LSTM 2021213946 韩子坚

实验目的

本实验旨在利用长短期记忆网络（LSTM）构建一个手写数字识别系统，具体目标是通过训练模型识别MNIST数据集中的数字，以此展示深度学习模型在计算机视觉任务中的应用及效果。

实验环境与库依赖

- conda用于管理虚拟环境
- jupyter用于交互式的界面
- PyTorch深度学习框架
- torchvision库用于数据集的加载与预处理

实验步骤与分析

1. 数据预处理

- **方法说明**：通过 `transforms.Compose` 组合了 `ToTensor` 和 `Normalize` 两个转换操作，前者将图像数据转换为PyTorch张量格式并归一化到[0,1]区间，后者进一步对数据进行了标准化处理，确保数据满足神经网络训练的期望分布。
- **重要性**：数据预处理是深度学习项目的关键步骤，它能增强模型的泛化能力，加速模型收敛。
- **代码**

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

2. 数据集加载与数据加载器创建

- **数据集**：MNIST数据集，包含60,000个训练样本和10,000个测试样本，每个样本是一张28x28像素的手写数字图像。
- **数据加载器**：使用 `DataLoader` 创建，实现了批量数据加载、随机打乱训练数据等操作，提高了训练效率和模型的泛化能力。
- **代码**

```python
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

3. 模型构建与配置

- **LSTM模型设计**：定义了一个基于LSTM的神经网络模型，包含两层LSTM结构和一个全连接层，用于从序列数据中提取特征并进行分类。
- **超参数设置**：合理设置了输入尺寸、隐藏层尺寸、LSTM层数和学习率等超参数，这些参数的选择对模型性能至关重要。
- **设备分配**：模型和数据被放置在CPU上运行
- **代码**

```python
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device) # 初始化隐藏状态
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device) # 初始化细胞状态

        out, _ = self.lstm(x, (h0, c0)) # 通过LSTM层
        out = self.fc(out[:, -1, :]) # 取最后一个时间步的输出用于分类
        return out

# 超参数设置
input_size = 28  # 由于我们将图像展平为一维，所以输入大小为28（28x28像素）
sequence_length = 28  # 序列长度，对于MNIST每个序列即为一行或一列的像素点
hidden_size = 128
num_layers = 2
num_classes = 10  # 手写数字共有10类

# 实例化模型并移动到GPU（如果可用）
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LSTMModel(input_size, hidden_size, num_layers, num_classes).to(device)
```

4. 损失函数与优化器选择

- **损失函数**：采用 `CrossEntropyLoss`，它适用于多分类问题，能够直接从模型输出计算类别概率，并与真实标签对比，计算损失。
- **优化器**：使用Adam优化器，因其自适应学习率机制，能够在不同参数上自动调整学习速率，简化调参过程，加速收敛。

- **代码**

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

5. 训练过程

- **训练循环**：模型进行了10个epochs的训练，每个epoch遍历整个训练集，每100个批次打印一次训练状态，包括当前epoch、批次序号和损失值。
- **前向传播与反向传播**：通过前向传播获得预测结果，计算损失，然后进行反向传播更新模型参数，这一过程是模型学习的核心机制。
- **代码**

```python
num_epochs = 10  # 训练轮数

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, sequence_length, input_size).to(device)
        labels = labels.to(device)

        # 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()

        if (i+1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}')
```

实验结果与讨论

- **预期结果**：经过多轮训练，模型应当能够在测试集上展现出较高的准确率，表明其学会了从手写数字图像中识别数字的技能。
- **后续改进**：可通过调整超参数、增加模型复杂度、使用更先进的网络架构或数据增强等手段进一步提升模型性能。

结论

本实验成功展示了使用LSTM网络在MNIST数据集上实现手写数字识别的过程，不仅加深了对深度学习模型构建、训练的理解，也为后续探索更复杂任务提供了基础。