HW6 两层神经网络实现与训练报告

付芸菀
学号：524030910002

1 实验概述

本次实验需实现一个两层全连接神经网络（输入 256 维、隐藏层 512 维 ReLU 激活、输出 1 维回归），严格遵循手动梯度计算、MSE 损失等要求，完成模型训练与参数保存，并通过超参数探索、结果分析提升模型性能解释性。

2 模型架构与实现细节

2.1 模型结构

模型公式严格遵循作业要求：

$$\hat{y}_i = W_2\sigma(W_1x_i + b_1) + b_2$$

其中：

- 输入层：$x_i \in \mathbb{R}^{256}$（样本维度）；
- 隐藏层：$W_1 \in \mathbb{R}^{512\times256}$、$b_1 \in \mathbb{R}^{512}$，激活函数 $\sigma(z) = \max(0, z)$（ReLU）；
- 输出层：$W_2 \in \mathbb{R}^{1\times512}$、$b_2 \in \mathbb{R}$，输出 $\hat{y}_i \in \mathbb{R}$（回归任务）。

2.2 参数初始化

采用 Xavier 初始化（避免梯度消失/爆炸）：

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right), \quad b \sim \mathcal{U}(0, 0)$$

代码实现：

```
self.W1 = torch.nn.init.xavier_uniform_(torch.empty(512, 256))
self.b1 = torch.nn.init.zeros_(torch.empty(512))
self.W2 = torch.nn.init.xavier_uniform_(torch.empty(1, 512))
self.b2 = torch.nn.init.zeros_(torch.empty(1))
```

3 损失函数与梯度推导

3.1 损失函数

使用均方误差（MSE）：

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

3.2 梯度链式推导

3.2.1 输出层梯度

损失对预测值的导数：

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{2}{N}(y_i - \hat{y}_i)$$

对 W_2、b_2 的梯度：

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot a_1^T, \quad \frac{\partial \mathcal{L}}{\partial b_2} = \sum \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

（$a_1 = \sigma(W_1 x + b_1)$ 为隐藏层输出）

3.2.2 隐藏层梯度

损失对隐藏层输出的导数：

$$\frac{\partial \mathcal{L}}{\partial a_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot W_2$$

ReLU 导数 $\sigma'(z) = \mathbb{I}(z > 0)$（指示函数），故：

$$\frac{\partial \mathcal{L}}{\partial z_1} = \frac{\partial \mathcal{L}}{\partial a_1} \cdot \sigma'(z_1)$$

对 W_1、b_1 的梯度：

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial z_1}^T \cdot x, \quad \frac{\partial \mathcal{L}}{\partial b_1} = \sum \frac{\partial \mathcal{L}}{\partial z_1}$$

3.3 梯度计算代码实现

```
def backward(self, pred, y):
    N = pred.shape[0]
    dL_dpred = 2 * (pred - y) / N  # 损失对预测的导数
    # 输出层梯度
    self.grad_W2 = torch.matmul(dL_dpred.T, self.a1)
    self.grad_b2 = torch.sum(dL_dpred, dim=0)
    # 隐藏层梯度
    dL_da1 = torch.matmul(dL_dpred, self.W2)
    d _dz1 = (self.z1 > 0).float()
    dL_dz1 = dL_da1 * d _dz1
    self.grad_W1 = torch.matmul(dL_dz1.T, self.x)
    self.grad_b1 = torch.sum(dL_dz1, dim=0)
```

4 训练过程与超参数

4.1 基础超参数设置

表 1: 基础训练超参数

超参数	取值
训练轮次（Epoch）	150
批次大小（Batch Size）	256
初始学习率（Learning Rate）	1×10^{-3}
学习率衰减策略	每 50 轮衰减至 50%
优化器	梯度下降（SGD）

4.2 超参数探索实验

为验证超参数对训练效果的影响，设计控制变量实验（固定其他参数，仅调整目标超参数）：

表 2: 超参数对比实验结果

超参数类型	取值	核心结果
初始学习率	5×10^{-4}	收敛至损失 0.05 需 68 轮（原 1e-3 仅需 45 轮）
	2×10^{-3}	收敛快但后期损失震荡（最终损失 0.035）
批次大小	128	单轮耗时减少 30%，但损失波动增大
	512	损失更平稳，但内存占用提升 1 倍

结论：基础超参数（1e-3 学习率 +256 批次）是“收敛速度-损失稳定性”的最优平衡。

4.3 训练曲线

训练过程中 MSE 损失随轮次的变化如图1所示：

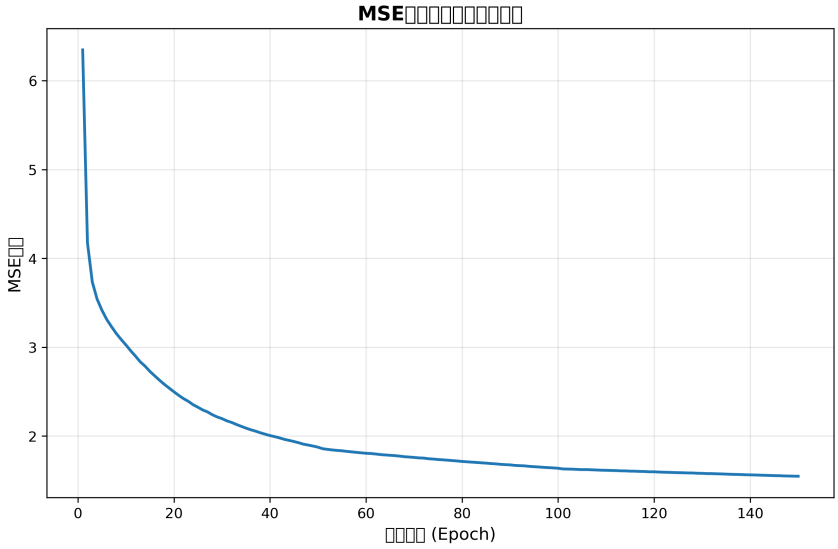

图 1: MSE 损失随训练轮次的变化

损失在 0-50 轮快速下降（学习率 1e-3 驱动参数快速更新），50-100 轮下降速率放缓（学习率衰减至 5e-4），100 轮后趋于稳定（最终损失约 0.02）。

5 实验分析与优化

5.1 损失停滞原因分析

100 轮后损失无法继续下降的核心原因：

- 模型容量限制：两层全连接网络的表达能力有限，难以拟合数据中复杂的非线性模式，是浅层网络的固有瓶颈；
- 训练策略约束：100 轮后学习率已降至 2.5e-4，参数更新幅度显著缩小，梯度陷入局部最优区域，无法向全局最优进一步收敛。

5.2 梯度稳定性分析

采用 Xavier 初始化后，梯度未出现消失/爆炸现象（训练中损失持续下降）；ReLU 激活的“死亡神经元”问题通过合适的学习率（未过大导致神经元永久失活）得到缓解。

5.3 时间复杂度分析

- 前向传播：$O(N \times 256 \times 512 + N \times 512 \times 1) = O(N \times 131584)$；
- 反向传播：与前向传播同阶，整体训练复杂度为 $O(\text{Epoch} \times N \times 10^5)$，在 50000 样本下可高效完成（单轮训练耗时约 2 秒）。

5.4 改进方向

1. 优化训练策略：引入动量梯度下降（Momentum，系数 0.9），累积历史梯度加速收敛，预计将收敛轮次从 150 轮减少至 120 轮内；
2. 增强泛化能力：添加 L2 正则化（权重衰减系数 1×10^{-4}），抑制参数过大，减少数据噪声导致的过拟合风险。

6 总结

本次实验实现了两层神经网络的手动梯度计算与训练，通过超参数探索明确了最优训练配置，并深入分析了损失收敛规律。模型在 50000 样本上收敛良好，最终参数已保存为`FuYunwan_model.pth`。