深 圳 大 学 实 验 报 告

课程名称：AI 边缘设计系统

实验项目名称：Python 神经网络手写数字识别

学院：电子与信息工程学院

专业：电子信息工程

指导教师：蒙山

报告人：陈应权 学号：2022280297 班级：06

实验时间：2024 年 9 月 27 日

实验报告提交时间：2023 年 9 月 28 日

教务部制

一、实验目的与要求：

实验目的

1. 掌握 PyTorch 框架：通过使用 PyTorch 计算框架，加深对深度学习工具的理解和应用。
2. 构建神经网络模型：快速构建一个用于手写体数字识别的神经网络模型，理解模型构建的基本流程。
3. 训练与识别程序：实现模型的训练过程，并对手写数字进行识别，以验证模型的有效性。
4. 理解实验环境：熟悉在 WSL Ubuntu22.04 环境下进行深度学习实验的硬件和软件配置。
5. 应用 MNIST 数据集：使用 MNIST 手写数字库进行实验，理解数据集的结构和如何加载使用。

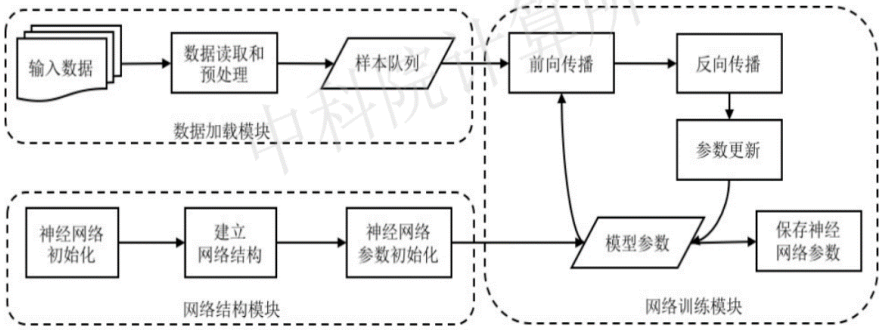
实验要求

1. 评估模型：分析程序代码，对主要功能单元的原理进行解析，并说明其应用方法。
2. 参数调优：
    - 提高模型精度：在不改变网络结构的前提下，尝试调节参数以提高模型的识别精度，并记录实验结果。
    - 原理分析：分析通过参数调节提高模型精度的原理和方法。
3. 网络结构优化：
    - 改进网络结构：思考并尝试通过修改网络结构来进一步提高模型精度。
    - 基本原理论述：讨论修改网络结构对提高模型精度的可能影响及其基本原理。
    - 实验结果：提供修改网络结构后的实验结果，对比改进前后的效果。

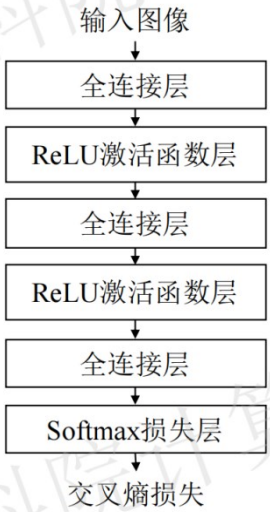
二、实验原理：

以下是实验指导书的内容。

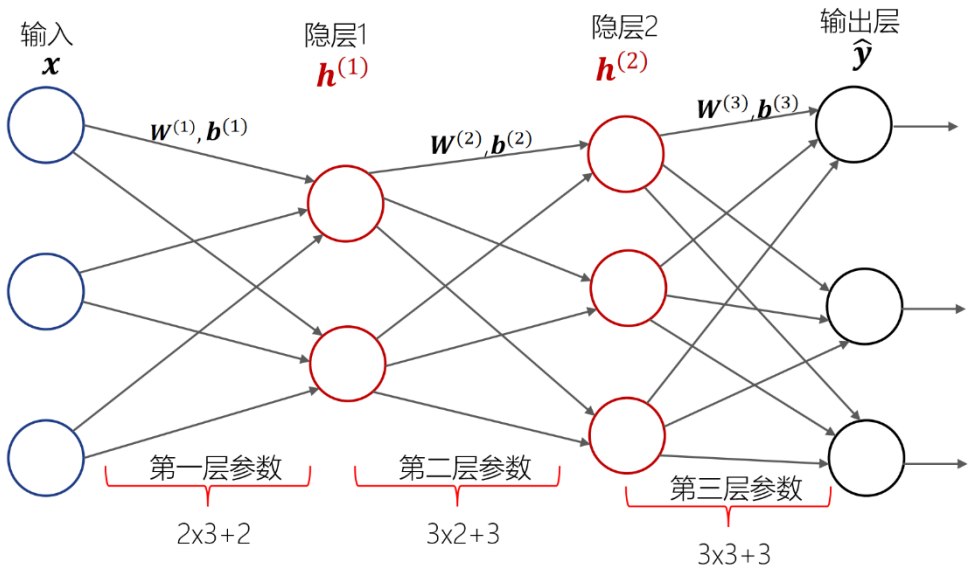
整体流程



具体流程



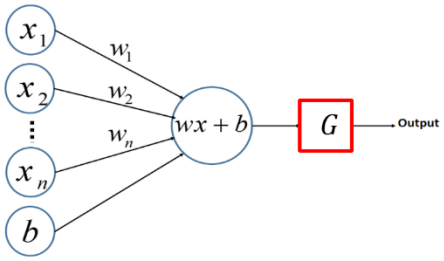
2.1 全连接层



需 6+6+9+8=29个参数

2.2 非线性激活

选择合适的激活函数



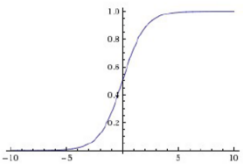
- 在神经元中，输入的数据通过加权求和后，还被作用了一个函数G，这个函数G就是激活函数（Activation Function）。
- 激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，因此神经网络可以应用到众多的非线性模型中
- 激活函数需具备的性质
  - 可微性： 当优化方法是基于梯度的时候，这个性质是必须的。
  - 输出值的范围： 当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的学习率。

sigmoid函数

数学表达式

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

几何图像



- Sigmoid 是最常见的非线性激活函数
- 能够把输入的连续实值变换为0和1之间的输出；如果是非常大的负数，那么输出就变为0；如果是非常大的正数，输出就变为1。

- 非0均值的输出，导致w计算的梯度始终都是正的
- 计算机进行指数运算速度慢
- 饱和性问题及梯度消失现象

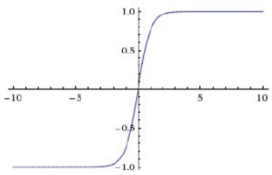
tanh函数

Sigmoid函数存在神经元会产生非0均值的输出的问题

寻找解决办法

$$\tanh(x) = \frac{\sinh(x)}{\cosh x} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$\tanh(x) = 2sigmoid(2x) - 1$$



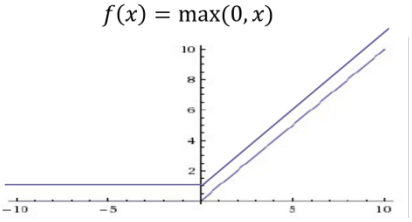
- 与sigmoid相比，tanh是0均值的
- 在输入很大或是很小的时候，输出几乎平滑，梯度很小，不利于权重更新

ReLU函数

$$f(x) = \max(0, x)$$

tanh函数虽然解决了sigmoid函数存在非0均值输出的问题，但仍然没改变梯度消失问题

寻找解决办法



- ReLU 能够在x>0时保持梯度不衰减，从而缓解梯度消失问题
- ReLU死掉。如果学习率很大，反向传播后的参数可能为负数，导致下一轮正向传播的输入为负数。当输入是负数的时候，ReLU是完全不被激活的，这就表明一旦输入到了负数，ReLU就会死掉
- 输出范围是无限的

2.3 损失函数

选择恰当的损失函数

损失函数 $L = f(\hat{y}, y)$，$\hat{y}$是模型预测值，是神经网络模型参数 W的函数，记作 $\hat{y} = H_w(x)$

从$\boldsymbol{w}$角度看，损失函数可以记为 $L(\boldsymbol{w}) = f(H_w(x), y)$

➢ 均方差损失函数

- 均方差损失函数是神经网络优化常用的损失函数

$$L=\frac{1}{2}(y-\hat{y})^2$$ ← 以一个神经元的均方差损失函数为例

假设使用sigmoid函数作为激活函数，则$\hat{y}=\sigma(z)$，其中$z=wx+b$，

$$\frac{\partial L}{\partial w}=(y-\hat{y})\sigma'(z)x \qquad \sigma'(z)=\big(1-\sigma(z)\big)\cdot\sigma(z)$$

$$\frac{\partial L}{\partial b}=(y-\hat{y})\sigma'(z)$$

所求的与 $\frac{\partial L}{\partial w}$ 和 $\frac{\partial L}{\partial b}$ 梯度中都含有$\sigma'(z)$，当神经元输出接近1时，梯度将趋于0，出现梯度消失，导致神经网络反向传播时参数更新缓慢，学习效率下降。

➢ 交叉熵损失函数

- 交叉熵损失函数能够有效克服使用sigmoid函数时，均方差损失函数出现的参数更新慢的问题

交叉熵损失函数：

$$L=-\frac{1}{m}\sum_{x\in D}\sum_{i}y_i\ln(\hat{y}_i)$$

其中，m 为训练样本的总数量，i 为分类类别

- 以二分类为例，交叉熵损失函数为：

$$L=-\frac{1}{m}\sum_{x\in D}(y\ln(\hat{y})+(1-y)\ln(1-\hat{y}))$$

2.4 前向传播与反向传播

示例

假定输入数据 $x_1 = 0.02$ 、$x_2 = 0.04$ 、$x_3 = 0.01$

固定偏置$\boldsymbol{b}^{(1)} = [0.4; 0.4; 0.4]$、$\boldsymbol{b}^{(2)} = [0.7; 0.7]$

期望输出$y_1 = 0.9$、$y_2 = 0.5$

未知权重 $\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix}$, $\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix}$

目的是为了能得到 $y_1 = 0.9$、$y_2 = 0.5$的期望的值，需计算出合适的 $\boldsymbol{W}^{(1)}$、$\boldsymbol{W}^{(2)}$ 的权重值

- 初始化权重值

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.15 & 0.30 \\ 0.25 & 0.20 & 0.35 \\ 0.10 & 0.25 & 0.15 \end{bmatrix}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.40 & 0.25 \\ 0.35 & 0.30 \\ 0.01 & 0.35 \end{bmatrix}$$

- 输入到隐层计算

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \boldsymbol{W}^{(1)^T}\boldsymbol{x} + \boldsymbol{b}^{(1)} = \begin{bmatrix} 0.25 & 0.25 & 0.10 \\ 0.15 & 0.20 & 0.25 \\ 0.30 & 0.35 & 0.15 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.04 \\ 0.01 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.4 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.416 \\ 0.4135 \\ 0.4215 \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \frac{1}{1 + e^{-\boldsymbol{v}}} = \begin{bmatrix} \frac{1}{1 + e^{-0.416}} \\ \frac{1}{1 + e^{-0.4135}} \\ \frac{1}{1 + e^{-0.4215}} \end{bmatrix} = \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix}$$

➢ 隐层到输出层计算

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \boldsymbol{W}^{(2)T}\boldsymbol{h} + \boldsymbol{b}^{(2)} = \begin{bmatrix} 0.40 & 0.35 & 0.01 \\ 0.25 & 0.30 & 0.35 \end{bmatrix} \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1.1577 \\ 1.2425 \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \frac{1}{1+e^{-\boldsymbol{z}}} = \begin{bmatrix} \frac{1}{1+e^{-1.1577}} \\ \frac{1}{1+e^{-1.2425}} \end{bmatrix} = \begin{bmatrix} 0.7609 \\ 0.7760 \end{bmatrix}$$

距离期望输出 $y_1 = 0.9$、$y_2 = 0.5$ 还有差距，通过反向传播修改权重

模型计算输出

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \begin{bmatrix} 0.7609 \\ 0.7760 \end{bmatrix}$$

期望输出

$y_1 = 0.9$

$y_2 = 0.5$

➢ 计算误差

$$L(\boldsymbol{W}) = L_1 + L_2 == \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

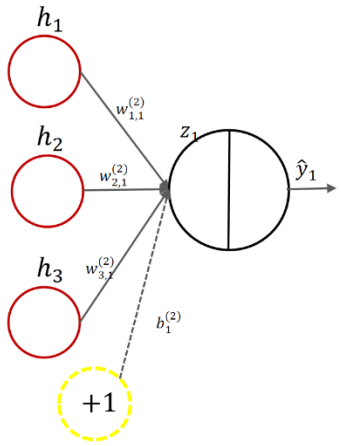
$$= \frac{1}{2}(0.7609 - 0.9)^2 + \frac{1}{2}(0.7760 - 0.5)^2 = 0.0478$$

计算值与真实值之间还有很大的差距，如何缩小计算值与真实值之间的误差?

通过反向传播进行反馈，调节权重值

反向传播

- 隐层到输出层的权值$\boldsymbol{W}^{(2)}$的更新



$$L(\boldsymbol{W}) = L_1 + L_2 == \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

- 以 $w_{2,1}^{(2)}$(记为 ω) 参数为例子，计算 ω 对整体误差的影响有多大，可以使用整体误差对 ω 参数求偏导

- 根据偏导数的链式法则推导

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1}\frac{\partial z_1}{\partial \omega}$$

$$L(\boldsymbol{W}) = \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} = -(y_1 - \hat{y}_1) = -(0.9 - 0.7609) = -0.1391$$

$$\hat{y}_1 = \frac{1}{1 + e^{-z_1}}$$

$$\frac{\partial \hat{y}_1}{\partial z_1} = \hat{y}_1(1 - \hat{y}_1) = 0.7609 * (1 - 0.7609) = 0.1819$$

➢ 根据偏导数的链式法则推导

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega}=\frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1}\frac{\partial z_1}{\partial \omega}$$

$$z_1=w_{1,1}^{(2)}\times h_1+\omega\times h_2+w_{3,1}^{(2)}\times h_3+b_1^{(2)}$$

$$\frac{\partial z_1}{\partial \omega}=h_2=0.6019$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega}=-(y_1-\hat{y}_1)\times\hat{y}_1(1-\hat{y}_1)\times h_2$$
$$=-0.1391\times 0.1819\times 0.6019=-0.0152$$

➢ 更新$w_{2,1}^{(2)}$（记为ω）的权重

初始值

$$\boldsymbol{W}^{(2)}=\begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} \end{bmatrix}=\begin{bmatrix} 0.40 & 0.25 \\ 0.35 & 0.30 \\ 0.01 & 0.35 \end{bmatrix}$$

$$\frac{\partial L(\boldsymbol{W})}{\partial \omega}=-0.0152$$

$$\omega=\omega-\alpha\times\frac{\partial L(\boldsymbol{W})}{\partial \omega}=0.35-(-0.0152)=0.3652$$

➢ 同理，可以计算新的$\boldsymbol{W}^{(2)}$的其他元素的权重值

三、实验过程及方法评估：
实验大致流程：

3.1 加载 MNIST 数据集

- 下载并加载 MNIST 手写数字数据集，该数据集包含训练集和测试集。
- 训练集包含 60,000 个样本，测试集包含 10,000 个样本。
- 每个样本是一个 28x28 像素的灰度图像及其对应的数字标签。

3.2 定义神经网络

- 设计一个神经网络架构，通常包括输入层、隐藏层和输出层。
- 确定网络中的神经元数量和连接方式。

3.3 损失函数和优化器

- 选择合适的损失函数（如交叉熵损失）来衡量模型预测与实际标签之间的差异。
- 选择一个优化算法（如 SGD 或 Adam）来更新网络权重，以最小化损失函数。

3.4 训练

- 使用训练集数据对神经网络进行训练。
- 在训练过程中，通过前向传播计算预测值，通过反向传播更新网络权重。
- 定期验证模型在验证集上的性能，以避免过拟合。

3.5 神经网络测试

- 在测试集上评估模型的性能，通常使用准确率作为评估指标。
- 分析模型的识别结果，确定模型的泛化能力。

3.6 绘制损失曲线

3.7 测试验证模型性能

- 拍摄自己的手写数字图片
- 在测试集上评估模型的性能，通常使用准确率作为评估指标。

主要功能单元的原理解析和应用方法

1. 数据加载与预处理
    - 原理：使用 torchvision.transforms 对 MNIST 数据集进行预处理，包括转换图像为张量(ToTensor())和归一化(Normalize())。

- 应用：归一化有助于神经网络更快地收敛，因为输入数据的分布被调整到一个较小的范围。

```
# 1. 加载MNIST数据集
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize( mean: (0.5,), std: (0.5,))
])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)
```

数据的加载与预处理

2. 神经网络模型定义
   - 原理：定义了一个包含三个全连接层的简单前馈神经网络。输入层将 28x28 图像展平为 784 维向量，然后通过两个隐藏层，最后通过输出层得到 10 个类别的预测。
   - 应用：ReLU 激活函数用于引入非线性，使得网络能够学习复杂的模式。

```
# 2. 定义神经网络模型
2 用法
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, out_features: 512)
        self.fc2 = nn.Linear( in_features: 512, out_features: 256)
        self.fc3 = nn.Linear( in_features: 256, out_features: 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

神经网络模型的定义

3. 损失函数和优化器
   - 原理：交叉熵损失函数(CrossEntropyLoss)用于多分类问题，衡量预测概率分布与真实标签之间的差异。Adam 优化器结合了动量和 RMSprop 的特点，自动调整学习率，以优化损失函数。
   - 应用：损失函数指导网络训练的方向，优化器负责更新网络权重以最小

化损失。

```
# 3. 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.01, weight_decay=1e-5)
```

损失函数与优化器

4. 模型训练
    - 原理：通过前向传播计算预测值，然后计算损失；通过反向传播传播损失，更新网络权重。
    - 应用：使用动态损失记录和打印，监控训练过程，每 100 个批次输出一次平均损失，以评估模型的学习进度。

```
# 4. 训练模型
losses = []  # 用于记录损失的列表
for epoch in range(30):  # 训练10个epoch
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 100 == 99:  # 每100个batch打印一次loss
            losses.append(running_loss / 100)  # 将平均loss添加到列表中
            print(f'[Epoch{epoch + 1}, Batch{i + 1}] loss: {running_loss / 100:.3f}')
            running_loss = 0.0
    print('Finished Training')
```

模型训练

5. 模型测试
    - 原理：在测试集上评估模型性能，计算准确率。
    - 应用：测试集上的准确率提供了模型泛化能力的指标。

```python
# 5. 测试模型
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy of the network on the 10000 test images: {accuracy:.2f}%')
```

模型的测试

6. 绘制损失曲线
    - 原理：使用 matplotlib 绘制训练过程中损失的变化，直观展示模型学习的效果。
    - 应用：损失曲线有助于识别模型是否过拟合或欠拟合，以及是否需要调整训练策略。

四、问题的思考

4.1 在不改变网络结构的条件下，尝试调节参数，进一步提高模型精度？请给出实验结果。

回答： 为了在不改变网络结构的条件下提高模型精度，可以尝试以下超参数调整：

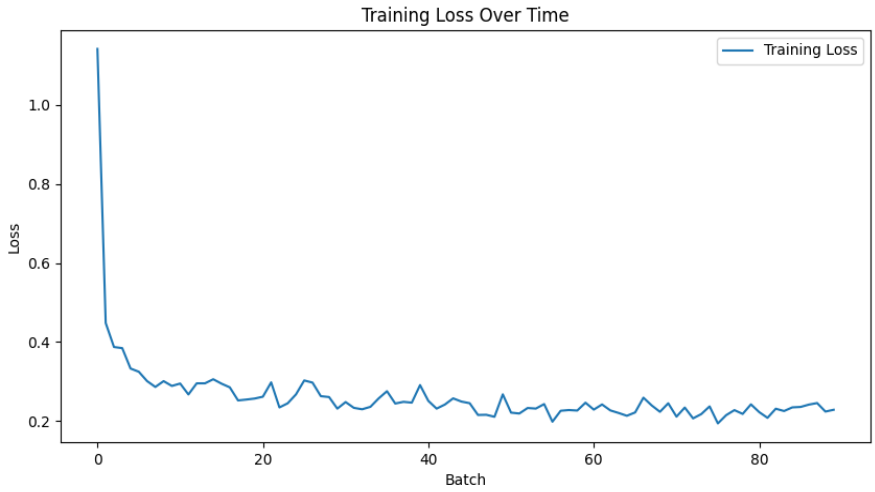
1. 学习率（Learning Rate）：降低学习率可能有助于模型更细致地逼近最优解。例如，可以尝试将学习率从 0.01 调整到 0.001 或更低。
2. 权重衰减（Weight Decay）：增加权重衰减的值可能会帮助正则化模型，减少过拟合的风险。但请注意，过大的权重衰减值可能导致模型欠拟合。
3. 批量大小（Batch Size）：增加批量大小可以提供更稳定的梯度估计，但同时也会增加内存的使用量。可以尝试将批量大小从 64 增加到 128 或更大。
4. 训练周期（Epochs）：增加训练周期可以使模型有更多的时间学习数据特征，但同时也可能导致过拟合。需要通过验证集来监控模型的性能，以确定最佳的训练周期。
5. 优化器（Optimizer）：虽然代码中使用的是 Adam 优化器，但可以尝试其他优化器，如 SGD 或 RMSprop，并调整其参数（例如动量和衰减率）。
6. 激活函数：虽然代码中使用了 ReLU 激活函数，但可以尝试其他激活函数，如 LeakyReLU 或 ELU，以观察它们对模型性能的影响。
7. 正则化技术：除了权重衰减，还可以尝试使用 Dropout 或批量归一化（Batch Normalization）等技术。

1.优化器表现

①当使用 Adam 优化器时，得到如下结果

Accuracy of the network on the 10000 test images: 94.29%

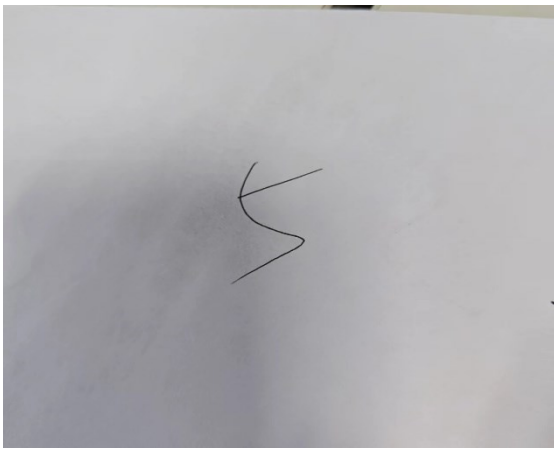
损失曲线：



使用 Adam 优化器下的损失曲线

此时用手写体图片来测试，结果如下

The predicted digit for the image is: 5

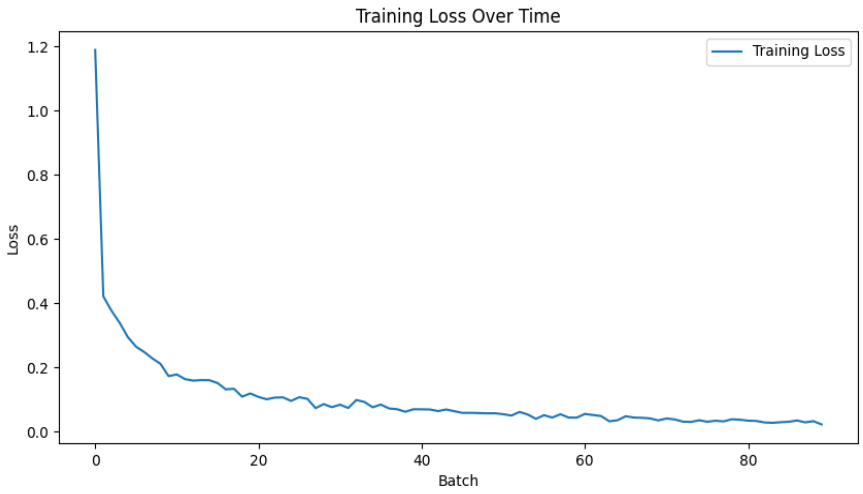


手写数字 5

可以看到模型是有效的。

②当使用 SGD 优化器时，得到如下结果，结果表明，无论是在损失曲线的收敛度还是模型预测的准确率上来看，SGD 都要优于 Adam。于是之后在调节学习率和 batch size，epochs 等我们都以 SGD 为基准。

Accuracy of the network on the 10000 test images: 97.76%



使用 SGD 优化器得到的损失曲线

2.经过不断调参，我们得到各个参数下的准确率，如下表所示。

优化器	Learning Rate	Weight Decay	Batch Size	Epochs	准确率
Adam	0.01	1e-5	64	10	94.29%
SGD	0.01	1e-5	64	10	97.76%
SGD	0.001	1e-5	64	10	95.24%
SGD	0.01	1e-5	64	30	98.30%
SGD	0.01	1e-5	128	30	98.30%

于是，我们可以得出以下结论：

在不改变网络结构的条件下，尝试调节参数，如学习率，权重下降，batchsize 和 epochs 会影响模型预测的准确率。通过多次调整参数，进行简要对比。发现，在适当的学习率下，适度地调高 epochs 和 batchsize 可能会提高模型的准确率，但不能过度训练，否则会加长训练时间，甚至造成过拟合。我得到的最优参数如下：

使用 SGD 优化器，学习率为 0.01 ，Batch Size 为 64 ，Epochs 为 30 下得到最优准确率为 98.30%。

4.2.思考如何通过修改网络结构进一步提高模型精度？论述修改网络结构取得效果的基本原理？请给出实验结果。

要通过修改网络结构提高模型精度，可以考虑以下几个方向：

1. 增加网络深度：增加更多的隐藏层可以使网络学习更复杂的特征，这通常能够提高模型的表达能力。
2. 调整网络宽度：增加隐藏层的神经元数量可以增加网络的学习能力，但同时也会增加计算复杂度。
3. 引入卷积层：尽管 MNIST 是灰度图像，但卷积神经网络（CNN）在图像识别任务中通常表现更好，因为它们能够捕捉局部空间特征。
4. 引入池化层：池化层可以降低特征维度，减少过拟合的风险，并提高模型对图像平移的不变性。

5. 正则化技术：如 Dropout 或批量归一化（Batch Normalization），可以帮助模型泛化得更好，减少过拟合。
6. 引入注意力机制：注意力机制可以帮助模型集中于图像中的重要部分，提高识别精度。

基本原理：

- 网络深度和宽度：更深或更宽的网络可以捕捉更复杂的特征，但同时也可能导致过拟合和增加训练难度。
- 卷积层：卷积层通过局部感受野捕捉空间特征，通常比全连接层更适合处理图像数据。
- 池化层：池化层通过降低特征维度减少参数数量，减少过拟合，同时保持对图像平移的不变性。
- 正则化技术：Dropout 随机丢弃一些神经元的激活，Batch Normalization 通过规范化层的输入来加速训练并提高泛化能力。
- 注意力机制：注意力机制使模型能够动态地关注图像的重要部分，提高识别的准确性。

于是我做了如下变动：
添加了一个卷积层和一个池化层来提取图像特征，同时增加了一个 Dropout 层来减少过拟合。需要注意的是，由于添加了卷积层，因此需要调整第一个全连接层的输入维度，以匹配卷积层输出的特征图维度。

最后得到的结果如下，发现在保持超参数不变下，由于我们作出上面的改动，最后的准确率得到了进一步的提升。从最优超参的 98.30%提高到 98.93%，进一步验证变动想法的有效性。最后修改的代码见附录。

Accuracy of the network on the 10000 test images: 98.93%

```
[Epoch 30, Batch 700] loss: 0.000
[Epoch 30, Batch 800] loss: 0.000
[Epoch 30, Batch 900] loss: 0.001
Finished Training
Accuracy of the network on the 10000 test images: 98.93%
```

修改网络框架后的最优结果

实验结论：

1. 神经网络基本单元的正确实现：
   通过本次实验，我们学习到了确保神经网络中每一层正确实现的重要性。关键在于确定每层的输入和输出维度，并理解前向传播的数学表达式以及反向传播中梯度的计算和权重的更新过程。这些理论基础对于构建有效的神经网络至关重要。
2. 不改变网络结构提高精度的方法：
   实验中，我们探索了多种在不改变网络结构的情况下提高模型精度的方法。通过对输入数据进行归一化和数据增强、对损失函数进行正则化、调整训练参数如 Dropout 比例、以及优化学习率等策略，我们能够有效提升模型的性能。这些方法证明了即使在不改变网络结构的前提下，通过细致的参数调整和数据预处理，也能显著提高模型的精度。
3. 通过修改网络结构提高精度：
   实验还展示了通过修改网络结构来提高模型精度的可能性。我们添加了一个卷积层和一个池化层来提取图像特征，同时增加了一个 Dropout 层来减少过拟合，成功提升了模型的识别精度。这些改进表明，网络结构的优化对于提高模型性能具有重要作用。
4. 实验结果：
   在本次实验中，我们通过使用 SGD 优化器、调整学习率、批量大小和训练周期等参数，最终在 MNIST 数据集上达到了 98.93%的准确率。这一结果不仅验证了我们对神经网络理论和实践的理解，也展示了通过合理调整参数和优化网络结构，可以显著提高手写数字识别模型的性能。
5. 实验反思：
   本次实验让我们深刻认识到深度学习模型构建和优化的复杂性。我们了解到，除了网络结构和参数调整，还有其他因素如数据预处理、模型正则化等也对模型性能有重要影响。此外，实验过程中我们也遇到了一些挑战，如过拟合和训练时间的平衡，这些经验将有助于我们在未来的深度学习项目中做出更好的决策。

指导教师批阅意见：

成绩评定：

指导教师签字：
年　　月　　日

备注：

附录:
修改网络结构后的代码

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from PIL import Image

# 1. 加载MNIST数据集
transform = transforms.Compose([
    transforms.ToTensor(),
```

```
        transforms.Normalize((0.5,), (0.5,))
    ])

    trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

    testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

    # 2. 定义神经网络模型
    class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            # 添加一个卷积层
            self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)  # 输入通道为1，输出通道为32
            # 添加一个池化层
            self.pool = nn.MaxPool2d(2, 2)  # 2x2 池化
            self.fc1 = nn.Linear(32 * 14 * 14, 512)  # 调整全连接层的输入维度
            self.fc2 = nn.Linear(512, 256)
            self.fc3 = nn.Linear(256, 10)
            # 添加一个 Dropout 层
            self.dropout = nn.Dropout(0.5)

        def forward(self, x):
            # 应用卷积层和池化层
            x = self.pool(torch.relu(self.conv1(x)))
            # 展平特征图以输入到全连接层
            x = x.view(-1, 32 * 14 * 14)
            x = torch.relu(self.fc1(x))
            x = self.dropout(torch.relu(self.fc2(x)))  # 应用 Dropout
            x = self.fc3(x)
            return x

    net = Net()

    # 3. 定义损失函数和优化器
    criterion = nn.CrossEntropyLoss()
```

```python
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

# 4. 训练模型
losses = []  # 用于记录损失的列表
for epoch in range(30):  # 训练30个epoch
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if (i+1) % 100 == 0:  # 每100个batch打印一次loss
            losses.append(running_loss / 100)  # 将平均loss添加到列表中
            print(f'[Epoch {epoch+1}, Batch {i+1}] loss: {running_loss / 100:.3f}')
            running_loss = 0.0
    print('Finished Training')

# 5. 测试模型
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy of the network on the 10000 test images: {accuracy:.2f}%')

# 6. 绘制损失曲线
plt.figure(figsize=(10, 5))
plt.plot(losses, label='Training Loss')
plt.xlabel('Batch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.legend()
```

```
    plt.show()

    # 7. 预测单张图片
    def predict_image(image_path):
        # 加载图片
        image = Image.open(image_path).convert('L')  # 转换为灰度图
        # 预处理
        transform = transforms.Compose([
            transforms.Resize((28, 28)),
            transforms.ToTensor(),
            transforms.Normalize((0.5,), (0.5,))
        ])
        image = transform(image).unsqueeze(0)  # 增加一个批次维度
        # 预测
        with torch.no_grad():
            output = net(image)
            _, predicted = torch.max(output, 1)
        return predicted.item()

    # 使用模型预测一张图片
    image_path = r"E:\Grade3\智能\number5.jpg"  # 替换为你的图片路径
    predicted_digit = predict_image(image_path)
    print(f'The predicted digit for the image is: {predicted_digit
    }')
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。