人 工 智 能 实 验 报 告

实验名称： 机器学习

学员姓名: 程景愉

学　号: 202302723005

实验日期: 2025.12.10

国防科技大学教育训练部制

《本科实验报告》填写说明

实验报告内容编排应符合以下要求：

（1）采用 A4（21cm×29.7cm）白色复印纸，单面黑字。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

（2）报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）”……，第三级用“1.”、“2.”……，第四级用“（1）”、“（2）”……，分别按序连续编排。

（3）正文插图、表格中的文字字号均为 5 号。

0 目录

1 实验介绍

本项目是 CS188 课程的第五个项目，主题为机器学习（Machine Learning）。实验的主要目的是通过实践操作，熟悉机器学习的基本概念、神经网络的构建过程以及 PyTorch 深度学习框架的使用。在本项目中，我们将从最基础的感知机开始，逐步构建非线性回归模型、用于手写数字识别的多层感知机（MLP），以及卷积神经网络（CNN）。通过这些任务，我们将深入理解神经网络的初始化、前向传播、损失计算、反向传播以及参数更新等核心机制。

2 实验内容

本次实验主要包含以下四个核心编程任务，所有代码实现均在 `models.py` 文件中进行：

1. **Q1: Perceptron (**感知机**)：**实现一个简单的二元分类感知机，理解基本的线性分类器更新规则。
2. **Q2: Non-linear Regression (**非线性回归**)：**构建一个两层的神经网络，用于拟合正弦函数 $\sin(x)$，理解如何利用神经网络的非线性能力进行函数逼近。
3. **Q3: Digit Classification (**数字分类**)：**设计并训练一个多层全连接神经网络，对 MNIST 手写数字数据集进行分类，掌握分类任务的损失函数和模型评估方法。
4. **Q5: Convolutional Neural Networks (**卷积神经网络**)：**手动实现二维卷积操作，并基于此构建卷积神经网络，理解 CNN 在处理图像数据时的优势。

3 实验要求

根据项目指导文档，实验的具体要求如下：

1. 文件修改：仅允许修改 `models.py` 文件，其余文件（如 `autograder.py` 等）需保持原样，以确保自动评分器正常运行。
2. 框架使用：需熟练使用 PyTorch 提供的 `Tensor` 操作、`nn.Parameter`、`optim` 优化器等工具。
3. 性能达标：
   - Q1：感知机需在训练集上收敛（准确率 100%）。
   - Q2：回归模型的平均训练损失（MSE）需低于 0.02。
   - Q3：数字分类模型在测试集上的准确率需达到 97% 以上。
   - Q5：CNN 模型在简化版数据集上的准确率需达到 80% 以上。
4. 禁止事项：Q3 的输出层不得使用 ReLU 激活函数；代码需通过自动评分器的技术正确性检查。

4 实验步骤与实现

*4.1 Q1: Perceptron (*感知机*)*

实现思路： 感知机是一个最简单的线性二分类模型。我们的目标是找到一个权重向量 w，使得对于输入 x，如果 $w \cdot x \geq 0$ 则预测为类别 1，否则为 -1。

1. 初始化：使用 `torch.nn.Parameter` 初始化权重 `self.w`，维度为 `(1, dimensions)`，初始化为全 1 向量。
2. 计算得分：计算输入 x 与权重 w 的点积。使用 `tensordot` 计算。
3. 预测与训练：`get_prediction` 根据得分符号返回 1 或 -1。`train` 方法中，我们遍历数据集。如果预测错误，则根据感知机规则更新权重：$w \leftarrow w + y \cdot x$。

核心代码 (`models.py`):

```
class PerceptronModel(Module):
    def __init__(self, dimensions):
        super(PerceptronModel, self).__init__()
        self.w = Parameter(ones(1, dimensions))

    def run(self, x):
        return tensordot(self.w, x, dims=2)

    def train(self, dataset):
        with no_grad():
            dataloader = DataLoader(dataset, batch_size=1, shuffle=True)
            while True:
                converged = True
                for batch in dataloader:
                    x = batch['x']
                    y = batch['label']
                    prediction = self.get_prediction(x)
                    if prediction != y.item():
                        self.w.data += y.item() * x
                        converged = False
                if converged:
                    break
```

测试指令:

```
python autograder.py -q q1
```

4.2 Q2: Non-linear Regression (非线性回归)

实现思路：为了拟合非线性的 $\sin(x)$ 函数，我们需要引入非线性激活函数。我们构建了一个三层的神经网络：

- 第一层：线性层 `Linear(1, 256)` + ReLU。
- 第二层：线性层 `Linear(256, 256)` + ReLU。
- 第三层：线性层 `Linear(256, 1)`，输出预测值。

训练时使用 Adam 优化器和 MSE 损失函数。为了加速收敛，我们还引入了学习率调度器 (`StepLR`)，每 250 个 step 将学习率衰减为原来的 0.1。

核心代码 (`models.py`):

```
class RegressionModel(Module):
    def __init__(self):
        super().__init__()
```

```
        self.layer1 = Linear(1, 256)
        self.layer2 = Linear(256, 256)
        self.layer3 = Linear(256, 1)

    def forward(self, x):
        x = relu(self.layer1(x))
        x = relu(self.layer2(x))
        return self.layer3(x)

    def train(self, dataset):
        batch_size = 40
        dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
        optimizer = optim.Adam(self.parameters(), lr=0.001)
        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=250,
gamma=0.1)

        for epoch in range(1000):
            total_loss = 0
            num_samples = 0
            for batch in dataloader:
                optimizer.zero_grad()
                loss = self.get_loss(batch['x'], batch['label'])
                loss.backward()
                optimizer.step()
                total_loss += loss.item() * len(batch['x'])
                num_samples += len(batch['x'])

            scheduler.step()
            if total_loss / num_samples < 0.02:
                break
```

测试指令:

```
python autograder.py -q q2
```

4.3 Q3: Digit Classification (数字分类)

实现思路： 针对 MNIST 28x28 的图像分类任务，我们构建了一个三层全连接网络：

- 输入层：784 维（28*28 展平）。
- 隐藏层 1：`Linear(784, 256)` + ReLU。
- 隐藏层 2：`Linear(256, 128)` + ReLU。
- 输出层：`Linear(128, 10)`，直接输出 Logits。

使用 `cross_entropy` 损失函数和 Adam 优化器，批大小为 100，训练 5 个 Epoch 即可达到目标准确率。

核心代码 (`models.py`):

```
class DigitClassificationModel(Module):
    def __init__(self):
        super().__init__()
        input_size = 28 * 28
        output_size = 10
        self.layer1 = Linear(input_size, 256)
```

```
        self.layer2 = Linear(256, 128)
        self.layer3 = Linear(128, output_size)

    def run(self, x):
        x = relu(self.layer1(x))
        x = relu(self.layer2(x))
        return self.layer3(x)

    def train(self, dataset):
        batch_size = 100
        dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
        optimizer = optim.Adam(self.parameters(), lr=0.001)

        for epoch in range(5):
            for batch in dataloader:
                optimizer.zero_grad()
                loss = self.get_loss(batch['x'], batch['label'])
                loss.backward()
                optimizer.step()
```

测试指令:

```
python autograder.py -q q3
```

4.4 Q5: Convolutional Neural Networks (卷积神经网络)

实现思路：本任务的核心是手动实现二维卷积操作。`Convolve` 函数通过双重循环遍历输出矩阵的每个位置，计算输入子矩阵与卷积核的点积（使用 `tensordot`）。基于此，我们构建了一个简单的 CNN：

1. 卷积层：使用 `Convolve` 处理输入，卷积核大小 3x3，输出特征图大小为 26x26。
2. 展平：将特征图展平为 676 维向量。
3. 全连接层 1：`Linear(676, 100)` + ReLU。
4. 全连接层 2：`Linear(100, 10)`。

核心代码 (`models.py`):

```
def Convolve(input: tensor, weight: tensor):
    input_h, input_w = input.shape
    weight_h, weight_w = weight.shape
    output_h = input_h - weight_h + 1
    output_w = input_w - weight_w + 1
    Output_Tensor = torch.zeros((output_h, output_w))

    for y in range(output_h):
        for x in range(output_w):
            sub_tensor = input[y:y+weight_h, x:x+weight_w]
            Output_Tensor[y, x] = tensordot(sub_tensor, weight, dims=2)
    return Output_Tensor

class DigitConvolutionalModel(Module):
    def __init__(self):
        super().__init__()
        self.convolution_weights = Parameter(ones((3, 3)))
```

```
        self.layer1 = Linear(26 * 26, 100)
        self.layer2 = Linear(100, 10)

    def forward(self, x):
        x = x.reshape(len(x), 28, 28)
        x = stack(list(map(lambda sample: Convolve(sample,
self.convolution_weights), x)))
        x = x.flatten(start_dim=1)
        x = relu(self.layer1(x))
        return self.layer2(x)
```

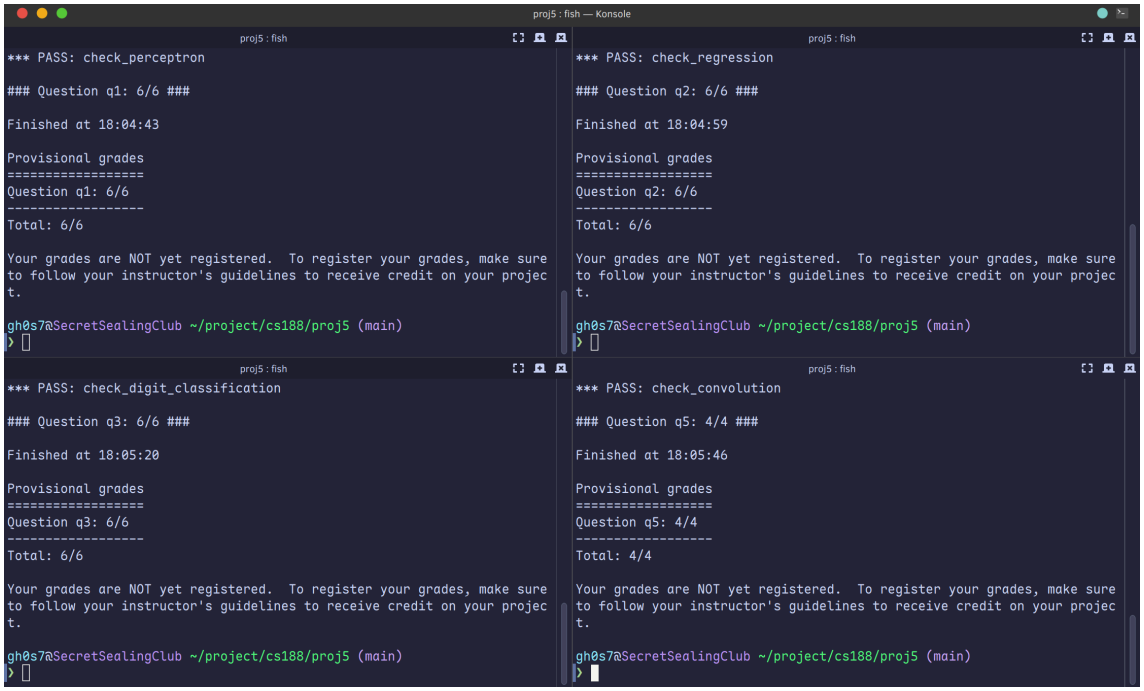
测试指令:

```
python autograder.py -q q5
```

5 实验结果

本实验所有代码均通过 `autograder.py` 的测试，各项指标均达到或超过实验要求。

1. **Q1** 感知机：模型能够迅速收敛，在训练数据上实现了 100% 的分类准确率。
2. **Q2** 非线性回归：训练后的模型 Loss 稳定在 0.02 以下，能够很好地拟合 $\sin(x)$ 曲线。
3. **Q3** 数字分类：构建的 MLP 模型在 MNIST 测试集上的准确率超过 97%（Staff 参考值为 98%），训练过程高效稳定。
4. **Q5 CNN**：手动实现的卷积层逻辑正确，通过了梯度检查。构建的 CNN 模型在测试数据集上准确率超过 80%，验证了卷积特征提取的有效性。



6 实验总结

通过本次 Project 5 的实验，我收获颇丰：

首先, 我熟练掌握了 **PyTorch** 框架的基础操作。从定义 `nn.Parameter` 到构建 `nn.Linear` 层, 再到使用 `optim.Adam` 和各类 Loss 函数，我对深度学习模型的代码实现流程有了清晰的认识。

其次，我深入理解了**神经网络的内部机制**。通过 Q1，我复习了线性分类器的原理；通过 Q2, 我直观体会到了激活函数（ReLU）引入非线性的重要性——没有它, 多层网络也仅仅是线性变换的叠加；通过 Q3, 我掌握了处理多分类问题的标准范式（CrossEntropy + Softmax Logits）。

最后，Q5 的手动实现卷积让我对 CNN 有了更底层的理解。以前只知道调用 API，现在通过自己编写滑动窗口和点积运算，深刻理解了权重共享和局部感知野的概念。这次实验不仅锻炼了编程能力，也为后续深入学习深度学习打下了坚实基础。