实验一：深度学习框架熟悉

姓名：李旻翀
学号：1190200208

日期：May 2, 2022

摘　　要

本实验为模式识别与深度学习课程的实验一：深度学习框架熟悉，主要任务是搭建一个MLP，并在 MNIST 数据集上对其进行训练与评价。在本实验中，我搭建了一个包含 2 个全连接层与 1 个 softmax 层的 MLP，并在 MNIST 数据集上取得了 98% 的准确率。

关键词： 模式识别与深度学习，MLP，MNIST

1　深度学习框架与实验环境

本实验采用的深度学习框架是 Pytorch，整个实验在 Pycharm + Anaconda 环境下完成。Anaconda 是一个开源的 Python 发行版本，可以很方便地安装许多深度学习所需要的包，而 Pycharm 则是一个功能强大的 IDE，可以在其中完成深度学习 python 代码的编写、测试等环节。由于我在大二时参加比赛接触到了深度学习相关的工具，因此在这次实验中无需重新配置环境。配置环境的具体流程比较繁琐，我在当时配置时记录在了个人博客中，可以参考我当时的博客。

2　实验背景知识

2.1　MLP 简介

多层感知机（MLP, Multilayer Perceptron）也叫人工神经网络（ANN, Artificial Neural Network），是一种前向结构的人工神经网络，包含输入层、输出层及多个隐藏层。MLP 神经网络的不同层之间是全连接的，即上一层的任何一个神经元与下一层的所有神经元都有连接。下图即是具有一层隐藏层的 MLP 的结构：

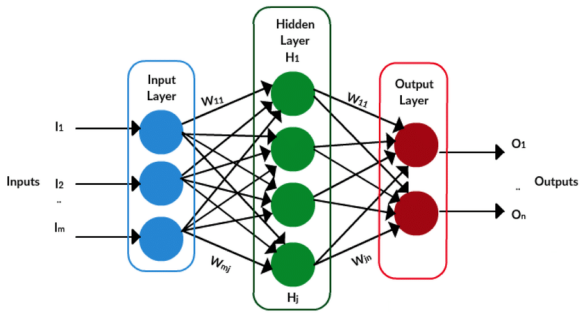

图 1: MLP 网络结构

2.2 MNIST 数据集介绍

MNIST 是一个内容为手写体数字的图片数据集，该数据集由美国国家标准与技术研究所（National Institute of Standards and Technology, NIST）发起整理，一共统计了来自 250 个不同的人手写数字图片，其中 50% 是高中生，50% 来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。在 MNIST 数据集中，训练集一共包含了 60000 张图像和标签，而测试集一共包含了 10000 张图像和标签。该数据集自 1998 年起，被广泛地应用于机器学习和深度学习领域，用来测试算法的效果。

图 2: MNIST 数据集中的图片

3 实验过程

3.1 数据读取

MNIST 是很经典的机器学习数据集，其已经被 Pytorch 收录于数据库中，可以直接通过 torchvision.datasets.MNIST( ) 直接调用，从而省去了繁琐的预处理步骤。

在本实验中，数据读取部分的代码如下：

```
# 加载MNIST数据集
transform = torchvision.transforms.Compose(
    [torchvision.transforms.ToTensor(), torchvision.transforms.Normalize((0.1307,), (0.3081,))])
train_data = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
train_loader = DataLoader(dataset=train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=batch_size, shuffle=True)
```

图 3: 数据读取部分代码

其中，torchvision.transforms.Compose( ) 的功能是通过 Compose 把一些对图像处理的方法集中起来。在这里，该行代码完成了数据转换为张量，以及 MNIST 数据集标准化的操作（0.1307 和 0.3081 是 MNIST 数据集的均值和标准差，由数据集提供方给出）。而通过 Pytorch 提供的 dataloader 方法，可以自动实现一个迭代器，每次返回一组 batch_size 个样本和标签。经过这些操作以后，数据集以样本和标签的形式保存在 train_loader 和 test_loader 中。

3.2 搭建网络结构

定义类 MLP 表示网络模型，在网络结构方面，我采用了两个隐藏层，设定每层的神经元数量为 512，dropout 为 0.2。两层的激活函数我选用 ReLU，在输出层我加了一个 softmax 进行归一化。

具体的代码如下图所示：

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()  # 调用父类的构造函数
        layer_1 = 512
        layer_2 = 512
        self.fc1 = nn.Linear(28 * 28, layer_1)
        self.fc2 = nn.Linear(layer_1, layer_2)
        self.fc3 = nn.Linear(layer_2, 10)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = x.view(-1, 28 * 28)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)

        x = F.relu(self.fc2(x))
        x = self.dropout(x)

        x = F.log_softmax(self.fc3(x), dim=1)
        return x
```

图 4: 网络结构代码

3.3 定义优化器

在本实验中，采用 Adam 优化器，学习率设为 0.001。

通过如下代码定义优化器：

```
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.001)
```

3.4 定义损失函数

本实验中的损失函数使用交叉熵损失函数。

通过如下代码定义损失函数：

```
loss_f = torch.nn.CrossEntropyLoss().cuda()
```

通过.cuda( ) 方法使 loss 函数的计算迁移到 GPU 上，能够一定程度上提升训练速度。

3.5 训练过程

一个 epoch 训练过程大体可以分为以下几个步骤：

1. 将数据输入模型并计算输出。可以通过.to(device) 方法将计算过程迁移到 GPU 上，从而加速计算。
2. 根据交叉熵函数计算损失。
3. 通过.backward( ) 完成反向传播。
4. 更新参数：optimizer.step( )
5. 每完成一遍训练后，用测试集测试模型效果，得到准确率。

整个训练过程，设置 batch size 为 128，进行 40 个 epoch 的训练，在所有训练完成后，保存模型参数为 MLP.ckpt 文件。

4 实验结果与分析

在本机 GPU 为 GTX 1660 Ti 6GB 的环境下，训练 1 个 epoch 大致需要 10s。训练完 40 个 epoch 后，结果如图所示：

```
Epoch:32   Training Loss: 0.0104   Acc:0.9810
Epoch:33   Training Loss: 0.0074   Acc:0.9813
Epoch:34   Training Loss: 0.0094   Acc:0.9797
Epoch:35   Training Loss: 0.0110   Acc:0.9808
Epoch:36   Training Loss: 0.0080   Acc:0.9830
Epoch:37   Training Loss: 0.0057   Acc:0.9814
Epoch:38   Training Loss: 0.0048   Acc:0.9818
Epoch:39   Training Loss: 0.0090   Acc:0.9827
Epoch:40   Training Loss: 0.0069   Acc:0.9791
Finished

Process finished with exit code 0
```

图 5: 命令行运行结果

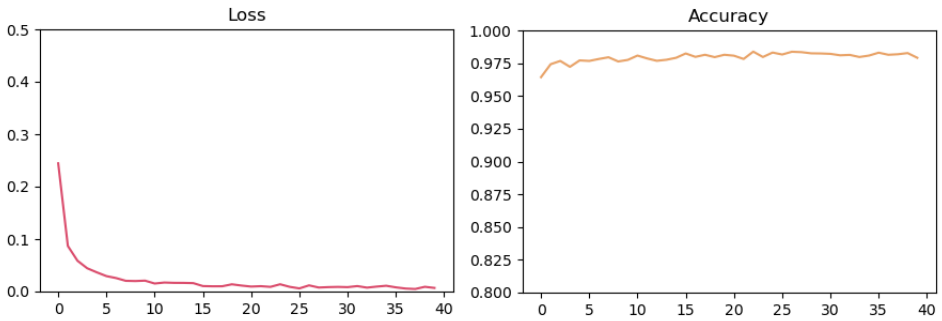

图 6: Loss 与 Acc 曲线

可以看出，在经过仅仅几个 epoch 后，MLP 模型在 MNIST 数据集上便有了比较好的效果，最终的准确率达到 0.98 左右。而损失也在一定数量的 epoch 后保持在 0.01 以下，没有出现过拟合的现象。

5 说明文档

在实验文件夹中，MLP.py 是程序文件。./data 中是下载下来的 MNIST 数据集，./model 中保存着训练好的模型。

在 MLP.py 的 main 函数中只保留 train_model( ) 函数，则可以进行 MLP 模型的训练过程，训练好的模型保存在./model/MLP_best.ckpt。

只保留 load_model( ) 函数，则会载入训练好的模型，并在测试集上验证其效果。效果如下图所示：

```
Load model | Acc:0.9838

Process finished with exit code 0
```

图 7: 测试验证效果