实验题目	MLP 实现			实验日期	2022.4.21
班级	1903104	学号	1190300321	姓名	郑晟赫

CS32262 模式识别与深度学习实验

实验报告

一、实验目的

1.配置 PyTorch 环境
2.使用 PyTorch 实现 MLP，并在 MNIST 数据集上验证。

二、实验环境

1.硬件设备：CPU：i7-9750H；GPU：NVIDIA GeForce GTX 1650；16G RAM；512GHD Disk
2.软件系统：Windows 10；Python 3.7.10；PyTorch 1.9.0
3.开发工具：Pycharm

三、实验内容

这一部分主要介绍实验实现过程以及结果分析。

1．数据集分析

MNIST 数据集包含各种手写数字图片，也包含每张图片对应的标签，标签表示这张图片中的数字是多少。MNIST 数据集中的图片是 28X28Pixel 的灰度图，故每一幅图就是 1 行 784（28X28）列的数据，括号中的每一个值代表一个像素。数据集使用过程中由于 PyTorch 已经内置了该数据集的信息，且已经分为训练集与测试集，因此直接使用 PyTorch 提供的接口读取即可。

2．代码编写

2.1 数据读取

使用 torchvision.datasets.MINIST 函数直接下载训练集和测试集，并使用 transforms 函数将读入的数据集转化为 PyTorch 训练需要的 tensor 形式，同时进行归一化处理。在构建训练集和测试集的时候使用 DataLoader 为训练集和测试集分别按照 batch_size 构造训练和测试的序列。

2.2 网络搭建

MLP 的构建过程中采用 4 层网络的结构，其中第一层将 28*28 维向量映射为 512 维，第二层将 512 维映射为 256 维，第三层将 256 维映射为 128 维，第四层将 128 维映射为 10 维向量，这就是最终的输出结果。其中每层都使用 ReLU 作为激活函数。选用 ReLU 的原因主要是首先，使用 ReLU 的时候优化算法的收敛速度比 sigmoid 和 tanh 快；其次，使用 ReLU 作为激活函数的时候计算复杂度较低，不存在指数或是开根号等较为复杂的运算，因此 ReLU 作为激活函数也更适合用于后向传播。

2.3 优化器

本实验的优化器采用 SGD 优化器，可以直接调用 PyTorch 中的 SGD 函数即可，设置的学习率为 0.01，momentum 为 0.5。SGD 是对原始的梯度下降算法的优化。在计算梯度的时候，只使用一个 mini batch 计算，也就是相当于使用 mini-batch 上的梯度去近似原始梯度。因此带来的优势在于效率的提升以及不太容易陷入局部最优解。但是缺点在于当 batch_size 太小的时候很可能带来学习过程中的震荡比较大。其中 momentum 变量可以理解为为优化过程添加一定的惯性，更有概率越过不太好的极小值点。具体数学证明不在此展示。

2.4 损失函数

损失函数采用交叉熵损失来计算预测值与期待值之间的差距。PyTorch 中也继承了这一损失函数，因此直接调用即可。但是需要注意的是 PyTorch 中的 CrossEntropyLoss 函数主要是将 softmax-log-NLLLoss

合并到一块得到的结果。也就是说在计算交叉熵之前已经计算了一遍 softmax，因此在网络构建的过程中需要注意网络的输出层并不需要再使用 softmax 作为激活函数，直接输出全连接层的计算结果即可。

3．实验验证

3.1 实验设置

Batch_size：64

Epoch：18

学习率：0.005

3.2 实验结果

实验结果见下图：

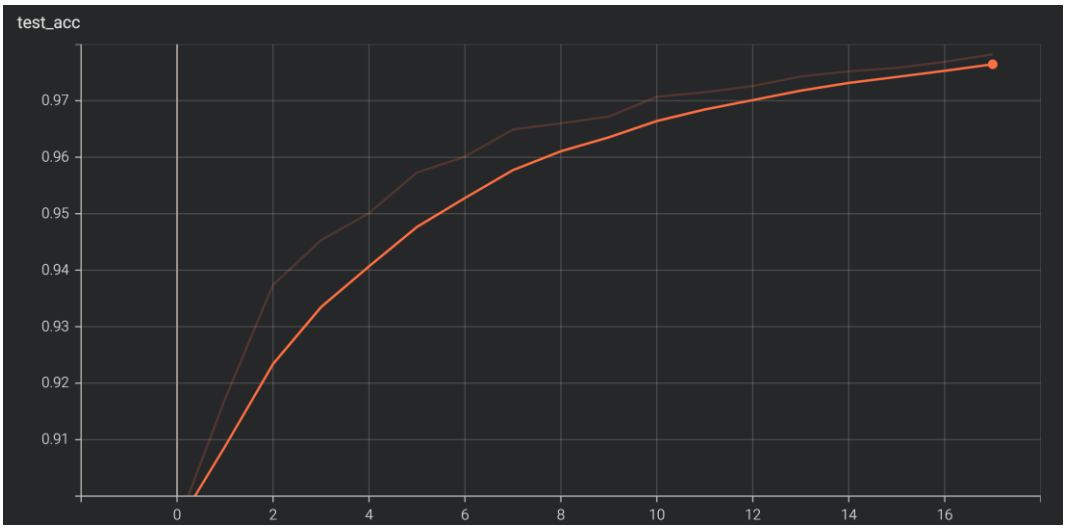

图 1.测试集准确率

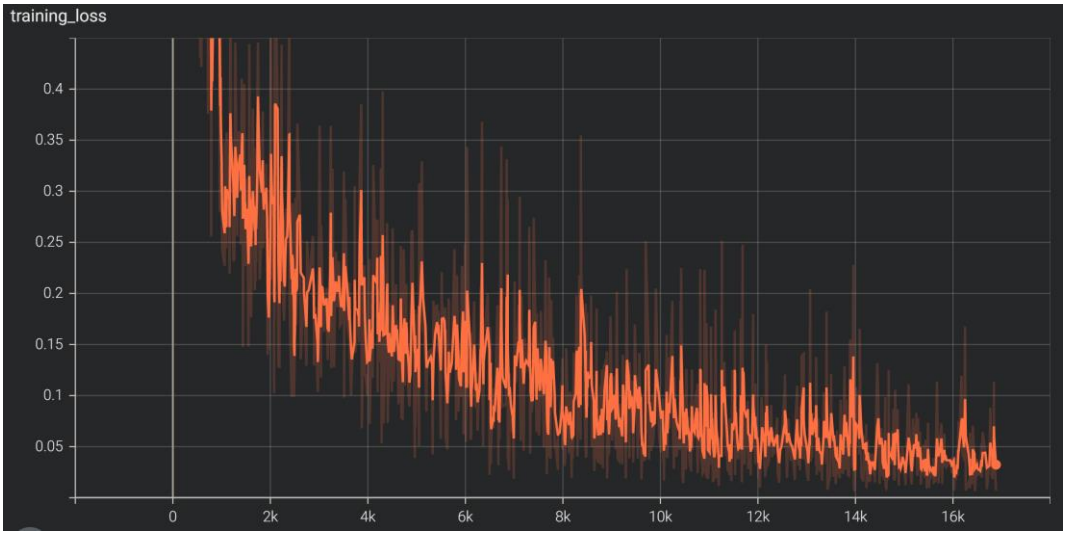

图 2.训练集 loss

从图中可以发现网络在训练集和测试集上的预测都趋近于收敛，最终在测试集上准确率达到 0.978。

四、实验结论（总结实验发现及结论）

通过动手实现 MLP 模型，对神经网络的认识更加深刻，同时对于 PyTorch 的使用有了一定的认识。