练习 2 实验报告

姓名：陈浩　学号：123106222839　学院：计算机科学与工程学院

1　实验目标

实现 LeNet-5 在 MINIST 数据集上的训练和测试，并进行分析，完成实验报告，提交代码。

2　实现说明

2.1 LeNet-5 网络结构

LeNet 由 Yann LeCun 提出，是一种经典的卷积神经网络，是现代卷积神经网络的起源之一。LeNet-5 中的 5 代表卷积核的尺寸，它具有一个输入层，两个卷积层，两个池化层，3 个全连接层（其中最后一个全连接层为输出层），其具体网络结果如图 1 所示。

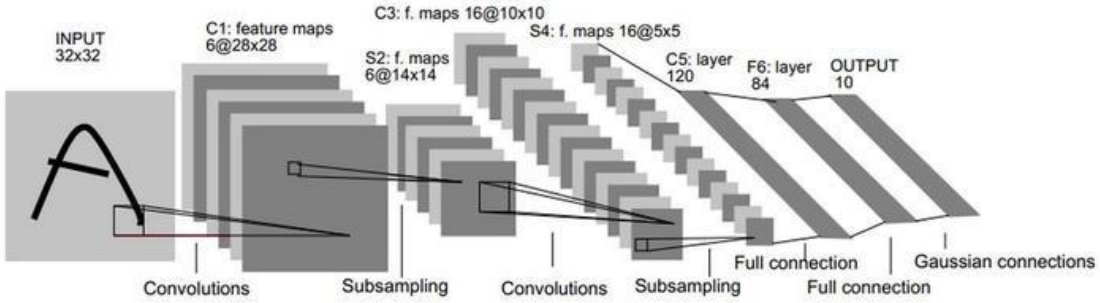

图 1 LeNet-5 网络结构

卷积层 C1。手写数字数据集是灰度图像，输入为 32×32×1 的图像，卷积核大小为 5×5，卷积核数量为 6，步长为 1，零填充。最终得到的 C1 的特征图大小为 28（32 − 5 + 1 = 28）。

下采样层 S2。卷积层 C1 之后是池化操作，每个窗口大小为 2×2，步长为 2。因此，每个池化操作会从每个 2×2 窗口中输出一个值，产生大小为 14×14 的特征图（输出通道数为 6）。这样可以减少特征图的大小，提高计算效率。

卷积层 C3。包含 16 个卷积核，每个卷积核的大小为 5×5 ，步长为 1，零填充。最终产生大小为 10×10 的特征图（输出通道数为 16）。

下采样层 S4。与下采样层 S 2 类似，采用大小为 2×2、步长为 2 的窗口进行下采样，输出特征图大小为 5×5（输出通道数为 16）。

全连接层 C5。C5 将大小为 5×5 、通道数为 16 的特征图拉成一个元素为 400 的向量，并通过一个带有 120 个神经元的全连接层进行连接。

全连接层 F6。全连接层 F6 将 120 个神经元连接到 84 个神经元。

输出层。输出层由 10 个神经元组成，每个神经元对应 0-9 中的一个数字，并输出最终的分类结果。在训练过程中，使用交叉熵损失函数计算输出层的误差，并通过反向传播算法更新卷积核和全连接层的权重参数。

2.2 实验流程

①导入所需库、设置随机数种子、加载数据并预处理。

导入所需库后设置随机数种子，方便复现结果。

由于 MNIST 数据集图片尺寸是 28×28 单通道的，而 LeNet-5 网络输入图片大小为 32×32，因此使用 transforms.Resize 将输入图片尺寸调整为 32×32。

数据集标准化。MINIST 数据集中原始数据不仅分布不均（噪声大）而且数值较大（数值范围是 0~255），我们打算将训练集和测试集标准化，使样本将呈现均值为 0 方差为 1 的数据分布，原因是均值为 0、方差为 1 的数据经过激活函数后能避免梯度消失问题。

标准化处理流程是样本减去它的均值，再除以它的标准差，最终样本将呈现均值为 0 方差为 1 的数据分布。根据提供的系数，均值 0.1307、标准差 0.3081。可以通过 transforms.Normalize((0.1307,), (0.3081,))实现 MINIST 数据集标准化。

②搭建 LeNet-5 神经网络结构，并定义前向传播的过程。

使用 torch.nn 中的内置函数实现定义卷积层、ReLU 函数、池化操作和全连接层。根据 LeNet-5 网络结构定义前向传播过程，

其中池化层中的池化操作可以使用最大池化和平均池化；在经过下 S4 后，需要使用 x.view(-1, 16*5*5)将特征图展平成一个元素为 400 的向量。

③将定义好的网络结构搭载到 GPU，并定义优化器。

使用 torch.device 函数定义好训练时使用 GPU，随后使用 LeNet().to(device)将 LeNet-5 部署到 GPU 上。

使用 torch.optim 中的内置函数可以选用各类优化器。

④定义训练过程。

初始化。使用 model.train()将模型设置为训练模式，enumerate 迭代已加载的数据集，初始化梯度。

损失函数。利用定义好的 LeNet-5 对输入进行预测，并开始计算模型输出与 Label 的损失和，其中多分类情况通常使用交叉熵损失函数，使用 cross_entropy 函数计算。

反向传播并更新参数。使用 loss.backward()函数实现反向传播，并使用 optimizer.step()根据优化器种类进行参数更新。

⑤定义测试过程。

初始化。使用 model.eval()将模型设置为验证模式，初始化精度值。

推理。在推理过程中不需要计算梯度也不进行反向传播，通过统计推理结果是否和 Label 一致来计算精度。

⑥训练与测试。

LeNet-5 网络模型定义好，训练函数、测试函数也定义好了，就可以直接使用 MNIST 数据集进行训练了。设置训练轮数 epoch，并将训练中的训练损失和测试过的精度可视化。

⑦保存模型。保存步骤⑥训练好的模型为.pth 文件。

⑧手写图片预测。读入⑦中保存的模型，对输入的手写数字图片进行预测。

3 实验与分析。

3.1 运行说明

实验所需数据存放在同目录的 data 文件夹中，模型保存在同目录的 models 文件夹中，实验代码为 LeNet-5.ipynb。

对于 LeNet-5.ipynb 版本，运行方法为使用 jupyter notebook 打开 ipynb 文件，顺序执行步骤 1-8。

3.2 消融实验

3.2.1 学习率

初始化训练中，使用 Adam 优化器，学习率设置为 0.01，进行 30 个 epoch 训练，每个 epoch 的训练损失如图 2。从图 2 的损失曲线可以看出，后半段训练中损失并没有收敛，初步分析是学习率过大导致的。

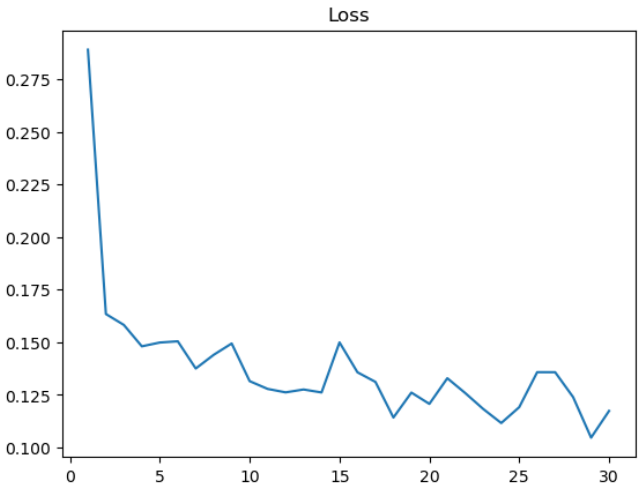

图 2 初始化训练的损失曲线

由于学习率设置为 0.01 会导致后阶段训练时损失函数不收敛的情况，所以在维持其它参数不变的前提下，重新设置学习率为 0.001，每个 epoch 的训练损失如图 3。由图 3 可知模型的损失曲线平滑且逐渐收敛，这也验证了参数的合理。

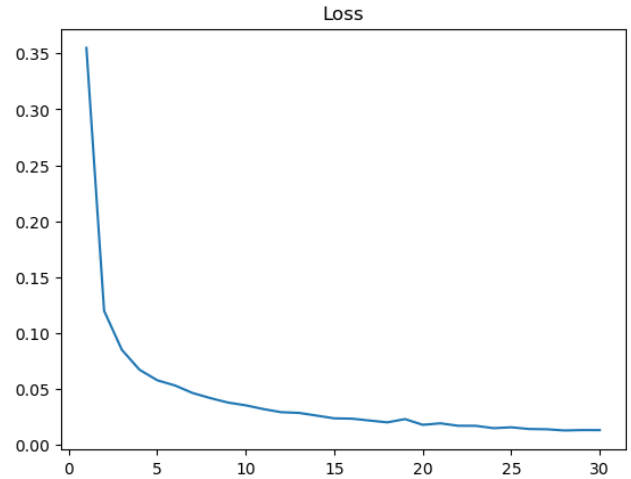

图 3 学习率为 0.001 的损失曲线

分析图 2 和图 3 中损失曲线的开始部分，发现图 2 开始时的训练损失更小，所以开始时更大的学习率能加快学习速度。为了综合开始时较大学习率能加快学习速度的优点以及最后较小学习率有助于损失函数收敛的优点，我使用学习率衰减策略，每过 10 轮学习率为旧学习率的 0.1 倍，训练时每个 epoch 损失如图 4。由图 4 可知虽然兼备了较大学习率和较小学习率的优点，也继承了其缺点，在第 5 到 10 轮损失函数在震荡，且在第 20 到 30 轮由于学习率过小导致收敛得慢。

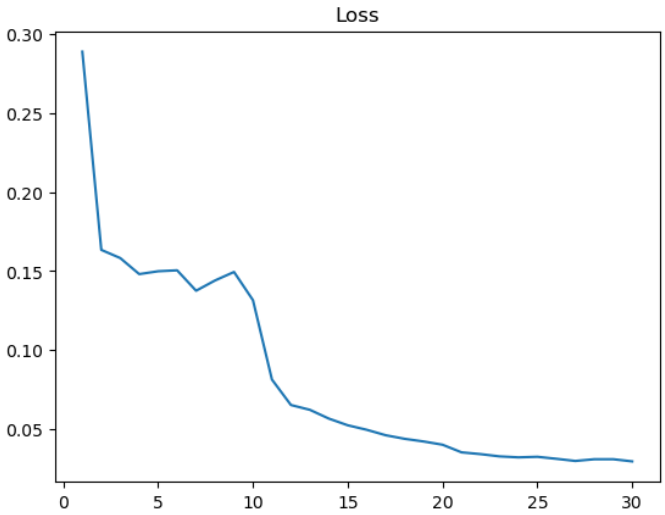

图 4 衰减学习率策略的损失曲线

统计三种学习率策略最后一轮的平均损失以及在测试集的精度，如表 1 所示，其中学习率[a, b]表示采用学习率衰减策略从 a 衰减到 b。由表 1 可知，三种策略中学习率设置为 0.001 的模型性能最优，因其最后一轮的平均损失最小且测试集的精度最高。

表 1 不同学习率模型的定量分析

Learning Rate	Average loss of Epoch 30	Accuracy
0.01	0.117377	95.67%
0.001	**0.013271**	**98.37%**
[0.01, 0.0001]	0.029381	98.15%

3.2.2 池化层策略

为了探究使用不同池化层策略对模型性能的影响，我在维持 3.2.1 中最优参数（学习率为 0.001），分别使用最大池化和平均池化进行 30 个 epoch 训练，训练过程的损失如图 5，发现与图 3 基本相同。

随后统计两种池化层策略最后一轮的平均损失以及在测试集的精度，结果如表 2 所示，使用平均池化操作训练的模型性能更优，因其最后一轮的平均损失最小且测试集的精度最高。

从结果分析，平均池化考虑了池化窗口中所有值的平均值，相比最大池化更加平滑，能够保留更多细节信息，且平均池化对噪声和局部变化的影响相对较小。

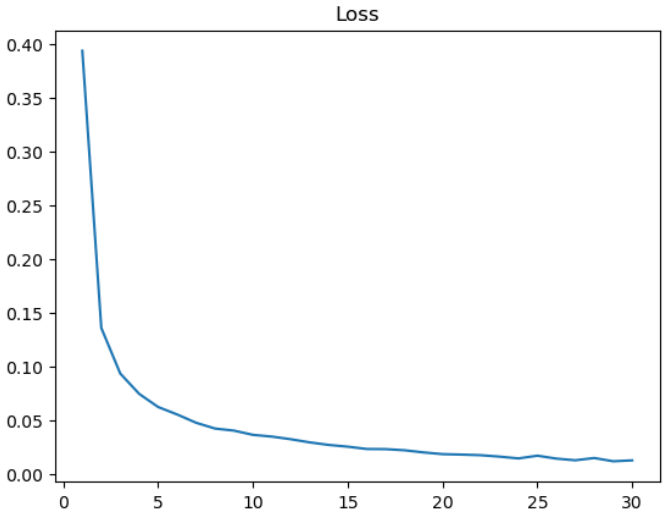

图 5 使用平均池化的损失曲线

表 2 两种池化层策略模型的定量分析

Method	Average loss of Epoch 30	Accuracy
Max Pooling	0.013271	98.37%
Average Pooling	**0.012505**	**98.58%**

3.3 手写图片预测

先加载训练好的模型，这里采用学习率为0.001和平均池化训练得到的模型，并将模型转为test模式。对于输入手写图片（如图6），将图片预处理后使用加载的模型进行预测，得到10个分类概率和最终预测类别如图7，可以看到模型预测结果正确，表明本次实验的成功！

图 6 待预测的手写图片

```
概率： tensor([[1.0907e-17, 7.6273e-10, 1.0000e+00, 1.8907e-10, 8.9585e-15, 3.5574e-10,
         8.9169e-14, 7.8419e-11, 1.2064e-17, 1.0133e-18]], device='cuda:0',
       grad_fn=<SoftmaxBackward0>)
预测类别： 2
```

图 7 预测结果