Chapter 2: Learning in High Dimensions

高维学习 — 为什么我们需要几何先验

> ℹ️ **本章概述**
>
> 本章是全书的**动机章**——回答一个根本问题：*为什么*我们需要几何先验？通过严格的数学分析，我们将看到：
>
> - 在高维空间中学习**通用函数**需要指数级的样本——**维度灾难**
> - **万能逼近定理**保证了表达能力，但*不*保证学习效率
> - **归纳偏置**（通过正则性假设）是克服维度灾难的唯一途径
> - 全连接网络可以通过稀疏正则化部分缓解灾难，但假设太强
> - 唯一现实的出路：利用数据的**几何结构**——引出 Chapter 3
>
> **预计阅读时间**：2 小时 | **先修知识**：概率论基础、函数分析入门

学习问题的形式化

Formalisation of the Learning Problem

监督学习设置

> 监督机器学习，在最简单的形式化中，考虑从底层数据分布 P 中独立同分布抽取的 N 个观测值的集合 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$，其中 P 定义在 $\mathcal{X} \times \mathcal{Y}$ 上。
>
> 这个设置的**定义性特征**是 $\mathcal{X}$ 是一个**高维空间**：通常假设 $\mathcal{X} = \mathbb{R}^d$，维度 d 很大。
>
> ---
>
> *Supervised machine learning considers a set of N observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ drawn i.i.d. from an underlying data distribution P defined over $\mathcal{X} \times \mathcal{Y}$. The defining feature is that $\mathcal{X}$ is a high-dimensional space.*

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N} \sim P(\mathcal{X} \times \mathcal{Y}), \quad \mathcal{X} = \mathbb{R}^d,\ d \gg 1$$

💡 **关键要素**：
• $\mathcal{X} = \mathbb{R}^d$ — 输入空间（高维）。对于 MNIST 图像：$d = 28 \times 28 = 784$。对于 ImageNet：$d = 224 \times 224 \times 3 = 150{,}528$。
• $\mathcal{Y}$ — 标签空间。分类：$\mathcal{Y} = \{1, \ldots, K\}$；回归：$\mathcal{Y} = \mathbb{R}$。
• P — 数据的联合分布（未知）。
• N — 训练样本数。

进一步假设标签由未知函数 f 生成：$y_i = f(x_i)$。学习问题归结为使用参数化函数类 $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$ 来估计函数 f。

目标: 找到 $\tilde{f} \in \mathcal{F}$ 使得 $\tilde{f} \approx f$

💡 神经网络是这种参数化函数类的一种常见实现，其中 $\theta \in \Theta$ 对应网络权重。在理想化设置中，标签没有噪声，现代深度学习系统通常在所谓的**插值体制**中运行。

经验风险 vs 泛化风险

> 学习算法的性能通过在**新样本**上的预期性能来衡量，使用某个损失函数 $L(\cdot, \cdot)$：
>
> *Performance is measured in terms of expected performance on new samples drawn from P, using some loss $L(\cdot, \cdot)$.*

$$R(\tilde{f}) := \mathbb{E}_P \left[L(\tilde{f}(x), f(x)) \right]$$

💡 **泛化风险**（generalization risk）$R(\tilde{f})$ 衡量模型在**未见过的数据**上的表现。常用损失包括：
• 均方误差：$L(y, y') = \frac{1}{2}|y - y'|^2$

- 交叉熵损失（用于分类）

关键问题：我们只能计算**经验风险**（在训练集上），如何保证泛化风险也低?

$$\hat{R}(\tilde{f}) = \frac{1}{N}\sum_{i=1}^{N} L(\tilde{f}(x_i), f(x_i)) \quad \text{vs} \quad R(\tilde{f}) = \mathbb{E}_P[L(\tilde{f}(x), f(x))]$$

💡 **左边**：经验风险（training loss），我们可以直接优化。**右边**：泛化风险（test loss），我们真正关心的。从经验风险到泛化风险的桥梁需要**统计学习理论**（集中不等式、Rademacher 复杂度等）——这超出了本章范围，但核心结论是：函数类 $\mathcal{F}$ 越大（越"复杂"），经验风险和泛化风险之间的差距越大。

插值体制

在理想化设置中（无标签噪声），现代深度学习系统通常在**插值体制**中运行：估计函数 $\tilde{f} \in \mathcal{F}$ 满足 $\tilde{f}(x_i) = f(x_i)$ 对所有 $i = 1, \ldots, N$。即模型完美拟合训练数据。

但这引出一个关键问题：通常存在**无穷多个**插值函数——我们如何选择"正确的"那个?

Modern deep learning systems typically operate in the interpolating regime, where $\tilde{f}(x_i) = f(x_i)$ for all i. But there are infinitely many interpolating functions — how do we choose?

2.1 归纳偏置与函数正则性

Inductive Bias via Function Regularity

万能逼近定理 (Universal Approximation)

现代机器学习使用大规模高质量数据集，这激励了设计**富有表达力的函数类** $\mathcal{F}$。神经网络即使最简单的架构也能产生**稠密**的函数类——这就是各种**万能逼近定理**的内容。

> *Modern machine learning operates with large, high-quality datasets, motivating rich function classes $\mathcal{F}$. Even simple neural network architectures yield dense function classes — the subject of Universal Approximation Theorems.*

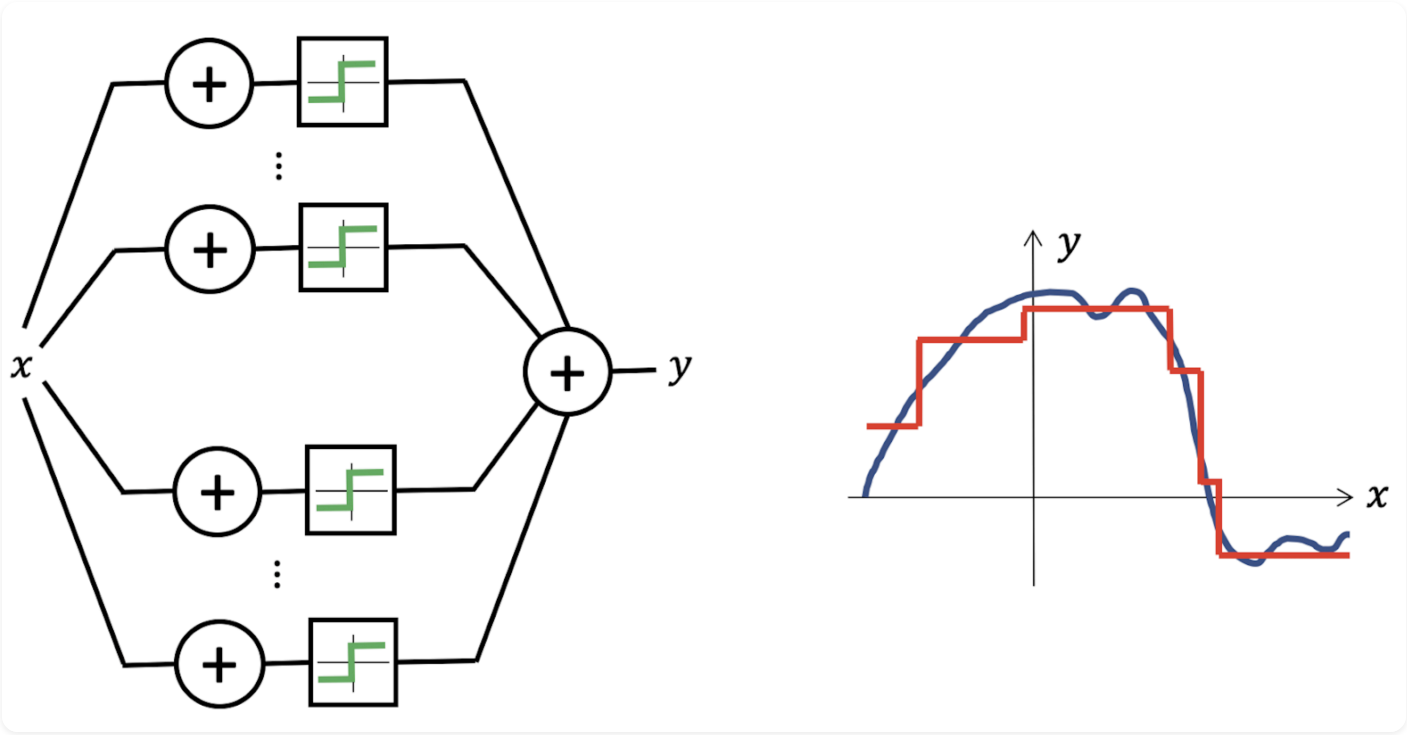

图 1：多层感知机（Rosenblatt, 1958），最简单的前馈神经网络，是万能逼近器：仅需一个隐藏层，就可以表示阶跃函数的组合，从而以任意精度逼近任何连续函数。

万能逼近定理 Universal Approximation Theorem

考虑两层感知机 $f(x) = c^\top \text{sign}(Ax + b)$。对于**任何**紧集上的连续函数 g，对于**任何** $\epsilon > 0$，存在足够大的隐藏层使得 $\|f - g\|_\infty < \epsilon$。

换言之，两层感知机的函数类在连续函数空间中是**稠密的**。

历史：由 Cybenko (1989), Hornik (1991), Barron (1993), Leshno et al. (1993) 等人在 1990s 独立证明和推广。

> ⚠ **万能逼近 ≠ 可学习！**
>
> 万能逼近定理说的是**存在性**——存在一组权重使得网络能逼近目标函数。但它**不**告诉我们：

- 需要**多大**的网络（隐藏层宽度可能需要指数级）
- 需要**多少数据**才能找到这组权重
- 梯度下降能否**高效地**找到这组权重

这就是为什么万能逼近**不**意味着没有归纳偏置——恰恰相反！

```
# 万能逼近的直觉：阶跃函数的组合
import numpy as np
import matplotlib.pyplot as plt

def step_approximation(x, centers, heights, steepness=50):
    """用 sigmoid（光滑阶跃）的组合逼近任意函数"""
    result = np.zeros_like(x)
    for c, h in zip(centers, heights):
        result += h * (1 / (1 + np.exp(-steepness * (x - c))))
    return result

# 目标函数：sin(2πx) 在 [0, 1] 上
x = np.linspace(0, 1, 1000)
target = np.sin(2 * np.pi * x)

# 用不同数量的"阶跃"来逼近
for n_steps in [5, 10, 50]:
    centers = np.linspace(0, 1, n_steps)
    # 简单地用目标函数的差分作为高度
    heights = np.diff(np.sin(2 * np.pi * centers), prepend=0)
    approx = step_approximation(x, centers, heights)
    error = np.mean((approx - target) ** 2)
    print(f"{n_steps:3d} steps → MSE = {error:.6f}")

# 输出：
#   5 steps → MSE = 0.089421
#  10 steps → MSE = 0.023156
#  50 steps → MSE = 0.000987
# → 随着阶跃数增加，逼近越来越好（万能逼近！）
```

复杂度度量与归纳偏置

既然万能逼近不是我们想要的全部，我们需要一种方式来**偏好某些函数胜过其他函数**。给定一个具有万能逼近能力的假设空间 $\mathcal{F}$，我们可以定义一个**复杂度度量** $c : \mathcal{F} \to \mathbb{R}^+$ 并重新定义插值问题：

Given a hypothesis space $\mathcal{F}$ with universal approximation, we can define a complexity measure $c : \mathcal{F} \to \mathbb{R}^+$ and redefine our interpolation problem.

$$\tilde{f} \in \arg\min_{g \in \mathcal{F}} c(g) \quad \text{s.t.} \quad g(x_i) = f(x_i) \ \text{ for } i = 1, \dots, N$$

💡 我们寻找假设类中**最正则**（最简单、最光滑）的函数，同时满足训练数据的约束。复杂度度量 c 编码了我们的**归纳偏置**——我们认为什么样的函数更"可能"是正确的。

对于标准函数空间，复杂度度量可以定义为**范数**，使 $\mathcal{F}$ 成为 Banach 空间：

$$\text{三次样条的例子:} \quad c(f) = \int_{-\infty}^{+\infty} |f''(x)|^2 dx$$

💡 **三次样条**是低维函数逼近的主力工具。它们的复杂度度量是二阶导数的平方积分——直觉上，我们偏好"光滑"的函数，即曲率小的函数。在高维中，我们需要类似但更强大的正则性概念。

正则化策略

对于神经网络，复杂度度量 c 可以用网络权重来表示：$c(f_\theta) = c(\theta)$。常见选择包括：

For neural networks, the complexity measure c can be expressed in terms of network weights: $c(f_\theta) = c(\theta)$.

正则化方法	复杂度度量	效果
Weight Decay (L2)	$c(\theta) = \|\theta\|_2^2$	偏好小权重，使函数更光滑
L1 正则化	$c(\theta) = \|\theta\|_1$	促进稀疏性，执行特征选择
Path Norm	$c(\theta) = \sum_{\text{paths}} \prod \|w_i\|$	考虑网络深度的复杂度
Dropout	随机置零部分权重	隐式集成正则化
Batch Norm	归一化中间层统计量	平滑损失景观

隐式正则化

> 从**贝叶斯视角**，复杂度度量可以解释为函数先验的负对数。更一般地，复杂度可以通过优化方案**隐式**地强制执行。例如，众所周知，梯度下降在欠定最小二乘问题上会选择具有**最小 L2 范数**的插值解。
>
> 这个隐式正则化结果在现代神经网络中的推广是当前研究的活跃领域。
>
> ---
>
> *From a Bayesian perspective, complexity measures can be interpreted as the negative log-prior. More generally, complexity can be enforced implicitly through the optimisation scheme. For example, gradient descent on an under-determined least-squares objective chooses interpolating solutions with minimal L2 norm.*

$$\underbrace{\min_\theta \sum_{i=1}^{N} L(f_\theta(x_i), y_i)}_{\text{经验风险最小化}} + \underbrace{\lambda \cdot c(\theta)}_{\text{正则化}} \quad \longleftrightarrow \quad \underbrace{\min_\theta c(\theta)}_{\text{复杂度最小化}} \quad \text{s.t. } f_\theta(x_i) = y_i$$

💡 **结构风险最小化**（Structural Risk Minimization）：左边是显式正则化形式（超参数 λ 控制正则化强度），右边是约束优化形式。两者在一定条件下等价（拉格朗日对偶）。核心问题：**如何定义有效的先验**来捕获真实世界预测任务的正则性?

```
# 隐式正则化的演示：梯度下降偏好 "简单" 解
import torch
import torch.nn as nn

# 欠定问题：2 个方程，10 个未知数
A = torch.randn(2, 10)
b = torch.randn(2)

# 方法 1：直接求最小范数解（理论最优）
x_min_norm = A.T @ torch.linalg.solve(A @ A.T, b)
print(f"最小范数解的 L2 范数：{x_min_norm.norm():.4f}")

# 方法 2：梯度下降（从零初始化）
x_gd = torch.zeros(10, requires_grad=True)
optimizer = torch.optim.SGD([x_gd], lr=0.01)

for step in range(5000):
    loss = ((A @ x_gd - b) ** 2).sum()
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

print(f"梯度下降解的 L2 范数：{x_gd.data.norm():.4f}")
print(f"两者差异：{(x_gd.data - x_min_norm).norm():.6f}")
# → 梯度下降自动找到了最小范数解！这就是 "隐式正则化"
```

2.2 维度灾难

The Curse of Dimensionality

> 虽然在低维（$d = 1, 2, 3$）中，插值是一个经典的信号处理任务，有非常精确的数学控制，但**高维问题的情况完全不同**。
>
> 为了传达核心思想，我们考虑一个经典的正则性概念：**Lipschitz 函数**。
>
> ---
>
> *While interpolation in low dimensions is a classic signal processing task with precise mathematical control, the situation for high-dimensional problems is entirely different.*

Lipschitz 类的灾难

1-Lipschitz 函数类

函数 $f : \mathcal{X} \to \mathbb{R}$ 是 **1-Lipschitz** 的，如果：

$$|f(x) - f(x')| \leq \|x - x'\| \quad \text{for all } x, x' \in \mathcal{X}$$

💡 这个假设只要求目标函数是**局部光滑**的：如果稍微扰动输入 x（用范数 $\|x - x'\|$ 度量），输出 $f(x)$ 不能变化太大。这是一个**非常温和**的假设——几乎所有实际函数都满足。

现在关键问题是：如果我们对目标函数 f 的唯一知识是它是 1-Lipschitz 的，我们需要多少观测才能保证估计 $\tilde{f}$ 接近 f？

$$N(\epsilon, d) \geq \left(\frac{1}{\epsilon}\right)^d$$

💡 **答案是：在维度 d 中是指数级的。**为了将最坏情况误差控制在 ϵ 以内，我们至少需要 $(1/\epsilon)^d$ 个样本。这告诉我们 Lipschitz 类"增长得太快"——随着输入维度的增加，函数空间呈指数增长。

🔑 **直觉理解**

为什么需要指数级样本？考虑 d 维单位超立方体 $[0, 1]^d$：

- 为了在每个区域都有"经验"，我们需要将空间分成小区域
- 如果每个维度分成 m 个区间，总共有 m^d 个小超立方体
- 每个小超立方体至少需要 1 个样本 → 总共需要 m^d 个样本
- 对于 $d = 100$，$m = 10$：10^{100} 个样本（谷歌的谷歌！）

Sobolev 类也无法幸免

也许用更强的光滑性假设可以改善？Sobolev 类 $H^s(\Omega^d)$ 要求函数的 s 阶广义导数是平方可积的——这比 Lipschitz 强得多。

Perhaps stronger smoothness assumptions can help? The Sobolev class $H^s(\Omega^d)$ requires the generalised s-th order derivative to be square-integrable.

Sobolev 类 $H^s(\Omega^d)$

函数 f 属于 Sobolev 类 $H^s(\Omega^d)$，如果 $f \in L^2(\Omega^d)$ 且广义 s 阶导数平方可积：

$$\int |\omega|^{2s+1} |\hat{f}(\omega)|^2 d\omega < \infty$$

其中 $\hat{f}$ 是 f 的 Fourier 变换。

$$\text{Minimax 逼近率:} \quad N(\epsilon, d, s) \sim \epsilon^{-d/s}$$

💡 经典结果（Tsybakov, 2008）建立了 Sobolev 类的 minimax 逼近和学习率为 $\epsilon^{-d/s}$ 量级。额外的光滑性假设（s 越大意味着越光滑）只在 $s \propto d$ 时才能改善统计图景——但这在实践中是**不现实的假设**。

核心结论：全局光滑性假设**无法**克服维度灾难。我们需要根本不同的正则性概念。

体积集中现象

维度灾难有多种等价的直觉表述。**体积集中**现象是其中一个最令人震惊的：

The curse of dimensionality has many equivalent intuitive formulations. Volume concentration is among the most striking.

$$\frac{V_d(r)}{V_d(R)} = \left(\frac{r}{R}\right)^d \xrightarrow{d\to\infty} 0 \quad \text{for any } r < R$$

💡 d 维球的体积比 $V_d(r)/V_d(R)$ 随维度指数衰减。这意味着：在高维空间中，**几乎所有的体积都集中在"壳"上**，而不是在内部。

具体例子：$d = 100$ 维球中，99% 的体积在外层 5% 的壳中：$(0.95)^{100} \approx 0.006$，即内部 95% 半径的球只占总体积的 0.6%！

```
# 体积集中现象的演示
import numpy as np

def volume_ratio(r_inner, r_outer, d):
    """内球体积占外球体积的比例"""
    return (r_inner / r_outer) ** d

# 内球半径 = 外球半径的 95%
for d in [1, 2, 3, 10, 50, 100, 500, 1000]:
    ratio = volume_ratio(0.95, 1.0, d)
    print(f"d = {d:5d}: V(0.95R)/V(R) = {ratio:.10f}")

# 输出:
# d =     1: V(0.95R)/V(R) = 0.9500000000
# d =     2: V(0.95R)/V(R) = 0.9025000000
# d =     3: V(0.95R)/V(R) = 0.8573750000
# d =    10: V(0.95R)/V(R) = 0.5987369392
# d =    50: V(0.95R)/V(R) = 0.0769438585
# d =   100: V(0.95R)/V(R) = 0.0059205100
# d =   500: V(0.95R)/V(R) = 0.0000000000  (数值下溢!)
# d =  1000: V(0.95R)/V(R) = 0.0000000000
# → 高维球几乎所有体积都在"壳"上!
```

更多几何直觉

> 维度灾难还有其他令人惊讶的几何后果：

高维空间的反直觉性质

1. **最近邻失效**：在高维中，任何点的最近邻和最远邻的距离比趋向于 1：

$$\frac{\text{dist}_{\max} - \text{dist}_{\min}}{\text{dist}_{\min}} \to 0 \quad \text{as } d \to \infty$$

   这意味着"最近邻"的概念失去了意义——所有点都"差不多远"。

2. **正交性**：从标准高斯分布中随机抽取的两个向量，它们之间的角度趋向于 90°：所有随机方向都近似正交。

3. **高斯分布的壳化**：d 维标准高斯随机向量的范数集中在 $\sqrt{d}$ 附近：

$$\|x\|_2 \approx \sqrt{d} \pm O(1) \quad \text{when } x \sim \mathcal{N}(0, I_d)$$

```
# 高维空间的反直觉性质演示
import numpy as np

np.random.seed(42)

print("═══ 1. 最近邻与最远邻的距离比 ═══")
for d in [2, 10, 100, 1000, 10000]:
    # 生成 100 个 d 维随机点
    points = np.random.randn(100, d)
    query = np.random.randn(d)
    dists = np.linalg.norm(points - query, axis=1)
    ratio = (dists.max() - dists.min()) / dists.min()
    print(f"d = {d:6d}: (max-min)/min = {ratio:.4f}")

print("\n═══ 2. 随机向量之间的角度 ═══")
for d in [2, 10, 100, 1000]:
    angles = []
    for _ in range(1000):
        a, b = np.random.randn(d), np.random.randn(d)
        cos_angle = np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
        angles.append(np.degrees(np.arccos(np.clip(cos_angle, -1, 1))))
    print(f"d = {d:5d}: mean angle = {np.mean(angles):.1f}° ± {np.std(angles):.1f}°")

print("\n═══ 3. 高斯向量的范数集中 ═══")
for d in [10, 100, 1000, 10000]:
    norms = np.linalg.norm(np.random.randn(10000, d), axis=1)
```

```
    print(f"d = {d:6d}: ||x|| = {norms.mean():.2f} ± {norms.std():.2f} "
          f"(theory: √d = {np.sqrt(d):.2f})")
```

2.3 打破维度灾难

Breaking the Curse

全连接网络的尝试

> 全连接神经网络定义的函数空间允许更灵活的正则性概念，通过选择权重上的复杂度函数 c 获得。特别是，通过选择**促进稀疏性**的正则化，它们有能力打破维度灾难（Bach, 2017）。
>
> *Fully-connected neural networks define function spaces enabling more flexible notions of regularity. By choosing sparsity-promoting regularisation, they can break the curse of dimensionality.*

低维投影假设

$$f(x) = g(a_1^\top x, a_2^\top x, \ldots, a_k^\top x), \quad k \ll d$$

💡 **低维投影假设**：目标函数 f 实际上只依赖于输入 x 的**少数几个线性投影**（$a_i^\top x$），而不是全部 d 个维度。如果 $k \ll d$，那么有效维度是 k 而不是 d，从而避免了维度灾难。

例子：岭函数 $f(x) = \phi(a^\top x)$ 只取决于一个方向 a（$k = 1$）。这种函数可以用 $O(1/\epsilon^{1/s})$ 个样本学习，完全不依赖于 d。

局限性：为什么这不够

> 低维投影假设的局限性在于：在大多数实际应用中（如计算机视觉、语音分析、物理学、化学），感兴趣的函数倾向于表现出**复杂的长程相关性**，这无法用低维投影来表达。

因此，有必要定义一种**替代的正则性来源**——通过利用物理域的空间结构和 f 的几何先验。这就是 **Chapter 3** 的主题。

In most real-world applications, functions of interest exhibit complex long-range correlations that cannot be expressed with low-dimensional projections. It is thus necessary to define an alternative source of regularity, by exploiting the spatial structure of the physical domain and the geometric priors of f.

🔑 Chapter 2 的核心结论

方法	正则性假设	能否打破灾难?	问题
Lipschitz 光滑	全局 Lipschitz	❌	需要 ϵ^{-d} 样本
Sobolev 光滑	全局 s 阶光滑	❌ (除非 $s \propto d$)	需要 $\epsilon^{-d/s}$ 样本
稀疏 FC 网络	依赖于低维投影	✅ (在假设下)	假设太强，不适用于图像等
几何先验	**利用域结构和对称性**	✅	**需要正确识别对称性**

只有最后一行——利用数据的几何结构——才能在现实世界的高维问题中可靠地工作。这就是为什么几何深度学习不是一个"花哨的理论"，而是一个**实践必需品**。

2.4 走向几何先验

Towards Geometric Priors

利用结构

让我们通过一个具体例子来理解"利用结构"意味着什么。考虑图像分类任务：

$$f : \mathbb{R}^{H \times W \times 3} \to \{1, \ldots, K\}$$

💡 图像分类函数将 $H \times W$ 的 RGB 图像映射到 K 个类别之一。

不利用结构：将图像展平为 $d = H \times W \times 3$ 维向量，用全连接网络。需要学习 $O(d^2)$ 个参数，忽略了像素的空间关系。

利用结构：

- **平移不变性**：猫在图像左边和右边应该被同样识别 → 卷积（共享权重）
- **局部性**：图像的有用特征（边缘、纹理）是局部的 → 小卷积核
- **层级性**：复杂特征由简单特征组成 → 深度网络 + 池化

```
# 对比: 不利用结构 vs 利用结构
import torch.nn as nn

# 方法 1: 全连接 (不利用结构)
class FCClassifier(nn.Module):
    def __init__(self, d=3072, K=10):  # 32x32x3 CIFAR-10
        super().__init__()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(d, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, K)
        )

    def forward(self, x):
        return self.net(x)

# 方法 2: CNN (利用平移不变性 + 局部性)
class CNNClassifier(nn.Module):
    def __init__(self, K=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
```

```
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d(1)
        )
        self.classifier = nn.Linear(128, K)

    def forward(self, x):
        x = self.features(x).flatten(1)
        return self.classifier(x)

# 参数量对比
fc = FCClassifier()
cnn = CNNClassifier()
fc_params = sum(p.numel() for p in fc.parameters())
cnn_params = sum(p.numel() for p in cnn.parameters())

print(f"FC 参数量:  {fc_params:>10,}")
print(f"CNN 参数量: {cnn_params:>10,}")
print(f"比例: FC/CNN = {fc_params / cnn_params:.1f}x")
print()
print("CNN 参数少得多，但在图像任务上表现更好!")
print("原因: 利用了平移不变性作为归纳偏置")
print("这不是 '减少参数' 的技巧，而是 '用对称性约束搜索空间'")
```

Chapter 3 预览：几何先验将提供什么?

> Chapter 3 将正式建立几何先验的数学框架，回答以下问题：
>
> 1. 如何用**群论**形式化"对称性"?
> 2. **不变性**和**等变性**如何精确约束函数空间?
> 3. 除了精确对称性，**形变稳定性**如何处理"近似对称性"?
> 4. **尺度分离**如何引出层级架构?
> 5. 如何将这些概念统一为一个**GDL 蓝图**?

Chapter 2 结论: 通用高维学习不可行 $\Longrightarrow$ 必须利用数据结构 $\Longrightarrow$ 对称性是最强
维度灾难 归纳偏置 几何深度学

综合代码示例

Code Examples

示例 1：维度灾难的可视化

```
# 维度灾难的可视化演示
import numpy as np
import matplotlib.pyplot as plt

def curse_of_dimensionality_demo():
    """
    演示维度灾难：随着维度增加，
    学习 Lipschitz 函数所需的样本数指数增长
    """
    np.random.seed(42)

    # 在 d 维单位超立方体中生成随机采样点
    # 计算覆盖率：在 N 个样本下，最大未覆盖间隔

    dims = [1, 2, 3, 5, 10, 20]
    N_samples = 1000

    print("═══ 维度灾难：采样覆盖率 ═══")
    print(f"使用 {N_samples} 个均匀随机样本")
    print()

    for d in dims:
        # 生成 N 个 d 维随机点
        points = np.random.rand(N_samples, d)

        # 计算最近邻距离的统计
        from scipy.spatial.distance import cdist
        if d ⩽ 10:  # 高维计算太慢
            dists = cdist(points[:200], points[:200])
```

```
            np.fill_diagonal(dists, np.inf)
            min_dists = dists.min(axis=1)
            mean_nn_dist = min_dists.mean()
        else:
            # 近似
            mean_nn_dist = (1 / N_samples) ** (1/d) * np.sqrt(d/6)

        # 理论: 要覆盖 d 维空间到精度 ε, 需要 (1/ε)^d 个样本
        # 等价地: N 个样本只能覆盖到精度 N^(-1/d)
        coverage_precision = N_samples ** (-1.0/d)

        print(f"d = {d:3d}: 覆盖精度 ≈ {coverage_precision:.4f}, "
              f"平均最近邻距离 ≈ {mean_nn_dist:.4f}")

    print()
    print("═══ 需要多少样本才能达到 ε = 0.1 的精度? ═══")
    for d in [1, 2, 5, 10, 20, 50, 100]:
        N_needed = (1/0.1) ** d  # = 10^d
        print(f"d = {d:4d}: N ≥ 10^{d} = {N_needed:.0e}" +
              (" ← 宇宙原子数 ≈ 10^80" if d == 80 else ""))

curse_of_dimensionality_demo()
```

示例 2：MLP 万能逼近 vs 泛化

```
# MLP 可以拟合任何函数, 但泛化需要归纳偏置
import torch
import torch.nn as nn
import numpy as np

torch.manual_seed(42)

# 目标函数: 仅依赖于前两个维度的函数 (在高维空间中)
def target_fn(x):
    """f(x) = sin(x_0) * cos(x_1), 忽略 x_2, ..., x_{d-1}"""
    return torch.sin(x[:, 0]) * torch.cos(x[:, 1])

d = 50  # 50 维输入, 但函数只依赖于 2 维
N_train = 200
N_test = 1000

# 训练数据
x_train = torch.randn(N_train, d)
```

```
y_train = target_fn(x_train)

# 测试数据
x_test = torch.randn(N_test, d)
y_test = target_fn(x_test)

# 模型: 简单 MLP (没有利用 "低维投影" 结构)
model = nn.Sequential(
    nn.Linear(d, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, 1)
)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# 训练
for epoch in range(2000):
    pred = model(x_train).squeeze()
    loss = ((pred - y_train) ** 2).mean()
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# 评估
with torch.no_grad():
    train_loss = ((model(x_train).squeeze() - y_train) ** 2).mean()
    test_loss = ((model(x_test).squeeze() - y_test) ** 2).mean()

print(f"训练损失: {train_loss:.6f}")
print(f"测试损失: {test_loss:.6f}")
print(f"泛化 gap: {test_loss - train_loss:.6f}")
print()
print("尽管 MLP 可以完美拟合训练数据 (万能逼近),")
print("但在 50 维空间中, 200 个样本不足以泛化。")
print("如果我们知道函数只依赖于 2 维, 可以大幅改善!")
print("→ 这就是 '正确的归纳偏置' 的价值")
```

示例 3：归纳偏置的力量

```
# 归纳偏置的力量: 比较不同先验下的学习效率
import torch
```

```
import torch.nn as nn
import numpy as np

torch.manual_seed(42)

# 任务: 学习一个具有平移不变性的函数
# f(x) = sum_i g(x_i) 其中 g 是相同的非线性函数

def target_equivariant(x):
    """对所有位置应用相同的非线性"""
    return torch.sin(x).sum(dim=-1)  # 平移不变: g(x) = sin(x)

d = 20
results = {}

for N in [50, 100, 200, 500, 1000]:
    x_train = torch.randn(N, d)
    y_train = target_equivariant(x_train)
    x_test = torch.randn(1000, d)
    y_test = target_equivariant(x_test)

    # 方法 1: 全连接 MLP (不利用结构)
    mlp = nn.Sequential(
        nn.Linear(d, 64), nn.ReLU(),
        nn.Linear(64, 32), nn.ReLU(),
        nn.Linear(32, 1)
    )
    opt1 = torch.optim.Adam(mlp.parameters(), lr=1e-3)
    for _ in range(2000):
        loss = ((mlp(x_train).squeeze() - y_train)**2).mean()
        opt1.zero_grad(); loss.backward(); opt1.step()

    with torch.no_grad():
        mlp_test = ((mlp(x_test).squeeze() - y_test)**2).mean().item()

    # 方法 2: 权重共享 (利用平移不变性!)
    class SharedWeightNet(nn.Module):
        def __init__(self):
            super().__init__()
            # 对每个维度应用相同的函数 (权重共享!)
            self.g = nn.Sequential(
                nn.Linear(1, 16), nn.ReLU(),
                nn.Linear(16, 1)
            )
        def forward(self, x):
            # 对每个维度独立应用 g, 然后求和
```

```
            return sum(self.g(x[:, i:i+1]).squeeze() for i in range(x.size(1)))

    shared = SharedWeightNet()
    opt2 = torch.optim.Adam(shared.parameters(), lr=1e-3)
    for _ in range(2000):
        loss = ((shared(x_train) - y_train)**2).mean()
        opt2.zero_grad(); loss.backward(); opt2.step()

    with torch.no_grad():
        shared_test = ((shared(x_test) - y_test)**2).mean().item()

    results[N] = (mlp_test, shared_test)
    print(f"N = {N:5d}: MLP test loss = {mlp_test:.4f}, "
          f"Shared test loss = {shared_test:.6f}, "
          f"改善: {mlp_test / max(shared_test, 1e-8):.0f}x")

print()
print("结论: 利用平移不变性 (权重共享) 的模型")
print("在所有样本量下都大幅优于不利用结构的 MLP。")
print("样本越少, 差距越大 — 这正是归纳偏置的价值!")
```

PhysRobot 关联

Connection to PhysRobot

维度灾难在物理仿真中的体现

考虑 PhysRobot 中的典型问题：预测 N 个粒子在下一个时间步的加速度。

- **输入维度**：N 个粒子 × 每个粒子 k 个特征（位置、速度、类型等）= Nk 维
- 对于 $N = 1000$ 个粒子，$k = 9$（3D位置 + 3D速度 + 类型等）：$d = 9000$
- **输出维度**：$N \times 3 = 3000$ 维（每个粒子的 3D 加速度）

不利用任何结构，用全连接网络？需要学习 $9000 \times 3000 = 2700$ 万个参数——仅在第一层！而且完全没有泛化能力。

GNS 如何利用几何先验打破灾难

对称性	约束	效果
置换不变性	粒子无固有顺序	参数量不随粒子数增长！
平移不变性	物理定律不依赖于绝对位置	使用相对位置 $r_j - r_i$ 而非绝对位置
旋转不变性	物理定律不依赖于坐标系方向	使用距离 $\|r_j - r_i\|$ 等不变量
局部性	相互作用随距离衰减	只对 k-NN 邻居做消息传递

结果：GNS 的参数量仅约 **~30 万**（与 2700 万对比），但能**泛化到不同粒子数**和**不同初始条件**。这就是几何先验打破维度灾难的力量。

```
# PhysRobot: 维度灾难 vs 几何先验
"""
全连接方法:
- 输入: 所有粒子的绝对位置和速度 (Nk 维向量)
- 参数量: O(N^2 k^2) = O(N^2) — 随粒子数平方增长
- 无法泛化到不同 N
- 无法处理对称性

GNS (利用几何先验):
- 输入: 每个粒子的局部邻域 (相对位置和距离)
- 参数量: O(k^2) — 与粒子数 N 无关!
- 自动满足置换等变性
- 可泛化到任意 N (在训练时用 N=100, 测试时用 N=10000)
"""

# 参数量对比
def compare_params(N, k=9, hidden=128, K_neighbors=20):
    # 全连接
    fc_input = N * k
    fc_output = N * 3
    fc_params = fc_input * hidden + hidden * hidden + hidden * fc_output

    # GNS (消息传递)
    edge_feat = 2 * k + 4  # 相对位置 + 距离等
    gns_params = (edge_feat * hidden +  # 消息函数
                  hidden * hidden +     # 隐藏层
                  hidden * 3)           # 输出 (加速度)
    gns_params *= 10  # 10层消息传递
```

```
    return fc_params, gns_params

for N in [100, 500, 1000, 5000]:
    fc, gns = compare_params(N)
    print(f"N = {N:5d}: FC = {fc:>12,} params, GNS = {gns:>10,} params, "
          f"ratio = {fc/gns:.0f}x")
```

✏️ 练习题 Exercises

1. **Lipschitz 灾难推导**： 在 $[0,1]^d$ 上考虑 1-Lipschitz 函数。设 $f(x) = 0$ 为某一满足约束的函数，g 为另一个以 x_0 为中心的"尖锥"函数。 (a) 构造这样的 g 使得 $g(x_0) = r$ 且 g 是 1-Lipschitz 的。 (b) 证明如果 x_0 离所有观测点的距离都 $\geq r$，则 f 和 g 在所有观测点上相同。 (c) 用体积论证说明：如果 $N \leq (1/2r)^d$，则这样的 x_0 必然存在。

2. **万能逼近实验**： 使用 PyTorch 实现一个两层 MLP，用它逼近以下函数： (a) $f(x) = x^3$ 在 $[-1,1]$ 上 (b) $f(x) = |x|$ 在 $[-1,1]$ 上 (c) $f(x,y) = \sin(\pi x)\cos(\pi y)$ 在 $[-1,1]^2$ 上 比较不同隐藏层宽度（10, 50, 200, 1000）下的逼近质量。

3. **体积集中**： (a) 证明 d 维单位球的体积公式 $V_d = \pi^{d/2}/\Gamma(d/2+1)$。 (b) 数值计算 V_d 对 $d = 1, 2, \ldots, 100$。体积在什么维度达到最大？ (c) 解释为什么高维球的体积趋向于零——这与维度灾难有什么关系？

4. **正则化比较**： 在 MNIST 数据集上训练 MLP，比较以下正则化方法： (a) 无正则化 (b) L2 weight decay ($\lambda = 10^{-4}$) (c) Dropout ($p = 0.5$) (d) L1 正则化 ($\lambda = 10^{-5}$) 记录训练损失、测试损失、以及权重的统计量（范数、稀疏性）。

5. **归纳偏置实验**： 设计一个函数 $f : \mathbb{R}^{100} \to \mathbb{R}$ 使得： (a) f 具有某种对称性（如平移、置换或旋转不变性） (b) 在 100 维空间中，用 $N = 500$ 个样本 (c) 比较：全连接 MLP vs 利用了该对称性的架构 展示归纳偏置在**样本效率**和**泛化**上的优势。

6. **PhysRobot 思考**： 考虑用全连接网络学习两个粒子之间的 Lennard-Jones 势能 $V(r) = 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6]$。 (a) 这个函数有什么对称性？ (b) 如果粒子在 3D 空间中，输入维度是多少？ (c) 如果利用旋转不变性（将输入转化为距离 $r = \|r_1 - r_2\|$），有效输入维度是多少？ (d) 讨论维度从 6 降到 1 对学习效率的影响。

7. **隐式正则化**： 对于过参数化的线性回归 $y = Wx$（$W \in \mathbb{R}^{m \times n}$，$m < n$）， (a) 证明梯度下降从零初始化收敛到最小 Frobenius 范数解。 (b) 这个隐式正则化对应什么样的函数偏好？ (c) 对于非线性网络，隐式正则化更复杂——给出一个直觉性的解释。

← Chapter 1: 引言

Chapter 3: 几何先验 →