苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 人工智能与知识工程 | | | | | | | 成绩 |  |
| 指导教师 | | 杨壮 | | 同组实验者 | | 无 | | 实验日期 | 2023.11.21 | |

|  |  |
| --- | --- |
| 实 验 名 称 | BP神经网络在手写字符识别中的应用 |

1. 实验目的

理解并应用深度学习基础，特别是在图像识别领域。

学习使用PyTorch框架建立和训练神经网络。

掌握使用MNIST手写数字数据集进行数字识别的方法。

熟悉机器学习项目的完整流程，包括数据预处理、模型训练、评估和预测。

1. 实验内容

使用PyTorch框架建立一个简单的神经网络（NeuralNet 类），用于手写数字识别。

在MNIST数据集上训练和评估该模型。

分析训练过程中的损失和准确率变化。

使用训练好的模型对新的手写数字图像进行预测。

展示和保存预测结果。

1. 实验步骤和结果

该项目的简介在项目文件夹中的readme.md中,readme.md中还有该项目部署和运行的详细过程

图形用户界面, 文本

描述已自动生成

项目结构图:

**MNIST Handwritten Digit Classification**

**|**

**|-- model.py # 定义神经网络结构**

**| |-- NeuralNet (class) # 构建用于MNIST数字分类的神经网络模型**

**|**

**|-- train\_model.py # 包含模型训练和评估的代码**

**| |-- train (function) # 训练模型，返回每次epoch的训练损失**

**| |-- test (function) # 评估模型，打印准确率和平均损失**

**| |-- main execution block # 执行模型训练和测试流程**

**|**

**|-- mnist\_predict.py # 使用训练好的模型进行预测**

**| |-- preprocess\_image (function) # 对新图像进行预处理以适应模型**

**| |-- predict (function) # 使用模型对预处理后的图像进行预测**

**| |-- main execution block # 执行图像预测流程**

**|**

**|-- data/**

**| |-- MNIST/ # 该文件夹用来存储MNIST中的数据**

**|**

**|-- examples/ # 该文件夹中用来存储100个样例图片**

**|**

**|-- predict\_images/ # 该文件夹中用来存储需要识别的图片**

**|**

**|-- docs/**

**| |-- MNIST手写字符识别实验报告.docx # 本次实验的word实验报告**

**| |-- MNIST手写字符识别实验报告.pdf # 本次实验的pdf实验报告**

**|**

**|-- model.pth # 训练出的神经网络模型**

**|-- readme.md # 项目简介**

**|-- requirements.txt # 项目中依赖的包**

项目所需要的依赖:

**Pillow==10.1.0**

**torch==2.1.1**

**torchvision==0.16.1**

**matplotlib~=3.8.2**

可以通过以下指令安装依赖:

**pip3 install -r requirements.txt**

项目运行方式:

**python3 train\_model.py**

1. 该python文件首先会下载MNIST数据到`./data`文件夹中,大小约为500MB.

2. 然后该项目会加载MNIST数据集,训练时会对数据集中的数据进行适当的转换(转为张量、标准化)

3. 其次为训练和测试数据集创建 `DataLoader`，以便在训练和评估过程中批量加载数据。

4.

`train` 函数用于训练模型。它遍历训练数据加载器中的批次数据，执行前向和后向传播，更新模型参数。

`test` 函数用于评估模型。它遍历测试数据加载器中的数据，计算模型的损失和准确率。

5.

设置训练的epochs数。

对于每个epoch，调用 `train` 函数进行训练，并调用 `test` 函数进行评估。

训练完成后，模型状态（权重和偏差）被保存到 `model.pth` 文件中。

6. 使用 `plot\_examples` 函数，从测试数据集中显示一些图片和模型的预测。

7. 使用 `save\_examples` 函数，将一些测试样本的图片及其模型预测保存到指定文件夹。

项目运行结果:

**Epoch 1**

**-------------------------------**

**loss: 2.354920 [ 0/60000]**

**loss: 0.412174 [ 6400/60000]**

**loss: 0.354146 [12800/60000]**

**loss: 0.344568 [19200/60000]**

**loss: 0.362949 [25600/60000]**

**loss: 0.451336 [32000/60000]**

**loss: 0.431929 [38400/60000]**

**loss: 0.251388 [44800/60000]**

**loss: 0.199853 [51200/60000]**

**loss: 0.130820 [57600/60000]**

**Test Error:**

**Accuracy: 93.4%, Avg loss: 0.225899**

**Epoch 2**

**-------------------------------**

**loss: 0.185689 [ 0/60000]**

**loss: 0.139291 [ 6400/60000]**

**loss: 0.355247 [12800/60000]**

**loss: 0.170598 [19200/60000]**

**loss: 0.220295 [25600/60000]**

**loss: 0.128450 [32000/60000]**

**loss: 0.065901 [38400/60000]**

**loss: 0.127051 [44800/60000]**

**loss: 0.072048 [51200/60000]**

**loss: 0.303864 [57600/60000]**

**Test Error:**

**Accuracy: 95.4%, Avg loss: 0.150283**

**Epoch 3**

**-------------------------------**

**loss: 0.124232 [ 0/60000]**

**loss: 0.115441 [ 6400/60000]**

**loss: 0.058861 [12800/60000]**

**loss: 0.083369 [19200/60000]**

**loss: 0.066679 [25600/60000]**

**loss: 0.118891 [32000/60000]**

**loss: 0.054929 [38400/60000]**

**loss: 0.210742 [44800/60000]**

**loss: 0.075034 [51200/60000]**

**loss: 0.110859 [57600/60000]**

**Test Error:**

**Accuracy: 96.1%, Avg loss: 0.133264**

**Epoch 4**

**-------------------------------**

**loss: 0.127429 [ 0/60000]**

**loss: 0.066850 [ 6400/60000]**

**loss: 0.241161 [12800/60000]**

**loss: 0.089738 [19200/60000]**

**loss: 0.193802 [25600/60000]**

**loss: 0.074207 [32000/60000]**

**loss: 0.272002 [38400/60000]**

**loss: 0.091803 [44800/60000]**

**loss: 0.097746 [51200/60000]**

**loss: 0.118522 [57600/60000]**

**Test Error:**

**Accuracy: 96.2%, Avg loss: 0.117603**

**Epoch 5**

**-------------------------------**

**loss: 0.162923 [ 0/60000]**

**loss: 0.081532 [ 6400/60000]**

**loss: 0.144496 [12800/60000]**

**loss: 0.197388 [19200/60000]**

**loss: 0.072401 [25600/60000]**

**loss: 0.020777 [32000/60000]**

**loss: 0.063514 [38400/60000]**

**loss: 0.086938 [44800/60000]**

**loss: 0.084312 [51200/60000]**

**loss: 0.032998 [57600/60000]**

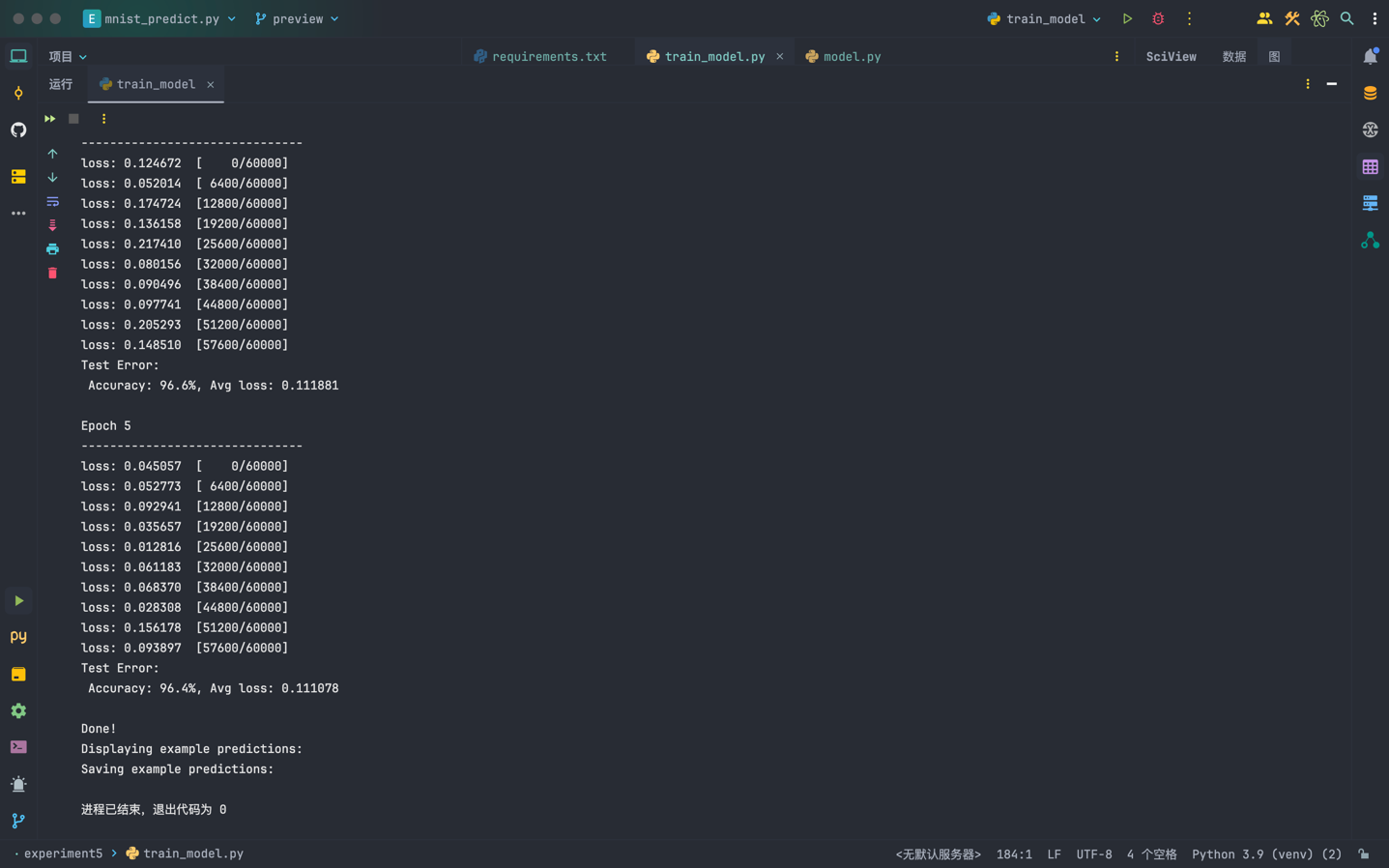
**Test Error:**

**Accuracy: 96.9%, Avg loss: 0.100546**

**Done!**

**Displaying example predictions:**

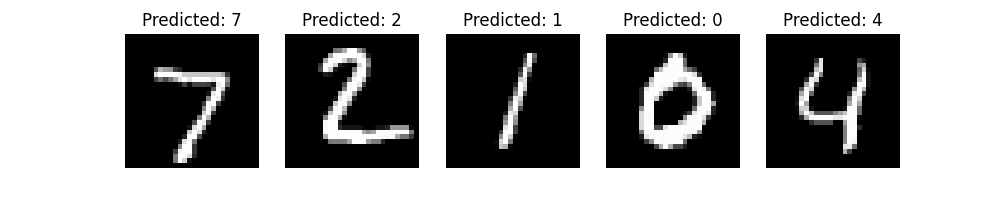
**Saving example predictions:**



可以看到,随着训练的增多,模型的正确率不断增高,而loss不断降低

该模型最终的正确率达到了96.9%,同时loss仅有0.1,可以认为该模型较精确

模型训练的样例:



文本

描述已自动生成

保存的一百个样例中的几个:

图片包含 图表

描述已自动生成

图表, 直方图

描述已自动生成

图表, 直方图

描述已自动生成

图表

低可信度描述已自动生成

图表, 直方图

描述已自动生成

图片包含 QR 代码

描述已自动生成

神经网络模型的定义:

**"""  
该模块定义了一个简单的神经网络模型，用于基本的图像识别任务。  
它包含一个神经网络类 `NeuralNet`，该类继承自 `torch.nn.Module`。  
  
Module: model.py  
"""  
  
from torch import nn  
  
  
# 定义神经网络模型  
class NeuralNet(nn.Module):  
 """  
 这个类实现了一个简单的神经网络，用于图像识别任务。  
 它包含两个主要部分：一个将输入图像展平的层和一个线性激活函数堆叠。  
  
 Class: NeuralNet  
 Extends: nn.Module  
 """  
 def \_\_init\_\_(self):  
 super(NeuralNet, self).\_\_init\_\_()  
 self.flatten = nn.Flatten()  
 self.linear\_relu\_stack = nn.Sequential(  
 nn.Linear(28 \* 28, 128),  
 nn.ReLU(),  
 nn.Linear(128, 10),  
 nn.LogSoftmax(dim=1)  
 )  
  
 def forward(self, x):  
 """  
 定义模型的前向传播逻辑。  
  
 Args:  
 x (Tensor): 输入数据。  
  
 Returns:  
 Tensor: 模型的输出。  
  
 Method: forward  
 """  
 x = self.flatten(x)  
 logits = self.linear\_relu\_stack(x)  
 return logits**

该类实现了对神经网络的定义.该类继承自torch.nn.Model,其中主要实现了将输入图像展平以及线性激活函数堆叠

用来训练神经网络模型的函数:

**def train(dataloader, model, loss\_fn, optimizer):  
 """  
 训练模型的函数  
  
 Args:  
 dataloader (DataLoader): 训练数据加载器  
 model (NeuralNet): 要训练的神经网络模型  
 loss\_fn (nn.Module): 损失函数  
 optimizer (optim.Optimizer): 优化器  
  
 Returns:  
 无返回值  
 """  
 size = len(dataloader.dataset)  
 for batch, (X, y) in enumerate(dataloader):  
 X, y = X.to(device), y.to(device)  
 pred = model(X)  
 loss = loss\_fn(pred, y)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
 if batch % 100 == 0:  
 loss, current = loss.item(), batch \* len(X)  
 print(f"loss: {loss:>7f} [{current:>5d}/{size:>5d}]")**

该函数输入参数分别为训练数据加载器,需要训练的神经网络模型,损失函数以及优化器,训练完成后会将结果输出

用来评估模型训练效果的函数

**def test(dataloader, model, loss\_fn):  
 """  
 评估模型的函数  
  
 Args:  
 dataloader (DataLoader): 测试数据加载器  
 model (NeuralNet): 要评估的神经网络模型  
 loss\_fn (nn.Module): 损失函数  
  
 Returns:  
 无返回值  
 """  
 size = len(dataloader.dataset)  
 num\_batches = len(dataloader)  
 model.eval()  
 test\_loss, correct = 0, 0  
 with torch.no\_grad():  
 for X, y in dataloader:  
 X, y = X.to(device), y.to(device)  
 pred = model(X)  
 test\_loss += loss\_fn(pred, y).item()  
 correct += (pred.argmax(1) == y).type(torch.float).sum().item()  
 test\_loss /= num\_batches  
 correct /= size  
 print(f"Test Error: \n Accuracy: {(100 \* correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")**

该函数输入测试数据加载器,神经网络模型以及损失函数,然后会计算模型的精确度,并且将结果输出

保存一些测试数据的函数:

**def save\_examples(model, dataloader, num\_examples=100, folder\_path='./examples'):  
 """保存MNIST数据集中的图片及模型预测的数字  
  
 Args:  
 model (NeuralNet): 训练好的模型  
 dataloader (DataLoader): 测试数据加载器  
 num\_examples (int): 要保存的样本数量  
 folder\_path (str): 保存图片的文件夹路径  
 """  
 if not os.path.exists(folder\_path):  
 os.makedirs(folder\_path)  
  
 model.eval()  
 examples\_shown = 0  
  
 for X, y in dataloader:  
 X, y = X.to(device), y.to(device)  
 with torch.no\_grad():  
 pred = model(X)  
 preds = pred.argmax(1) # 获取每个样本的预测标签  
  
 for i in range(X.size(0)):  
 if examples\_shown >= num\_examples:  
 break  
  
 img = X[i].squeeze().cpu().numpy()  
 plt.imshow(img, cmap='gray')  
 plt.title(f"Predicted: {preds[i].item()}")  
 plt.savefig(os.path.join(folder\_path, f"example\_{examples\_shown}.png"))  
 plt.close()  
 examples\_shown += 1  
  
 if examples\_shown >= num\_examples:  
 break**

该函数用于保存一些训练结果,在不设置参数num\_examples时默认保存100个测试结果,将结果保存至./examples文件夹中

模型性能分析:

准确率: 模型在MNIST测试数据集上达到了较高的准确率，说明模型能有效识别手写数字。

损失下降: 训练过程中损失逐渐下降，表明模型学习到了数据中的特征。

改进空间:

可以尝试更复杂的网络结构（如卷积神经网络）来提高准确率。

实施更细致的参数调整和优化，比如调整学习率、使用不同的优化器。

关于改进空间的具体分析可以从以下几个方面入手：

1. 网络结构优化

使用更复杂的网络结构：目前模型使用的是相对简单的全连接网络。虽然这对于MNIST数据集已经足够，但对于更复杂的图像识别任务，可以考虑使用卷积神经网络（CNN）。CNN在处理图像数据方面更为高效和有效，因为它能更好地捕捉图像的空间层次结构和特征。

增加隐藏层和神经元数量：增加模型的深度和宽度可以帮助捕获更复杂的特征，但同时也要注意避免过拟合。

2. 超参数调整

学习率调整：学习率是影响模型训练效率和性能的关键参数。可以尝试使用不同的学习率，或者使用学习率调度器，在训练过程中动态调整学习率。

优化器选择：尽管Adam优化器是一个强大且广泛使用的选择，但根据具体任务和数据，其他优化器（如SGD、RMSprop）可能会带来更好的结果。

批量大小：增大或减小批量大小可能会影响训练的稳定性和速度。较大的批量可以提供更稳定的梯度估计，但可能会增加训练时间。

3. 正则化和数据增强

应用正则化技术：例如，使用Dropout或者L2正则化可以帮助减少过拟合，提高模型的泛化能力。

数据增强：对于图像数据，常用的数据增强技术包括旋转、缩放、裁剪等。这些技术可以帮助模型学习到从不同角度观察数字的能力，提高模型的鲁棒性。

1. 实验总结

关键知识点和技能

深度学习基础: 通过构建和训练神经网络，我加深了对深度学习，尤其是用于图像识别任务的基础知识的理解。

PyTorch框架的应用: 学习了如何使用PyTorch进行模型的设计、训练和测试，包括处理数据集、定义网络结构、选择损失函数和优化器等。

数据预处理: 理解了将数据转换为模型可接受格式的重要性，包括图像的规范化和转换为张量。

模型评估: 学习了如何评估模型的性能，主要关注损失和准确率，并了解如何解释这些指标。

实际应用: 掌握了如何将训练好的模型应用到新数据上，进行预测和结果分析。

模型性能分析

准确率: 模型在MNIST测试数据集上达到了较高的准确率，说明模型能有效识别手写数字。

损失下降: 训练过程中损失逐渐下降，表明模型学习到了数据中的特征。

改进空间:

可以尝试更复杂的网络结构（如卷积神经网络）来提高准确率。

实施更细致的参数调整和优化，比如调整学习率、使用不同的优化器。

反思和挑战

调参挑战: 在实验过程中，找到最佳的学习率和优化器设置是一个挑战。通过多次实验和结果比较，我逐渐理解了这些参数对模型性能的影响。

数据处理: 刚开始时对数据预处理不够熟悉，通过学习和实验，我更好地理解了如何准备数据以提高模型性能。

理论与实践的结合: 通过这个实验，我将理论知识应用于实践，更深入地理解了神经网络和深度学习的工作原理。

结论

总的来说，这次实验不仅加深了我对深度学习的理解，也提高了我的编程和问题解决能力。它对我的学术和职业发展都是一次宝贵的经历。