哈爾濱工業大學

模式识别与深度学习实验报告

实验 一

题　　目　深度学习框架熟悉

学　　院　计算学部

专　　业

学　　号

学　　生

任课教师　左旺孟

哈尔滨工业大学计算机科学与技术学院

2023 年春季

实验一 报告

一、 实验目的

使用 PyTorch 实现 MLP，并在 MNIST 数据集上验证。

二、 实验环境

硬件配置：

CPU：Intel Core i7-8550U

GPU：NVIDIA Quadro P500

软件配置：

OS：Windows 10

IDE：Visual Studio Code 1.70.2

Packages in Environment：

python=3.9.16; torch=1.7.1+cu110; torchvision=0.8.2+cu110; scikit-learn=1.2.2

三、 实验内容

3.1 实验基本参数

由于在实验过程中，可能需要要对多层感知机（MLP）进行超参数的反复调节、改变迭代次数、调整降维维数等，因此可以通过命令行向 ArgumentParser 对象添加参数。

```
parser = argparse.ArgumentParser('MNIST_MLP')
parser.add_argument('--batch_size', type=int, default=20)
parser.add_argument('--PCA_dim', type=int, default=784) # 通过序号选择用哪一个多层感知，用来对比效果
parser.add_argument('--nn', type=str, default='0') # 通过序号选择用哪一个多层感知，用来对比效果
parser.add_argument('--gpu', type=int, default=0) # -1时为CPU
parser.add_argument('--niters', type=int, default=5) # 训练的epoch数
args = parser.parse_args()
```

3.2 MNIST 数据读取与预处理

3.2.1 数据读取

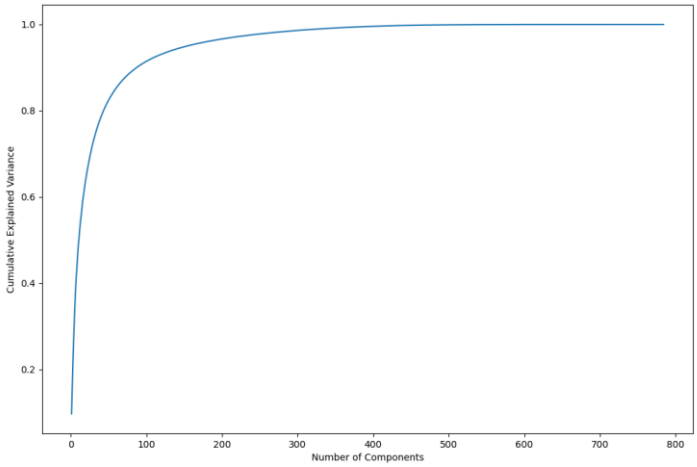
使用 torchvision.datasets.MNIST 函数下载训练集和测试集，并使用 transform 对数据预处理。将数据转化为张量，并做归一化处理。此外，在之后的过程中，对训练集和测试集始终采取同样的方式操作和处理。

```python
# 定义数据转换器
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 加载 MNIST 训练集和测试集
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
```

3.2.2 数据降维

由于在数据集是图片，尺寸为 28*28，且具有较强的相似性。为了提高分类过程的训练速度，使用主成分分析（PCA）方法对原始数据进行降维。为了确定数据应该降维到几维，对训练集降到各维的贡献率进行计算，如下图所示。




```
前595个主成分解释了99.99%的方差
前596个主成分解释了99.99%的方差
前597个主成分解释了99.99%的方差
前598个主成分解释了99.99%的方差
前599个主成分解释了99.99%的方差
前600个主成分解释了100.00%的方差
前601个主成分解释了100.00%的方差
前602个主成分解释了100.00%的方差
前603个主成分解释了100.00%的方差
```

```
前260个主成分解释了98.00%的方差
前261个主成分解释了98.01%的方差
前262个主成分解释了98.03%的方差
前263个主成分解释了98.05%的方差
前264个主成分解释了98.07%的方差
前265个主成分解释了98.08%的方差
前266个主成分解释了98.10%的方差
前267个主成分解释了98.12%的方差
前268个主成分解释了98.14%的方差
```

从中可以看出，降维到 600 维已经可以解释 100%的方差，而降维到 260 维就可以解释 98%的方差。在后续的分类训练过程中也同样能证明，降维到 100 维就可以实现 90%以上的准确度效果，大部分信息对分类任务实际上是冗余的。

3.2.3 数据加载

将降维后的图像数据和原本的标签数据通过 TensorDataset 函数组合，并通过 Datalodaer 将数据分批次，因为神经网络需要按批传入数据和反向传播。其中，命令行传入的 args.batch_size 参数可以作为超参数用来调节训练效果。

```
# 定义训练集和测试集的 DataLoader
batch_size = args.batch_size
train_dataset = TensorDataset(torch.tensor(train_images_pca).to(device), train_labels)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

test_dataset = TensorDataset(torch.tensor(test_images_pca).to(device), test_labels)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

3.3 搭建网络

由于实验要求使用多层感知机（MLP）进行分类，因此全部采用线性层 nn.Linear()和最常用的激活函数 nn.ReLU()组合的神经网络。

损失函数使用多分类问题常用的交叉熵 nn.CrossEntropyLoss()。优化器选用 Adam 优化器。学习率通过 ReduceLROnPlateau 动态调整，如果验证损失在一段时间内不再下降，学习率将会减少。

在实验中，为验证神经网络宽度和深度对分类效果的影响，固定神经元数量为 1000 个，设计了不同深度的 3 个多层感知机（MLP）作为对比，各感知机每层神经元个数如下所示：

nn=0 : 2 隐藏层+1 输出层：PCA_dim --> 500 --> 500 --> 10；

nn=1 : 7 隐藏层+1 输出层：PCA_dim --> 500 --> 250 --> 125 --> 65 --> 30 --> 15 --> 15 --> 10；

nn=2 : 4 隐藏层+1 输出层：PCA_dim --> 300 --> 300 --> 200 --> 200 --> 10.

nn=3 : 4 隐藏层+1 输出层：PCA_dim --> 500 --> 300 --> 100 --> 100 --> 10.

```
class MLP_0(nn.Module):
    # 2隐藏层+1输出层: PCA_dim --> 500 --> 500 --> 10
    def __init__(self, input_dim = 28 * 28, hidden_num = 500):
        super(MLP_0, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_num),
            nn.ReLU(inplace=True),
            nn.Linear(hidden_num, hidden_num),
            nn.ReLU(inplace=True),
            nn.Linear(hidden_num, 10)
            )
        for m in self.net.modules():
            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=0.1)
                nn.init.constant_(m.bias, val=0)
    def forward(self, x):
        out = self.net(x)
        return F.softmax(out, dim=1)
```

示例：nn=0 时的多层感知机 MLP_0

四、 实验结果与分析

4.1 实验基本设置

训练轮数（Epoch）：20

训练设备：GPU

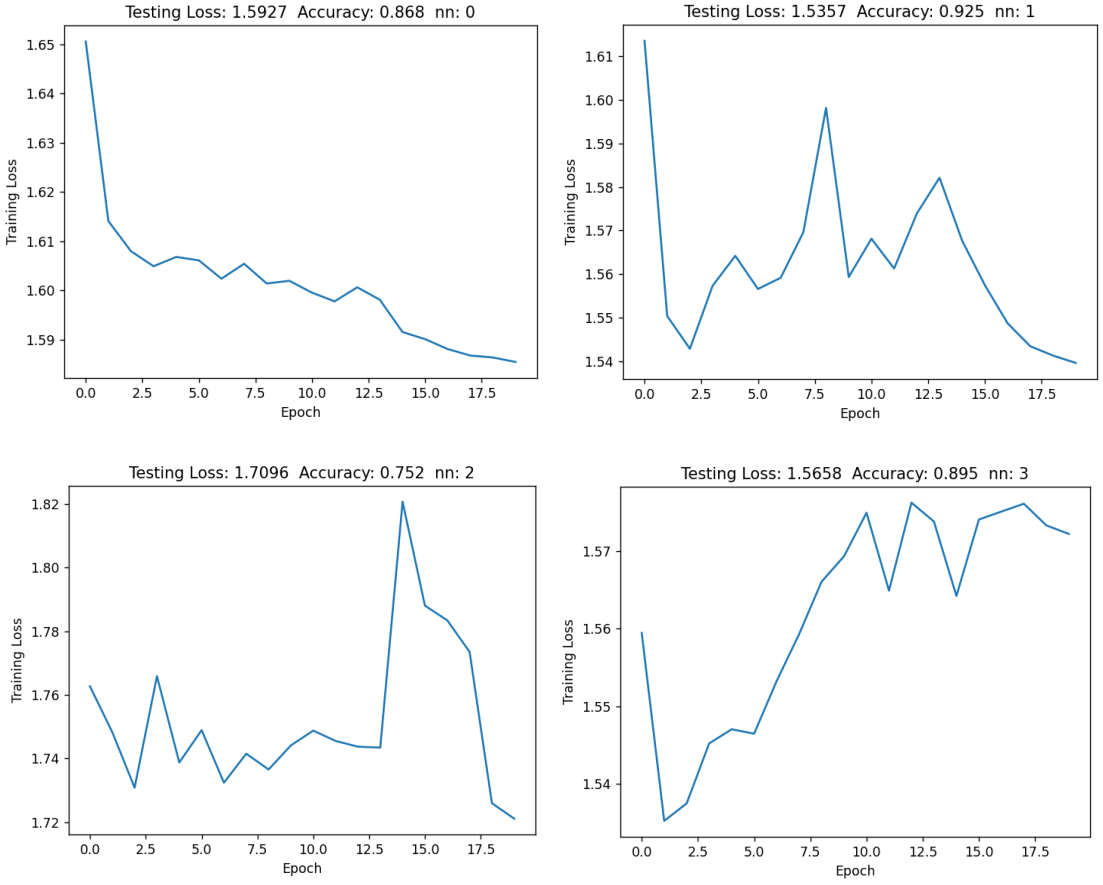
验证频率：20（即训练结束之后再进行验证，中途不验证）

4.2 结果与分析

4.2.1 感知机不同、其他参数相同结果

相同参数：降维到 300 维，Batch_size=20.

不同参数：感知机（图中的 nn 表示选用了 3.3 中的哪一个神经网络）

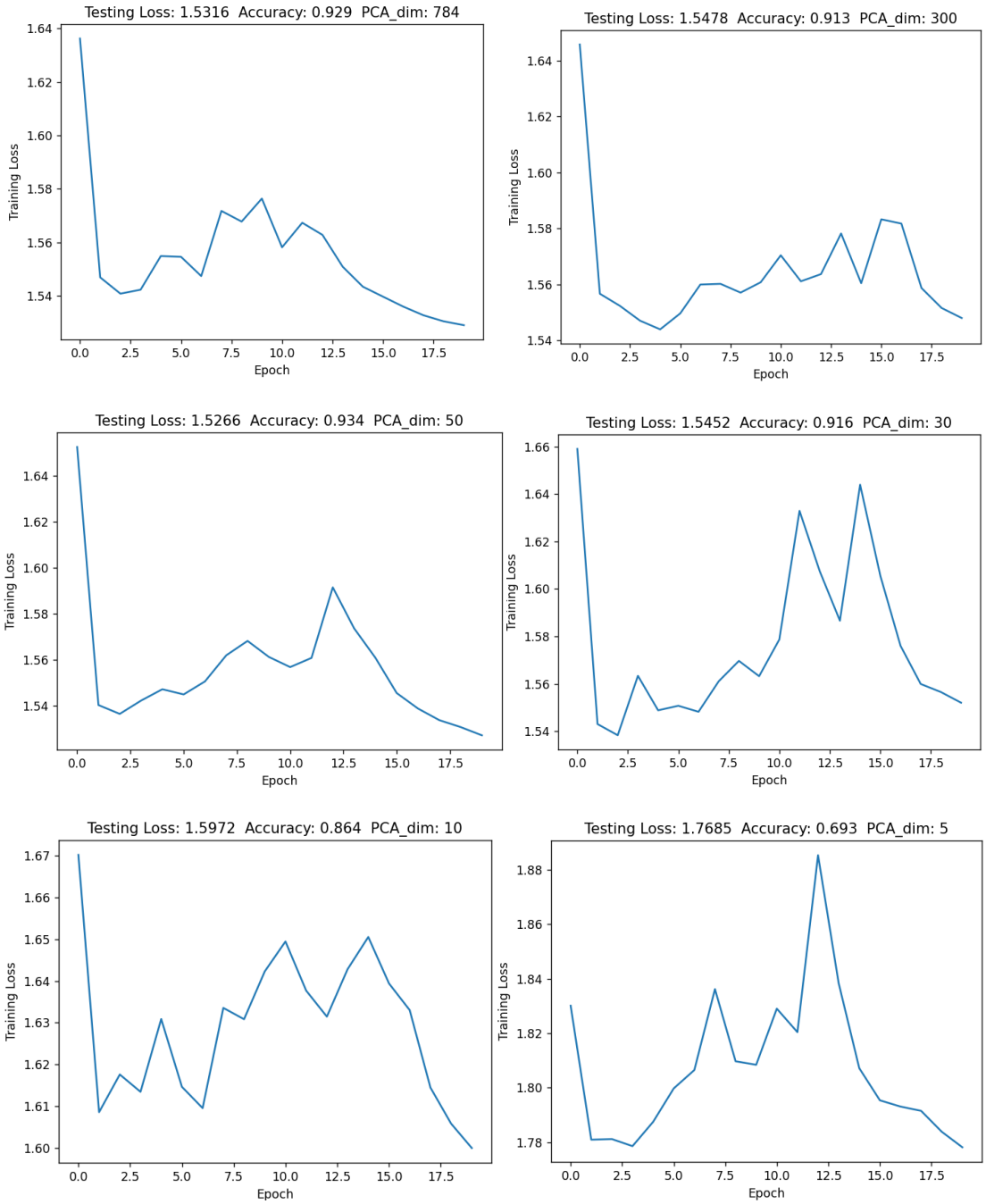


3 个感知机都有 1000 个神经元，但是隐藏层数分别是 2、7、4，在测试集上的准确率分别是 0.868、0.925、0.752 和 0.895. 从中可以看出，相对层数最多的神经网络 nn=1 表现最好，但也并不是随着层数的增加，准确率逐渐上升。对于第一个隐藏层神经元数量最小的 nn=2，准确率也最低，可能第一个隐藏层神经元数量不宜过小。由于算力限制，只迭代了 20 次，不一定每个神经网络都达到了收敛状态、发挥出全部的能力。

4.2.2 降维维数不同、其他参数相同结果

相同参数：nn=1(选用之前实验中表现最好的神经网络)，Batch_size=20.

不同参数：降维维数（PCA_dim 表示将原始数据降维到几维）

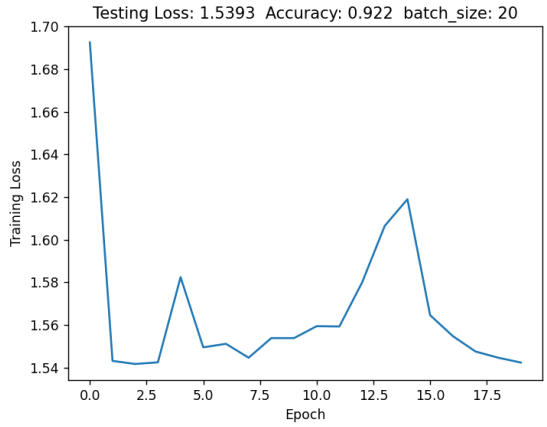


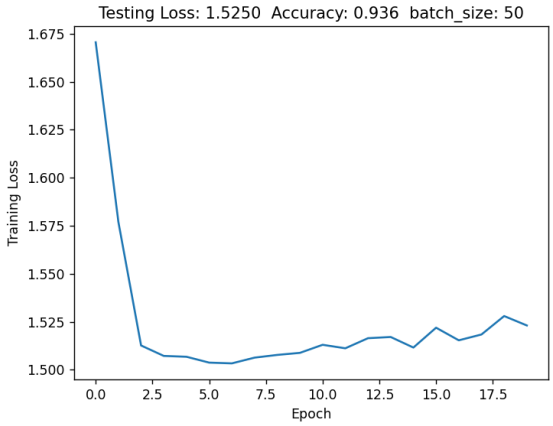
从中可以看出，从原始的 784 维降到 50 维依然有很高（甚至更高）的准确率 0.934，不过继续降维会逐渐降低分类准确率。因此，可以降到 50 维之后再进行分类。此外，在维数大于 30 维之后，每次都能达到 90%以上的准确率，也佐证了 nn=1 有很好的拟合效果。

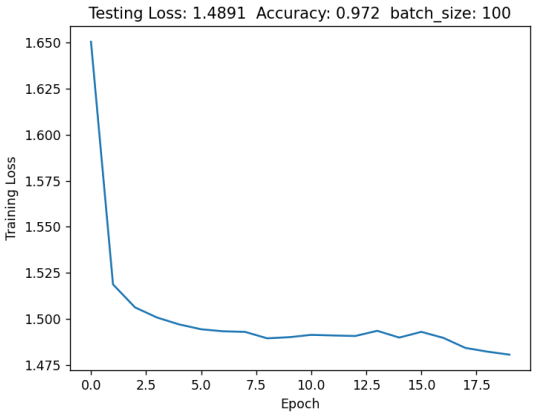
4.2.3 batch_size 不同、其他参数相同结果

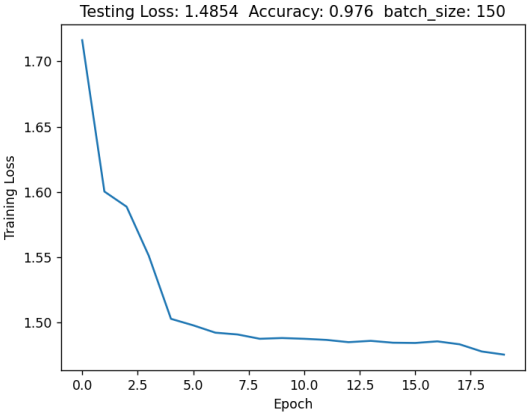
相同参数：nn=1(表现最好的神经网络)，降维到 50 维(兼顾准确率与运算速度)。

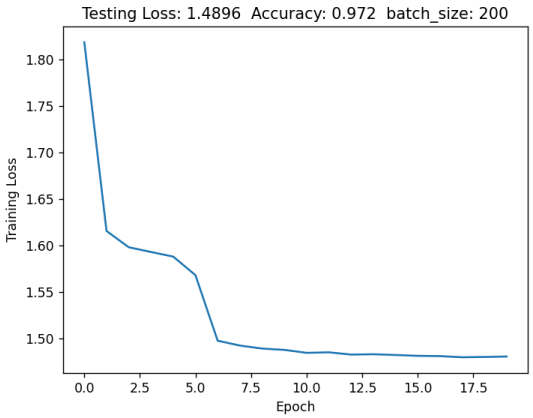
不同参数：batch_size.

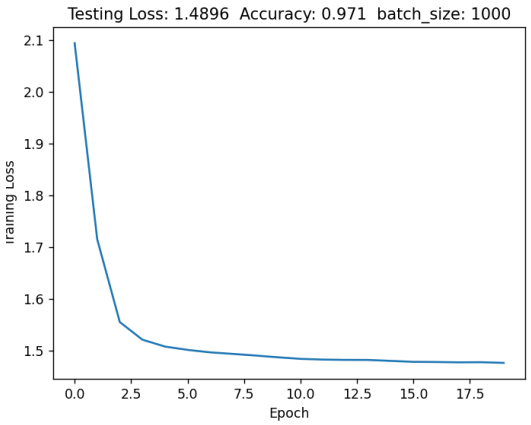


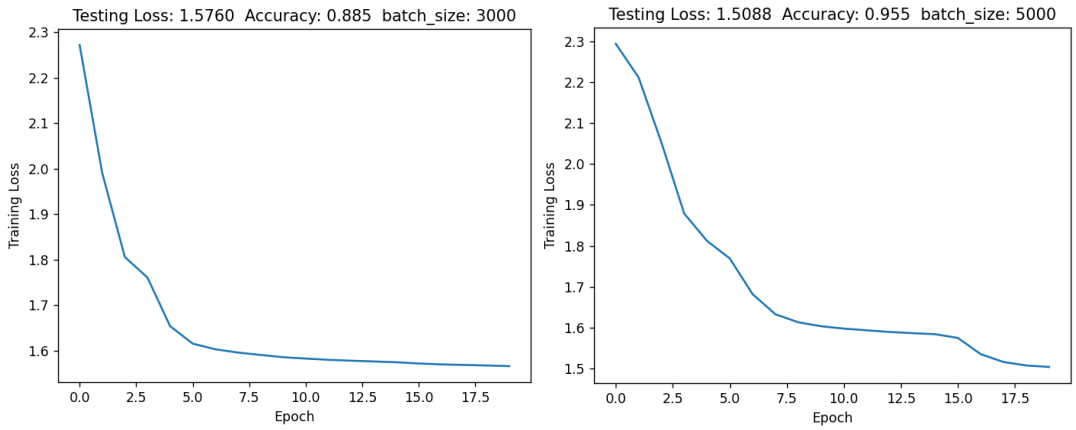












从中可以看出，随着 batch_size 的增大，准确率经历了从上升到下降的过程，在 batch_size 在 100-200 之间取到最大的准确率 0.972。如果 batch_size 过小，训练速度会变慢，同时模型的泛化能力也会降低；如果 batch_size 过大，训练速度会变快，但是单个 epoch 的迭代次数减少了，参数的调整也慢了。

总的来说，可能是由于训练轮数（Epoch）较小的原因，训练得出的 Testing Loss 和 Training Loss 非常接近，没有出现过拟合现象。在训练的过程中，尤其是选用的模型或参数效果不佳时，训练效果对初值较为敏感，Loss 下降不显著。此外，一些情况下也出现了 Loss 震荡和增加的情况，这可能是当前学习率过大导致的。最后，算力不足导致运行较慢、训练轮数较少，也会影响最佳参数的反复调节，降低分类准确率。

五、 结语

通过该实验，我有了一个亲自探究感知机结构和效果机会。之所以把 nn=1 设计成 8 层，是因为比较好奇要是节点少、层数很多会怎样，但是要是平均分配各层神经元数量担心一开始就把数据在神经网络里提取的特征太少，会起到类似降维的作用，因此就把 nn=1 设计成神经元数量越来越少的形式了。没想到效果还是最好的，在 PCA 到 50 维（不 PCA 实在是运行的太慢了）的情况下，还能达到最高 97.6%的准确率。此外，也体会到了超参数 batch_size 和其他参数的相互作用、对训练效果的影响程度，收获颇丰。