

中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称: Artificial Intelligence

教学班级

专业(方向)

学号

姓名

计科三班 计算机科学与技术 21307185 张礼贤

一、实验题目

用卷积神经网络(CNN)实现手写数字识别, 数据集为MNIST, 网络结构自行设计

二、实验原理

算法原理

根据给定的代码, 利用CNN实现手写数字识别的算法原理可以分为以下几个步骤:

1. 数据加载与预处理:

- 使用 `torchvision.datasets.MNIST` 加载MNIST数据集, 其中 `train=True` 表示加载训练集, `train=False` 表示加载测试集。
- 使用 `transforms.ToTensor()` 将图像转换为Tensor类型, 并进行归一化处理。
- 创建训练集和测试集的数据加载器 `train_data_loader` 和 `test_data_loader`, 用于批量加载数据。

2. 构建卷积神经网络模型:

- 定义 `ConvNet` 类, 继承自 `nn.Module`。
- 在 `ConvNet` 类的初始化函数中定义各个层的结构, 包括卷积层、激活函数、池化层和全连接层等。

- 在前向传播函数 `forward` 中定义网络的前向传播过程，按照卷积、激活、池化和全连接的顺序进行计算。

3. 定义损失函数和优化器：

- 使用交叉熵损失函数 `nn.CrossEntropyLoss()`，适用于多分类问题。
- 定义优化器 `optim.Adam`，使用Adam算法进行参数优化，学习率为0.001。

4. 模型训练：

- 设置训练的总轮数 `num_epochs`。
- 在每个epoch循环中，遍历训练集数据加载器 `train_data_loader`，获取批量的图像和标签。
- 清除梯度，防止梯度累积。
- 进行前向传播，计算模型输出和损失。
- 反向传播和优化，更新模型参数。
- 统计损失和准确率，累计计算每个epoch的总损失和总正确预测数。
- 计算平均损失和准确率。

5. 模型评估：

- 设置模型为评估模式，使用 `model.eval()`。
- 在 `torch.no_grad()` 上下文中，禁止梯度计算，加快推理速度。
- 遍历测试集数据加载器 `test_data_loader`，获取批量的图像和标签。
- 进行前向传播，计算模型输出。
- 根据输出结果和真实标签计算正确预测数。
- 计算测试集的准确率。

6. 绘制曲线图：

- 创建空列表 `losses` 和 `acc`，用于保存每个epoch的损失和准确率。
 - 在每个epoch循环中，将平均损失和准确率添加到相应的列表中。
 - 使用 `plt.plot` 绘制损失和准确率曲线。
 - 设置x轴和y轴的标签和标题。
 - 使用 `plt.legend()` 显示图例。
 - 使用 `plt.show()` 显示绘制的曲线图。
-

代码实现及注释

```
import gzip # 用于gzip压缩和解压缩
import os # 用于操作文件和目录路径
import torch # PyTorch深度学习框架
import torchvision # Torchvision提供了处理图像和视频数据的工具
import numpy as np # 用于处理数值计算和数组操作
import torch.nn as nn # PyTorch中的神经网络模块
import torch.optim as optim # 用于定义优化器
from PIL import Image # 用于图像处理
from matplotlib import pyplot as plt # 用于绘制图像和图表
from torchvision import datasets, transforms # Torchvision中的数据集和数据转换工具
from torch.utils.data import DataLoader, Dataset # PyTorch中的数据加载工具

transform = transforms.Compose([
    transforms.ToTensor(), # 将图像转换为Tensor格式
    transforms.Normalize((0.5,), (0.5,)) # 归一化操作
])

train_data = datasets.MNIST(
    root="./data/",
    train=True,
    transform=transform, # 应用数据转换操作
    download=True
)

test_data = datasets.MNIST(
    root="./data/",
    train=False,
    transform=transform, # 应用数据转换操作
    download=True
)

train_data_loader = torch.utils.data.DataLoader(
    dataset=train_data,
    batch_size = 64,
    shuffle = True,
    drop_last = True)
```

```
test_data_loader = torch.utils.data.DataLoader(  
    dataset=test_data,  
    batch_size = 64,  
    shuffle = False,  
    drop_last = False)
```

pytorch网络输入图像的格式为（C，H，W），而numpy中的图像的shape为（H,W,C）。
故需要变换通道才能有效输出

```
class ConvNet(nn.Module):  
    def __init__(self):  
        super(ConvNet, self).__init__()  
        # 定义第一个卷积层，卷积核大小为3，步长为1，零填充大小为1  
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1)  
        # 对卷积后的特征图进行ReLU激活函数，将负值部分置为0，保留正值  
        self.relu1 = nn.ReLU()  
        # 定义第一个池化层，利用最大池化方法，将图像划分为不重叠的区域，输出区域中的最大值  
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  
        # 定义第二个卷积层  
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1,  
padding=1)  
        # 对卷积后的图像进行激活  
        self.relu2 = nn.ReLU()  
        # 定义第二个池化层，进一步缩小图像尺寸  
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  
        # 经过两次卷积和池化后，得到7 * 7 尺寸图像，通道数为32，映射到隐藏层  
        self.fc1 = nn.Linear(7 * 7 * 32, 128)  
        # 对隐藏层的输出进行激活  
        self.relu3 = nn.ReLU()  
        # 将隐藏层连接到输出层，由于有10个输出 0 ~ 9，故输出为10  
        self.fc2 = nn.Linear(128, 10)  
  
    def forward(self, x):  
        x = self.conv1(x)  
        x = self.relu1(x)  
        x = self.pool1(x)  
        x = self.conv2(x)  
        x = self.relu2(x)  
        x = self.pool2(x)  
        x = x.view(-1, 7 * 7 * 32)  
        x = self.fc1(x)
```

```
x = self.relu3(x)
x = self.fc2(x)
return x
```

```
def smooth(data, window_size):
    window = np.ones(window_size) / float(window_size)
    smoothed_data = np.convolve(data, window, mode='same')
    return smoothed_data
```

```
model = ConvNet()
#使用交叉熵损失函数
criterion = nn.CrossEntropyLoss()
#定义优化器，使用随机梯度下降算法，最小化损失函数
optimizer = optim.Adam(model.parameters(), lr=0.001)
num_epochs = 20
losses = []
acc = []
for epoch in range(num_epochs):
    total_loss = 0.0
    total_correct = 0

    for batch_images, batch_labels in train_data_loader:
        # 清除梯度，确保每个batch的梯度都是新计算的
        optimizer.zero_grad()
        # 前向传播
        outputs = model(batch_images)
        # 计算损失
        loss = criterion(outputs, batch_labels)
        # 反向传播和优化
        loss.backward()
        # 根据梯度更新模型的参数，从而根据损失减少的方向调整参数
        optimizer.step()
        # 统计损失和准确率
        total_loss += loss.item() * batch_images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        total_correct += (predicted == batch_labels).sum().item()

    # 计算平均损失和准确率
    avg_loss = total_loss / len(train_data)
    accuracy = total_correct / len(train_data)
    acc.append(accuracy)
    losses.append(avg_loss)
    # 打印每个epoch的结果
```

```
print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f},  
Accuracy: {accuracy:.4f}")  
  
model.eval() # 设置模型为评估模式  
total_correct = 0  
  
with torch.no_grad():  
    for batch_images, batch_labels in test_data_loader:  
        outputs = model(batch_images)  
        _, predicted = torch.max(outputs.data, 1)  
        total_correct += (predicted == batch_labels).sum().item()  
  
accuracy = total_correct / len(test_data)  
print(f"Test Accuracy: {accuracy:.4f}")  
  
# 绘制误差曲线  
plt.plot(losses, label='Loss')  
plt.plot(acc, label='Accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Value')  
plt.title('Training Loss and Accuracy')  
plt.xticks(np.arange(0, num_epochs, step=1)) # 设置横坐标刻度为整数  
plt.legend()  
plt.show()
```

三、实验结果及分析

实验结果展示

- loss曲线图与acc正确率曲线图：



- 测试准确率： 0.9908

实验结果分析

- 代码中的ConvNet模型采用了经典的卷积神经网络架构，能够对MNIST手写数字图像进行分类。
- 训练过程中的损失逐渐减小，准确率逐渐增加，表明模型在学习过程中逐渐改善性能。
- 在测试阶段，模型在MNIST测试数据集上达到了一定的准确率。