，0.001 学习率的 loss 逐渐减小，降至 0.1 以下的区间；而 0.0001 学习率的 loss 下降相对较慢；

(3) 不同网络层数对模型学习带来的影响。

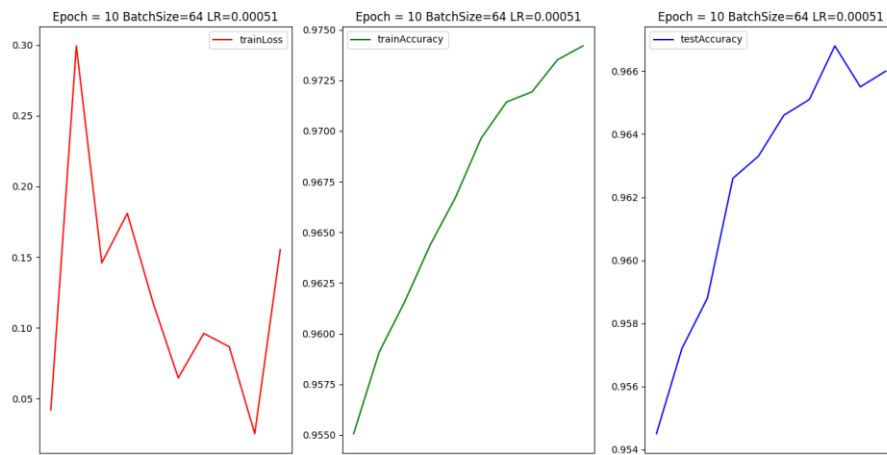


图 9 网络层数为 2 的效果

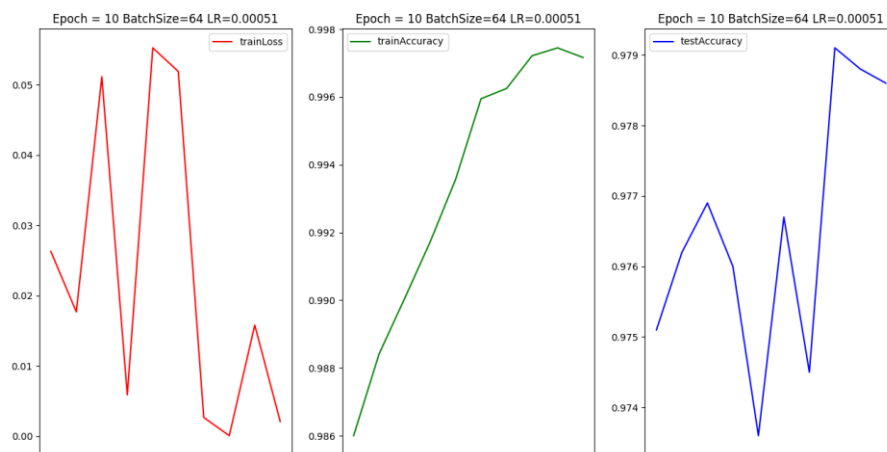


图 10 网络层数为 3 的效果

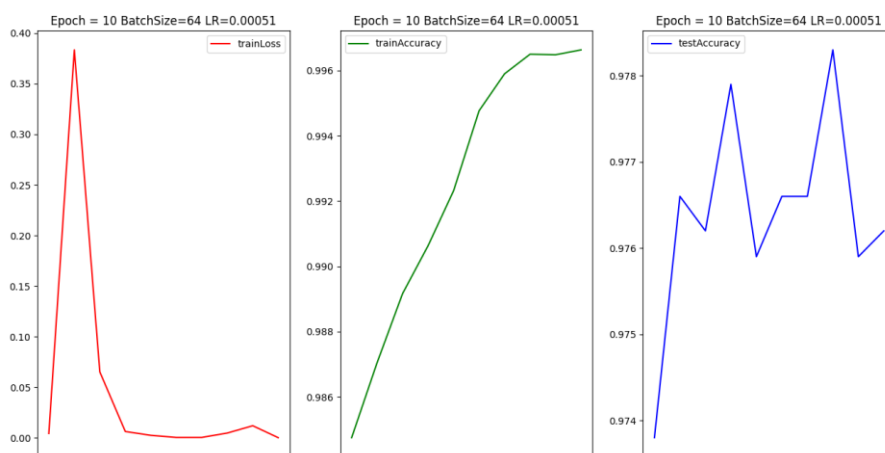


图 11 网络层数为 4 的效果

上图是网络层数分别为 2、3、4，训练 10 轮之后的结果曲线；图 7 和图 8 比较可以发现，当网络层数增加的时候，模型可以学习到更多的信息，网络层数为 3 的模型在测试集上可以达到更高的正确率，为 0.98 左右；而网络层数为 2 的模型在测试集上只有 0.966 左右；除此之外，在训练集上，二者的正确率也有明显差异，明显网络更深的结构，学习效果更好。

但是比较图 7 和图 8，后者网络层数更深，loss 下降更快，loss 更小，但是在量化标准上，二者没有显著的差异；这可能是由于参数量增加，导致了过拟合现象；这也意味着，网络并不是越深越好，更深的网络需要更多的信息；

4 心得与体会

PyTorch 的官方文档写得非常好，很容易理解；同时，他们提供了大量的示例代码和教程，帮助我快速上手。通过这次实验，我对使用 PyTorch 构建一个深度学习模型的过程有了一个基本的了解，并且能够设置一些简单的对照实验，用现有的理论知识对实验现象进行说明和分析。