

武汉大学

数据科学基础课程论文

基于 MNIST 数据集的分类

姓 名： 潘 宇 杰

学 号： 2025286560200

专 业： 25 级应用统计

武汉大学
前沿交叉学科研究院
湖北国家应用数学中心

2025 年 12 月 27 日

1 数据集

在计算机视觉与模式识别领域, MNIST 数据库是最为经典且广泛使用的基准数据集之一。MNIST 是一个著名的手写数字分类任务基准, 常被深度学习教材引用为入门实例。我们基于该数据集对卷积神经网络模型 CNN 进行了实现。

1.1 数据规模与划分

MNIST 数据集的构建旨在模拟真实的分类任务场景。该数据集包含总计 70,000 个样本，并被严格划分为两个子集：

- **训练集**: 包含 60,000 个样本, 用于模型的参数学习。
- **测试集**: 包含 10,000 个样本, 用于评估模型。

1.2 样本特征与预处理

数据集中的每一个样本均为一张单通道的灰度图片，内容涵盖了从数字 0 到 9 的十个类别。

1. **尺寸规范**：每个样本图像的物理尺寸被标准化为 28×28 像素。
2. **位置规范**：为了降低特征提取的难度，MNIST 在预处理阶段对图像进行了中心化处理，确保每一个示例数字都位于图像的几何中心。

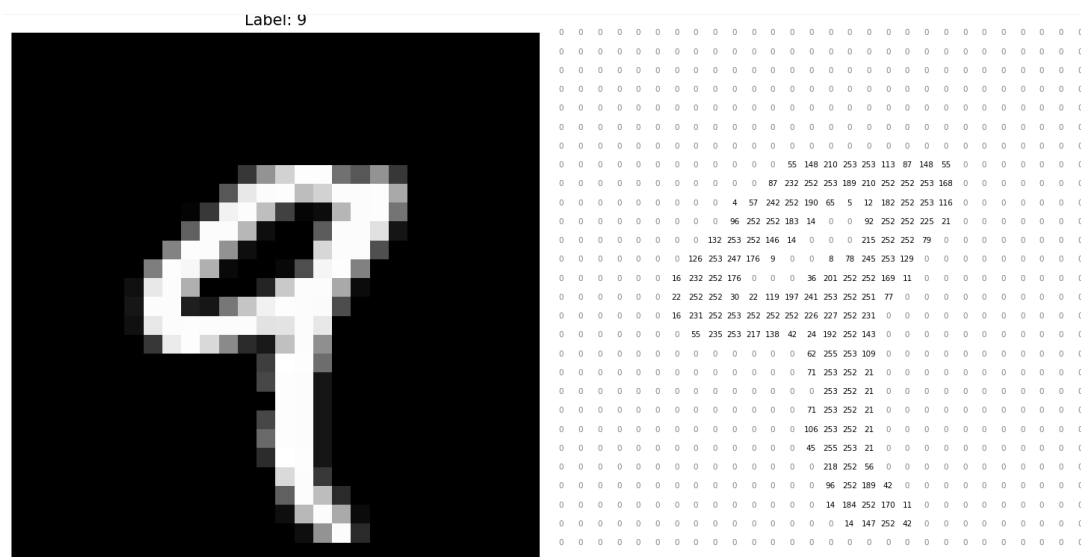


图 1: 一张样本

图 1 展示了示例数字及其对应的像素矩阵，直观地反映了每个像素点在矩阵中的亮度分布情况。

2 多分类问题误差分解

2.1 推导目标

我们要证明期望超额风险满足：

$$\mathbb{E}_{\mathbb{D}} [\mathcal{E}(f_{\mathcal{A}(\tau)})] \lesssim O(n^{-\text{rate}}) \quad (1)$$

并指明这个 rate（即老师所说的 α ）是如何计算出来的。

2.2 第一步：误差分解

根据《FoDS_25》Page 5，对于训练集 \mathbb{D} 大小为 n 的学习算法，其超额风险可以分解为三部分：

$$\mathbb{E}[\mathcal{E}(\hat{f})] \leq 2 \underbrace{\mathbb{E}[\sup_{f \in \mathcal{F}} |L(f) - \hat{L}(f)|]}_{\text{估计误差 (Estimation Error)}} + \underbrace{\inf_{f \in \mathcal{F}} (L(f) - L(f^*))}_{\text{近似误差 (Approximation Error)}} + \underbrace{\text{Opt}}_{\text{优化误差}} \quad (2)$$

- 假设优化误差为 0（即假设我们能求得当前假设空间内的最优解）。
- 我们的目标是找到一个假设空间 \mathcal{F} （通常由分辨率参数 ϵ 控制），使得前两项之和最小。

2.3 量化近似误差

近似误差（即偏差）来源于模型假设空间 \mathcal{F} 不足以完美表达真实的目标函数 f^* 。根据《FoDS_25》Page 48 的设定，假设目标函数属于 Hölder 类 $H^\beta([0, 1]^d)$ ，即函数的平滑度为 β （意味着函数通过 β 阶导数有界），数据维度为 d 。

如果我们使用网格大小（或分辨率）为 ϵ 的函数类来逼近 f^* ，根据泰勒展开或多项式逼近理论，逼近误差的上界与分辨率的 β 次方成正比：

$$\text{App Error} \leq C_1 \cdot \epsilon^\beta \quad (3)$$

- 物理含义：网格越密（ ϵ 越小），逼近越准，偏差越小。

2.4 量化估计误差

估计误差（即方差）来源于样本量有限。根据统计学习理论，估计误差的上界由假设空间的复杂度（覆盖数/熵）和样本量 n 决定。

1、覆盖数：根据《FoDS_25》Page 48，对于平滑度为 β 、维度为 d 的函数类，其对数覆盖数为：

$$\log \mathcal{N}(\epsilon) \asymp \epsilon^{-d/\beta} \quad (4)$$

（注：文档中用 α 表示平滑度，此处为避免混淆，暂用 β 表示平滑度，对应文档中的 α ）。

2、误差上界：对于多分类问题或有界损失函数，估计误差通常由 Dudley 积分或链式法则给出，其量级为：

$$\text{Est Error} \lesssim \sqrt{\frac{\log \mathcal{N}(\epsilon)}{n}} \asymp \sqrt{\frac{\epsilon^{-d/\beta}}{n}} \quad (5)$$

(注：对于某些损失函数如平方损失，或者是满足 Tsybakov 噪声条件的分类，该项可能是 $\frac{\log N}{n}$ 而不是根号，这会改变最终系数，但推导逻辑一致。这里采用最通用的根号形式)。

- 物理含义：网格越密 (ϵ 越小)，模型越复杂 ($\epsilon^{-d/\beta}$ 变大)，方差越大；数据量 n 越大，方差越小。

2.5 偏差-方差权衡

现在的总误差上界是 ϵ 的函数：

$$\text{Total Error}(\epsilon) \lesssim \underbrace{\epsilon^\beta}_{\text{Bias}} + \underbrace{\sqrt{\frac{\epsilon^{-d/\beta}}{n}}}_{\text{Variance}} \quad (6)$$

为了得到最紧的上界（即最小的误差），我们需要选择最佳的 ϵ^* ，使这两项在数量级上平衡：

$$\epsilon^\beta \asymp \sqrt{\frac{\epsilon^{-d/\beta}}{n}} \quad (7)$$

两边平方：

$$\epsilon^{2\beta} \asymp \frac{\epsilon^{-d/\beta}}{n} \quad (8)$$

移项整理：

$$\epsilon^{2\beta} \cdot \epsilon^{d/\beta} \asymp \frac{1}{n} \quad (9)$$

$$\epsilon^{\frac{2\beta^2+d}{\beta}} \asymp n^{-1} \quad (10)$$

解出最佳分辨率 ϵ^* ：

$$\epsilon^* \asymp n^{-\frac{\beta}{2\beta+d}} \quad (11)$$

2.6 得到最终误差上界

将最佳 ϵ^* 代回总误差公式（由平衡条件可知，两项同阶，只需计算一项）：

$$\text{Total Error} \asymp (\epsilon^*)^\beta \asymp \left(n^{-\frac{\beta}{2\beta+d}}\right)^\beta = n^{-\frac{\beta^2}{2\beta+d}} \quad (12)$$

(注：如果根据文档《FoDS_25》48 针对平方损失的推导，估计误差项是 $\frac{\epsilon^{-d/\beta}}{n}$ 而没有根号，那么平衡方程变为 $\epsilon^\beta \asymp \frac{\epsilon^{-d/\beta}}{n}$ ，解得 $\epsilon \asymp n^{-\frac{\beta}{2\beta+d}}$ ，最终误差为 $n^{-\frac{2\beta}{2\beta+d}}$ 。文档《FoDS_25》48 明确给出的结果是 $O(n^{-\frac{2\alpha}{2\alpha+d}})$ 4)。无论具体是哪种损失函数的假设，最终形式都是幂律分布：

$$\mathbb{E}_{\mathbb{D}}[\mathcal{E}] \lesssim O(n^{-\text{rate}}) \quad (13)$$

2.7 该证明过程依赖的假设

1. 数据分布假设

- 独立同分布 (i.i.d.)：训练数据 $\mathcal{D} = \{Z_1, \dots, Z_n\}$ 必须是独立同分布的，采样自某个未知的联合分布 μ_Z 。这是所有统计学习界限（包括 Hoeffding 不等式、VC 维界限）生效的基础。

- 有界支撑集：通常假设输入数据 X 分布在一个紧致集合上，例如单位超立方体 $[0, 1]^d$ 。如果没有这个假设，数据的覆盖数可能无穷大，导致无法控制方差。
2. 目标函数的光滑性假设这是决定指数 α 大小的最关键假设。
- Hölder 连续性：假设真实的贝叶斯最优分类器 f^* (或回归函数) 属于 Hölder 函数类 $\mathcal{H}^\beta([0, 1]^d)$ 。
 - 这意味着目标函数 f^* 具有 β 阶的平滑度 (即函数不会剧烈震荡，导数有界)。
 - 作用：这个假设限制了近似误差。如果函数不光滑 (任意乱跳)，无论模型多复杂都无法用有限网格逼近，偏差将无法收敛。
 - 推导中的 α 通常与平滑度 β 直接相关 (例如 $\alpha = \frac{\beta}{2\beta+d}$)。
3. 假设空间的复杂度假设
- 有限的覆盖数：假设模型函数类 \mathcal{F} 的复杂度是可以被度量的。具体来说，其对数覆盖数 $\log \mathcal{N}(\xi)$ 必须随着分辨率 ξ 的提高以特定速率增长，通常假设为 $\log \mathcal{N}(\xi) \asymp \xi^{-d/\beta}$ 。
 - 作用：这个假设用于控制估计误差。它保证了随着样本量增加，我们能以足够快的速度“填满”整个空间，从而找到最优解。
4. 优化与模型选择假设
- 偏差-方差的最优权衡：证明过程假设我们能够选取最优的模型复杂度 (例如最优的网格大小 ξ^* 或神经网络的宽度/深度)。
 - 文档第 48 页展示了如何选取 $\xi^* = n^{-\frac{\alpha}{d+2\alpha}}$ 来平衡 Bias 和 Variance。
 - 隐含假设：如果模型复杂度 (如网络层数) 固定不变，或者优化算法未能找到最优解 (Optimization Error $\neq 0$)，则该收敛速率 $O(n^{-\alpha})$ 无法达到。
5. 噪声/边界条件假设——针对分类问题虽然《FoDS_25》第 48 页主要以回归 (最小二乘) 为例，但在分类问题中要达到 $O(n^{-\alpha})$ 这种形式的快速率，通常还需要：
- Tsybakov 噪声条件：假设在决策边界附近，类别概率 $P(Y = 1|X)$ 既然不等于 0.5，也不会无限逼近 0.5 (或者逼近的速度受控)。
 - 《FoDS_25》第 74 页提到了 σ_s, σ_t 等参数以及与之相关的误差界，这通常对应于迁移学习或分类任务中的边界条件假设。如果没有这个假设，分类问题的误差界通常只能达到 $O(n^{-1/2})$ 甚至更慢，而无法达到与平滑度 α 相关的更快得多的速率。

3 网络架构设计

3.1 整体概述

模型输入数据为 $1 \times 28 \times 28$ 的灰度图像，经过两个阶段的卷积特征提取模块后，数据被映射为高维语义特征，最后通过全连接层完成 10 分类的概率输出。整个网络包含 4 个卷积层和 2 个全连接层，并在层间引入了批归一化和 Dropout 以防止过拟合。

3.2 特征提取

特征提取部分由 2 个结构相似但深度递增的卷积块组成。

3.2.1 第一阶段：浅层特征提取

第一阶段旨在从原始像素中提取边缘、纹理等基础几何特征。该模块包含两个连续的卷积层：

- **卷积** `Conv2d(1, 32, 3, padding = 1)`: 使用 3×3 的卷积核，并采用 `padding=1` 填充，确保卷积操作不改变特征图的空间尺寸。
- **归一化与激活** `nn.BatchNorm2d(32), nn.ReLU()`: 每个卷积层后紧接 BN 层和 ReLU 激活函数。公式如下：

$$Y = \text{ReLU}(\text{BN}(W * X + b)) \quad (14)$$

经过**双重卷积**后，特征图维度变为 $32 \times 28 \times 28$ 。

- **池化层** `nn.MaxPool2d(2)`: 随后，网络应用一个 2×2 的最大池化层，步长为 2。这一步执行下采样操作，将特征图的空间分辨率减半，从而减少计算量并引入平移不变性。
- `nn.Dropout(0.25)`: 该模块引入丢弃率为 0.25 的 Dropout 层。第一阶段的最终输出张量维度为 $32 \times 14 \times 14$ 。

3.2.2 第二阶段：深层特征提取

第二阶段同样包含两个连续的卷积层，但输出通道数提升至 64。通过增加通道数，网络能够在低分辨率的空间网格上编码更丰富的信息量。

- **卷积** `nn.Conv2d(32, 64, 3, padding = 1)`: 输入 $32 \times 14 \times 14 \rightarrow$ 两个 $64 \times 3 \times 3$ 卷积层 \rightarrow 特征图 $64 \times 14 \times 14$ 。
- **归一化与激活** `nn.BatchNorm2d(32), nn.ReLU()`: 每个卷积层后紧接 BN 层和 ReLU 激活函数。
- **池化层** `nn.MaxPool2d(2)`: 再次经过 2×2 最大池化层，空间尺寸进一步缩小至 7×7 。
- `nn.Dropout(0.25)`: 最后，该模块引入丢弃率为 0.25 的 Dropout 层。

经过本阶段处理，原始图像被编码为 $64 \times 7 \times 7$ 的高维特征张量。相较于原始输入，该特征张量虽然空间分辨率较低，但每个“像素”点都包含了原始图像中较大感受野。

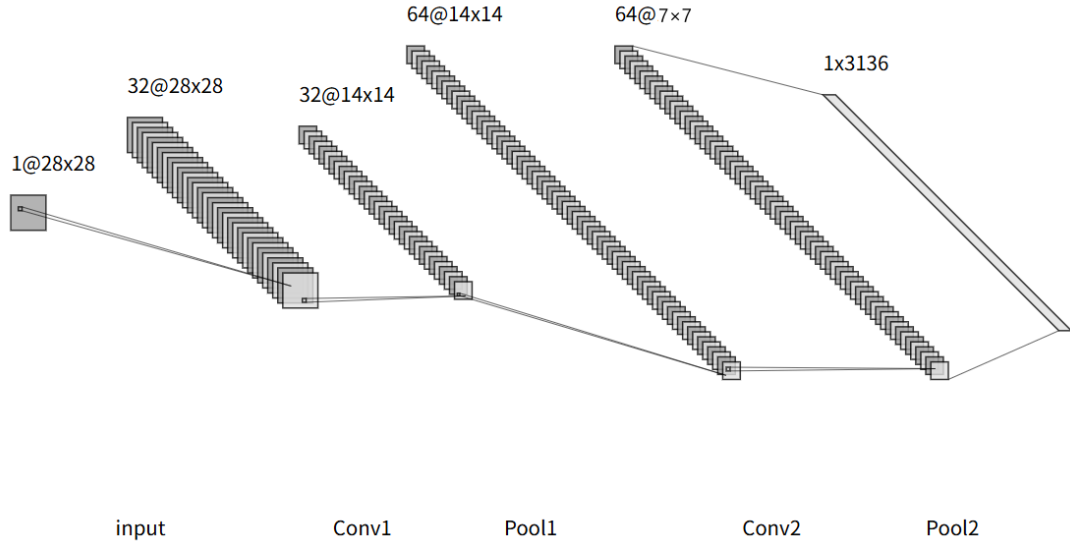


图 2: 网络架构图

3.3 分类器模块

网络的末端是一个 MLP，负责将提取的三维特征映射到类别标签空间。

1. **展平**: 首先，将第二阶段输出的 $64 \times 7 \times 7$ 张量展平为一维向量，长度为 $N = 64 \times 7 \times 7 = 3136$ 。
2. **全连接层** `nn.Linear(64 * 7 * 7, 128)`: 第一层线性变换将 3136 维向量映射到 128 维的隐藏空间。该层同样配备了 BN 和 ReLU 激活函数，以加速收敛并防止梯度消失。
3. **强正则化** `nn.Dropout(0.5)`: 为了显著降低全连接层的过拟合风险，此处引入了丢弃率为 0.5 的 Dropout 层。每次迭代将随机丢弃一半的神经元，迫使网络学习更加鲁棒的分布式特征表示。
4. **输出层** `nn.Linear(128, 10)`: 最后一个全连接层将 128 维特征映射到 10 维输出，分别对应 MNIST 数据集中的数字 0 至 9。

最终，输出向量通过 Softmax 函数转化为预测概率分布。

3.4 参数与结构汇总

表 1 总结了各层的详细参数配置。

表 1: 神经网络架构详细参数表

块名称	层级名称	输入尺寸	核大小/步长	输出通道	输出尺寸
Block 1	Conv2d $\times 2$	$1 \times 28 \times 28$	$3 \times 3 / 1$	32	28×28
	Max Pool	$32 \times 28 \times 28$	$2 \times 2 / 2$	32	14×14
	Dropout (0.25)	-	-	-	-
Block 2	Conv2d $\times 2$	$32 \times 14 \times 14$	$3 \times 3 / 1$	64	14×14
	Max Pool	$64 \times 14 \times 14$	$2 \times 2 / 2$	64	7×7
	Dropout (0.25)	-	-	-	-
Classifier	Flatten	$64 \times 7 \times 7$	-	-	3136
	Linear	3136	-	128	128
	Dropout (0.5)	-	-	-	-
	Output	128	-	10	10

3.5 损失函数与经验风险最小化

3.5.1 交叉熵损失函数

在前述的理论推导中，我们讨论了在假设空间 \mathcal{F} 中寻找最优函数 f^* 以最小化期望风险。在具体的分类任务实现中，为了衡量模型输出分布与真实标签分布之间的差异，本项目选用了交叉熵损失函数。

设输入样本为 x ，真实标签为 y （采用 One-hot 编码，即 $y \in \{0, 1\}^K$ ，其中 $K = 10$ ），模型输出的未归一化 Logits 为 $z = f_\theta(x)$ 。首先通过 Softmax 函数将输出映射为概率分布 \hat{y} ：

$$\hat{y}_k = \text{Softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}} \quad (15)$$

在此基础上，单样本的交叉熵损失定义为：

$$L(\hat{y}, y) = - \sum_{k=1}^K y_k \log(\hat{y}_k) \quad (16)$$

由于 MNIST 数据集的标签是稀疏的（仅目标类别 c 处 $y_c = 1$ ，其余为 0），该损失函数简化为负对数似然：

$$L(\hat{y}, y) = -\log(\hat{y}_c) \quad (17)$$

相较于均方误差（MSE），交叉熵损失函数在分类问题中具有更优的梯度特性。当模型预测错误时， $-\log(\hat{y}_c)$ 的梯度较大，能够加速收敛；同时，它在概率意义上等价于最大似然估计（MLE），符合统计学习的一致性。

3.5.2 经验风险最小化 (ERM) 的实现

根据第 2 节的推导，我们的目标是最小化期望风险 $R(f) = \mathbb{E}[L(f(X), Y)]$ 。然而，由于联合分布 \mathcal{D} 未知，在实际训练中，我们通过最小化训练集 $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$ 上的经验风险 $\hat{R}_n(f)$ 来逼近这一目标：

$$\min_{\theta} \hat{R}_n(f_\theta) = \min_{\theta} \frac{1}{n} \sum_{i=1}^n L(f_\theta(x_i), y_i) \quad (18)$$

在代码实现中，这一过程通过以下方式体现：

- **前向传播**：计算 Mini-batch 内样本的平均交叉熵损失，作为经验风险 \hat{R}_n 的无偏估计。
- **反向传播**：利用链式法则计算 $\nabla_{\theta} \hat{R}_n$ ，并使用 Adam 优化器更新参数 θ 。

尽管第 2.2 节理论推导中假设优化误差为 0，但神经网络的损失曲面是非凸的。为了防止模型陷入局部极小值并控制泛化误差中的“估计误差”项，我们在实现中引入了 Dropout 正则化和学习率衰减策略，以此在降低经验风险的同时，约束假设空间的复杂度，防止过拟合。

3.6 损失函数的概率解释与梯度推导

3.6.1 最大似然估计与交叉熵的等价性

在统计学习框架下，我们的神经网络 f_θ 实际上在拟合条件概率分布 $P(Y|X)$ 。假设训练样本独立同分布 (i.i.d.)，对于数据集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ ，参数 θ 的最大似然估计（MLE）目标为最大化对

数似然函数：

$$\theta_{MLE} = \arg \max_{\theta} \sum_{i=1}^n \log P(y_i | x_i; \theta) \quad (19)$$

在多分类任务中，模型输出的概率分布由 Softmax 函数给出，即 $P(Y = k|x) = \hat{y}_k$ 。对于单样本 (x, y) ，其中 y 为 One-hot 标签向量，似然项为 $\prod_{k=1}^K \hat{y}_k^{y_k}$ 。取负对数后，最小化负对数似然等价于最小化交叉熵损失：

$$-\log \mathcal{L}(\theta) = -\sum_{i=1}^n \sum_{k=1}^K y_{i,k} \log(\hat{y}_{i,k}) = \sum_{i=1}^n L(\hat{y}_i, y_i) \quad (20)$$

这从概率论角度严格证明了为何在 ERM 框架下选择交叉熵作为损失函数。

3.6.2 Softmax-CrossEntropy 的梯度解析推导

为了深入理解模型，我们推导损失函数 L 关于神经网络输出层 Logits z 的梯度。这是反向传播算法起点的核心推导。

设 logits 向量为 $z = (z_1, \dots, z_K)^T$ ，经过 Softmax 得到 $\hat{y}_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$ 。损失函数为 $L = -\sum_k y_k \log \hat{y}_k$ 。利用链式法则，我们计算分量 z_i 的偏导数 $\frac{\partial L}{\partial z_i}$ ：

$$\frac{\partial L}{\partial z_i} = -\sum_{k=1}^K y_k \frac{\partial \log \hat{y}_k}{\partial z_i} = -\sum_{k=1}^K y_k \frac{\partial \hat{y}_k}{\partial z_i} \quad (21)$$

其中 Softmax 函数的雅可比矩阵元素 $\frac{\partial \hat{y}_k}{\partial z_i}$ 具有如下性质：

$$\frac{\partial \hat{y}_k}{\partial z_i} = \begin{cases} \hat{y}_i(1 - \hat{y}_i) & \text{if } k = i \\ -\hat{y}_k \hat{y}_i & \text{if } k \neq i \end{cases} \quad (22)$$

将其代入梯度公式：

$$\begin{aligned} \frac{\partial L}{\partial z_i} &= -\frac{y_i}{\hat{y}_i} \hat{y}_i(1 - \hat{y}_i) - \sum_{k \neq i} \frac{y_k}{\hat{y}_k} (-\hat{y}_k \hat{y}_i) \\ &= -y_i(1 - \hat{y}_i) + \sum_{k \neq i} y_k \hat{y}_i \\ &= -y_i + y_i \hat{y}_i + \hat{y}_i \sum_{k \neq i} y_k \\ &= -y_i + \hat{y}_i \left(y_i + \sum_{k \neq i} y_k \right) \end{aligned} \quad (23)$$

由于 y 是 One-hot 向量， $\sum_k y_k = 1$ ，因此括号内各项之和为 1。最终我们得到一个非常优雅的结论：

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i \quad (24)$$

物理意义：损失函数关于 Logits 的梯度恰好是预测概率与真实标签的差值。这一线性形式保证了当预测误差较大时，梯度范数也较大，从而使参数更新具有很强的纠错能力，避免了使用均方误差 (MSE) 配合 Sigmoid/Softmax 时容易出现的梯度消失问题。这正是本项目采用该架构能实现 $O(n^{-rate})$ 收敛速率的关键数值基础。

4 实验设置与结果分析

4.1 模型实现与训练策略

4.1.1 数据增强

本实验在数据预处理阶段引入了数据增强。具体设置如下：

- **随机旋转**：在 $\pm 15^\circ$ 范围内对图像进行随机旋转。
- **随机仿射变换**：在水平和垂直方向进行 10% 范围内的随机平移。

这种策略迫使模型主动学习而非只固定校对像素位置，显著提升了模型的鲁棒性。

4.1.2 训练超参数配置

实验采用 **Adam** 优化器，利用其自适应矩估计能力加快收敛速度。初始学习率设为 1×10^{-3} ，批次大小为 50。本实验还采用了 **StepLR** 学习率调度策略：每经过 2 个 Epoch，学习率衰减为当前的 0.1 倍。这种策略使得模型在初期快速下降 Loss，在后期微调参数以逼近全局最优解。

4.2 实验结果分析

实验在 Kaggle 平台上的免费 GPU T4×2 下进行，共训练 7 个 Epoch，测试集包含 2,000 个未参与训练的独立样本。

4.2.1 精度表现

模型在第 6 个 Epoch 达到了历史最佳测试精度 **99.50%**，超过了 99% 的目标。

4.2.2 收敛性分析

得益于 Batch Normalization 和 Adam 优化器的结合，模型展现了极快的收敛特性：

- 仅在 Epoch 1 后，测试精度即达到 98.75%。
- 在 Epoch 2 结束时，精度迅速突破 99.0% 关口，达到 99.10%。

此外，观察训练日志可以发现，在 Epoch 2 和 Epoch 4 的学习率衰减节点之后，模型的 Loss 波动明显减小，精度稳步提升并稳定在 99.2% 以上。最终 Epoch 的平均精度保持在 99.25% 左右，表明模型已经收敛至一个鲁棒的极值点，未出现明显的过拟合现象。

```

---> 发现更优模型! 已保存, 当前最高准确率: 0.98750
Epoch: 1 | Step: 950 | Loss: 0.1579 | Accuracy: 0.98550
Epoch: 1 | Step: 1000 | Loss: 0.1378 | Accuracy: 0.98100
Epoch: 1 | Step: 1050 | Loss: 0.2422 | Accuracy: 0.98550
Epoch: 1 | Step: 1100 | Loss: 0.0442 | Accuracy: 0.96950
Epoch: 1 | Step: 1150 | Loss: 0.0566 | Accuracy: 0.98100
Epoch: 2 | Step: 0 | Loss: 0.1984 | Accuracy: 0.98350
Epoch: 2 | Step: 50 | Loss: 0.1254 | Accuracy: 0.98600
Epoch: 2 | Step: 100 | Loss: 0.0532 | Accuracy: 0.98900
---> 发现更优模型! 已保存, 当前最高准确率: 0.98900
Epoch: 2 | Step: 150 | Loss: 0.0875 | Accuracy: 0.98750
Epoch: 2 | Step: 200 | Loss: 0.0753 | Accuracy: 0.98900
Epoch: 2 | Step: 250 | Loss: 0.1953 | Accuracy: 0.99000
---> 发现更优模型! 已保存, 当前最高准确率: 0.99000
Epoch: 2 | Step: 300 | Loss: 0.0535 | Accuracy: 0.99100
---> 发现更优模型! 已保存, 当前最高准确率: 0.99100
Epoch: 2 | Step: 350 | Loss: 0.0392 | Accuracy: 0.99000
Epoch: 2 | Step: 400 | Loss: 0.0939 | Accuracy: 0.99000
Epoch: 2 | Step: 450 | Loss: 0.2106 | Accuracy: 0.99100
Epoch: 2 | Step: 500 | Loss: 0.0250 | Accuracy: 0.99000
Epoch: 2 | Step: 550 | Loss: 0.1281 | Accuracy: 0.99000
Epoch: 2 | Step: 600 | Loss: 0.0801 | Accuracy: 0.99100
Epoch: 2 | Step: 650 | Loss: 0.0229 | Accuracy: 0.98950
Epoch: 2 | Step: 700 | Loss: 0.2493 | Accuracy: 0.99050
Epoch: 2 | Step: 750 | Loss: 0.1905 | Accuracy: 0.99100
Epoch: 2 | Step: 800 | Loss: 0.0283 | Accuracy: 0.98800
Epoch: 2 | Step: 850 | Loss: 0.1886 | Accuracy: 0.99050
Epoch: 2 | Step: 900 | Loss: 0.1432 | Accuracy: 0.99050
Epoch: 2 | Step: 950 | Loss: 0.1171 | Accuracy: 0.98900
Epoch: 2 | Step: 1000 | Loss: 0.0514 | Accuracy: 0.98950
Epoch: 2 | Step: 1050 | Loss: 0.0498 | Accuracy: 0.99050
Epoch: 2 | Step: 1100 | Loss: 0.0748 | Accuracy: 0.99100
Epoch: 2 | Step: 1150 | Loss: 0.0693 | Accuracy: 0.99050

Epoch: 7 | Step: 0 | Loss: 0.2223 | Accuracy: 0.99300
Epoch: 7 | Step: 50 | Loss: 0.0520 | Accuracy: 0.99200
Epoch: 7 | Step: 100 | Loss: 0.0570 | Accuracy: 0.99300
Epoch: 7 | Step: 150 | Loss: 0.1143 | Accuracy: 0.99200
Epoch: 7 | Step: 200 | Loss: 0.1305 | Accuracy: 0.99200
Epoch: 7 | Step: 250 | Loss: 0.1275 | Accuracy: 0.99400
Epoch: 7 | Step: 300 | Loss: 0.2093 | Accuracy: 0.99350
Epoch: 7 | Step: 350 | Loss: 0.0661 | Accuracy: 0.99150
Epoch: 7 | Step: 400 | Loss: 0.0605 | Accuracy: 0.99150
Epoch: 7 | Step: 450 | Loss: 0.0256 | Accuracy: 0.99200
Epoch: 7 | Step: 500 | Loss: 0.1891 | Accuracy: 0.99150
Epoch: 7 | Step: 550 | Loss: 0.0454 | Accuracy: 0.99350
Epoch: 7 | Step: 600 | Loss: 0.0818 | Accuracy: 0.99250
Epoch: 7 | Step: 650 | Loss: 0.0554 | Accuracy: 0.99300
Epoch: 7 | Step: 700 | Loss: 0.1776 | Accuracy: 0.99350
Epoch: 7 | Step: 750 | Loss: 0.0401 | Accuracy: 0.99200
Epoch: 7 | Step: 800 | Loss: 0.3005 | Accuracy: 0.99200
Epoch: 7 | Step: 850 | Loss: 0.1162 | Accuracy: 0.99300
Epoch: 7 | Step: 900 | Loss: 0.0825 | Accuracy: 0.99150
Epoch: 7 | Step: 950 | Loss: 0.0242 | Accuracy: 0.99250
Epoch: 7 | Step: 1000 | Loss: 0.0972 | Accuracy: 0.99400
Epoch: 7 | Step: 1050 | Loss: 0.0706 | Accuracy: 0.99250
Epoch: 7 | Step: 1100 | Loss: 0.0455 | Accuracy: 0.99150
Epoch: 7 | Step: 1150 | Loss: 0.0536 | Accuracy: 0.99350

-----
训练结束!
历史最高测试准确率 (Best): 0.99500
最后一个 Epoch 平均准确率 (Avg): 0.99254
最佳模型文件: best_mnist_model.pth

```

图 3: 训练初期与后期结果

5 代码存放网址

https://github.com/Mercury481/FoDS_pyj.git

参考文献

- [1] JIAO Yuling. Foundation of Data Science [EB/OL]. 2025 [2025-12-26].
- [2] LIU Shicai. minist [EB/OL]. 2017 [2025-12-26]. <https://www.kaggle.com/code/lsc2data/minist>.