作业题目：使用PyTorch实现Logistic Regression和MLP模型对MNIST数据集进行分类

1. 任务描述

1. 使用PyTorch下载MNIST数据集，包括训练集和测试集。
2. 使用PyTorch分别实现Logistic Regression模型和MLP（多层感知机）模型。
3. 分别使用Logistic Regression模型和MLP模型进行训练，调试参数并获得较好的训练结果。
4. 分别记录训练集的Loss并画出曲线。
5. 针对测试集使用Accuracy评估模型的性能，讨论两个模型的训练结果并给出自己的结论。

2. 编程模块模板

2.1 数据加载模块

```
import torch
from torchvision import datasets, transforms

# 定义数据预处理
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 下载并加载训练集和测试集
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=tran
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=tran

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffl
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64, shuffle=
```

2.2 Logistic Regression模型

```
import torch.nn as nn

class LogisticRegression(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegression, self).__init__()
        # your coding here

    def forward(self, x):
        # your coding here
```

2.3 MLP模型

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(MLP, self).__init__()
        # your coding here

    def forward(self, x):
        # your coding here
```

2.4 训练模块

```
import torch.optim as optim

def train_model(model, train_loader, criterion, optimizer, num_epochs=10):
    """
    训练模型
    :param model: 模型
    :param train_loader: 训练数据加载器
    :param criterion: 损失函数
    :param optimizer: 优化器
    :param num_epochs: 训练轮数
    :return: 训练过程中的损失列表
    """
    model.train()  # 设置模型为训练模式
    losses = []  # 用于存储每个batch的损失

    for epoch in range(num_epochs):
        epoch_loss = 0.0
        for i, (images, labels) in enumerate(train_loader):
            # 将图像展平为向量（适用于全连接网络）
            images = images.view(-1, 28 * 28)

            # 前向传播
            outputs = model(images)
            loss = criterion(outputs, labels)

            # 反向传播和优化
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # 记录损失
            epoch_loss += loss.item()
            if (i + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(train_load

        # 记录每个epoch的平均损失
        losses.append(epoch_loss / len(train_loader))
        print(f'Epoch [{epoch + 1}/{num_epochs}], Average Loss: {losses[-1]:.4f}')

    return losses
```

2.5 测试模块

```
# 测试模型
def test_model(model, test_loader):
    """
    测试模型
    :param model: 模型
    :param test_loader: 测试数据加载器
    :return: 模型在测试集上的准确率
    """
    model.eval()  # 设置模型为评估模式
    correct = 0
    total = 0

    with torch.no_grad():  # 禁用梯度计算
        for images, labels in test_loader:
            # 将图像展平为向量（适用于全连接网络）
            images = images.view(-1, 28 * 28)

            # 前向传播
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)  # 获取预测结果

            # 统计正确预测的数量
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Accuracy of the model on the test images: {accuracy:.2f}%')
    return accuracy
```

3. 评分标准

任务	评分标准	分值
数据加载	正确下载并加载MNIST数据集，包括训练集和测试集	10
Logistic Regression模型实现	正确实现Logistic Regression模型，并能够进行训练	20
MLP模型实现	正确实现MLP模型，并能够进行训练	20
训练过程	能够正确训练模型，调试参数并获得较好的训练结果	20
Loss曲线	正确记录并画出训练集的Loss曲线	10

任务	评分标准	分值
测试集评估	使用Accuracy评估模型性能，并给出合理的讨论和结论	20

4. 讨论与结论

- 比较Logistic Regression模型和MLP模型在MNIST数据集上的表现。
- 讨论两个模型的优缺点，并给出自己的结论。
- 可以进一步探讨如何改进模型性能（如调整超参数、使用更复杂的模型等）。

5. 提交要求

- 提交代码文件（.ipynb格式）。
- 提交实验报告（以markdown格式嵌入到.ipynb文件），包括训练过程的Loss曲线、测试集的Accuracy结果以及讨论与结论。