多层感知机及其在鸢尾花分类中的应用

基于 SimpleNN 框架的实验报告

实验时间: 2025 年 4 月 10 日

1. 实验目的

本实验旨在通过构建多层感知机(Multi-Layer Perceptron, MLP)模型来解决鸢尾花数据集的分类问题，加深对神经网络基本原理的理解。具体目标包括：

1. 掌握多层感知机的基本结构和工作原理
2. 理解神经网络的前向传播和反向传播算法
3. 学习使用 SimpleNN 框架构建并训练神经网络模型
4. 通过可视化分析模型的训练过程和性能表现
5. 培养解决实际分类问题的能力

2. 实验原理

2.1. 多层感知机(MLP)简介

多层感知机是一种前馈神经网络，由输入层、一个或多个隐藏层和输出层组成。每一层由若干个神经元构成，相邻层之间的神经元全连接。MLP 的基本结构如图 1 所示：

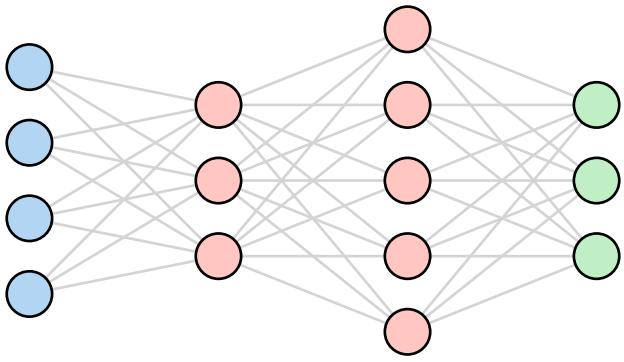

Figure 1: 多层感知机的基本结构

2.2. 神经网络的数学表达

2.2.1. 前向传播

前向传播是信息从输入层经过隐藏层到输出层的过程。对于第l层的神经元，其输出可表示为：

$$z^l = W^l a^{l-1} + b^l$$

$$a^l = g(z^l)$$

其中：

- a^l: 第l层的激活值
- W^l: 连接第$l-1$层和第l层的权重矩阵
- b^l: 第l层的偏置向量
- $g(c\cdot)$: 激活函数（如 ReLU、Sigmoid、Tanh 等）

2.2.2. 反向传播

反向传播是计算损失函数对网络参数梯度的过程，用于权重更新。对于输出层，误差项为：

$$\delta^L = \nabla_a C \odot g'(z^L)$$

其中C为损失函数，g'为激活函数的导数，$\odot$为 Hadamard 积。

对于隐藏层l，误差项为：

$$\delta^l = \left((W^{l+1})^T \delta^{l+1}\right) \odot g'(z^l)$$

然后，对于每一层的参数，梯度为：

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T$$

$$\nabla_{b^l} C = \delta^l$$

2.2.3. 参数更新

使用梯度下降法更新参数：

$$W^l := W^l - \eta \nabla_{W^l} C$$

$$b^l := b^l - \eta \nabla_{b^l} C$$

其中η是学习率。

2.3. 鸢尾花数据集

鸢尾花数据集是一个经典的多分类数据集，包含三种鸢尾花（Setosa、Versicolor 和 Virginica）的 150 个样本，每个样本有 4 个特征：花萼长度、花萼宽度、花瓣长度和花瓣宽度。

Figure 2: 鸢尾花数据集的特征分布可视化

3. 代码实现

3.1. SimpleNN 框架简介

SimpleNN 是一个基于 NumPy 实现的轻量级神经网络框架，采用静态计算图设计。该框架封装了前向传播、反向传播、参数更新等核心功能，通过简洁的 API 使用户能够方便地构建和训练神经网络模型。

3.1.1. SimpleNN 框架的核心组件

SimpleNN 框架由以下几个核心模块组成，它们共同构成了一个完整的神经网络训练系统：

1. 参数管理（**Parameter**）：
   - 封装网络中的可学习参数（权重和偏置）
   - 管理参数的数据（data）和梯度（grad）
   - 支持梯度清零等操作

2. 层（**Layer**）：
   - 基础 Layer 类定义了所有层的通用接口
   - Dense：全连接层，实现 $y = xW + b$ 的变换
   - BatchNorm：批归一化层，提高训练稳定性和收敛速度
   - 各种激活函数层：ReLU、Sigmoid、Tanh、Softmax 等
   - Dropout：随机失活层，防止过拟合
3. 损失函数（**Loss**）：
   - MSE：均方误差，用于回归问题
   - SoftmaxCrossEntropy：Softmax 交叉熵损失，用于多分类问题
   - BinaryCrossEntropy：二元交叉熵，用于二分类问题
   - L1Loss、HuberLoss 等其他损失函数
4. 优化器（**Optimizer**）：
   - SGD：随机梯度下降
   - MomentumSGD：带动量的随机梯度下降
   - Adam：自适应矩估计算法，结合动量法和 RMSProp 的优势
   - RMSProp：均方根传播算法
5. 学习率调度器（**Scheduler**）：
   - LinearDecayScheduler：线性衰减学习率
6. 评估指标（**Metric**）：
   - Accuracy：准确率
   - F1Score：F1 分数
7. 模型（**Model**）：
   - 组合以上组件，提供端到端训练接口
   - 支持模型编译、训练、评估和预测
   - 提供模型结构可视化和训练过程记录

在本实验中，我们使用这些组件构建了一个能够对鸢尾花进行分类的多层感知机模型。

3.2. 数据预处理

数据预处理代码

```
def load_iris_dataset():
    """加载鸢尾花数据集"""
    # 加载数据集
    iris = load_iris()
    X = iris.data
    y = iris.target
    class_names = iris.target_names

    # 数据标准化
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # 划分训练集和验证集
    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    return X_train, y_train, X_val, y_val, class_names
```

主要的预处理步骤包括：
1. 加载鸢尾花数据集
2. 对特征进行标准化处理
3. 按照 8:2 的比例划分训练集和验证集

3.3. 模型构建

模型构建代码

```
# 构建MLP模型
model = snn.Model(
    layers=[
        snn.Dense(4, 16),  # 输入层到隐藏层，4个输入特征，16个隐藏单元
        snn.ReLU(),        # ReLU激活函数
        snn.Dense(16, 3),  # 隐藏层到输出层，3个输出类别
        snn.Tanh()         # Tanh激活函数
    ]
)

# 编译模型
model.compile(
    loss=snn.SoftmaxCrossEntropy(),  # 多分类使用Softmax交叉熵损失
    optimizer=snn.Adam(lr=0.01),     # Adam优化器，初始学习率0.01
    scheduler=snn.LinearDecayScheduler(final_lr=0.001, total_steps=300), # 学习率调度器
    metrics=[snn.Accuracy()]         # 使用准确率作为评估指标
)
```

模型结构：
- 输入层：4 个神经元（对应 4 个特征）
- 隐藏层：16 个神经元，使用 ReLU 激活函数
- 输出层：3 个神经元（对应 3 个类别），使用 Tanh 激活函数
- 损失函数：SoftmaxCrossEntropy

3.3.1. 损失函数原理

在本实验中，我们使用 Softmax 交叉熵损失函数，这是多分类问题中的标准选择。其数学原理如下：

1. **Softmax 函数**：将神经网络输出转换为概率分布

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

其中，z_i是神经网络对第i类的原始输出，K是类别总数。Softmax 确保所有输出的概率之和为 1。

2. 交叉熵损失：衡量预测概率分布与真实分布之间的差异

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$$

其中，y_i是真实标签（one-hot 编码），$\hat{y}_i$是模型预测的概率（Softmax 输出）。

对于单个样本的梯度计算：

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i$$

这种简洁的梯度形式是 Softmax 交叉熵在多分类问题中广泛应用的原因之一。

3.3.2. 优化器原理

本实验采用 Adam（Adaptive Moment Estimation）优化器，它结合了动量法（Momentum）和 RMSProp 的优势。其数学原理如下：

1. 参数更新规则：

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

其中：

- g_t：当前梯度
- m_t：一阶矩（梯度的移动平均）
- v_t：二阶矩（梯度平方的移动平均）
- β_1, β_2：衰减率（通常为 0.9 和 0.999）
- η：学习率
- ε：数值稳定性常数（避免除零）

2. **Adam** 的优势：

- 结合了动量法的惯性，可以突破局部极小值
- 自适应调整各参数的学习率，适应不同尺度的特征
- 通过偏差修正（$\hat{m}_t$和$\hat{v}_t$）提高初始训练阶段的稳定性

3. 学习率调度：本实验还采用了线性衰减的学习率调度策略

$$\eta_t = \eta_0 + (\eta_f - \eta_0) \times \frac{t}{T}$$

其中，η_0是初始学习率（0.01），η_f是最终学习率（0.001），t是当前步数，T是总步数（300）。

学习率从初始值逐渐线性衰减到最终值，这有助于在训练初期快速收敛，后期精细调整以获得更高精度。

3.4. 模型训练

模型训练代码

```
# 训练模型
history = model.fit(
    x=X_train,
    y=y_train,
    batch_size=64,
    epochs=100,
    validation_data=(X_val, y_val),
    shuffle=True,
    verbose=1
)
```

训练参数：
- 批次大小：64
- 训练轮数：100
- 每轮随机打乱数据

3.5. 模型评估和可视化

模型评估和可视化代码

```python
# 评估模型
eval_results = model.evaluate(X_val, y_val)
print(f"验证损失: {eval_results['loss']:.4f}")
print(f"验证准确率: {eval_results['accuracy']:.4f}")

# 预测
predictions = model.predict(X_val)
predictions_class = np.argmax(predictions, axis=1)

# 计算准确率
accuracy = np.mean(predictions_class == y_val)
print(f"验证准确率: {accuracy:.4f}")

# 绘制混淆矩阵
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_val, predictions_class)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.savefig('confusion_matrix.png')
```

4. 实验结果

4.1. 训练过程

下图展示了模型训练过程中损失函数和准确率的变化：

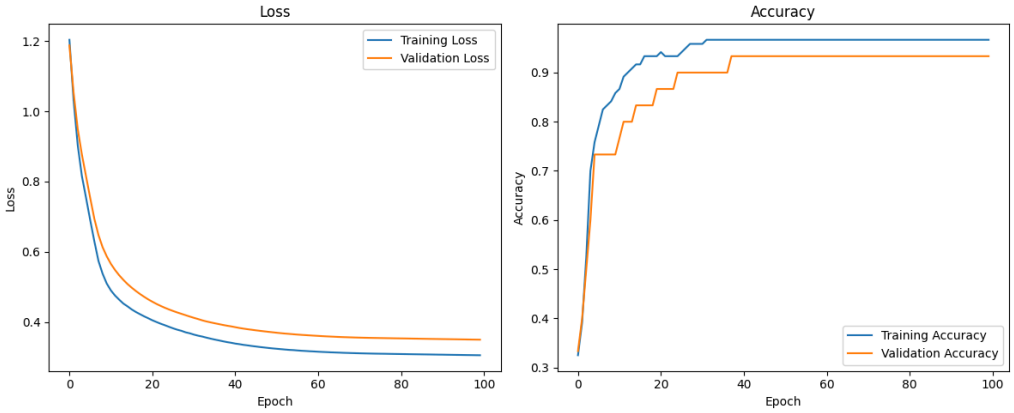

Figure 3: 模型训练过程中的损失和准确率变化

从图中可以观察到：
1. 训练损失和验证损失都随着训练轮数增加而降低，最终趋于稳定
2. 训练准确率和验证准确率都随着训练轮数增加而提高，最终达到较高水平
3. 训练集和验证集表现接近，说明模型没有明显过拟合

4.2. 分类性能

模型在验证集上的最终性能评估如下：

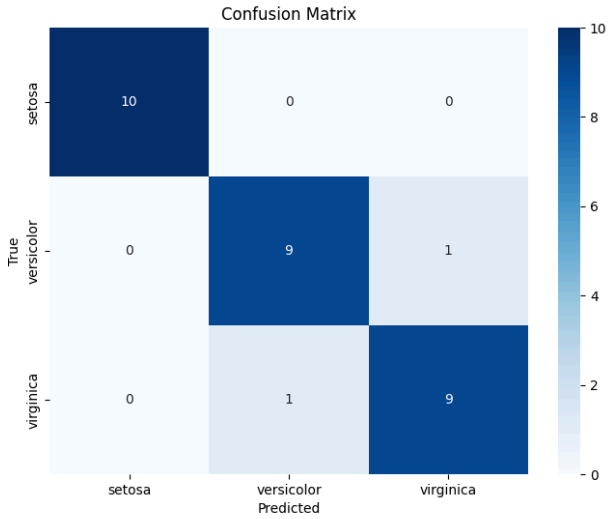

Figure 4: 混淆矩阵

混淆矩阵显示了模型对各类别的预测情况：

- 对角线元素代表正确分类的样本数
- 非对角线元素代表错误分类的样本数

模型在验证集上达到了较高的准确率，对三种鸢尾花都有很好的识别能力。特别是对 Setosa 类的识别准确率达到 100%，而对 Versicolor 和 Virginica 的分类也取得了良好的效果。

5. 心得体会

通过本次多层感知机实验，我对神经网络的工作原理有了更深入的理解：

1. 理论与实践结合：亲自实现和训练 MLP 模型，使我对前向传播、反向传播等理论知识有了具象的理解。SimpleNN 框架的设计清晰，有助于理解神经网络的内部工作机制。
2. 参数调优的重要性：在实验过程中，我发现网络结构、学习率、批次大小等超参数的选择对模型性能有显著影响。合理的参数配置可以加速训练过程并提高模型性能。
3. 数据预处理的必要性：数据标准化等预处理步骤对模型训练至关重要，它可以加速收敛并提高模型稳定性。
4. 可视化分析的价值：通过损失曲线、准确率曲线和混淆矩阵等可视化工具，可以直观地评估模型性能并及时发现问题。
5. 框架设计的思考：SimpleNN 框架采用模块化设计，将复杂的神经网络训练过程分解为清晰的组件，这种设计思想值得在今后的工程实践中借鉴。

通过这次实验，我不仅掌握了 MLP 的基本原理和应用，更重要的是体会到了深度学习“理论指导实践，实践深化理论”的学习方法。在今后的学习中，我将继续深入研究更复杂的神经网络结构和更高级的训练技巧。

6. Appendix

6.1. Train Log:

```
Model Summary:
================================================================================
Layer(input_dim, output_dim)  Params              Trainable Params
--------------------------------------------------------------------------------
0: Dense(4, 16)               80                  80
1: ReLU                       0                   0
2: Dense(16, 3)               51                  51
3: Tanh                       0                   0
--------------------------------------------------------------------------------
Total params: 131
Trainable params: 131
Non-trainable params: 0
================================================================================
Epoch 1/100 - 0.00s - loss: 1.2036 - accuracy: 0.3250 - val_loss: 1.1877 -
val_accuracy: 0.3333
Epoch 2/100 - 0.00s - loss: 1.0320 - accuracy: 0.3917 - val_loss: 1.0540 -
val_accuracy: 0.4000
Epoch 3/100 - 0.00s - loss: 0.9038 - accuracy: 0.5250 - val_loss: 0.9500 -
val_accuracy: 0.5000
Epoch 4/100 - 0.00s - loss: 0.8162 - accuracy: 0.7000 - val_loss: 0.8775 -
val_accuracy: 0.6000
Epoch 5/100 - 0.00s - loss: 0.7543 - accuracy: 0.7583 - val_loss: 0.8157 -
val_accuracy: 0.7333
Epoch 6/100 - 0.00s - loss: 0.6911 - accuracy: 0.7917 - val_loss: 0.7539 -
val_accuracy: 0.7333
Epoch 7/100 - 0.00s - loss: 0.6307 - accuracy: 0.8250 - val_loss: 0.6948 -
val_accuracy: 0.7333
Epoch 8/100 - 0.00s - loss: 0.5743 - accuracy: 0.8333 - val_loss: 0.6480 -
val_accuracy: 0.7333
Epoch 9/100 - 0.00s - loss: 0.5379 - accuracy: 0.8417 - val_loss: 0.6133 -
val_accuracy: 0.7333
Epoch 10/100 - 0.00s - loss: 0.5097 - accuracy: 0.8583 - val_loss: 0.5872 -
val_accuracy: 0.7333
Epoch 11/100 - 0.00s - loss: 0.4903 - accuracy: 0.8667 - val_loss: 0.5658 -
val_accuracy: 0.7667
Epoch 12/100 - 0.00s - loss: 0.4756 - accuracy: 0.8917 - val_loss: 0.5484 -
val_accuracy: 0.8000
Epoch 13/100 - 0.00s - loss: 0.4638 - accuracy: 0.9000 - val_loss: 0.5334 -
val_accuracy: 0.8000
Epoch 14/100 - 0.00s - loss: 0.4529 - accuracy: 0.9083 - val_loss: 0.5203 -
val_accuracy: 0.8000
Epoch 15/100 - 0.00s - loss: 0.4450 - accuracy: 0.9167 - val_loss: 0.5084 -
val_accuracy: 0.8333
Epoch 16/100 - 0.00s - loss: 0.4363 - accuracy: 0.9167 - val_loss: 0.4983 -
val_accuracy: 0.8333
Epoch 17/100 - 0.00s - loss: 0.4290 - accuracy: 0.9333 - val_loss: 0.4888 -
val_accuracy: 0.8333
Epoch 18/100 - 0.00s - loss: 0.4227 - accuracy: 0.9333 - val_loss: 0.4802 -
val_accuracy: 0.8333
Epoch 19/100 - 0.00s - loss: 0.4166 - accuracy: 0.9333 - val_loss: 0.4722 -
val_accuracy: 0.8333
Epoch 20/100 - 0.00s - loss: 0.4111 - accuracy: 0.9333 - val_loss: 0.4647 -
val_accuracy: 0.8667
Epoch 21/100 - 0.00s - loss: 0.4052 - accuracy: 0.9417 - val_loss: 0.4581 -
val_accuracy: 0.8667
Epoch 22/100 - 0.00s - loss: 0.4003 - accuracy: 0.9333 - val_loss: 0.4518 -
val_accuracy: 0.8667
```

```
Epoch 23/100 - 0.00s - loss: 0.3955 - accuracy: 0.9333 - val_loss: 0.4461 -
val_accuracy: 0.8667
Epoch 24/100 - 0.00s - loss: 0.3913 - accuracy: 0.9333 - val_loss: 0.4409 -
val_accuracy: 0.8667
Epoch 25/100 - 0.00s - loss: 0.3867 - accuracy: 0.9333 - val_loss: 0.4361 -
val_accuracy: 0.9000
Epoch 26/100 - 0.00s - loss: 0.3823 - accuracy: 0.9417 - val_loss: 0.4316 -
val_accuracy: 0.9000
Epoch 27/100 - 0.00s - loss: 0.3784 - accuracy: 0.9500 - val_loss: 0.4275 -
val_accuracy: 0.9000
Epoch 28/100 - 0.00s - loss: 0.3753 - accuracy: 0.9583 - val_loss: 0.4237 -
val_accuracy: 0.9000
Epoch 29/100 - 0.00s - loss: 0.3711 - accuracy: 0.9583 - val_loss: 0.4198 -
val_accuracy: 0.9000
Epoch 30/100 - 0.00s - loss: 0.3683 - accuracy: 0.9583 - val_loss: 0.4159 -
val_accuracy: 0.9000
Epoch 31/100 - 0.00s - loss: 0.3646 - accuracy: 0.9583 - val_loss: 0.4123 -
val_accuracy: 0.9000
Epoch 32/100 - 0.00s - loss: 0.3615 - accuracy: 0.9667 - val_loss: 0.4089 -
val_accuracy: 0.9000
Epoch 33/100 - 0.00s - loss: 0.3590 - accuracy: 0.9667 - val_loss: 0.4052 -
val_accuracy: 0.9000
Epoch 34/100 - 0.00s - loss: 0.3560 - accuracy: 0.9667 - val_loss: 0.4020 -
val_accuracy: 0.9000
Epoch 35/100 - 0.00s - loss: 0.3531 - accuracy: 0.9667 - val_loss: 0.3994 -
val_accuracy: 0.9000
Epoch 36/100 - 0.00s - loss: 0.3505 - accuracy: 0.9667 - val_loss: 0.3968 -
val_accuracy: 0.9000
Epoch 37/100 - 0.00s - loss: 0.3481 - accuracy: 0.9667 - val_loss: 0.3944 -
val_accuracy: 0.9000
Epoch 38/100 - 0.00s - loss: 0.3455 - accuracy: 0.9667 - val_loss: 0.3920 -
val_accuracy: 0.9333
Epoch 39/100 - 0.00s - loss: 0.3432 - accuracy: 0.9667 - val_loss: 0.3898 -
val_accuracy: 0.9333
Epoch 40/100 - 0.00s - loss: 0.3412 - accuracy: 0.9667 - val_loss: 0.3879 -
val_accuracy: 0.9333
Epoch 41/100 - 0.00s - loss: 0.3389 - accuracy: 0.9667 - val_loss: 0.3856 -
val_accuracy: 0.9333
Epoch 42/100 - 0.00s - loss: 0.3373 - accuracy: 0.9667 - val_loss: 0.3834 -
val_accuracy: 0.9333
Epoch 43/100 - 0.00s - loss: 0.3353 - accuracy: 0.9667 - val_loss: 0.3814 -
val_accuracy: 0.9333
Epoch 44/100 - 0.00s - loss: 0.3337 - accuracy: 0.9667 - val_loss: 0.3797 -
val_accuracy: 0.9333
Epoch 45/100 - 0.00s - loss: 0.3320 - accuracy: 0.9667 - val_loss: 0.3782 -
val_accuracy: 0.9333
Epoch 46/100 - 0.00s - loss: 0.3306 - accuracy: 0.9667 - val_loss: 0.3766 -
val_accuracy: 0.9333
Epoch 47/100 - 0.00s - loss: 0.3291 - accuracy: 0.9667 - val_loss: 0.3752 -
val_accuracy: 0.9333
Epoch 48/100 - 0.00s - loss: 0.3278 - accuracy: 0.9667 - val_loss: 0.3737 -
val_accuracy: 0.9333
Epoch 49/100 - 0.00s - loss: 0.3264 - accuracy: 0.9667 - val_loss: 0.3724 -
val_accuracy: 0.9333
Epoch 50/100 - 0.00s - loss: 0.3252 - accuracy: 0.9667 - val_loss: 0.3711 -
val_accuracy: 0.9333
Epoch 51/100 - 0.00s - loss: 0.3241 - accuracy: 0.9667 - val_loss: 0.3698 -
```

```
val_accuracy: 0.9333
Epoch 52/100 - 0.00s - loss: 0.3231 - accuracy: 0.9667 - val_loss: 0.3685 -
val_accuracy: 0.9333
Epoch 53/100 - 0.00s - loss: 0.3220 - accuracy: 0.9667 - val_loss: 0.3674 -
val_accuracy: 0.9333
Epoch 54/100 - 0.00s - loss: 0.3210 - accuracy: 0.9667 - val_loss: 0.3665 -
val_accuracy: 0.9333
Epoch 55/100 - 0.00s - loss: 0.3204 - accuracy: 0.9667 - val_loss: 0.3655 -
val_accuracy: 0.9333
Epoch 56/100 - 0.00s - loss: 0.3194 - accuracy: 0.9667 - val_loss: 0.3646 -
val_accuracy: 0.9333
Epoch 57/100 - 0.00s - loss: 0.3185 - accuracy: 0.9667 - val_loss: 0.3637 -
val_accuracy: 0.9333
Epoch 58/100 - 0.00s - loss: 0.3178 - accuracy: 0.9667 - val_loss: 0.3630 -
val_accuracy: 0.9333
Epoch 59/100 - 0.00s - loss: 0.3171 - accuracy: 0.9667 - val_loss: 0.3622 -
val_accuracy: 0.9333
Epoch 60/100 - 0.00s - loss: 0.3165 - accuracy: 0.9667 - val_loss: 0.3615 -
val_accuracy: 0.9333
Epoch 61/100 - 0.00s - loss: 0.3158 - accuracy: 0.9667 - val_loss: 0.3608 -
val_accuracy: 0.9333
Epoch 62/100 - 0.00s - loss: 0.3152 - accuracy: 0.9667 - val_loss: 0.3601 -
val_accuracy: 0.9333
Epoch 63/100 - 0.00s - loss: 0.3147 - accuracy: 0.9667 - val_loss: 0.3595 -
val_accuracy: 0.9333
Epoch 64/100 - 0.00s - loss: 0.3141 - accuracy: 0.9667 - val_loss: 0.3589 -
val_accuracy: 0.9333
Epoch 65/100 - 0.00s - loss: 0.3136 - accuracy: 0.9667 - val_loss: 0.3583 -
val_accuracy: 0.9333
Epoch 66/100 - 0.00s - loss: 0.3132 - accuracy: 0.9667 - val_loss: 0.3578 -
val_accuracy: 0.9333
Epoch 67/100 - 0.00s - loss: 0.3127 - accuracy: 0.9667 - val_loss: 0.3574 -
val_accuracy: 0.9333
Epoch 68/100 - 0.00s - loss: 0.3123 - accuracy: 0.9667 - val_loss: 0.3570 -
val_accuracy: 0.9333
Epoch 69/100 - 0.00s - loss: 0.3119 - accuracy: 0.9667 - val_loss: 0.3566 -
val_accuracy: 0.9333
Epoch 70/100 - 0.00s - loss: 0.3116 - accuracy: 0.9667 - val_loss: 0.3562 -
val_accuracy: 0.9333
Epoch 71/100 - 0.00s - loss: 0.3113 - accuracy: 0.9667 - val_loss: 0.3559 -
val_accuracy: 0.9333
Epoch 72/100 - 0.00s - loss: 0.3110 - accuracy: 0.9667 - val_loss: 0.3556 -
val_accuracy: 0.9333
Epoch 73/100 - 0.00s - loss: 0.3107 - accuracy: 0.9667 - val_loss: 0.3553 -
val_accuracy: 0.9333
Epoch 74/100 - 0.00s - loss: 0.3105 - accuracy: 0.9667 - val_loss: 0.3551 -
val_accuracy: 0.9333
Epoch 75/100 - 0.00s - loss: 0.3102 - accuracy: 0.9667 - val_loss: 0.3549 -
val_accuracy: 0.9333
Epoch 76/100 - 0.00s - loss: 0.3101 - accuracy: 0.9667 - val_loss: 0.3547 -
val_accuracy: 0.9333
Epoch 77/100 - 0.00s - loss: 0.3099 - accuracy: 0.9667 - val_loss: 0.3545 -
val_accuracy: 0.9333
Epoch 78/100 - 0.00s - loss: 0.3097 - accuracy: 0.9667 - val_loss: 0.3542 -
val_accuracy: 0.9333
Epoch 79/100 - 0.00s - loss: 0.3095 - accuracy: 0.9667 - val_loss: 0.3540 -
val_accuracy: 0.9333
```

```
Epoch 80/100 - 0.00s - loss: 0.3093 - accuracy: 0.9667 - val_loss: 0.3538 -
val_accuracy: 0.9333
Epoch 81/100 - 0.00s - loss: 0.3091 - accuracy: 0.9667 - val_loss: 0.3537 -
val_accuracy: 0.9333
Epoch 82/100 - 0.00s - loss: 0.3089 - accuracy: 0.9667 - val_loss: 0.3535 -
val_accuracy: 0.9333
Epoch 83/100 - 0.00s - loss: 0.3087 - accuracy: 0.9667 - val_loss: 0.3533 -
val_accuracy: 0.9333
Epoch 84/100 - 0.00s - loss: 0.3086 - accuracy: 0.9667 - val_loss: 0.3531 -
val_accuracy: 0.9333
Epoch 85/100 - 0.00s - loss: 0.3084 - accuracy: 0.9667 - val_loss: 0.3529 -
val_accuracy: 0.9333
Epoch 86/100 - 0.00s - loss: 0.3082 - accuracy: 0.9667 - val_loss: 0.3527 -
val_accuracy: 0.9333
Epoch 87/100 - 0.00s - loss: 0.3080 - accuracy: 0.9667 - val_loss: 0.3525 -
val_accuracy: 0.9333
Epoch 88/100 - 0.00s - loss: 0.3079 - accuracy: 0.9667 - val_loss: 0.3522 -
val_accuracy: 0.9333
Epoch 89/100 - 0.00s - loss: 0.3077 - accuracy: 0.9667 - val_loss: 0.3521 -
val_accuracy: 0.9333
Epoch 90/100 - 0.00s - loss: 0.3075 - accuracy: 0.9667 - val_loss: 0.3519 -
val_accuracy: 0.9333
Epoch 91/100 - 0.00s - loss: 0.3074 - accuracy: 0.9667 - val_loss: 0.3517 -
val_accuracy: 0.9333
Epoch 92/100 - 0.00s - loss: 0.3071 - accuracy: 0.9667 - val_loss: 0.3516 -
val_accuracy: 0.9333
Epoch 93/100 - 0.00s - loss: 0.3070 - accuracy: 0.9667 - val_loss: 0.3514 -
val_accuracy: 0.9333
Epoch 94/100 - 0.00s - loss: 0.3068 - accuracy: 0.9667 - val_loss: 0.3512 -
val_accuracy: 0.9333
Epoch 95/100 - 0.00s - loss: 0.3066 - accuracy: 0.9667 - val_loss: 0.3510 -
val_accuracy: 0.9333
Epoch 96/100 - 0.00s - loss: 0.3065 - accuracy: 0.9667 - val_loss: 0.3508 -
val_accuracy: 0.9333
Epoch 97/100 - 0.00s - loss: 0.3063 - accuracy: 0.9667 - val_loss: 0.3506 -
val_accuracy: 0.9333
Epoch 98/100 - 0.00s - loss: 0.3061 - accuracy: 0.9667 - val_loss: 0.3504 -
val_accuracy: 0.9333
Epoch 99/100 - 0.00s - loss: 0.3059 - accuracy: 0.9667 - val_loss: 0.3502 -
val_accuracy: 0.9333
Epoch 100/100 - 0.00s - loss: 0.3058 - accuracy: 0.9667 - val_loss: 0.3501 -
val_accuracy: 0.9333
验证损失: 0.3501
验证准确率: 0.9333
验证准确率: 0.9333

分类报告:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```