信息与软件工程学院上机实验报告

（第 1 次）

一、实验名称

Pytorch 环境配置和使用、线性模型的欠拟合与过拟合问题

二、实验目的及要求

1. 掌握 PyTorch 的基本操作。
2. 掌握线性模型，会用 PyTorch 平台实现线性回归模型。
3. 能够模拟出欠拟合和过拟合的情况。

三、实验环境

1. Python
2. Anaconda
3. PyTorch

四、实验内容

1. 练习 PyTorch 的基本操作，包含 PyTorch Tensor 的基本操作、自动求导操作。
2. 自主设计一个包含 5 个及以上特征的函数，并基于该函数生成模拟数据。
3. 自主实现线性模型、损失函数和评价函数。
4. 在数据中加入噪声维度，将输入扩展到 20 维以上，并模拟欠拟合和过拟合现象。

五、实验设计内容要求

1. 实验步骤

本实验围绕“在线性模型条件下观察欠拟合与过拟合现象”这一目标展开。为

了使实验现象足够明显，我先自行构造了一组带有非线性关系的模拟数据，再通过改变特征表达方式和训练样本数量，分别得到欠拟合、正常拟合和过拟合三种结果。这样做的好处是，实验过程完全可控，结论也更容易分析。

本实验设计的函数为:

$$y = 2.0x_1 - 1.5x_2^2 + 1.2x_3x_4 + 0.8\sin(\pi x_5) + 0.6x_1x_5 + 0.3$$

该函数一共使用了 5 个有效特征，其中既有线性项，也有平方项、乘积项和三角函数项，因此能够较好地体现非线性关系。在线性模型无法直接表示这类关系的前提下，模型的拟合效果就会明显受到特征设计方式的影响，这正适合用来研究欠拟合与过拟合问题。

1.1 设计思想

程序实现时，先随机生成 5 个有效特征，再根据上述函数计算标签值。为了使实验更接近真实情况，标签中加入了少量随机噪声。随后又额外引入 20 个纯噪声特征，将输入数据扩充到 25 维。这样一来，数据中既包含真正有效的信息，也包含干扰项，更有利于观察模型在不同条件下的表现。

模型部分没有直接调用高级封装，而是自行实现了线性模型、损失函数和评价函数。实验中仍然使用线性模型作为基础，但通过改变输入特征的构造方式来控制模型的有效表达能力：直接使用原始输入时容易欠拟合；按照真实函数结构构造特征时能够较好拟合；引入大量高阶特征并减少训练样本时则容易出现过拟合。

1.2 实现流程

程序的整体实现流程如图所示。

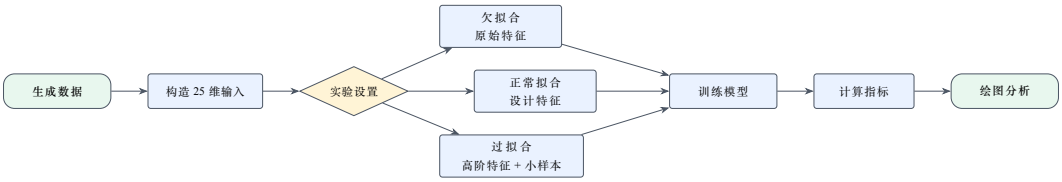


图 1: 实验流程图

1.3 主要实现过程

在具体实现中，首先利用 `torch.rand()` 生成 5 个有效特征，并将其映射到 `[-2, 2)` 区间，再根据设计函数计算输出 y，同时向标签中加入少量高斯噪声。之后使用 `torch.randn()` 额外生成 20 个噪声特征，并与前面的有效特征按列拼接，从而得到 25 维输入数据。

在模型训练阶段，程序先手动划分训练集和测试集，再分别构造三种不同的特征形式。欠拟合实验直接使用原始 25 维特征；正常拟合实验根据真实函数构造出更合理的特征；过拟合实验则构造大量平方项和交叉项，同时减少训练样本数。在线性模型部分，程序使用 $\hat{y} = Xw + b$ 作为预测形式，并通过 PyTorch 的自动求导功能实现梯度下降更新。评价时采用 RMSE 和 R^2 两个指标，并最终绘制对比图用于分析三种拟合状态之间的差异。

2. **调试过程及实验结果**

调试过程中，我主要关注两个问题：一是实验是否真的能够稳定地呈现三种不同的拟合状态，二是最终图像是否能够清楚说明这些差异。起初直接使用原始特征训练时，模型在训练集和测试集上的效果都较差。结合真实函数进行分析后可以发现，数据关系中包含平方项、乘积项和 sin 项，而原始线性输入无法表达这些非线性结构，因此欠拟合现象是必然的。

随后，在正常拟合实验中，我把真实函数中的主要结构重新构造成新的输入特征，例如 x_2^2、x_3x_4 和 $\sin(\pi x_5)$。这样虽然模型本身仍然是线性模型，但由于输入表达更加合理，实验结果得到了明显改善。为了进一步制造过拟合现象，我又增加了平方项和交叉项的数量，并把训练样本数压缩到较小规模，从而使模型容易在训练集上记住噪声，最终得到训练集指标很好而测试集明显变差的结果。

2.1 实验结果

本实验最终运行结果如下：

```
Code/NN/Lab01 [🐍 v3.13.12(Lab01)]
❯ uv run src/nn01.py
真实函数：
y = 2.0*x1 - 1.5*x2^2 + 1.2*x3*x4 + 0.8*sin(pi*x5) + 0.6*x1*x5 + 0.3
数据总维度：25 维 = 5 个有效特征 + 20 个噪声特征

1. 欠拟合
train RMSE: 2.3026, train R2: 0.5037
test  RMSE: 3.2053, test  R2: 0.2847

2. 正常拟合
train RMSE: 0.2056, train R2: 0.9960
test  RMSE: 0.1848, test  R2: 0.9976

3. 过拟合
train RMSE: 0.0000, train R2: 1.0000
test  RMSE: 3.2404, test  R2: 0.1216

图片已保存到：/l          /Code/NN/Lab01/src/fit_comparison.png
```

图 2: 程序运行结果截图

表 1: 三种拟合情况的实验结果

实验情况	train RMSE	train R2	test RMSE	test R2
欠拟合	2.3026	0.5037	3.2053	0.2847
正常拟合	0.2056	0.9960	0.1848	0.9976
过拟合	0.0000	1.0000	3.2404	0.1216

2.2 结果分析

从表中可以看出，欠拟合时训练集和测试集的 RMSE 都较大，同时 R^2 值偏低，说明模型本身表达能力不足，连训练数据中的主要规律都没有学到。正常拟合时，训练集和测试集的 RMSE 都很小，R^2 接近 1，表明模型已经较好地学习了数据中的真实关系，并且具有较好的泛化能力。过拟合时，训练集上的 RMSE 接近于 0，R^2 接近于 1，但测试集上的误差明显增大，这说明模型对训练数据拟合过度，同时也把噪声当成了有效规律。

为了更直观地说明这一点，程序还生成了训练集与测试集指标对比图。

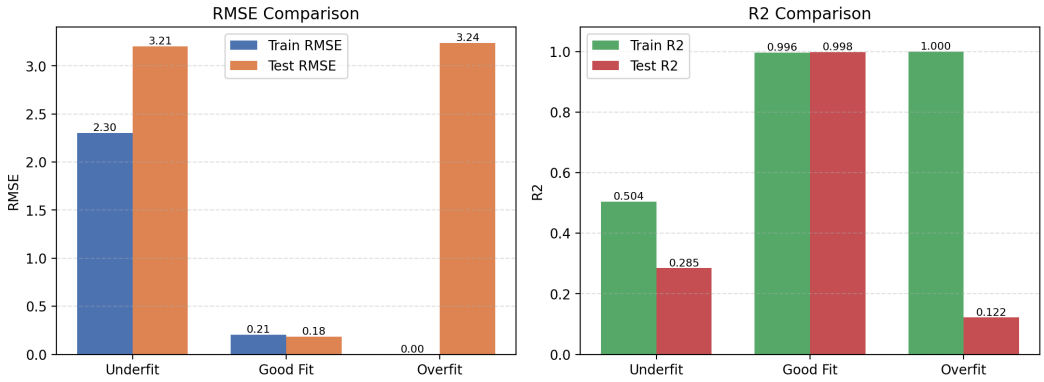


图 3: 欠拟合、正常拟合与过拟合的指标对比图

欠拟合的特点是训练集和测试集都表现较差；正常拟合的特点是训练集和测试集都表现较好；过拟合的特点则是训练集效果很好，但测试集明显变差。借助这种对比方式，可以更加准确地展示三种拟合状态之间的差异。

3. 总结

通过本次实验，我对 PyTorch 的基本用法、线性回归模型以及欠拟合与过拟合问题有了更加直观的认识。从实验结果来看，线性模型虽然形式简单，但在特征设计合理时仍然能够较好地拟合非线性关系；如果模型只能看到原始线性输入，就容易出现欠拟合。另一方面，当特征数量过多而训练样本过少时，模型虽然能在训练集上取得很好的结果，却会在测试集上明显退化，这正是典型的过拟合现象。由此可以看出，判断模型效果时不能只看训练集指标，还必须结合测试集结果进行综合分析。

通过人为加入 20 个噪声维度，我进一步观察到了高维噪声对模型泛化能力的影响。尤其是在过拟合实验中，复杂特征和小样本共同作用，使模型不仅学习了真实规律，也学习了随机噪声，最终导致测试结果下降。这次上机给我的一个直接体会是，课堂上关于欠拟合和过拟合的概念只有在自己动手实现之后才会真正清晰。通过亲自构造数据、训练模型并观察指标变化，我对特征工程、样本数量和模型复杂度之间的关系有了更深的理解。

如果继续改进本实验，可以从三个方面入手。第一，可以增加验证集，用来进一步观察模型在不同训练轮数下的变化；第二，可以尝试加入正则化项，以

减轻过拟合现象；第三，可以继续比较不同学习率、样本数量和噪声强度对实验结果的影响，使实验结论更加完整。

4. 附录

```python
from __future__ import annotations

import math
from pathlib import Path

import matplotlib.pyplot as plt
import torch

OUTPUT_PATH = Path(__file__).resolve().parent / "fit_comparison.png"


# Step 1: 生成模拟数据
def make_dataset(num_samples: int = 300, seed: int = 42) -> tuple[torch.Tensor, torch.Tensor]:
    torch.manual_seed(seed)

    useful_x = torch.rand(num_samples, 5) * 4 - 2

    x1 = useful_x[:, 0:1]
    x2 = useful_x[:, 1:2]
    x3 = useful_x[:, 2:3]
    x4 = useful_x[:, 3:4]
    x5 = useful_x[:, 4:5]

    y = (
        2.0 * x1
        - 1.5 * x2**2
        + 1.2 * x3 * x4
        + 0.8 * torch.sin(math.pi * x5)
        + 0.6 * x1 * x5
        + 0.3
    )

    y = y + 0.2 * torch.randn(num_samples, 1)

    noise_x = torch.randn(num_samples, 20)

    all_x = torch.cat([useful_x, noise_x], dim=1)
    return all_x, y
```

```python


# Step 2: 划分训练集和测试集
def train_test_split(
    x: torch.Tensor, y: torch.Tensor, train_size: int, seed: int
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]:
    generator = torch.Generator().manual_seed(seed)

    index = torch.randperm(x.shape[0], generator=generator)

    train_index = index[:train_size]
    test_index = index[train_size:]
    return x[train_index], y[train_index], x[test_index], y[test_index]


# Step 3: 构造不同的特征映射
def underfit_features(x: torch.Tensor) -> torch.Tensor:
    return x


def good_features(x: torch.Tensor) -> torch.Tensor:
    x1 = x[:, 0:1]
    x2 = x[:, 1:2]
    x3 = x[:, 2:3]
    x4 = x[:, 3:4]
    x5 = x[:, 4:5]

    return torch.cat(
        [x1, x2**2, x3 * x4, torch.sin(math.pi * x5), x1 * x5],
        dim=1,
    )


def overfit_features(x: torch.Tensor) -> torch.Tensor:
    parts = [x, x**2]

    cross_terms = []
    for i in range(15):
        for j in range(i + 1, 15):
            cross_terms.append(x[:, i : i + 1] * x[:, j : j + 1])

    return torch.cat(parts + cross_terms, dim=1)

```

```python

# Step 4: 数据标准化
def standardize(
    train_x: torch.Tensor, test_x: torch.Tensor
) -> tuple[torch.Tensor, torch.Tensor]:
    mean = train_x.mean(dim=0, keepdim=True)
    std = train_x.std(dim=0, keepdim=True, unbiased=False)

    std[std < 1e-6] = 1.0
    return (train_x - mean) / std, (test_x - mean) / std


# Step 5: 实现损失函数和评价函数
def mse_loss(pred: torch.Tensor, target: torch.Tensor) -> torch.
    Tensor:
    return ((pred - target) ** 2).mean()


def rmse(pred: torch.Tensor, target: torch.Tensor) -> float:
    return float(torch.sqrt(mse_loss(pred, target)))


def r2_score(pred: torch.Tensor, target: torch.Tensor) -> float:
    ss_res = ((pred - target) ** 2).sum()
    ss_tot = ((target - target.mean()) ** 2).sum()
    return float(1 - ss_res / ss_tot)


# Step 6: 实现线性模型
def train_linear_model(
    train_x: torch.Tensor,
    train_y: torch.Tensor,
    epochs: int,
    lr: float,
) -> tuple[torch.Tensor, torch.Tensor]:
    w = torch.zeros(train_x.shape[1], 1, requires_grad=True)
    b = torch.zeros(1, requires_grad=True)

    for _ in range(epochs):
        pred = train_x @ w + b
        loss = mse_loss(pred, train_y)
        loss.backward()

        with torch.no_grad():
            w -= lr * w.grad
```

```
            b -= lr * b.grad

            w.grad.zero_()
            b.grad.zero_()

    return w.detach(), b.detach()


def evaluate_case(
    name: str,
    plot_name: str,
    feature_fn,
    x: torch.Tensor,
    y: torch.Tensor,
    train_size: int,
    epochs: int,
    lr: float,
    seed: int,
) -> dict[str, torch.Tensor | float | str]:
    train_x, train_y, test_x, test_y = train_test_split(x, y, train_size=train_size, seed=seed)

    train_x = feature_fn(train_x)
    test_x = feature_fn(test_x)

    train_x, test_x = standardize(train_x, test_x)

    w, b = train_linear_model(train_x, train_y, epochs=epochs, lr=lr)

    train_pred = train_x @ w + b
    test_pred = test_x @ w + b

    print(f"\n{name}")
    print(f"train RMSE: {rmse(train_pred, train_y):.4f}, train R2: {r2_score(train_pred, train_y):.4f}")
    print(f"test  RMSE: {rmse(test_pred, test_y):.4f}, test  R2: {r2_score(test_pred, test_y):.4f}")

    return {
        "name": name,
        "plot_name": plot_name,
        "train_rmse": rmse(train_pred, train_y),
        "test_rmse": rmse(test_pred, test_y),
        "train_r2": r2_score(train_pred, train_y),
```

```
        "test_r2": r2_score(test_pred, test_y),
    }


# Step 7: 绘图
def draw_result_figure(results: list[dict[str, torch.Tensor | float | str]]) -> None:
    labels = [str(result["plot_name"]) for result in results]
    train_rmse = [float(result["train_rmse"]) for result in results]
    test_rmse = [float(result["test_rmse"]) for result in results]
    train_r2 = [float(result["train_r2"]) for result in results]
    test_r2 = [float(result["test_r2"]) for result in results]

    x = list(range(len(results)))
    width = 0.36

    fig, axes = plt.subplots(1, 2, figsize=(11, 4.6))
    ax_rmse, ax_r2 = axes

    rmse_train_bars = ax_rmse.bar(
        [value - width / 2 for value in x],
        train_rmse,
        width=width,
        label="Train RMSE",
        color="#4C72B0",
    )
    rmse_test_bars = ax_rmse.bar(
        [value + width / 2 for value in x],
        test_rmse,
        width=width,
        label="Test RMSE",
        color="#DD8452",
    )
    ax_rmse.set_title("RMSE Comparison")
    ax_rmse.set_ylabel("RMSE")
    ax_rmse.set_xticks(x)
    ax_rmse.set_xticklabels(labels)
    ax_rmse.grid(axis="y", linestyle="--", alpha=0.4)
    ax_rmse.legend()

    r2_train_bars = ax_r2.bar(
        [value - width / 2 for value in x],
        train_r2,
```

```
        width=width,
        label="Train R2",
        color="#55A868",
    )
    r2_test_bars = ax_r2.bar(
        [value + width / 2 for value in x],
        test_r2,
        width=width,
        label="Test R2",
        color="#C44E52",
    )
    ax_r2.set_title("R2 Comparison")
    ax_r2.set_ylabel("R2")
    ax_r2.set_ylim(0.0, 1.08)
    ax_r2.set_xticks(x)
    ax_r2.set_xticklabels(labels)
    ax_r2.grid(axis="y", linestyle="--", alpha=0.4)
    ax_r2.legend()

    for bars, axis, fmt in (
        (rmse_train_bars, ax_rmse, "{:.2f}"),
        (rmse_test_bars, ax_rmse, "{:.2f}"),
        (r2_train_bars, ax_r2, "{:.3f}"),
        (r2_test_bars, ax_r2, "{:.3f}"),
    ):
        for bar in bars:
            height = bar.get_height()
            axis.text(
                bar.get_x() + bar.get_width() / 2,
                height,
                fmt.format(height),
                ha="center",
                va="bottom",
                fontsize=8,
            )

    fig.tight_layout(rect=[0, 0.05, 1, 0.95])
    fig.savefig(OUTPUT_PATH, dpi=200, bbox_inches="tight")
    plt.close(fig)


def main() -> None:
    x, y = make_dataset()

    results = []
```

```
254
255     print("真实函数:")
256     print("y = 2.0*x1 - 1.5*x2^2 + 1.2*x3*x4 + 0.8*sin(pi*x5) +
        0.6*x1*x5 + 0.3")
257     print("数据总维度: 25 维 = 5 个有效特征 + 20 个噪声特征")
258
259     results.append(
260         evaluate_case(
261             name="1. 欠拟合",
262             plot_name="Underfit",
263             feature_fn=underfit_features,
264             x=x,
265             y=y,
266             train_size=200,
267             epochs=400,
268             lr=0.05,
269             seed=1,
270         )
271     )
272
273     results.append(
274         evaluate_case(
275             name="2. 正常拟合",
276             plot_name="Good Fit",
277             feature_fn=good_features,
278             x=x,
279             y=y,
280             train_size=200,
281             epochs=400,
282             lr=0.05,
283             seed=1,
284         )
285     )
286
287     results.append(
288         evaluate_case(
289             name="3. 过拟合",
290             plot_name="Overfit",
291             feature_fn=overfit_features,
292             x=x,
293             y=y,
294             train_size=30,
295             epochs=800,
296             lr=0.03,
297             seed=2,
```

```
            )
        )

        draw_result_figure(results)
        print(f"\n图片已保存到: {OUTPUT_PATH}")


if __name__ == "__main__":
    main()
```