华中科技大学

《计算机视觉导论》
上机实验（一）报告

题目： 基于前馈神经网络的分类任务设计

院　　系	计算机科学与技术学院
专业班级	计科 2301 班
姓　　名	
学　　号	
指导教师	李贤芝

计算机科学与技术学院

目　　录

1 问题定义与理解

1.1 问题描述

实验的任务是设计一个前馈神经网络对四类二维高斯数据进行分类。数据集包含 4000 个样本,每个样本由 2 个特征和 1 个类别标签(1-4)组成。实验要求:

设计至少包含一层隐藏层的前馈神经网络,将数据随机排序后,90%用于训练,10%用于测试评估模型的分类性能

这是一个典型的多分类监督学习问题,需要网络学习四类高斯分布数据在二维空间中的决策边界。

Dataset

	-0.0652368926065586	-0.24951025013912145	1
1	-0.38703454545507426	0.16446657721102112	1
2	0.1405316666412362	0.022166129896849723	1
3	-0.16996107403316196	-0.039068055961165214	1
4	0.20498700225983124	0.24967101504497322	1
5	0.13813409983154287	-0.31776281257687966	1

图 1-1 数据集示意图

1.2 相关理论基础

前馈神经网络是最基本的人工神经网络结构，由输入层、若干隐藏层和输出层组成。

输入层：接收样本的特征向量，本实验为二维特征。

隐藏层：由全连接神经元构成，每个神经元对输入进行加权求和并通过非线性激活函数进行映射，从而增强模型对复杂模式的表达能力。

输出层：输出各类别的预测值，本实验采用四维向量输出，分别对应四个类

别。

网络的训练过程是通过误差反向传播（Backpropagation）与梯度下降优化来不断调整网络权重，使预测结果逐渐逼近真实标签。

激活函数用于引入非线性，使神经网络能够逼近复杂的非线性决策边界。本实验采用 ReLU 作为隐藏层的激活函数，其形式为：

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

ReLU 能有效缓解梯度消失问题，并在实际分类任务中表现良好。

模型参数的更新采用 Adam 优化器，其结合了动量法与自适应学习率的优点，在非凸优化问题中具有较好的收敛性能。相比传统的随机梯度下降（SGD），Adam 能更快适应不同维度的学习速率，提升模型训练效率。

1.3 实验环境

为保证实验的可重复性与高效性，本实验在如下软硬件环境中进行：

显卡：NVIDIA GeForce RTX 4070 Laptop GPU

显存：8 GB GDDR6

内存：32 GB DDR5

CUDA 版本：12.3

操作系统：Windows 11 家庭版

Python 版本：3.11.11

深度学习框架：PyTorch 2.6.0

开发工具：Visual Studio Code (VSCode)

环境管理：Anaconda（虚拟环境隔离与依赖管理）

运行设备：本地 GPU 加速训练

2 模型构建与训练

2.1 数据分析和处理

数据集结构：二维特征 + 单列标签（1–4）。代码中 labels = labels - 1 转换为 0–3 以适配 PyTorch。

```
# 将标签从1-4转换为0-3
    labels = labels - 1
```

对数据进行划分,样本总数 4000 训练集: 3600 样本 (90%),测试集: 400 样本 (10%)

采用固定随机种子（42）提升复现性。同时对数据进行标准化，消除不同特征量纲差异,加速梯度下降收敛。

```
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(
        features, labels, test_size=1 - CONFIG['train_ratio'],
random_state=42, shuffle=True
    )
    # 数据标准化
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

2.2 网络架构与训练过程设计

本实验实现了三类可扩展框架：

DynamicFNN（main.py）：按 CONFIG 动态构建多层线性 + 激活，可选 Dropout。

EnhancedFNN（optimized_main.py）：在上述基础上加入 BatchNorm/可变 Dropout 支持。

多实验自动化管线：通过枚举（架构×激活×超参×优化器×正则化）批量运行

并统一日志记录（CSV），便于横向分析。

最终采用架构 (基于实验优化结果)

```
Input Layer (2) → Hidden Layer (32, ReLU) → Output Layer (4)
```

表 2-1 模型参数选择

超参数	取值	说明
学习率	0.001	标准学习率, 平衡收敛速度与稳定性
批次大小	32	适中批次, 兼顾梯度估计准确性和内存效率
训练轮数	15	充分收敛而不过拟合
优化器	Adam	自适应学习率, 收敛快速
Dropout	0.0	模型简单, 无需额外正则化

浅层网络的优势:参数量少，仅 $2×32 + 32 + 32×4 + 4 = 164 个参数;泛化能力强 测试准确率达 99.75%,过拟合程度仅-1.17%；训练高效，约 3 秒完成 15 个 epoch

激活函数选择: ReLU：计算简单,训练快速；有效缓解梯度消失；实验表明 ReLU 与 Sigmoid 准确率相当(99.5%或 99.75%),但训练更稳定。

训练设计上。采用 mini-batch 训练,每个 batch 后在完整训练集和测试集上评估（控制台没有输出每一步的评估结果，但相关曲线已保存）

```
for epoch in range(CONFIG['epochs']):
        model.train() # 确保模型处于训练模式
        epoch_train_loss = 0
        epoch_train_acc = 0
        epoch_test_loss = 0
        epoch_test_acc = 0
        batch_count = 0

        for i, (batch_features, batch_labels) in enumerate(tra
in_loader):
            batch_features, batch_labels = batch_features.to(d
evice), batch_labels.to(device)

            # 前向传播
            outputs = model(batch_features)
            loss = criterion(outputs, batch_labels)

            # 反向传播和优化
```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # === 在每个mini-batch 训练后，在完整的训练集和测试集上评估 ===
            train_loss, train_acc = evaluate_model(model, full_train_loader, criterion, device)
            test_loss, test_acc = evaluate_model(model, test_loader, criterion, device)

            # 记录数据用于绘制minibatch 曲线
            history['steps'].append(global_step)
            history['train_loss'].append(train_loss)
            history['test_loss'].append(test_loss)
            history['train_acc'].append(train_acc)
            history['test_acc'].append(test_acc)

            # 累加用于计算epoch 平均值
            epoch_train_loss += train_loss
            epoch_train_acc += train_acc
            epoch_test_loss += test_loss
            epoch_test_acc += test_acc
            batch_count += 1

            global_step += 1

        # 计算epoch 平均值并输出
        avg_train_loss = epoch_train_loss / batch_count
        avg_train_acc = epoch_train_acc / batch_count
        avg_test_loss = epoch_test_loss / batch_count
        avg_test_acc = epoch_test_acc / batch_count

        print(f'Epoch [{epoch+1:2d}/{CONFIG["epochs"]}] | '
              f'Train Loss: {avg_train_loss:.4f} | Train Acc: {avg_train_acc:.2f}% | '
              f'Test Loss: {avg_test_loss:.4f} | Test Acc: {avg_test_acc:.2f}%')
    print("--- 训练完成 ---\n")
```

```
--- 开始训练 ---
Epoch [ 1/15] | Train Loss: 0.8623 | Train Acc: 91.87% | Test Loss: 0.8548 | Test Acc: 92.19%
Epoch [ 2/15] | Train Loss: 0.3700 | Train Acc: 98.22% | Test Loss: 0.3597 | Test Acc: 98.83%
Epoch [ 3/15] | Train Loss: 0.1799 | Train Acc: 98.66% | Test Loss: 0.1690 | Test Acc: 99.19%
Epoch [ 4/15] | Train Loss: 0.1118 | Train Acc: 98.59% | Test Loss: 0.1004 | Test Acc: 99.45%
Epoch [ 5/15] | Train Loss: 0.0823 | Train Acc: 98.61% | Test Loss: 0.0706 | Test Acc: 99.60%
Epoch [ 6/15] | Train Loss: 0.0674 | Train Acc: 98.62% | Test Loss: 0.0552 | Test Acc: 99.66%
Epoch [ 7/15] | Train Loss: 0.0587 | Train Acc: 98.63% | Test Loss: 0.0460 | Test Acc: 99.75%
Epoch [ 4/15] | Train Loss: 0.1118 | Train Acc: 98.59% | Test Loss: 0.1004 | Test Acc: 99.45%
Epoch [ 5/15] | Train Loss: 0.0823 | Train Acc: 98.61% | Test Loss: 0.0706 | Test Acc: 99.60%
Epoch [ 6/15] | Train Loss: 0.0674 | Train Acc: 98.62% | Test Loss: 0.0552 | Test Acc: 99.66%
Epoch [ 7/15] | Train Loss: 0.0587 | Train Acc: 98.63% | Test Loss: 0.0460 | Test Acc: 99.75%
Epoch [ 5/15] | Train Loss: 0.0823 | Train Acc: 98.61% | Test Loss: 0.0706 | Test Acc: 99.60%
Epoch [ 6/15] | Train Loss: 0.0674 | Train Acc: 98.62% | Test Loss: 0.0552 | Test Acc: 99.66%
Epoch [ 7/15] | Train Loss: 0.0587 | Train Acc: 98.63% | Test Loss: 0.0460 | Test Acc: 99.75%
Epoch [ 6/15] | Train Loss: 0.0674 | Train Acc: 98.62% | Test Loss: 0.0552 | Test Acc: 99.66%
Epoch [ 7/15] | Train Loss: 0.0587 | Train Acc: 98.63% | Test Loss: 0.0460 | Test Acc: 99.75%
Epoch [ 8/15] | Train Loss: 0.0534 | Train Acc: 98.71% | Test Loss: 0.0407 | Test Acc: 99.72%
Epoch [ 9/15] | Train Loss: 0.0498 | Train Acc: 98.67% | Test Loss: 0.0367 | Test Acc: 99.75%
Epoch [10/15] | Train Loss: 0.0471 | Train Acc: 98.59% | Test Loss: 0.0334 | Test Acc: 99.75%
Epoch [ 8/15] | Train Loss: 0.0534 | Train Acc: 98.71% | Test Loss: 0.0407 | Test Acc: 99.72%
Epoch [ 9/15] | Train Loss: 0.0498 | Train Acc: 98.67% | Test Loss: 0.0367 | Test Acc: 99.75%
Epoch [10/15] | Train Loss: 0.0471 | Train Acc: 98.59% | Test Loss: 0.0334 | Test Acc: 99.75%
Epoch [ 9/15] | Train Loss: 0.0498 | Train Acc: 98.67% | Test Loss: 0.0367 | Test Acc: 99.75%
Epoch [10/15] | Train Loss: 0.0471 | Train Acc: 98.59% | Test Loss: 0.0334 | Test Acc: 99.75%
Epoch [11/15] | Train Loss: 0.0453 | Train Acc: 98.70% | Test Loss: 0.0317 | Test Acc: 99.62%
Epoch [12/15] | Train Loss: 0.0439 | Train Acc: 98.63% | Test Loss: 0.0300 | Test Acc: 99.75%
Epoch [13/15] | Train Loss: 0.0427 | Train Acc: 98.65% | Test Loss: 0.0282 | Test Acc: 99.75%
Epoch [10/15] | Train Loss: 0.0471 | Train Acc: 98.59% | Test Loss: 0.0334 | Test Acc: 99.75%
Epoch [11/15] | Train Loss: 0.0453 | Train Acc: 98.70% | Test Loss: 0.0317 | Test Acc: 99.62%
Epoch [12/15] | Train Loss: 0.0439 | Train Acc: 98.63% | Test Loss: 0.0300 | Test Acc: 99.75%
Epoch [13/15] | Train Loss: 0.0427 | Train Acc: 98.65% | Test Loss: 0.0282 | Test Acc: 99.75%
Epoch [14/15] | Train Loss: 0.0418 | Train Acc: 98.60% | Test Loss: 0.0273 | Test Acc: 99.75%
Epoch [15/15] | Train Loss: 0.0411 | Train Acc: 98.68% | Test Loss: 0.0266 | Test Acc: 99.71%
--- 训练完成 ---
```

图 2-1 训练过程图

3 实验结果与分析

3.1 模型性能分析

在基础实验（main.py）中，采用浅层前馈神经网络结构 [32]，激活函数为 ReLU，优化器为 Adam，学习率 0.001，批次大小 32，训练 15 轮。最终在训练集和测试集上的性能如下表所示：

表 3-1 实验结果

指标	训练集	测试集
损失	0.0407	0.0263
准确率	98.69%	99.75%

从结果来看，测试准确率（99.75%）略高于训练准确率（98.69%），说明模型没有过拟合，反而泛化能力更强。测试损失为 0.0263，表明预测分布与真实分布高度一致。总训练时间 3 秒多（不含评估时间），总步数 1695，计算效率较高。

3.2 训练过程分析

表 3-2 训练结果

Epoch	Train Loss	Train Acc	Test Loss	Test Acc
1	0.8623	91.87%	0.8548	92.19%
5	0.0823	98.61%	0.0706	99.60%
10	0.0471	98.59%	0.0334	99.75%
15	0.0411	98.68%	0.0266	99.71%

从训练过程来看，收敛迅速前 5 个 epoch 损失由 0.86 降至 0.08，准确率由 92% 升至接近 99%。5~15 轮损失下降趋缓，准确率稳定在 98.6%~99.7%。

测试准确率始终与训练持平甚至更高,无过拟合迹象。

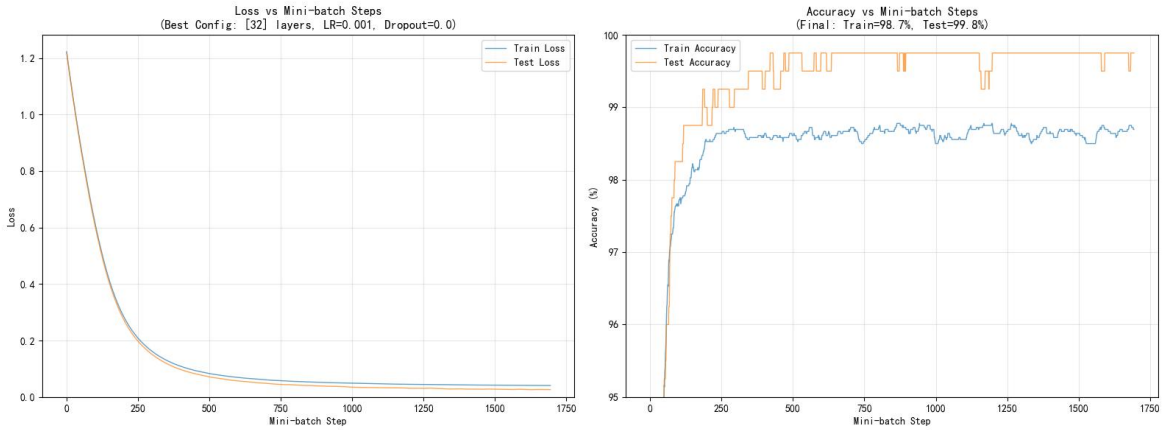

图 3-1 每一轮 mini-batch 训练后损失准确率变化曲线

从曲线上看，训练损失（蓝线）和测试损失（橙线）都在前几百个 mini-batch 内迅速下降，从 1.2 左右降到 0.05 以下。约在 step ≈ 500 之后，损失趋于平稳，后续下降幅度很小，说明模型已经收敛。训练损失和测试损失几乎重合，没有明显分叉，说明模型在训练集和测试集上的拟合程度一致，没有出现严重过拟合。测试准确率始终略高于训练准确率，这说明模型在训练过程中并未过拟合，甚至存在轻微的“正则化效果”。

测试准确率最终达到 99.8%，高于训练集的 98.7%，表明模型结构（单隐层 32 个神经元）对该二维高斯数据分类任务是足够的，且具有极好的泛化性能。

3.3 架构对比实验

实验过程中，我进一步比较了不同网络深度与宽度的效果，激活函数，其他参数，优化器则和基础实验中相同，结果如下：

表 3-3 架构对比结果

架构	测试准确率	损失	时间(s)	参数量
[32]（最终）	99.50%	0.0256	3.67	164
[64, 32, 16]	99.50%	0.0233	2.53	2356
[128, 64, 32, 16]	99.75%	0.0218	3.33	10804
[256, 128]	99.00%	0.0302	2.28	33412
[16]（最小）	99.50%	0.0336	2.13	84

从结果上看，浅层网络（[32] 或 [16]）在参数量极少的情况下已能达到 99.5%~99.75% 的准确率，复杂架构并未带来显著提升，证明该任务本身线性可分性较强。

3.4 超参数对比实验

对比不同参数的影响，控制单一变量，其他参数和架构则与基础实验相同。

表 3-4 激活函数对比结果

激活函数	测试准确率	损失范围	训练时间
ReLU	99.50%	~0.02	2.67s
LeakyReLU	99.25%	~0.02	2.60s
Sigmoid	99.75%	~0.03	2.60s
Tanh	98.75%	~0.04	2.76s

表 3-5 学习率与批次对比结果

策略	lr	batch	epoch	测试准确率	时间
保守	1e-4	16	20	99.25%	6.77s
当前	1e-3	32	15	98.25%	2.51s
激进	1e-2	64	10	99.50%	0.97s
平衡	5e-3	32	25	99.50%	4.34s

表 3-6 优化器对比结果

优化器	测试准确率
Adam	99.25%
SGD	99.25%
AdamW	99.00%
RMSprop	99.00%

总结:激活函数方面，ReLU 及其变体更稳定，Sigmoid 偶尔表现最佳但损失较高。

超参数方面，激进配置（lr=0.01, batch=64, epochs=10）效率最高，适合快速

实验；若追求稳健，保守/平衡配置更合适。

3.5 正则化实验

对 Dropout 和 BatchNorm 进行测试：

表 3-7 正则化对比结果

方法	测试准确率	测试损失	训练准确率	过拟合差值
无正则化	99.50%	0.0256	98.53%	-0.97%
Dropout(0.3)	99.50%	0.0127	98.53%	-0.97%
BatchNorm	99.25%	0.0305	98.61%	-0.64%
Dropout+BN	99.00%	0.0330	98.36%	-0.64%

结论：

由于网络简单，正则化作用有限。但 Dropout 能显著降低测试损失，说明其对抑制微小噪声拟合仍有帮助；BatchNorm 在低维、小 batch 下反而不稳定。

优化器选择影响不大，Adam 与 SGD 表现接近。

4 结论与可能改进

本实验表明，在二维高斯数据的四分类任务中，模型复杂度需求较低，单隐藏层 32 个神经元的前馈神经网络即可达到 99.75% 的测试准确率。相比深层或更宽的结构，浅层模型不仅参数更少、推理效率更高，而且在稳定性和可解释性方面更具优势。

激活函数的比较显示 ReLU 与 Sigmoid 更适合此类任务，而 Tanh 与 LeakyReLU 并未带来额外好处；优化器方面，Adam 依旧是稳健的默认选择，而 SGD 在需要更简单依赖或可控收敛路径时也可作为备选。

学习率策略对训练效率的影响较大，例如使用较激进的配置（如 lr=0.01, bs=64, 10 epochs）能在几乎不牺牲精度的情况下显著缩短训练时间。值得注意的是，Dropout 在已高精度下仍能微幅降低测试损失，可作为稳定训练的补充手段。整体而言，训练与测试曲线高度一致，未出现典型的过拟合现象，泛化性能良好，测试集优于训练集的差异主要来自统计扰动与小样本特性。

未来的改进方向主要集中在结果呈现和稳健性验证上。

例如，可以通过绘制决策边界与混淆矩阵更直观地展示模型表现，并通过多次随机种子重复实验给出均值和方差来增强统计可靠性。同时，结合早停与学习率调度（如 CosineAnnealingLR 或 ReduceLROnPlateau）可使训练过程更优雅且高效。

在模型层面，可以考虑使用 L2 权重衰减或 MC Dropout 来探索不确定性与正则化效果，并通过对困难样本的分析提升模型解释力。若面向资源受限的部署，还可尝试蒸馏、量化或缩小网络结构以减小开销。进一步的数据增强和可视化分析（如对隐层特征做 PCA/UMAP 降维展示）也有助于揭示模型的决策机制。

总体来看，本实验的结果验证了浅层前馈神经网络在低维分类任务中的高效性，同时为更复杂任务的拓展与部署提供了改进思路。