机器学习 · 开卷复习手册

Machine Learning · 2026 春 · 授课老师 顾小东

全 19 讲 · 知识点 + 公式 + 手把手例题

19	318	410
讲	知识点 / 例题	配套习题(在线)

本手册由课程 PPT 整理生成，含逐题分步详解的例题（橙色框）。在线版含可练习题库：
nssmd.github.io/ml-course

目录

第 1 讲 机器学习导论

Introduction

本讲从 AI 发展史出发，阐明机器学习（Machine Learning）的本质——让计算机在不显式编程的前提下从数据中自动发现统计规律并优化自身性能，系统讲解数据、模型、损失函数、优化算法这四大核心要素 $\theta^* = \arg\min_\theta \mathcal{L}(\theta \mid \mathcal{D})$，并介绍机器学习系统的构建流程、监督/无监督/强化学习三大范式，以及泛化性、过拟合与正则化等关键概念。

1 人工智能、机器学习与深度学习的关系

三者是层层嵌套的包含关系，常被画成同心圆：**人工智能（AI）⊃ 机器学习（ML）⊃ 深度学习（DL）**。

- **人工智能（Artificial Intelligence）**：能够处理一般需要人类知识和智力介入任务的计算机系统。由 John McCarthy 等人于二十世纪五十年代奠基。
- **机器学习（Machine Learning）**：从若干输入-输出样例中学习数据规律的统计技术，是实现 AI 的一类主流方法。
- **深度学习（Deep Learning）**：一种自动学习数据表征（representation）的机器学习技术，以深度神经网络为代表。

直觉上：AI 是目标（造出智能系统），ML 是当前最有效的手段（让机器自己从数据中学），DL 是 ML 中近年最成功的子流派（用多层网络自动提取特征）。

AI　机器学习　深度学习　基本概念

2 AI 与机器学习发展简史

理解机器学习的来龙去脉有助于把握其方法论演变：

1. **二十世纪四五十年代（Early Days）**：1943 年 McCulloch & Pitts 提出大脑的布尔电路模型；1950 年图灵发表《Computing Machinery and Intelligence》并提出图灵测试。
2. **二十世纪五十至七十年代**：早期兴奋期，逻辑推理与符号主义。
3. **二十世纪七十至九十年代（Knowledge-based）**：知识系统时代，把人总结的知识体系交给计算机，代表是**专家系统**；同时归纳与统计学习兴起（符号主义、联结主义、统计学习）。
4. **二十一世纪、2012 年起**：**深度学习**爆发，深度神经网络、卷积网络、表征学习。
5. **2022 年起**：**大语言模型**时代，以 Transformer、ChatGPT 为代表。

核心趋势：从「人写规则」逐步走向「让机器从数据中自动学规律」。

发展史　专家系统　深度学习　大语言模型

3 机器学习 vs 传统编程

这是理解机器学习价值的关键对比：

- **传统编程**：人编写一段程序，按照**显式定义**的一组指令执行。输入 = 数据 + 程序，计算机输出**结果**。
- **机器学习**：人只提供数据和期望结果，让程序**自动从样例中总结**出处理过程。输入 = 数据 + 结果，计算机输出**程序（模型）**。

可以记成一句话：**机器学习** = 「**学习不用编程完成任务的能力**」。

适用场景：当任务的决策规则太多、太复杂，人难以穷举显式写出时（例如识别一张图里是不是猫、判断一张发票），机器学习就比手写规则更有效。

传统编程　机器学习　范式对比

4 机器学习的典型应用

机器学习如今**无处不在**，课程列举了它在各领域的代表性落地场景：

- **机器视觉（Computer Vision）**：目标检测、语义分割、医疗诊断、自动驾驶等。
- **自然语言处理（NLP）**：机器翻译、问答、文本摘要、报告生成，甚至中文诗歌生成。
- **语音助手（Speech）**：语音识别、智能助理。
- **推荐系统（Recommendation）**：短视频推荐、商品推荐、好友推荐。
- **用户画像（User Profiling）**：刻画用户特征以支撑精准营销与个性化服务。
- **AI 编程**：自动生成程序代码。
- **AI 作画**：根据文字生成图像，如 DALL·E。
- **AI 影片**：生成以假乱真的视频内容（如 NVIDIA「真假发布会」）。

这些应用的共同点是：任务的规则极其复杂、难以人工显式编写，但有大量数据可供学习，正是机器学习大显身手之处。

应用　机器视觉　NLP　推荐系统

5 机器学习的定义

课程给出的较完整定义：

> 机器学习是一种人工智能技术，使机器能够在**无需显式人工编程**的情况下，通过从数据中**自动发现潜在的统计规律（模型）**，并**不断优化自身性能（优化）**，以实现特定的目标（如最小化损失函数）。

拆解这句话对应了四个关键动作：

- 「从数据中」→ **数据**
- 「发现潜在规律（模型）」→ **模型 / 假设**
- 「实现特定目标」→ **损失函数 / 目标**
- 「不断优化自身性能」→ **优化算法**

用人类学习作类比：实践（数据/经验）→ 考试（测试/指标）→ 总结纠错（训练/优化）→ 知识（模型），机器学习是对人类「从经验中学习」过程的算法化模拟。

定义　类比　核心思想

6 机器学习四大核心要素

任何机器学习系统都可拆成四个要素，**仅有数据不能学习，数据 + 假设才能学习**：

1. **数据（经验）**：训练集 $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}$，$x$ 是输入特征，y 是标签/目标值。
2. **模型（假设）**：对数据规律的假设形式 $y = g(x \mid \theta) + \epsilon$，例如线性模型 $\hat{y} = wx + b$，其中 θ（如 w, b）是待学习的**参数**。
3. **损失函数（目标）**：衡量模型预测好坏的指标，例如平方损失

$$\mathcal{L}(\theta \mid \mathcal{D}) = \frac{1}{2} \sum_{i=1}^{N} [(wx_i + b) - y_i]^2$$

4. **优化算法（提升）**：寻找使损失最小的最优参数

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta \mid \mathcal{D})$$

四要素分别回答：用什么经验、假设什么规律、如何评估效果、如何找到最优模型。

四要素 数据 模型 损失函数 优化

7 特征提取与向量化

机器学习算法处理的是数值，因此各种形式的原始数据（图像、文本、声音等）都需要先转成数值向量：

- **向量化 / 特征提取（feature extraction）**：把每个样本表示成高维向量空间中的一个**点**。
- 例如水果分拣中，一个水果可表示为特征向量：颜色（红=1）、形状（圆=0）、大小（=3），即 $[1, 0, 3]$。

为什么重要：把样本统一表示为特征空间中的点后，「学习」就变成了在这个空间里寻找划分边界（分类）或拟合曲面（回归）的几何问题。特征的好坏直接决定模型上限——好特征让简单模型也能работать，差特征让复杂模型也学不动。深度学习的一大价值正是**自动学习特征表征**，省去人工特征工程。

特征提取 向量化 特征空间

8 构建机器学习系统的通用流程

构建一个机器学习系统通常按以下步骤进行：

1. **考虑数据可行性和规模**：能获得多少数据？获取代价（时间、算力、人力）有多大？
2. **选择合理的数据表示形式**：数据预处理、特征提取等。
3. **选择合理的模型假设**。
4. **选择合适的损失函数**。
5. **选择或设计一个学习（优化）算法**。

以「自动水果分拣」为例（机器学习的 Hello-World）：

- **步骤一 数据收集**：给每个水果样本打标签（Apple / Orange）。
- **步骤二 数据预处理**：不同水果在大小、亮度、位置上有差异，需去背景、调光照等。
- **步骤三 特征提取**：把每个水果表示为特征空间中的点。
- **步骤四 训练**：在特征空间中划分区域，得到**决策边界（decision boundary）**。
- **步骤五 测试评估**：用训练好的分类器对**新图像**预测，并用分类错误率等指标评估。

完整流程串起来是：数据收集 → 特征提取 → 训练 → 验证 → 评估 → 预测。

系统流程 决策边界 工作示例 评估

9 机器学习的三大范式

按是否有标签、如何获得反馈，机器学习分为三类：

- **监督学习（Supervised Learning）**：每个输入样本都有目标输出标签，相当于「有老师教」。数据形式 (x, y)，学习目标是函数映射 $x \to y$。典型任务：分类、回归、目标检测、语义分割。类比：练习题提供标准答案。
- **无监督学习（Unsupervised Learning）**：训练样本只有特征 x、**没有标签**，相当于「无老师」。学习目标是发现数据隐含或潜在的结构。典型任务：聚类、降维（如 PCA）、概率密度估计、数据生成、异常检测。
- **强化学习（Reinforcement Learning）**：智能体（Agent）通过与**环境（Environment）交互**来学习，学习从状态（state）到动作（action）的映射，以**最大化长期奖励（reward）**。类比：刷题只给总分、不给参考答案。代表案例：AlphaGo、机器人避障。一种经典的更新方式是**蒙特卡洛强化学习**，按采样轨迹的实际长期回报 R_t 修正状态价值估计：

  $$v(s_t) \leftarrow v(s_t) + \alpha \left(R_t - v(s_t)\right)$$

  其中 α 是学习率，R_t 是从状态 s_t 出发在一条采样轨迹中获得的实际长期回报。

一句话区分：监督学习给「标准答案」，无监督学习给「一堆无答案的数据」，强化学习给「分数/奖励信号」。

监督学习 无监督学习 强化学习 范式

10 监督学习：分类与回归

监督学习按输出 y 的类型分为两大类问题：

- **分类（Classification）**：输出是**离散的类别**。例如水果分拣输出「苹果 / 橙子」，本质是把特征空间划分成若干区域，每个区域对应一个类别，区域之间的界线就是**决策边界**。常用评估指标：分类错误率（被分错类别的样本比例）。
- **回归（Regression）**：输出是**连续的数值**。例如根据平均收入预测房价、预测车速等，本质是拟合一条曲线/曲面。常用评估指标：平方误差等。

区分要点：判断一个监督学习任务是分类还是回归，只看标签 y 是「类别（可数、无大小关系）」还是「数值（连续、有大小关系）」。例如「预测明天气温多少度」是回归，「预测明天是否下雨」是分类。

监督学习 分类 回归

11 无监督学习的典型任务

无监督学习从无标签数据中挖掘结构，常见任务包括：

- **聚类（Clustering）**：把相似的数据分组。常用于研究初期的探索性数据分析（exploratory data analysis），帮助洞察数据的内在结构。
- **降维与特征选择（Dimensionality Reduction）**：选择关键属性、降低数据维度。代表方法是**主成分分析（PCA）**，经典应用如「特征脸（eigenfaces）」。
- **概率密度估计（Density Estimation）**：估计数据的概率分布。
- **数据生成（Data Generation）**：寻找模型构造的分布 $p_{\text{model}}(x)$ 去近似真实数据分布 $p_{\text{data}}(x)$。
- **异常检测（Abnormal/Anomaly Detection）**：找出数据集中最不可能（least likely）的观测，例如网络入侵检测。

这些任务的共同点是**没有「正确答案」标签**，算法靠数据本身的分布和相似性来组织数据。

无监督学习 聚类 降维 PCA 异常检测

12 泛化性、欠拟合与过拟合

机器学习的根本追求是**泛化性（generalization）**：模型能否正确处理**没见过的**新样本？这引出「学习 vs 记忆」的核心矛盾——模型应学习**普遍规律**，而不是死记训练数据的**特定细节**。

- **欠拟合（Underfitting）**：模型**不够复杂**，无法刻画真实规律，模型假设可能根本不成立。表现为训练误差和测试误差都高。
- **过拟合（Overfitting）**：模型**过于复杂**，记住了数据的细节（如随机噪声）而非潜在规律。表现为训练误差很低但测试误差高。

随模型复杂度（Complexity）增加，训练误差持续下降，但测试误差先降后升，呈 U 形——最佳复杂度在 U 形谷底。

生动类比（驾考宝典）：靠「有『应』字选 B、三短一长选最长」这类应试技巧答题，就是典型的过拟合——记住了题目表面规律而没真正理解交规，遇到新题就失效。

泛化性 过拟合 欠拟合 模型复杂度

13 避免过拟合的常见方法

目标是选择**合理复杂度**的模型，让模型拟合**总体数据分布**而非仅训练集分布。常见手段：

1. **增加训练数据**：数据越多，越难记住噪声。
2. **数据划分（Hold-Out）**：保留未见数据评估泛化误差。典型划分为训练集（约 50%，「平时练习」）、验证集（约 25%，「模拟考」）、测试集（约 25%，「高考」）。
3. **交叉验证（Cross Validation）**：尤其数据规模小时，用 **K 折交叉验证**既能划分又能充分利用数据，给出对未见数据表现更可靠的估计，代价是更多计算量和时间。
4. **早停（Early Stopping）**：监控验证损失，在出现过拟合迹象（验证损失开始上升）时及时停止训练。
5. **正则化（Regularization）**：在损失函数中加入惩罚项，惩罚模型复杂度，例如

$$L(W, \lambda_1, \lambda_2) = \|y - WX\|^2 + \lambda_2 \|w\|_2^2 + \lambda_1 \|w\|_1$$

   其中 $\|w\|_2^2$ 是 L2 正则、$\|w\|_1$ 是 L1 正则。
6. **引入先验知识**：如贝叶斯先验。

正则化　交叉验证　早停　数据划分　过拟合

14 没有免费午餐定理（No Free Lunch）

没有免费午餐定理（No Free Lunch Theorem） 的核心结论：

- **没有一种通用的、在所有问题上都最好的模型**。
- **不同模型需要适配数据特征和具体应用**。

实践含义：不存在「万能算法」，选择模型时必须结合数据的结构、规模和任务目标，通过实验比较来确定。这也解释了为什么要学习这么多不同的机器学习算法——它们各有适用的场景和假设。

这是对「哪个模型更好？」这一问题的根本回答：脱离具体数据和任务谈「最好的模型」是没有意义的。

No Free Lunch　模型选择　理论

15 下节预告：线性回归

本讲建立了机器学习的整体框架（四要素、三大范式、泛化与过拟合）。下一讲将进入第一个具体算法——**线性回归（Linear Regression）**：

- 它是**最简单而典型**的机器学习算法，恰好把本讲的四要素串起来：用线性模型 $\hat{y} = wx + b$ 作假设、用平方损失作目标、用优化算法求最优参数。
- 其求解思想类似经典的**最小二乘法（Least Squares）**。

可以把线性回归当作本讲抽象概念的「第一个落地实例」，从它入手最容易理解机器学习「从数据中学参数」的完整闭环。

线性回归　最小二乘　预告

16 📝 例题1：用「房价预测」一步步拆出机器学习四大要素

很多同学背得出「数据、模型、损失、优化」四个词，但一到具体场景就懵。我们用一个**最小的房价预测**例子，把四要素一个一个对上号。

场景：某小区有 3 套房子，我们想根据「面积」预测「房价」。已知数据如下：

房子	面积 x（百平米）	房价 y（百万元）
1	1	2
2	2	3
3	3	5

第 1 步：数据（经验） —「用什么经验？」

就是上面这张表，写成集合：

$$\mathcal{D} = \{(1, 2),\ (2, 3),\ (3, 5)\}$$

每个括号里前一个数是输入特征 x（面积），后一个是标签 y（真实房价）。**为什么先要数据**：机器没有先验常识，它对「面积和房价的关系」一无所知，只能从这些样例里去归纳。

第 2 步：模型（假设） —「假设是什么规律？」

我们**假设**房价和面积近似是一条直线关系：

$$\hat{y} = wx + b$$

这里 $\hat{y}$ 是模型的**预测值**（带帽子，区别于真实 y），w 和 b 是**待学习的参数** θ。**为什么需要假设**：课件强调「仅凭数据不能学习，数据 + 假设才能学习」。如果不假设是直线，机器面对无穷多种可能的曲线，根本无从下手。这里我们就先押一个最简单的假设：直线。

第 3 步：损失函数（目标） —「怎么评估好不好？」

假设此刻机器随便猜了一组参数 $w = 1,\ b = 0$，即 $\hat{y} = x$。我们逐条算预测值和误差：

- 房子1：预测 $\hat{y} = 1$，真实 2，差 $1 \times 1 = 1$，平方 1
- 房子2：预测 $\hat{y} = 2$，真实 3，差 1，平方 1
- 房子3：预测 $\hat{y} = 3$，真实 5，差 2，平方 4

把平方误差加起来再乘 $\frac{1}{2}$（乘 $\frac{1}{2}$ 只是为了求导时好看，不影响谁大谁小）：

$$\mathcal{L} = \frac{1}{2}(1 + 1 + 4) = 3$$

为什么用平方：误差有正有负，直接相加会抵消；平方后都变正，且错得越离谱、惩罚越大。这个数字 3 就是衡量「当前模型有多差」的总分，**越小越好**。

第 4 步：优化算法（提升） —「怎么找到最好的参数？」

现在的目标写成一行就是：

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta \mid \mathcal{D})$$

意思是「找到那组让损失 $\mathcal{L}$ 最小的 w, b」。我们试着把直线调好一点，比如 $w = 1.5,\ b = 0$：

- 房子1：预测 1.5，差 0.5，平方 0.25
- 房子2：预测 3，差 0，平方 0
- 房子3：预测 4.5，差 0.5，平方 0.25
- 新损失 $\mathcal{L} = \frac{1}{2}(0.25 + 0 + 0.25) = 0.25$

损失从 3 降到了 0.25，说明 $w = 1.5$ 这条线明显更好！优化算法做的就是这件事：一点点调整参数，让损失不断往下走，直到几乎不能再降。

> **一句话总结**：数据告诉机器「事实长什么样」，模型给出「猜测的形式」，损失函数打出「这次猜得多差」的分数，优化算法负责「不断改参数把分数降到最低」。四者缺一不可。

例题

17 📝 例题2：监督 / 无监督 / 强化，用「学游泳」一个场景全分清

三大范式名字很像，最容易记混。我们不堆定义，用**同一件事——学游泳**，演三个版本，你立刻就能区分。

版本 A：监督学习（有老师、有标准答案）

教练站在池边，**每一个动作都告诉你对错**：「你这一下手臂角度错了，应该这样」。这里：

- 输入 x = 你做的动作
- 标签 y = 教练给的「标准动作」
- 你在学一个映射 $x \to y$

对应到机器：给一堆**带标签**的数据 (x, y)，比如「这张图是猫(标签=猫)、那张是狗(标签=狗)」，让模型学会输入到标签的映射。**关键词：每个样本都有正确答案**。

版本 B：无监督学习（没老师、没答案，自己找规律）

没有教练，泳池里有一群人在游，**没人告诉你谁游得对**。但你观察久了，自己发现：「哦，这帮人动作很像，是蛙泳；那帮人动作像另一类，是自由泳」。你把人**自动分成了几组**，但你并不知道这些组「叫什么名字」。

对应到机器：只给**无标签**数据 x（只有动作，没有「对/错」「叫什么」），让模型自己发现结构。比如把一堆没贴标签的客户**聚类**成几群。**关键词：没有答案，只能靠数据相互的相似性分组**。

版本 C：强化学习（没标准答案，只给分/反馈）

没人教你标准动作，但你每次尝试后，**水会给你反馈**：呛水了（负奖励 / pain），或者顺利往前游了一段（正奖励 / reward）。你不知道「标准答案」是什么，只能根据这些奖惩，**慢慢摸索出能拿高分的一整套动作**。

对应到机器：智能体(Agent)和环境(Environment)交互，做一个动作(action)，环境返回一个奖励(reward)和新状态(state)，目标是**最大化长期总奖励**。AlphaGo 就是这样：没人告诉它每一步「标准下法」，只有最后的胜负当奖励信号。**关键词：没有逐步答案，只有事后的分数/奖惩**。

对照速记表：

范式	老师给你什么	数据形式	生活类比
监督	标准答案	(x, y) 有标签	教练逐动作纠正
无监督	啥也不给，自己看	只有 x	自己观察把人分组
强化	只给分数/奖惩	状态-动作-奖励	呛水还是顺畅当反馈

> **一句话区分**：监督给「标准答案」，无监督给「一堆没答案的数据」，强化给「分数/奖励信号」。判断题目属于哪种，先问一句：**有没有标签 y？如果没有标签但有奖励反馈，就是强化**。

例题

18 📝 例题3：过拟合 = 背题不理解，用「驾考」看清三种状态

「过拟合」这个词很抽象，但课件里那个**驾考宝典**的例子特别传神。我们把它讲透，再配上具体数字看「训练好≠考得好」。

比喻：考驾照理论题

小王要考科目一（交规理论），有两种备考方式：

- **方式①（欠拟合 / 没学进去）**：小王根本没认真看，连基本规则都没记住。结果**平时刷题错一半，真考也错一半**——练习和考试都差。这就是**欠拟合**：模型太简单/没学到东西，训练误差和测试误差都高。
- **方式②（过拟合 / 背题套路）**：小王不理解交规，但发现了一堆「应试技巧」——「题里有『应』字选B」「三短一长选最长」「看图有小女孩选A」。靠这些套路，他把**练习题库刷到了 100% 全对**！但真考时题目换了说法、换了图，套路全失效，**实际只考了 60 分**。这就是**过拟合**：模型记住了训练数据的**表面细节和噪声**，而不是真正的规律，所以「平时很好，一上新题就崩」。
- **方式③（理想 / 真正理解）**：小李认真理解了每条交规背后的道理（「非正常行驶场合应选谨慎的选项」）。他**练习题对 95%，真考也对 93%**——训练和测试都好，且差距不大。这才是我们想要的**良好泛化**。

用数字一眼分辨（误差表）：

状态	训练误差	测试误差	诊断
欠拟合	高（如 45%）	高（如 48%）	太简单，没学到
过拟合	极低（如 1%）	高（如 35%）	背题，记住了噪声
刚好	低（如 6%）	低（如 8%）	真正学到规律

判断口诀：

1. 训练、测试**都差** → 欠拟合
2. 训练**很好**但测试**差很多**（两者差距大）→ 过拟合
3. 训练、测试**都好且接近** → 泛化良好

为什么模型越复杂越容易过拟合：复杂模型「记忆力」太强，连训练数据里的随机噪声（个别异常点）都给硬记下来了。所以随复杂度增加，训练误差一路降，但测试误差**先降后升呈 U 形**——最佳复杂度就在 U 形的谷底。

> **怎么治过拟合**（课件给的药方）：增加训练数据、正则化（惩罚复杂度）、数据划分 + 交叉验证、早停、引入先验知识。本质都是一句话：**别让模型死记，逼它学普遍规律**。

例题

19 📝 例题4：水果分拣——把一张图变成「点」，再画一条决策边界

课件把「自动水果分拣」叫机器学习的 Hello-World。我们用**具体特征数字**走一遍，重点搞懂两个常懵的概念：**特征向量化** 和 **决策边界**。

任务：给一筐水果，自动分成「苹果🍎」还是「橙子🍊」。

第 1 步：数据收集（打标签）

先人工标好几个样本（这一步必须有标签，所以这是**监督学习-分类**任务）：

水果	颜色(红=1,橙=2)	大小	标签
A	1	3	苹果
B	2	2	橙子
C	1	3.2	苹果
D	2	1.8	橙子

第 2 步：预处理

不同照片的背景、亮度、位置都不同，先**去背景、调光照**，否则机器会被无关因素干扰。

第 3 步：特征提取 / 向量化（核心！）

机器看不懂图片，只懂数字。我们把每个水果变成一个**特征向量**——就是高维空间里的**一个点**。比如水果 A 就是点 [1, 3]（颜色=1，大小=3）。于是四个水果变成坐标系里的四个点：

- 苹果 A、C 落在「颜色=1」这一侧
- 橙子 B、D 落在「颜色=2」这一侧

为什么要向量化：一旦每个样本都是空间里的点，「学习」就变成了一个**几何问题**——在点之间画一条线把两类分开。

第 4 步：训练 → 得到决策边界

训练就是在特征空间里找一条线，把苹果区和橙子区分开。这条分界线就叫**决策边界（decision boundary）**。比如这里可以画一条竖线「颜色 = 1.5」：左边判苹果，右边判橙子。

第 5 步：测试 / 评估（用没见过的新水果）

来了一个**新水果** E，特征 [1, 2.9]。它落在决策边界左边（颜色=1 < 1.5）→ 模型判定为**苹果**。

怎么评估好坏？用**分类错误率** = 被分错的样本比例。假设拿 10 个新水果测试，错了 1 个，错误率就是 $\frac{1}{10} = 10\%$。课件还提醒：除了准确率，有时也要考虑计算复杂度、易用性等。

> **串起来记**：数据收集 → 特征提取（图变成点）→ 训练（画决策边界）→ 验证 → 评估（用新数据测错误率）→ 预测。

> **最容易懵的两点**：(1)「向量化」= 把任意数据（图/文/声）变成一串数字，成为空间里的点；(2)「决策边界」= 分隔不同类别区域的那条线/面。

例题

第 2 讲 线性回归

Linear Regression

线性回归用线性函数 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$ 拟合实值目标，以均方误差为损失，可通过正规方程 $\mathbf{w}^\star = (X^\top X)^{-1} X^\top \mathbf{r}$ 解析求解或用梯度下降迭代优化，并借助 L_2 正则化缓解不可逆与过拟合问题。

1 回归问题与机器学习四要素

回归 (Regression) 属于**监督学习**，目标是预测**实值标签** (real-valued label)，与输出离散类别的分类问题相对。

直觉上，回归用来描述一个**因变量** (target) 与若干**自变量** (features) 之间的定量关系：测量总有噪声，我们希望从有限的带噪样本中“回归”出背后的真实规律。

任何机器学习方法都可拆成四个要素，本讲将围绕它们展开：

- **数据 (Experience)**：训练集 $\mathcal{D} = \{(\mathbf{x}^{(\ell)}, r^{(\ell)})\}_{\ell=1}^{N}$
- **模型 (Hypothesis)**：线性函数 $f(\mathbf{x})$
- **损失函数 (Objective)**：均方误差 MSE
- **优化算法 (Improve)**：梯度下降 / 正规方程

学习的本质即求解 $\theta^\star = \arg\min_\theta \mathcal{L}(\theta \mid \mathcal{D})$。回归的典型应用包括**预测**（如房价、营收）、**归因分析**（哪个自变量影响大）以及**控制**。

回归　监督学习　机器学习要素

2 线性回归模型结构

线性回归假设目标值是输入特征的**线性组合**：

$$y = f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

其中 $\mathbf{w} = (w_1, \ldots, w_d)^\top$ 是各特征的**权重 (slope)**，w_0 是**偏置 (bias / intercept)**。

几何上，对 d 维输入而言，该模型在 $(d+1)$ 维空间中是一张**超平面 (hyperplane)**；当 $d = 1$ 时退化为一条直线 $y = wx + w_0$。

常用技巧是引入常数特征 $x_0 \equiv 1$，把偏置吸收进权重向量，写成更紧凑的 $y = \mathbf{w}^\top \mathbf{x}$（此时 $\mathbf{w}$、$\mathbf{x}$ 都是 $(d+1)$ 维）。

模型工作流程：

- **训练过程**：根据数据上的误差，估计合适的参数 $\mathbf{w}$ 与 w_0。
- **预测过程**：对新输入 $\mathbf{x}$ 计算 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$ 得到回归值。

线性模型　超平面　权重　偏置

3 平方误差与均方误差损失 (MSE)

对单个样本 $\mathbf{x}$，模型输出预测值 y，真实目标值记为 r，定义**平方误差 (squared error)**：

$$\ell(\mathbf{w}, w_0 \mid \mathbf{x}, r) = (r - y)^2$$

对整个数据集 $\mathcal{D} = \{(\mathbf{x}^{(\ell)}, r^{(\ell)})\}_{\ell=1}^{N}$，损失函数取**均方误差 (Mean Squared Error, MSE)**：

$$\mathcal{L}(\mathbf{w}, w_0 \mid \mathcal{D}) = \frac{1}{2N} \sum_{\ell=1}^{N} \left(r^{(\ell)} - y^{(\ell)}\right)^2$$

要点解释：

- **平方**让正负误差都变正且重罚大误差，同时使损失**可导**，便于梯度优化。
- 系数 $\frac{1}{N}$ 做平均，使损失不随样本量变大而膨胀；额外的 $\frac{1}{2}$ 纯粹是为了求导后约掉平方带来的因子 2，不影响最优解位置。

最小二乘　MSE　损失函数

4 梯度下降法优化损失

当无法或不愿解析求解时，用**梯度下降 (Gradient Descent)** 迭代逼近最优。

优化目标 $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$，迭代公式：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

沿**负梯度方向**（损失下降最快的方向）走一步，η_t 为**学习率 (learning rate)**。

关键参数与直觉：

- 学习率太大会**震荡发散**，太小则**收敛过慢**。
- 梯度为零时停止，对凸的 MSE 而言即全局最优。

求梯度需用**链式法则 (chain rule)**：

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{1}{N} \sum_{\ell} \left(r^{(\ell)} - y^{(\ell)}\right) \frac{\partial y^{(\ell)}}{\partial w_j} = -\frac{1}{N} \sum_{\ell} \left(r^{(\ell)} - y^{(\ell)}\right) x_j^{(\ell)}$$

因为 $y^{(\ell)} = \mathbf{w}^\top \mathbf{x}^{(\ell)}$，所以 $\partial y^{(\ell)} / \partial w_j = x_j^{(\ell)}$。

代入迭代得分量更新规则：

$$w_j^{new} = w_j^{old} + \frac{\eta}{N} \sum_{\ell=1}^{N} \left(r^{(\ell)} - y^{(\ell)}\right) x_j^{(\ell)}$$

直觉：误差 $(r - y)$ 越大、对应特征 x_j 越大，则该方向修正越多。

梯度下降　学习率　链式法则　迭代优化

5 梯度下降训练算法 (伪代码)

用梯度下降训练线性回归（含偏置 w_0，输入约定 $x_0 \equiv 1$）的完整流程：

```
Input: D = {(x^(l), r^(l))}, l = 1..N
// 1. 初始化参数
for j = 0..d:
    w_j ← rand(-0.01, 0.01)
repeat:
    // 2. 清空梯度累加器
    for j = 0..d: Δw_j ← 0
    // 3. 遍历所有样本累加梯度
    for l = 1..N:
        y ← 0
        for j = 0..d: y ← y + w_j * x_j^(l)   // 前向预测
        for j = 0..d: Δw_j ← Δw_j + (r^(l) - y) * x_j^(l)
    // 4. 取平均
    for j = 0..d: Δw_j ← Δw_j / N
    // 5. 参数更新
    for j = 0..d: w_j ← w_j + η * Δw_j
until convergence
```

要点：

- 参数初始化为接近 0 的**小随机数**，避免对称性问题。
- 此处每轮遍历**全部样本**后再更新一次，属**批量梯度下降 (Batch GD)**。
- 收敛判据可用损失变化小于阈值或达到最大迭代次数。

伪代码 批量梯度下降 参数初始化

6 矩阵形式与最小二乘

把数据堆叠成矩阵能写出简洁的闭式解。设设计矩阵 $X \in \mathbb{R}^{N\times d}$（第 ℓ 行为样本 $\mathbf{x}^{(\ell)\top}$），目标向量 $\mathbf{r} = (r^{(1)}, \dots, r^{(N)})^\top$。

- **预测**：$\mathbf{y} = X\mathbf{w}$
- **损失**（向量化的 MSE）：

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{r}-\mathbf{y})^\top(\mathbf{r}-\mathbf{y}) = \frac{1}{2}(\mathbf{r}-X\mathbf{w})^\top(\mathbf{r}-X\mathbf{w})$$

这正是**最小二乘 (Least Squares)** 目标：最小化残差向量的平方范数 $\|\mathbf{r}-X\mathbf{w}\|_2^2$。

展开并对 $\mathbf{w}$ 求导：

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = -X^\top(\mathbf{r}-X\mathbf{w})$$

矩阵形式让我们既能一次性写出全样本梯度（用于优化），又能令梯度为零直接解析求解（下一知识点）。

矩阵形式 设计矩阵 最小二乘 向量化

7 正规方程 (闭式解)

令损失梯度为零即得**正规方程 (Normal Equation)**，无需迭代：

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = 0 \;\Rightarrow\; X^\top(\mathbf{r} - X\mathbf{w}) = 0 \;\Rightarrow\; X^\top X\mathbf{w} = X^\top \mathbf{r}$$

当 $X^\top X$ 可逆时，最优参数为：

$$\boxed{\mathbf{w}^\star = (X^\top X)^{-1} X^\top \mathbf{r}}$$

对比梯度下降：

- **优点**：一步到位、无需选学习率、无需迭代。
- **缺点**：需要求 $d \times d$ 矩阵的逆，复杂度约 $O(d^3)$，**高维特征**或**大数据**时代价高；且要求 $X^\top X$ 可逆。

经验法则：特征维度 d 较小（如几千以内）时用正规方程更省心，d 很大或样本极多时偏向（随机）梯度下降。

正规方程 闭式解 最小二乘 矩阵求逆

8 几何解释：投影矩阵

把最优解代回模型，预测向量为：

$$\mathbf{y} = X(X^\top X)^{-1} X^\top \mathbf{r} = H\mathbf{r}, \qquad H = X(X^\top X)^{-1} X^\top$$

H 称为**帽子矩阵 (hat matrix)** 或**投影矩阵**。

几何直觉：

- 预测 $\mathbf{y} = X\mathbf{w}$ 一定是 X 各列向量 $[\mathbf{x}_1, \ldots, \mathbf{x}_d]$ 张成的**列空间 (column space)** 中的一点。
- 最小化 $\|\mathbf{r} - X\mathbf{w}\|^2$ 等价于在该子空间中找一点 $\mathbf{y}$ 离目标 $\mathbf{r}$ **距离最近**。
- 距离最近的点恰是 $\mathbf{r}$ 在列空间上的**正交投影**，因此残差 $\mathbf{r} - \mathbf{y}$ 与列空间正交——这正是正规方程 $X^\top(\mathbf{r} - X\mathbf{w}) = 0$ 的含义。

H 的性质：对称 ($H^\top = H$) 且幂等 ($H^2 = H$)，是投影矩阵的标志。

几何解释 投影矩阵 列空间 正交

9 正则化：可逆性与过拟合

问题：当特征间存在**线性相关**（如 $x_2 = 3x_1$，即多重共线性）时，$X^\top X$ **奇异不可逆**，正规方程 $\mathbf{w}^\star = (X^\top X)^{-1} X^\top \mathbf{r}$ 无法直接计算；即便近似可逆，解也会数值不稳定、模型**过拟合**。

解决方法：正则化 (Regularization) —— 给损失加一个惩罚项（这里是 L_2，即岭回归 Ridge）：

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{r} - X\mathbf{w})^\top (\mathbf{r} - X\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

惩罚大权重，鼓励模型“简单”。

新梯度：

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = -X^\top (\mathbf{r} - X\mathbf{w}) + \lambda \mathbf{w}$$

令其为零：

$$X^\top \mathbf{r} = (X^\top X + \lambda I)\mathbf{w} \;\Rightarrow\; \boxed{\mathbf{w}^\star = (X^\top X + \lambda I)^{-1} X^\top \mathbf{r}}$$

关键作用：

- λI 给对角线“加料”，使 $X^\top X + \lambda I$ **一定正定可逆**，彻底解决奇异问题。
- λ 越大，权重被压得越小，模型越平滑；$\lambda = 0$ 退化为普通最小二乘。
- 给模型加惩罚等价于**给参数加先验 (adding data prior)**，相当于**数据增强 / 引入先验**。

正则化　岭回归　过拟合　多重共线性

10 📝 例题1：用正规方程手算房价直线

目标：给 4 个 (面积, 房价) 样本，手算最佳拟合直线 $\hat{r} = w_0 + w_1 x$ 的斜率和截距。

面积 x	房价 r
1	2
2	2
3	4
4	5

单特征下，最小二乘有一条好记的公式：

$$w_1 = \frac{\sum (x - \bar{x})(r - \bar{r})}{\sum (x - \bar{x})^2}, \qquad w_0 = \bar{r} - w_1 \bar{x}$$

第1步：算两个平均值（为什么？后面公式都围绕“偏离平均多少”来算）

- $\bar{x} = \dfrac{1+2+3+4}{4} = 2.5$
- $\bar{r} = \dfrac{2+2+4+5}{4} = 3.25$

第2步：算分子 $S_{xr} = \sum (x - \bar{x})(r - \bar{r})$（衡量 x 和 r 是否“一起变大”）

$x - \bar{x}$	$r - \bar{r}$	乘积
-1.5	-1.25	1.875
-0.5	-1.25	0.625
0.5	0.75	0.375
1.5	1.75	2.625

求和：$S_{xr} = 1.875 + 0.625 + 0.375 + 2.625 = 5.5$

第3步：算分母 $S_{xx} = \sum (x - \bar{x})^2$（衡量 x 自身的离散程度）

$$S_{xx} = (-1.5)^2 + (-0.5)^2 + (0.5)^2 + (1.5)^2 = 2.25 + 0.25 + 0.25 + 2.25 = 5.0$$

第4步：求斜率与截距

$$w_1 = \frac{5.5}{5.0} = 1.1, \qquad w_0 = 3.25 - 1.1 \times 2.5 = 3.25 - 2.75 = 0.5$$

所以最佳直线是 $\boxed{\hat{r} = 0.5 + 1.1\,x}$。

第5步：用它预测。面积 $x = 5$ 的房价预测为 $\hat{r} = 0.5 + 1.1 \times 5 = 6.0$。

> 小检验：把训练点代回算残差 $(r - \hat{r})$ 得 $0.4, -0.7, 0.2, 0.1$，有正有负、都很小，说明直线确实“穿过”了数据中间，拟合合理。

例题

11 📝 例题2：MSE 损失到底怎么算

目标：给定真实值和预测值，亲手算一遍均方误差，搞清楚课件里 $\frac{1}{2N}$ 的含义。

假设有 3 个样本，真实目标 r 与模型预测 y 如下：

样本	真实 r	预测 y	误差 $r-y$
1	3.0	2.5	0.5
2	5.0	5.5	−0.5
3	7.0	6.0	1.0

第1步：每个样本的平方误差 $(r-y)^2$（为什么平方？让正负误差都变正，且大误差被重罚）

- 样本1：$0.5^2 = 0.25$
- 样本2：$(-0.5)^2 = 0.25$
- 样本3：$1.0^2 = 1.00$

第2步：求和

$$\sum_{\ell=1}^{3}(r^{(\ell)}-y^{(\ell)})^2 = 0.25 + 0.25 + 1.00 = 1.5$$

第3步：套课件的损失公式 $\mathcal{L} = \frac{1}{2N}\sum(r-y)^2$，这里 $N=3$：

$$\mathcal{L} = \frac{1}{2\times 3}\times 1.5 = \frac{1.5}{6} = 0.25$$

为什么是 $\frac{1}{2N}$ 而不是 $\frac{1}{N}$？

- $\frac{1}{N}$ 是取**平均**，让损失不随样本数变多而膨胀（普通 MSE $= \frac{1.5}{3} = 0.5$）。
- 多出来的 $\frac{1}{2}$ 只是为了**求导方便**：对 $(r-y)^2$ 求导会冒出因子 2，乘上 $\frac{1}{2}$ 正好抵消，让梯度公式更干净。它**不改变**最优解的位置。

> 记住：损失越小说明预测越贴近真实。这里 $\mathcal{L}=0.25$ 只是一个数值，真正有用的是在训练中“想办法把它降下去”。

例题

12 📝 例题3：手推一步梯度下降的参数更新

目标：给定学习率和初始参数，代入数字算出走一步之后的新参数，并验证损失确实下降了。

模型 $y = w_0 + w_1 x$（约定常数特征 $x_0 \equiv 1$），3 个样本（真实规律是 $r = 2x + 1$）：

x	r
1	3
2	5
3	7

设定：初值 $w_0 = 0$, $w_1 = 0$，学习率 $\eta = 0.1$，$N = 3$。

更新规则（课件版，注意是 $+$ 号）：

$$w_j^{new} = w_j^{old} + \frac{\eta}{N} \sum_{\ell=1}^{N} \left(r^{(\ell)} - y^{(\ell)}\right) x_j^{(\ell)}$$

第1步：用当前参数做预测。因为 $w_0 = w_1 = 0$，所以全部 $y = 0$。

第2步：算每个样本的误差 $r - y$

- $3 - 0 = 3, \quad 5 - 0 = 5, \quad 7 - 0 = 7$

第3步：算 w_0 的更新量（对应特征 $x_0 \equiv 1$，所以直接对误差求和）

$$\frac{\eta}{N} \sum (r - y) \cdot 1 = \frac{0.1}{3} \times (3 + 5 + 7) = \frac{0.1}{3} \times 15 = 0.5$$

所以 $w_0^{new} = 0 + 0.5 = 0.5$。

第4步：算 w_1 的更新量（对应特征 x，要乘上各自的 x）

$$\sum (r - y) \cdot x = 3 \cdot 1 + 5 \cdot 2 + 7 \cdot 3 = 3 + 10 + 21 = 34$$

$$\frac{\eta}{N} \times 34 = \frac{0.1}{3} \times 34 \approx 1.133$$

所以 $w_1^{new} = 0 + 1.133 = 1.133$。

第5步：验证损失真的下降了

- 更新前预测全是 0，损失 $\mathcal{L} = \frac{1}{2 \times 3}(3^2 + 5^2 + 7^2) = \frac{83}{6} \approx 13.83$。
- 更新后 $y = 0.5 + 1.133x$，预测约 $1.63, 2.77, 3.90$，损失降到约 2.74。

> 直觉：误差 $(r - y)$ 越大、对应特征 x_j 越大，这个方向就被推得越猛。这里 w_1 走得比 w_0 远，正是因为大 x 把大误差“放大”了。真实参数是 $(1, 2)$，再迭代很多次后 (w_0, w_1) 会越来越靠近它。

例题

13 📝 例题4：正则化 λ 如何“压小”系数

目标：用同一组数据，改变正则化强度 λ，亲眼看系数被压缩的过程，理解岭回归。

为了能手算，用过原点的单特征模型 $y = w\,x$。数据 $x = [1, 2, 3]$，$r = [2, 4, 6]$（完美满足 $r = 2x$，所以不加正则时 w 应该正好是 2）。

岭回归（L_2 正则）的闭式解，单特征版就是：

$$w^\star = \frac{\sum x\,r}{\sum x^2 + \lambda} = \frac{X^\top \mathbf{r}}{X^\top X + \lambda}$$

第1步：先算两个不变的量

- 分子 $X^\top \mathbf{r} = \sum x\,r = 1\cdot 2 + 2\cdot 4 + 3\cdot 6 = 2 + 8 + 18 = 28$
- 分母里的 $X^\top X = \sum x^2 = 1 + 4 + 9 = 14$

第2步：代入不同的 λ（λ 加在分母上，分母变大 → w 变小）

λ	$w^\star = \frac{28}{14+\lambda}$	结果
0	28/14	2.00（=普通最小二乘）
1	28/15	≈ 1.87
7	28/21	≈ 1.33
14	28/28	1.00

第3步：读懂规律

- $\lambda = 0$：退化为普通最小二乘，$w = 2$ 完美拟合。
- λ 越大：分母越大，w 被越压越小（$2.0 \to 1.87 \to 1.33 \to 1.0$），模型变“平缓”。

为什么要这么做？ 当特征线性相关（如 $x_2 = 3x_1$）时 $X^\top X$ 不可逆，正规方程算不出来；多特征版的解 $\mathbf{w}^\star = (X^\top X + \lambda I)^{-1} X^\top \mathbf{r}$ 在对角线上加了 λI，**保证可逆**，同时压制过大权重、缓解过拟合。

> 一句话记忆：λ 像一个“惜字如金”的旋钮，越大越逼着模型把系数往小了写。

例题

第 3 讲 决策树

Decision Tree

决策树是一种通过一系列 if-then 决策规则对数据进行分类的模型，采用「分而治之」自顶向下递归构建。其核心是选择划分属性以最大化标签纯度，常用指标包括信息增益 $\mathrm{Gain}(D, A) = \mathrm{Ent}(D) - \mathrm{Ent}(D \mid A)$、信息增益率和基尼指数，分别对应 ID3、C4.5、CART 三种算法。决策树易过拟合，可用随机森林等集成方法降低方差。

1 从线性到分段线性：决策树的动机

回归预测实值标签，分类预测离散类别标签。最简单的分类是用一条线性决策边界把特征空间一分为二，但现实数据常常：

- **非线性可分**：无法用一条直线分开（can not be linearly separated）。
- **存在非数值属性**：如「male/female」「红/绿」等离散类别特征。

决策树的思想是**分段线性（piecewise linear）**：把输入空间切分成若干互不相交的区域，每个区域内局部线性可分，再组合多个简单模型。每一次切分都对应一条与坐标轴平行的划分，最终用一组 if-then 规则逼近复杂边界。

> 直觉：与其找一条复杂曲线，不如反复用简单的「阈值判断」把问题拆小，直到每个子问题足够简单。

分类　决策边界　动机

2 决策树的基本思想与模型结构

决策树的核心思想是**分而治之（Divide and Conquer）**：不断划分数据属性、创建 if-then 规则，把大问题拆成小问题。

一棵决策树包含三种节点：

1. **根节点（root node）**：包含整个数据集。
2. **中间节点（internal node）**：每个非叶节点包含一个**属性测试条件**，根据样本在该属性上的取值把数据划分到不同分支。
3. **叶子节点（leaf / terminal node）**：每个叶子节点对应一个**类别标签**。

预测时，从根节点出发，按样本的属性取值沿分支下行，到达某个叶子节点即输出该叶子的类别。例如「width>6.5cm? → height>9.5cm? → orange/lemon」。

> 决策树本质上把特征空间划分成一组与坐标轴平行的矩形区域，每个区域贴一个类别标签。

模型结构　节点　if-then

3 决策树的训练：自顶向下递归构建

训练（构建）一棵决策树采用**自顶向下的分治学习**，步骤如下：

1. 构造一个根节点，包含整个训练数据集。
2. 选择一个**最合适的划分属性**。
3. 根据所选属性的不同取值，把当前节点的样本划分成若干子集。
4. 对每个子集创建一个孩子节点，并把子集数据传给该孩子。
5. **递归重复 2~4**，直到满足停止条件。

关键观察：**按不同的划分顺序，同一份数据可以构建出多种不同的树**。因此核心问题是：

> 如何高效地构建一棵「好」的决策树？

答案是：每次选择划分属性 X 时，尽可能**最大化标签 Y 的纯度**——理想情况下，叶节点对应的样本全部属于同一类别（节点纯净）。如何量化「纯度提升」，就引出了信息增益、基尼指数等指标，每种指标对应一种具体算法。

训练 递归 分治

4 信息熵（Entropy）：度量不纯度

信息熵刻画一个数据集合的**不纯净度（impurity）**。对数据集 D，设其中第 k 类样本所占比例为 p_k（共 K 类），熵定义为：

$$\mathrm{Ent}(D) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

含义与性质：

- 熵越**小**，数据越**纯**（不纯度低）；熵越大，类别越混杂。
- 当所有样本属于同一类（$p = 1$）时，$\mathrm{Ent}(D) = 0$，纯度最高。
- 当各类均匀分布时熵最大；两类时若 $p = 0.5$，则 $\mathrm{Ent} = 1$ bit（最不纯）。
- 约定 $0 \log_2 0 = 0$。

实际计算用类别计数：$p_k = \frac{|C_k|}{|D|}$，其中 $|C_k|$ 是 D 中第 k 类的样本数。

> 例：3 个柠檬、2 个橙子，$\mathrm{Ent}(D) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \approx 0.97$。

信息熵 不纯度 公式

5 条件熵与信息增益（Information Gain）—— ID3

用属性 A 划分后，整体不纯度用**条件熵**度量。设 A 有 n 个取值，把 D 划分为 $D_1, \dots, D_n$：

$$\mathrm{Ent}(D \mid A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \mathrm{Ent}(D_i)$$

即各子集熵按样本占比加权平均。**信息增益**为划分前后熵的减少量：

$$\mathrm{Gain}(D, A) = \mathrm{Ent}(D) - \mathrm{Ent}(D \mid A)$$

ID3 算法：对每个候选属性计算信息增益，选择信息增益**最大**的属性作为划分属性——即划分后纯度提升最多。

计算流程：

1. 算根节点熵 $\mathrm{Ent}(D)$。
2. 对每个属性，按取值划分子集、算各子集熵、加权得条件熵、再求增益。
3. 选增益最大的属性扩展分支，递归。

> 课件例子：$\mathrm{Ent}(D) = 0.97$，按 height 划分条件熵 $= \frac{2}{5} \times 0 + \frac{3}{5} \times 0.92 = 0.55$，$\mathrm{Gain} = 0.97 - 0.55 = 0.42$；按 width 划分 $\mathrm{Gain} = 0.97 - 0.95 = 0.02$。height 增益更大，故先选 height。

信息增益　ID3　条件熵

6 三种算法对比：ID3 / C4.5 / CART

按划分准则（纯度度量指标）不同，常见决策树算法有：

算法	划分准则
ID3	信息增益（Information Gain）最大
C4.5	信息增益率（Gain Ratio）
CART	基尼指数（Gini Index）

要点：

- 三者的本质都是「贪心地选当前最优的划分属性」，区别在于**纯度度量指标不同**——每一种指标对应一种决策树构造算法。
- 本讲重点展开 **ID3（基于信息增益）** 的完整计算流程；C4.5、CART 仅作为基于不同准则的算法了解。

对比　ID3　C4.5　CART

7 停止划分条件

递归构建决策树时，满足以下任一条件即停止划分，把当前节点设为叶节点：

- **纯度足够**：叶子节点包含的样本全部属于同一类别。
- **数据量过小**：叶子节点包含的样本数小于一个预设阈值。
- **没有属性可分（ID3）**：所有属性已用完，无法再继续划分。

停止后，叶节点的类别通常取该节点样本中的多数类。

> 决策树**易过拟合**：它能以任意精度拟合训练集，某些噪声样本也会被树「记住」，导致泛化能力下降。

停止条件 过拟合

8 决策树的优缺点

优点：

- **可解释性强**：树结构直观，便于可视化分析，结果是一组人类可读的 if-then 规则。
- **数据预处理要求低**：对特征缩放、归一化不敏感，能混合处理数值与类别属性。
- 能轻易处理**非线性决策边界**。
- 数据驱动，可以**以任意精度拟合训练集**。

缺点：

- **容易过拟合**：某些噪声样本总能被树涵盖并学习，导致树过深、泛化差。

过拟合（高方差）正是**集成方法（如随机森林）**的动机。

优缺点 过拟合 可解释性

9 随机森林（Random Forest）

随机森林通过**集成多棵决策树**降低单棵树的方差（「三个臭皮匠，顶一个诸葛亮」）。构建流程：

1. **Bootstrap 采样**：从原始数据集**有放回**地抽取 N 个样本，构成一个随机训练集。
2. 用该随机数据集训练一棵决策树；对每个节点：

a. **随机抽取 d 个属性**（特征子集）；

b. 仅在这 d 个属性中按信息增益等准则选择最优划分。

1. 重复步骤 1~2 共 k 次，得到 k 棵树。
2. **聚合预测**：分类用**多数投票（majority voting）**得到最终结果（回归则取平均）。

两重随机性（样本随机 + 特征随机）使各树差异化、相关性低，集成后更稳健。

优点：

- 通常比单棵决策树更**准确**。
- 训练快、**易并行**（树之间相互独立）。
- 能处理**高维数据**且无需特征选择（特征子集随机选）。
- 能处理缺失属性；可评估**特征重要性**；能检测特征间相互作用；对**不平衡数据**能平衡误差。

随机森林 集成学习 Bagging

10 📝 例题1：手算根节点信息熵 Ent(D)

我们用经典的「**14天是否打网球**」数据集（基础薄弱的同学就盯着这一张表，全程不换数据）。一共 14 条样本，标签 `Play`（是否打球）取值 Yes / No。

天	Outlook(天气)	Humidity(湿度)	Wind(风)	Play(打球?)
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No

第1步：数标签。 把 14 条样本的 Play 列数一遍：

- Yes（打球）：第 3,4,5,7,9,10,11,12,13 天，共 **9** 个；
- No（不打）：第 1,2,6,8,14 天，共 **5** 个。

所以根节点 D 一共 $9+5=14$ 个样本。

第2步：算每一类的比例 p_k。 为什么要先算比例？因为熵公式吃的是「比例」，不是个数。

$$p_{\text{Yes}}=\frac{9}{14}\approx 0.643,\qquad p_{\text{No}}=\frac{5}{14}\approx 0.357$$

第3步：套信息熵公式，把每个 log 值都写出来。

$$\text{Ent}(D)=-\sum_{k} p_k\log_2 p_k=-p_{\text{Yes}}\log_2 p_{\text{Yes}}-p_{\text{No}}\log_2 p_{\text{No}}$$

逐项计算（$\log_2 x$ 怎么来的：$\log_2 x=\ln x/\ln 2$，记不住就背几个常用值）：

- $\log_2 0.643\approx -0.637$，于是 $-\frac{9}{14}\log_2\frac{9}{14}\approx -0.643\times(-0.637)=0.410$；
- $\log_2 0.357\approx -1.485$，于是 $-\frac{5}{14}\log_2\frac{5}{14}\approx -0.357\times(-1.485)=0.530$。

第4步：相加得结果。

$$\text{Ent}(D)=0.410+0.530=\boxed{0.940\text{ bit}}$$

> 直觉检查：熵介于 0~1 之间，越接近 1 越混乱。这里 9:5 比较接近一半一半，所以熵 0.94 很高（很不纯），说明根节点确实需要继续划分。如果是 14:0（全是 Yes），熵就会是 0。

例题

11 📝 例题2：按 Outlook 划分，手算条件熵与信息增益 Gain

继续用例题1的网球数据（$\text{Ent}(D) = 0.940$，9 Yes / 5 No）。现在试着用 **Outlook（天气）** 这个特征来划分，看能把熵降多少。

第1步：按 Outlook 取值把 14 条样本分成 3 堆。 Outlook 有三个取值：Sunny / Overcast / Rain。

子集	包含的天	Yes	No	小计
D_{Sunny}	1,2,8,9,11	2 (第9,11)	3 (第1,2,8)	5
D_{Overcast}	3,7,12,13	4	0	4
D_{Rain}	4,5,6,10,14	3 (第4,5,10)	2 (第6,14)	5

第2步：分别算三个子集自己的熵。 注意每个子集都要重新数自己的 Yes/No 比例。

Sunny 子集（2 Yes, 3 No）：

$$\text{Ent}(D_{\text{Sunny}}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = -0.4 \times (-1.322) - 0.6 \times (-0.737) = 0.529 + 0.442 = 0.971$$

Overcast 子集（4 Yes, 0 No）：全是 Yes，**纯净**！按约定 $0\log_2 0 = 0$，所以

$$\text{Ent}(D_{\text{Overcast}}) = -1 \times \log_2 1 - 0 = 0$$

Rain 子集（3 Yes, 2 No）：和 Sunny 的比例对称（3:2 vs 2:3），熵相同：

$$\text{Ent}(D_{\text{Rain}}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.442 + 0.529 = 0.971$$

第3步：加权平均得条件熵。 为什么要加权？因为大子集更重要，权重就是「子集样本数 / 总样本数」。

$$\begin{aligned}\text{Ent}(D \mid \text{Outlook}) &= \frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.971 \\ &= 0.347 + 0 + 0.347 = 0.694\end{aligned}$$

第4步：信息增益 = 划分前熵 − 划分后条件熵。

$$\text{Gain}(D, \text{Outlook}) = \text{Ent}(D) - \text{Ent}(D \mid \text{Outlook}) = 0.940 - 0.694 = \boxed{0.246}$$

> 含义：用 Outlook 一刀切下去，整体不纯度从 0.940 降到 0.694，纯度提升了 0.246 bit。其中 Overcast 子集直接变纯净（熵=0），是这次划分的最大功臣。

例题

12 📝 例题3：对比 Humidity 与 Wind，谁的增益大就选谁

光算一个特征不够，ID3 要**对每个特征都算一遍增益，挑增益最大的当划分属性**。我们再算 Humidity 和 Wind，和例题2的 Outlook(0.246) 一起 PK。仍然用同一份网球数据，$\text{Ent}(D) = 0.940$。

A. 按 Humidity（湿度）划分 —— 取值 High / Normal。

子集	天	Yes	No	小计
D_{High}	1,2,3,4,8,12,14	3 (第3,4,12)	4 (第1,2,8,14)	7
D_{Normal}	5,6,7,9,10,11,13	6	1 (第6)	7

- $\text{Ent}(D_{\text{High}}) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.524 + 0.461 = 0.985$
- $\text{Ent}(D_{\text{Normal}}) = -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.191 + 0.401 = 0.592$

条件熵（两个子集各 7 个，权重都是 7/14）：

$$\text{Ent}(D \mid \text{Humidity}) = \frac{7}{14} \times 0.985 + \frac{7}{14} \times 0.592 = 0.492 + 0.296 = 0.788$$

$$\text{Gain}(D, \text{Humidity}) = 0.940 - 0.788 = \boxed{0.152}$$

B. 按 Wind（风）划分 —— 取值 Weak / Strong。

子集	天	Yes	No	小计
D_{Weak}	1,3,4,5,8,9,10,13	6	2 (第1,8)	8
D_{Strong}	2,6,7,11,12,14	3	3 (第2,6,14)	6

- $\text{Ent}(D_{\text{Weak}}) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = 0.311 + 0.5 = 0.811$
- $\text{Ent}(D_{\text{Strong}}) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 0.5 + 0.5 = 1.0$（3:3 完全对半，熵最大=1）

条件熵（权重 8/14 与 6/14）：

$$\text{Ent}(D \mid \text{Wind}) = \frac{8}{14} \times 0.811 + \frac{6}{14} \times 1.0 = 0.463 + 0.429 = 0.892$$

$$\text{Gain}(D, \text{Wind}) = 0.940 - 0.892 = \boxed{0.048}$$

第3步：把三个增益排个序，选最大的。

特征	信息增益 Gain
Outlook	**0.246** ✅
Humidity	0.152
Wind	0.048

> 结论：$\text{Gain}(\text{Outlook}) = 0.246$ 最大，所以**根节点应该选 Outlook 来划分**。这正是 ID3 的核心：每一步都贪心地选「让纯度提升最多」的特征。划分完后，对 Sunny、Rain 这两个还不纯的子集递归重复同样的步骤即可（Overcast 已纯净，直接做叶节点标 Yes）。

例题

第 4 讲　贝叶斯分类

Bayes Classifier

本讲从概率视角看待分类：利用贝叶斯定理 $P(c \mid x) = \frac{P(x|c)P(c)}{P(x)}$ 把分类转化为后验概率最大化(MAP)问题，并通过朴素贝叶斯的条件独立假设 $P(x_1, \ldots, x_d \mid c) \approx \prod_{i=1}^{d} P(x_i \mid c)$ 大幅减少参数，再用极大似然估计参数、用拉普拉斯平滑处理零计数，最终应用于文本/垃圾邮件分类。

1 分类的概率视角与联合分布

前几讲的决策树通过递归切分特征空间来分类。但当数据呈现 $Y_1 \sim N(\mu_1, \Sigma_1)$、$Y_2 \sim N(\mu_2, \Sigma_2)$ 这类重叠的概率分布时，**无法用线性或递归方式干净地切开**，却存在清晰的概率分布与依赖关系。

- 概率视角认为机器学习是在**推断数据特征之间的概率关系**，模型即联合分布 $p(x, y)$。
- **联合分布(joint distribution)足以回答任意涉及这些变量的概率推断问题**。例如已知联合概率表，可算条件概率：

$$P(\text{Browsing} = \text{GitHub} \mid \text{Major} = \text{CS}) = \frac{P(\text{GitHub}, \text{CS})}{P(\text{CS})}$$

- 这正是贝叶斯/条件概率思想的雏形：用频率在训练集上估计概率，再做推断。

> 直觉：分类问题的本质是问“给定特征 x，它属于哪一类 c 的概率最大”。

概率视角　联合分布　条件概率

2 贝叶斯定理：先验、似然、证据、后验

贝叶斯分类的核心公式：

$$P(c \mid x) = \frac{p(x \mid c)\, P(c)}{p(x)}, \quad i = 1, 2, \ldots$$

四个量的含义务必分清：

- **先验概率(prior)** $P(c)$：在观测 x 之前，类别 c 出现的概率。可由训练集频率直接估计，$\hat{P}(c) = \frac{\#(C=c)}{N}$。
- **似然(likelihood)** $p(x \mid c)$：又称类条件概率(class-conditional density)，即在类别 c 下观测到特征 x 的概率分布。
- **证据(evidence)** $p(x)$：数据 x 被观测到的总概率，对二分类 $p(x) = P(c_1)p(x \mid c_1) + P(c_2)p(x \mid c_2)$。它对所有类别相同，是**归一化常数**。
- **后验概率(posterior)** $P(c \mid x)$：在观测 x 之后，属于类别 c 的概率，是我们真正想要的判别依据。

> 记忆口诀：后验 $\propto$ 似然 $\times$ 先验。证据只起归一化作用，比较大小时可忽略。

贝叶斯定理　先验　似然　后验

3 贝叶斯决策：MAP 判别规则

有了后验概率就可以做决策。**最大后验(MAP, Maximum A Posteriori)** 规则：选择后验概率最大的类别。

$$\hat{c} = \arg\max_i \ P(c_i \mid x)$$

由于 $p(x)$ 对各类相同，等价地：

$$\hat{c} = \arg\max_c \ p(x \mid c)P(c)$$

- 二分类判别规则：若 $p(x \mid c_1)P(c_1) > p(x \mid c_2)P(c_2)$ 则判为 c_1，否则 c_2。
- **没有特征信息时**(只看先验)：例如收件箱 85 封正常、15 封垃圾，$P(C_1) = 85\% > P(C_2) = 15\%$，则一律预测“正常”，连邮件都不用看。
- **有特征信息时**：例如统计“金钱相关词”数量 x，在 $x = 6$ 处算得 $p(C_1 \mid x) = 0.92$、$p(C_2 \mid x) = 0.08$，于是判为 C_1。

> MAP 与极大似然(ML)的区别：ML 只看似然 $\arg\max p(x \mid c)$，MAP 额外乘上先验 $P(c)$。当先验均匀时两者一致。

贝叶斯决策 MAP 判别规则

4 多维特征与参数爆炸问题

现实中特征 x 往往是多维的，如垃圾邮件的 x_1=地址、x_2=主题、x_3=长度⋯。此时

$$c_{MAP} = \arg\max_{c \in C} P(c \mid x) = \arg\max_{c \in C} \frac{P(x \mid c)P(c)}{P(x)} = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_d \mid c)P(c)$$

问题在于直接估计联合类条件概率 $P(x_1, \ldots, x_d \mid c)$：

- 需要为**每一种特征取值组合**统计一个概率，联合概率表的参数个数随维度 d **指数级增长**。
- 若每个特征有 k 种取值，则每个类需要约 $k^d - 1$ 个参数，**参数太多、数据稀疏，几乎无法可靠估计**。

> 这正是引入“朴素(naïve)”条件独立假设的动机：用简化假设换取可计算性。

多维特征 参数爆炸 联合概率

5 朴素贝叶斯的条件独立假设

朴素贝叶斯(Naïve Bayes) 的关键假设：给定类别 c，各特征 x_j 之间**条件独立**。

由链式法则本应是：

$$P(x_1, \ldots, x_d \mid c) = P(x_1 \mid c)P(x_2 \mid x_1, c) \cdots P(x_d \mid x_1, \ldots, x_{d-1}, c)$$

条件独立假设把它简化为连乘：

$$P(x_1, \ldots, x_d \mid c) \approx \prod_{i=1}^{d} P(x_i \mid c)$$

于是分类器为：

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

- “朴素”二字即指这个**通常并不严格成立**的独立假设。
- 好处：参数从指数级降到线性级，每个类只需估计 $P(c)$ 与 d 个一维条件概率 $P(x_i \mid c)$。
- 它把复杂的联合分布拆成若干简单边缘分布的乘积，是用偏差换方差的典型做法。

朴素贝叶斯　条件独立　连乘

6 训练：极大似然估计参数

训练朴素贝叶斯 = **估计参数** $P(c)$ 以及每个 $P(x_i \mid c)$。采用频率近似的**极大似然估计(MLE)**：

- 先验：

$$\hat{P}(c) = \frac{\#(C = c)}{N} = \frac{n_c}{N}$$

其中 n_c 是类别为 c 的样本数，N 是样本总数。

- 类条件概率(离散特征)：

$$\hat{P}(x_i \mid c) = \frac{\#(x_i, c)}{\sum_{x \in |x|} \#(x, c)} = \frac{\text{类别为 } c \text{ 且取值为 } x_i \text{ 的样本数}}{\text{类别为 } c \text{ 的样本数}}$$

- 训练**只需对训练集扫描一遍**计数即可，非常快。
- 这本质上是用样本频率作为概率的极大似然估计：在多项分布假设下，频率正是 MLE 解。

> 文本分类中常用词频版本：$P(w_k \mid c_j) = \frac{n_k}{n}$，$n_k$ 为词 w_k 在该类文本中出现次数，n 为该类文本总词数。

训练　极大似然估计　参数估计

7 零计数问题与拉普拉斯平滑

零计数(Zero Counts)问题：若训练集中类别 c 的样本里**从未出现过**某属性取值 x_i，则 $\hat{P}(x_i \mid c) = 0$。

由于朴素贝叶斯是连乘，$\hat{P}(c)\prod_i \hat{P}(x_i \mid c) = 0$，**只要有一项为 0，整体后验就被强行归零**——即使其它所有特征都强烈支持类别 c，也永远不可能被判为 c。这显然不合理。

拉普拉斯平滑(Laplace Smoothing)：给每个属性取值加一个虚拟计数。

$$\hat{P}(x_i \mid c) = \frac{\#(x_i, c) + 1}{\sum_{x \in |x|} \#(x, c) + |x|}$$

其中 $|x|$ 是该属性不同取值的个数(文本中即词表大小 $|Vocab|$)。

- 文本分类一般写作加法平滑参数 α：$P(w_k \mid c_j) = \frac{n_k + \alpha}{n + \alpha|Vocab|}$，$\alpha = 1$ 即拉普拉斯平滑。
- 作用：消除零概率、避免一票否决，同时保证 $\sum_i \hat{P}(x_i \mid c) = 1$ 仍成立。

> 直觉：平滑相当于给每个取值一点先验信念“它至少可能出现一次”。

零计数　拉普拉斯平滑　平滑

8 朴素贝叶斯算法流程

完整算法分训练与预测两段：

训练阶段：

```
for each class c do
    p̂(c) ← estimate P(c)                // 频率
    for each attribute value x_i do
        p̂(x_i | c) ← estimate P(x_i|c)  // 计数 + 平滑
    end
end
```

预测阶段：

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

- 训练只需一**遍扫描计数**，是典型的 **eager learning(急切学习)**，模型一旦建好预测极快。
- 模型可在**增删训练样本**时只调整计数，易于维护更新。

算法流程　急切学习　增量更新

9 朴素贝叶斯的典型应用场景

PPT 总结了朴素贝叶斯三类典型应用：

- **文本分类(Text Classification)**：如垃圾邮件过滤、情感分析。朴素贝叶斯**最常用于文本分类**——由于在多分类问题上表现更好、又有独立性规则简化计算，相比其它算法往往有更高的成功率。
- **实时预测(Real-Time Prediction)**：朴素贝叶斯是**急切学习(eager learning)** 分类器，模型训练好后预测**极快**，适合需要实时响应的场景。
- **多分类预测(Multi-Class Prediction)**：可直接给出目标变量**多个类别各自的概率**，天然支持多分类。

> 这三点共同根源于它"训练快、预测快、天然多分类"的特性，下面的文本分类正是其最经典的落地。

应用场景　文本分类　实时预测　多分类

10 文本分类与垃圾邮件过滤实例

朴素贝叶斯最经典的应用是**文本分类**(垃圾邮件、情感分析)。把文档看作词袋 $d=\{w_1,\dots,w_{|d|}\}$，则

$$score(c)=P(c)\prod_{k=1}^{|d|}P(w_k\mid c),\quad \hat{c}=\arg\max_c score(c)$$

垃圾邮件示例：训练得正常邮件中 $P(\text{Dear}\mid N)=0.44$、$P(\text{Friend}\mid N)=0.31$；垃圾邮件中 $P(\text{Dear}\mid S)=0.29$、$P(\text{Friend}\mid S)=0.14$。先验 $P(N)=\frac{6}{6+3}=0.67$、$P(S)=\frac{3}{9}=0.33$。

对新邮件“dear friend”：

- $P(N\mid \text{Dear,Friend})\propto 0.67\times 0.44\times 0.31\approx 0.09$
- $P(S\mid \text{Dear,Friend})\propto 0.33\times 0.29\times 0.14\approx 0.03$

因 $0.09>0.03$，判为 **Normal(正常邮件)**。

> 注意 $P(\text{Lunch}\mid S)=0$ 这类零概率会造成一票否决，正是需要平滑的地方。

文本分类　垃圾邮件　词袋模型

11 优点、缺点与贝叶斯网络

优点：

- **快**：训练只需一遍扫描，测试也快；急切学习器。
- **性能有竞争力**：当独立假设近似成立时表现很好，尤其擅长**多分类**问题与文本分类。
- **易更新维护**：增删样本只改计数。

缺点：

- **条件独立假设常被违反**——但实践中“出奇地好用”。原因在于：分类只需 $\arg\max$ **排序正确**即可，并不要求后验概率本身准确：

$$\arg\max_c P(c)\prod_i P(x_i \mid c) = \arg\max_c P(c)P(x_1,\ldots,x_d \mid c)$$

即使估计的后验有偏，只要最大者对应的类不变，分类结果就正确。

- **欠拟合(underfitting)**：模型复杂度固定且偏低。

放松独立假设：**贝叶斯网络(Bayesian/belief network)** 用有向图显式建模特征之间的依赖关系，可在朴素贝叶斯与完整联合分布之间折中。

优缺点　贝叶斯网络　欠拟合

12 📝 例题1：疾病检测——用贝叶斯定理算"阳性但其实没病"

题目：某种疾病在人群中的患病率为 1%。有一种检测试剂：

- 真有病的人，检测呈阳性的概率(灵敏度)是 99%；
- 没病的人，检测也可能误报阳性(假阳性率)，概率是 5%。

现在小明体检结果是**阳性**，问：他**真的患病**的概率 $P(\text{患病} \mid \text{阳性})$ 是多少？很多人凭直觉会说"99% 吧"，我们来算一下真实答案。

第 1 步：把已知信息翻译成概率符号（先认清谁是先验、谁是似然）

- 先验 $P(\text{病}) = 0.01$，$P(\text{无病}) = 0.99$ —— 还没看检测结果时的患病概率。
- 似然 $P(\text{阳} \mid \text{病}) = 0.99$ —— 有病时测出阳性。
- 似然 $P(\text{阳} \mid \text{无病}) = 0.05$ —— 没病却误报阳性。

> 为什么要分清：贝叶斯定理就是"由 $P(\text{阳} \mid \text{病})$ 反推 $P(\text{病} \mid \text{阳})$"，方向千万别弄反。

第 2 步：写出贝叶斯定理

$$P(\text{病} \mid \text{阳}) = \frac{P(\text{阳} \mid \text{病})\,P(\text{病})}{P(\text{阳})} = \frac{\text{似然} \times \text{先验}}{\text{证据}}$$

第 3 步：先算分子（似然 × 先验）

$$P(\text{阳} \mid \text{病})\,P(\text{病}) = 0.99 \times 0.01 = 0.0099$$

第 4 步：算分母——证据 $P(\text{阳})$（用全概率公式把两条路径加起来）

一个人测出阳性，要么"有病且测对"，要么"没病但误报"：

$$P(\text{阳}) = \underbrace{P(\text{阳} \mid \text{病})P(\text{病})}_{0.99\times0.01=0.0099} + \underbrace{P(\text{阳} \mid \text{无病})P(\text{无病})}_{0.05\times0.99=0.0495} = 0.0594$$

第 5 步：代入相除，得到后验

$$P(\text{病} \mid \text{阳}) = \frac{0.0099}{0.0594} \approx 0.1667 = 16.67\%$$

结论与直觉：检测阳性后，真患病概率只有约 **16.7%**，远不到 99%！

为什么这么低？因为**健康人太多了**（99%）。即使每个健康人只有 5% 误报，乘以庞大的健康人群，误报的绝对人数（0.0495）反而**远超**真病人测对的人数（0.0099）。

> 一句话记忆：**后验 = 先验 × 似然 ÷ 证据**。先验很小（病罕见）时，再准的检测也救不回来——这就是为什么稀有病要做"二次复检"。

例题

13 📝 例题2：朴素贝叶斯天气分类——今天该不该打网球？

题目：根据下面 10 天的历史记录（特征：天气、湿度；标签：是否打球 Play），用朴素贝叶斯预测：当**天气=晴(Sunny)、湿度=高(High)** 时，今天会 **打球(Yes)** 还是 **不打(No)**？

编号	天气Outlook	湿度Humidity	打球Play
1	晴	高	No
2	晴	高	No
3	阴	高	Yes
4	雨	高	Yes
5	雨	正常	Yes
6	雨	正常	No
7	阴	正常	Yes
8	晴	正常	Yes
9	晴	正常	Yes
10	雨	高	Yes

要预测的样本：$x = (\text{天气} = \text{晴},\ \text{湿度} = \text{高})$。

第 1 步：数一数标签，算先验 $P(c)$（先验 = 类别频率）

10 天中 Yes 有 7 天（编号 3,4,5,7,8,9,10），No 有 3 天（编号 1,2,6）：

$$P(\text{Yes}) = \frac{7}{10} = 0.7, \qquad P(\text{No}) = \frac{3}{10} = 0.3$$

第 2 步：算"天气=晴"在每类下的条件概率（似然之一）

在 **7 个 Yes** 里，天气=晴的有编号 8、9，共 2 个：

$$P(\text{晴} \mid \text{Yes}) = \frac{2}{7} \approx 0.286$$

在 **3 个 No** 里，天气=晴的有编号 1、2，共 2 个：

$$P(\text{晴} \mid \text{No}) = \frac{2}{3} \approx 0.667$$

第 3 步：算"湿度=高"在每类下的条件概率（似然之二）

在 **7 个 Yes** 里，湿度=高的有编号 3、4、10，共 3 个：

$$P(\text{高} \mid \text{Yes}) = \frac{3}{7} \approx 0.429$$

在 **3 个 No** 里，湿度=高的有编号 1、2，共 2 个：

$$P(\text{高} \mid \text{No}) = \frac{2}{3} \approx 0.667$$

第 4 步：套用朴素贝叶斯公式（假设两特征条件独立，似然相乘）

$$score(c) = P(c) \times P(\text{晴} \mid c) \times P(\text{高} \mid c)$$

- Yes 类：

$$score(\text{Yes}) = 0.7 \times 0.286 \times 0.429 \approx 0.0859$$

- No 类：

$$score(\text{No}) = 0.3 \times 0.667 \times 0.667 \approx 0.1334$$

第 5 步：比较两类得分，取大者

$$score(\text{No}) \approx 0.133 > score(\text{Yes}) \approx 0.086$$

结论：预测为 **No（不打球）**。

> 为什么不打球胜出？因为"晴 + 高湿"这两个特征在历史里都更偏向 No（晴天 67% 是 No，高湿在 No 中也占 67%），虽然 Yes 的先验更大(0.7)，但两个似然乘起来后被反超了。

> 想得到真正的概率值，可再除以证据 $P(x) = 0.0859 + 0.1334 = 0.2193$：$P(\text{No} \mid x) = 0.1334 / 0.2193 \approx 60.8\%$。但只为"比大小做决策"，这一步可省略。

例题

14 📝 例题3：拉普拉斯平滑——拯救被"一票否决"的预测

题目（紧接例题2的数据）：现在来一封新邮件，要判断是 **正常(N)** 还是 **垃圾(S)**。训练得到的词概率如下（来自 PPT 的垃圾邮件例子）：

词	$P(\cdot \mid N)$	$P(\cdot \mid S)$
Dear	0.44	0.29
Friend	0.31	0.14
Lunch	0.20	**0**
Dollar	0.08	0.57

先验 $P(N)=\frac{6}{9}\approx 0.67$、$P(S)=\frac{3}{9}\approx 0.33$。注意 **训练集的垃圾邮件里从没出现过 "Lunch"**，所以 $P(\text{Lunch}\mid S)=0$。

新邮件内容是：**"Lunch Dollar"**（一次午餐词 + 一次金钱词）。

第 1 步：不平滑，直接算——看看灾难怎么发生

- 正常类：

$$score(N)=P(N)\times P(\text{Lunch}\mid N)\times P(\text{Dollar}\mid N)=0.67\times 0.20\times 0.08\approx 0.0107$$

- 垃圾类：

$$score(S)=P(S)\times \underbrace{P(\text{Lunch}\mid S)}_{=0}\times P(\text{Dollar}\mid S)=0.33\times 0\times 0.57=0$$

问题来了：\"Dollar\" 是强烈的垃圾信号（$P(\text{Dollar}\mid S)=0.57$ 很高！），但只因为 \"Lunch\" 那一项是 0，**整个连乘被归零**，$score(S)=0$。于是无论金钱词多么可疑，这封邮件都**永远不可能被判为垃圾**——这就是"零计数一票否决"。

第 2 步：引入拉普拉斯平滑（加 1）重新估计词概率

平滑公式（$|Vocab|$ 是词表大小，这里有 Dear/Friend/Lunch/Dollar 共 4 个词，$|Vocab|=4$）：

$$\hat{P}(w\mid c)=\frac{\#(w,c)+1}{n_c+|Vocab|}$$

假设垃圾邮件 S 类训练文本总词数 $n_S=7$，其中 Lunch 计数为 0、Dollar 计数为 4：

- $\hat{P}(\text{Lunch}\mid S)=\frac{0+1}{7+4}=\frac{1}{11}\approx 0.091$ ← **不再是 0 了！**
- $\hat{P}(\text{Dollar}\mid S)=\frac{4+1}{7+4}=\frac{5}{11}\approx 0.455$

同理对正常类 N（设 $n_N=16$，Lunch 计数 4、Dollar 计数 1）：

- $\hat{P}(\text{Lunch}\mid N)=\frac{4+1}{16+4}=\frac{5}{20}=0.25$
- $\hat{P}(\text{Dollar}\mid N)=\frac{1+1}{16+4}=\frac{2}{20}=0.10$

第 3 步：用平滑后的概率重新计算得分

- 正常类：

$$score(N)=0.67\times 0.25\times 0.10\approx 0.0168$$

- 垃圾类：

$$score(S)=0.33\times 0.091\times 0.455\approx 0.0137$$

第 4 步：比较

$$score(N) \approx 0.0168 > score(S) \approx 0.0137$$

结论：平滑后判为 **正常(N)**，但关键是——$score(S)$ **不再是 0 了**，垃圾类重新有了"参赛资格"。如果再多几个金钱词，$score(S)$ 完全可能反超。

> 一句话记忆：平滑就是给每个词"预存 1 票"，让从没见过的词也有一点点小概率，避免某一项为 0 把整个乘积"清零"。分子加 1、分母加 $|Vocab|$，是为了让所有概率加起来仍然等于 1。

例题

第 5 讲 K近邻

K-Nearest Neighbors

K近邻 (KNN) 是一种基于实例的惰性学习方法：对一个待分类样本，找出训练集中与它最近的 K 个邻居，用多数投票决定类别。它不显式训练模型，核心在于**距离度量**、K **值选择**与**特征归一化**。

1 从“建模”到“直接用数据”——KNN 的动机

前几讲的分类器（决策树、朴素贝叶斯）都先从数据中**学习一个模型**，再用模型预测。KNN 抛开这种思路，直接利用数据本身做判断。

核心思想可以用一句俗语概括：**“物以类聚，人以群分”**——告诉我你的邻居是谁，我就能告诉你你是谁。

- **基于实例 (instance-based)**：把训练样本全部存下来，预测时才参与计算。
- **惰性学习 (lazy learning)**：没有显式的训练阶段，几乎所有计算都推迟到预测时进行；与之相对的“积极学习 (eager learning)”（如决策树）在训练时就把模型构建好。
- **非参数方法 (non-parametric)**：不对数据分布做参数化假设，决策边界由数据直接“撑”出来。

> 直觉：相似的输入往往有相似的输出。KNN 把“相似”量化为“距离近”。

KNN　惰性学习　基于实例

2 KNN 分类的基本流程

KNN 用**多数投票 (majority vote)** 分类：一个对象被判为其 K 个最近邻中出现最多的类别。算法步骤：

1. **确定参数** K（邻居个数）。
2. 取一个待分类的测试样本，计算它到**所有训练样本**的距离。
3. 对距离**排序**，取距离最小的 K 个样本。
4. 由这 K 个邻居**多数投票**，将测试样本判为得票最多的类别。

例如柠檬/橙子分类中，若 $K=3$，最近的三个邻居是「柠檬、橙子、柠檬」，则按 2:1 投票判为**柠檬**。

> KNN 也可用于**回归**：把“多数投票”换成“邻居标签的平均（或加权平均）”。

注意：训练阶段几乎什么都不做（只存数据），全部开销在预测阶段。

分类流程　多数投票　算法

3 距离度量 (1)：欧氏距离与曼哈顿距离

“最近”需要一个**距离函数**来定义。设两条记录 $x = (x_1, \dots, x_n)$、$y = (y_1, \dots, y_n)$。

欧氏距离 (Euclidean Distance，欧氏距离)——直线距离：

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

它在高维空间和处理类别型 (categorical) 变量时表现不佳，因为它只看坐标差，忽略了属性之间的相似性。

曼哈顿距离 (Manhattan Distance，城市街区距离 city block)——沿坐标轴走的距离：

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

名字来源于在棋盘格街道上只能横平竖直行走。相比欧氏距离，它对单一维度上的大偏差不那么敏感。

例（二维点）：$p_1 = (0, 2)$，$p_2 = (2, 0)$，欧氏距离 $= \sqrt{2^2 + 2^2} = 2.828$；曼哈顿距离 $= |0 - 2| + |2 - 0| = 4$。

距离度量　欧氏距离　曼哈顿距离

4 距离度量 (2)：闵可夫斯基距离

闵可夫斯基距离 (Minkowski Distance，闵氏距离) 是欧氏与曼哈顿距离的统一推广：

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

参数 p 控制距离的“形状”：

- $p = 1$：退化为**曼哈顿距离 (Manhattan distance)**。
- $p = 2$：退化为**欧氏距离 (Euclidean distance)**。

直觉：p 越大，越强调差异最大的那个维度；p 越小，各维度贡献越均衡。

距离度量　闵可夫斯基距离　范数

5 相似度与距离：余弦相似度

距离 (distance) 越小表示越像；**相似度 (similarity)** 则相反——越像值越大，常落在 $[0,1]$ 区间。

余弦相似度 (Cosine Similarity) 衡量两个向量方向的夹角，而非长度：

$$\cos(\theta) = \frac{x \cdot y}{\|x\|\,\|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\,\sqrt{\sum_i y_i^2}}$$

例：$d_1 = [3,2,0,5,0,0,0,2,0,0]$，$d_2 = [1,0,0,0,0,0,0,1,0,2]$。

- $d_1 \cdot d_2 = 3{\times}1 + 2{\times}1 = 5$
- $\|d_1\| = \sqrt{3^2+2^2+5^2+2^2} = 6.481$，$\|d_2\| = \sqrt{1^2+1^2+2^2} = 2.245$
- $\cos(d_1, d_2) = 5/(6.481{\times}2.245) = 0.315$

余弦相似度对向量长度不敏感，特别适合**文本/高维稀疏数据**（如词频向量）：只关心“成分比例”，不关心文档长短。

相似度　余弦相似度　文本

6 归一化 (Normalization)

KNN 直接依赖距离，而距离会被**量纲大、取值范围大的特征主导**。例如「年龄(20~25)」与「分数(0~100)」一起算欧氏距离时，分数差几乎完全决定结果，年龄被淹没。**归一化**把各特征缩放到可比的范围。

Min-max 归一化：缩放到固定区间 $[0,1]$：

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Z-score 标准化：缩放成均值 0、标准差 1：

$$x' = \frac{x - \mu}{\sigma}$$

- Min-max 保留原始分布形状，但对**异常值敏感**（极值会压缩其余数据）。
- Z-score 对异常值更稳健，不限定输出范围。

> 务必只用**训练集**的统计量 $(x_{\min}, x_{\max}, \mu, \sigma)$ 去变换测试集，否则会造成数据泄漏。KNN 的劣势之一就是“必须做归一化”。

归一化　标准化　预处理

7 决策边界与 Voronoi 图

Voronoi 划分 (Voronoi Tessellation，维诺图/沃罗诺伊分割)：把空间划分成若干区域，每个区域内的点离某一个训练样本最近。区域之间的边界是“到两个训练样本等距”的点集。

当 $K=1$（最近邻）时，分类结果完全由 Voronoi 区域决定：测试点落在谁的格子里就归谁的类。**决策边界 (Decision Boundary)** 就是分隔不同类别的那些 Voronoi 边。

- $K=1$：边界曲折、紧贴样本，能完美拟合训练点，但对噪声极其敏感。
- $K>1$：边界更平滑。

> 当训练样本很多且标签含噪声时，$K=1$ 的决策边界会变得非常“破碎”，泛化差。增大 K 相当于对邻域做平滑。

决策边界　Voronoi图　最近邻

8 K 值的影响 (Effect of K)

K（邻居个数）是 KNN 的关键参数，它直接控制决策边界的形状：

- **K 小（如 $K=1$）**：决策边界曲折、紧贴样本，能完美拟合训练点，但对噪声极其敏感。
- **K 大**：邻域平均效应增强，决策边界更**平滑**。
- **极端 $K=N$（N 为训练样本总数）**：所有样本都参与投票，无论输入是什么，永远预测**全局多数类**，模型完全失去判别力。

> 讨论：$K=1$ 与 $K=15$ 谁更好？没有绝对答案——当训练样本很多、标签含噪声时，$K=1$ 的边界会非常破碎；增大 K 相当于对邻域做平滑，但过大又会忽略局部结构。

K值选择　决策边界　平滑

9 KNN 的优缺点

优点 (Advantages)：

- **简单**，易于理解、解释和实现。
- **无需训练**（惰性学习），没有训练阶段的计算开销。
- **可增量更新**：新数据可无缝加入，不影响已有模型、无需重训。

缺点 (Disadvantages)：

- **不适合大数据集**：每次预测都要算到所有样本的距离，计算代价高。
- 对**维度灾难**高度敏感。
- **存储开销大**：需保存全部训练数据。
- **必须做数据归一化**，否则大量纲特征主导距离。

> 一句话：KNN 把“学习成本”全部转嫁到了“预测成本”和“存储成本”上。

优缺点　适用场景

10 📝 例题1：手算欧氏距离并排序（找最近邻的第一步）

KNN 的第一步就是：把待分类点到**每一个**训练点的距离都算出来，再排序。我们用 5 个二维带标签的点来手算一遍。

训练集（5 个点，类别为 A 或 B）：

点	坐标 (x_1, x_2)	类别
A点	$(2, 3)$	A
B点	$(5, 4)$	B
C点	$(4, 7)$	A
D点	$(8, 6)$	B
E点	$(7, 3)$	A

待分类点（测试点）： $z = (5, 5)$，类别未知。

欧氏距离公式（两点直线距离）：

$$d(z, p) = \sqrt{(z_1 - p_1)^2 + (z_2 - p_2)^2}$$

下面**逐个**代入计算（每一步都把数字写出来）：

1. **到 B点** $(5, 4)$：横坐标差 $5 - 5 = 0$，纵坐标差 $5 - 4 = 1$。

$$d = \sqrt{0^2 + 1^2} = \sqrt{1} = 1.000$$

1. **到 C点** $(4, 7)$：差为 $5 - 4 = 1$ 和 $5 - 7 = -2$。

$$d = \sqrt{1^2 + (-2)^2} = \sqrt{1 + 4} = \sqrt{5} \approx 2.236$$

1. **到 E点** $(7, 3)$：差为 $5 - 7 = -2$ 和 $5 - 3 = 2$。

$$d = \sqrt{(-2)^2 + 2^2} = \sqrt{4 + 4} = \sqrt{8} \approx 2.828$$

1. **到 D点** $(8, 6)$：差为 $5 - 8 = -3$ 和 $5 - 6 = -1$。

$$d = \sqrt{(-3)^2 + (-1)^2} = \sqrt{9 + 1} = \sqrt{10} \approx 3.162$$

1. **到 A点** $(2, 3)$：差为 $5 - 2 = 3$ 和 $5 - 3 = 2$。

$$d = \sqrt{3^2 + 2^2} = \sqrt{9 + 4} = \sqrt{13} \approx 3.606$$

> 小提示：差是负数没关系，因为要**平方**，结果一定非负。所以 $(-2)^2$ 和 2^2 一样都是 4。

按距离从小到大排序（这一步是为下一步投票做准备）：

排名	点	距离	类别
1	B点	1.000	B
2	C点	2.236	A
3	E点	2.828	A
4	D点	3.162	B
5	A点	3.606	A

这张排序表是 KNN 的核心中间结果，**例题2、例题3 都基于它**。

例题

11 📝 例题2：K=1 与 K=3 投票，结果竟然不同！

接着例题1 的排序表，我们来做最关键的一步——**多数投票 (majority vote)**。我们会发现：**换一个 K，分类结果可能完全相反**。

沿用排序表（测试点 $z=(5,5)$）：

排名	点	距离	类别
1	B点	1.000	**B**
2	C点	2.236	A
3	E点	2.828	A
4	D点	3.162	B
5	A点	3.606	A

情形一：$K=1$（只看最近的 1 个邻居）

1. 取排名第 1 的点：B点，距离 1.000。
2. 只有它 1 票，类别是 **B**。
3. 所以 z 被判为 **B 类**。

> 为什么？$K=1$ 就是「离谁最近就跟谁同类」。这里 z 离 B点 仅 1.0，是绝对最近，于是直接判 B。

情形二：$K=3$（看最近的 3 个邻居）

1. 取排名前 3 的点：B点(B)、C点(A)、E点(A)。
2. 数票：A 有 **2 票**（C、E），B 有 **1 票**（B）。
3. $2>1$，多数是 A，所以 z 被判为 **A 类**。

对比结论：

K	参与投票的邻居	票数	判定结果
$K=1$	B	B:1	**B**
$K=3$	B, C, E	A:2, B:1	**A**

这说明什么？ K 不是随便选的：

- $K=1$ 时，结果被**最近的那一个点**完全决定。如果 B点 恰好是个**噪声/异常点**，z 就被带偏了——这就是 $K=1$ 紧贴噪声、**易过拟合**的直观体现。
- $K=3$ 时，多看了几个邻居，发现周围其实 A 更多，结果更**稳健**。

> 增大 K 相当于对邻域做平滑，能降低对单个噪声点的敏感；但 K 也不是越大越好，过大（极端到 N）会只预测全局多数类。

例题

12 📝 例题3：不归一化的「翻车」现场——量纲大的特征独霸距离

这是 KNN 最常见的坑：如果两个特征**量纲差很多**（比如「年收入」单位是元、「年龄」单位是岁），不归一化时距离几乎只由大数那一维决定，另一维形同虚设。我们用具体数字演示。

训练集（特征：年收入(元)、年龄(岁)）：

人	年收入(元)	年龄(岁)	类别
小明	30000	25	A
小红	31000	45	B

测试点： 小测 $(30500, 26)$。直觉上，小测年龄 26 岁，跟 **45 岁的小红差很远**、跟 25 岁的小明很近，按常识应更像谁？年龄上明显更像小明（A）。我们算算看。

第一步：不归一化，直接算欧氏距离

1. **到小明** $(30000, 25)$：收入差 $30500 - 30000 = 500$，年龄差 $26 - 25 = 1$。

$$d = \sqrt{500^2 + 1^2} = \sqrt{250000 + 1} = \sqrt{250001} \approx 500.001$$

1. **到小红** $(31000, 45)$：收入差 $30500 - 31000 = -500$，年龄差 $26 - 45 = -19$。

$$d = \sqrt{(-500)^2 + (-19)^2} = \sqrt{250000 + 361} = \sqrt{250361} \approx 500.361$$

观察问题所在： 两个距离都约等于 500，差别极小（500.001 vs 500.361）。原因是**收入差 500 的平方 = 250000 把一切淹没了**，年龄差（1 和 19）平方后才 1 和 361，对总距离几乎没有影响。也就是说——**年龄这个维度基本被忽略了，距离完全由收入主导**。

> 直观感受：$\sqrt{250000 + 1}$ 和 $\sqrt{250000 + 361}$ 几乎一样。年龄差了 19 岁，距离却只差了 0.36，这显然不合理。

第二步：做 Min-max 归一化，把两维都缩放到 $[0, 1]$

公式：$x' = \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}$。这里收入范围取 $[30000, 31000]$，年龄范围取 $[25, 45]$。

- 收入：$30000 \to 0$，$30500 \to \frac{500}{1000} = 0.5$，$31000 \to 1$。
- 年龄：$25 \to 0$，$26 \to \frac{1}{20} = 0.05$，$45 \to 1$。

归一化后的坐标：小明 $(0, 0)$、小红 $(1, 1)$、小测 $(0.5, 0.05)$。

第三步：用归一化后的坐标重新算距离

1. **到小明** $(0, 0)$：

$$d = \sqrt{(0.5 - 0)^2 + (0.05 - 0)^2} = \sqrt{0.25 + 0.0025} = \sqrt{0.2525} \approx 0.502$$

1. **到小红** $(1, 1)$：

$$d = \sqrt{(0.5 - 1)^2 + (0.05 - 1)^2} = \sqrt{0.25 + 0.9025} = \sqrt{1.1525} \approx 1.074$$

结果对比：

	到小明(A)	到小红(B)	$K = 1$ 判定
不归一化	500.001	500.361	小明(A)，但**险胜，年龄被忽略**
归一化后	**0.502**	1.074	小明(A)，**差距明显，符合常识**

归一化后，「年龄相近」这个本该重要的信息终于起作用了：到小明 0.502，到小红 1.074，**明确更像小明**。两维公平地参与了距离计算。

> 结论：KNN **必须**先归一化，否则取值范围大的特征会独霸距离，让其他特征白白浪费。常用方法是 Min-max（缩放到 $[0,1]$）或 Z-score（均值 0、标准差 1）。

例题

13 📝 例题4：欧氏距离 vs 曼哈顿距离，同一对点两种算法

「最近」取决于用哪种**距离函数**。同样两个点，欧氏距离和曼哈顿距离算出来的值不一样。我们手算对比一次。

两个点： $p=(1,2)$，$q=(4,6)$。

先求每一维的差：

- 第一维：$1-4=-3$
- 第二维：$2-6=-4$

方法一：欧氏距离（直线距离，先平方再开方）

$$d_{\text{欧}}=\sqrt{(-3)^2+(-4)^2}=\sqrt{9+16}=\sqrt{25}=5.000$$

方法二：曼哈顿距离（沿坐标轴走，取绝对值再相加）

$$d_{\text{曼}}=|-3|+|-4|=3+4=7.000$$

为什么不一样?

- **欧氏距离**是「抄近路斜着走」的直线长度，对应勾股定理的斜边，所以是 5。
- **曼哈顿距离**像在棋盘格街道上开车，只能**横平竖直**走：先横着走 3 格，再竖着走 4 格，总共 $3+4=7$ 格。

> 它们都是**闵可夫斯基距离** $\left(\sum_i |x_i-y_i|^p\right)^{1/p}$ 的特例：

> - $p=1$ → 曼哈顿距离（这里 $|-3|^1+|-4|^1=7$，再开 1 次方还是 7）；

> - $p=2$ → 欧氏距离（这里 $(9+16)^{1/2}=5$）。

> 一般 p 越大，越突出差异最大的那一维；这里第二维差 4 最大，p 越大它的「话语权」越重。

例题

第 6 讲 逻辑回归

Logistic Regression

本讲从判别函数(Discriminant Function)的视角引入线性分类器，直接建模决策边界(Decision Boundary)，并由感知机(Perceptron)的硬判决缺陷过渡到逻辑回归。逻辑回归用 Sigmoid 函数 $y=\sigma(w^Tx+w_0)=\frac{1}{1+e^{-(w^Tx+w_0)}}$ 估计后验概率 $P(C_1\mid x)$，以交叉熵/对数似然为损失，用梯度下降训练，并自然推广到多分类的 Softmax 回归。

1 从生成式到判别式：为什么直接估计决策边界

传统分类器各自依赖数据的某种结构来做决策：

- **决策树**：靠数据**不纯度(impurity)**划分；
- **贝叶斯分类**：靠**概率密度** $p(x\mid C_i)$ 与先验 $P(C_i)$ 比较后验；
- **K近邻**：靠样本间**距离**。

核心质疑是：我们真的需要知道每类数据的完整结构(密度/距离/不纯度)吗？分类的最终目的只是把样本分到正确的一侧。

> 判别式思想(Decision Boundaries Are All You Need)：**直接估计决策边界**，而不去费力建模每一类的完整概率分布。确定类间边界(判别)通常比估计类条件密度更容易、参数更少。

- **生成式(generative)**：先学 $p(x\mid C_i)$、$P(C_i)$，再用贝叶斯反推后验，建模整体数据分布。
- **判别式(discriminative)**：直接学 $P(C_i\mid x)$ 或边界 $g(x)$。逻辑回归正是典型的判别式模型。

判别式模型　决策边界　动机

2 判别函数与决策规则

对于 c 个类别的分类问题，定义 c 个**判别函数(discriminant function)** $g_i(x\mid\theta_i)$，每个函数刻画样本 x 属于类别 C_i 的程度，由一组参数 θ_i 决定。

决策规则：把 x 分到使判别函数最大的那一类：

$$\text{assign } x\to C_j \quad \text{if} \quad g_j(x)>g_i(x),\ \forall i\neq j$$

以贝叶斯决策为例，判别函数有多种等价写法(对单调变换不变)：

- $g_i(x)=P(C_i\mid x)=\dfrac{p(x\mid C_i)P(C_i)}{\sum_j p(x\mid C_j)P(C_j)}$
- $g_i(x)=p(x\mid C_i)P(C_i)$
- $g_i(x)=\ln p(x\mid C_i)+\ln P(C_i)$

两类情形可合并为单个判别函数 $g(x)=g_1(x)-g_2(x)$：

- $g(x)>0\Rightarrow C_1$，否则 C_2；
- **决策边界**即 $g(x)=0$。

判别函数　决策规则　二分类

3 线性判别函数与超平面几何

线性判别函数(Linear Discriminant Function) 用线性函数刻画样本的类别归属——直观地说就是"画一条线把数据分开":

$$g(x) = w^T x + w_0 = w_1 x_1 + w_2 x_2 + \cdots + w_0$$

几何含义：$g(x) = 0$ 定义了一个**超平面(hyperplane)**，把输入空间切成两个半空间：

- $g(x) > 0$ 一侧为决策区域 R_1(判 C_1)；
- $g(x) < 0$ 另一侧为 R_2(判 C_2)；
- 权向量 w 与超平面**正交**，指向 $g(x) > 0$ 的一侧。

优点：

- **简单(simplicity)**：时间与空间复杂度均为 $O(d)$；
- **可理解(understandability)**：输出是属性的加权和；
- **准确(accuracy)**：在一定假设满足时相当准确。

这与线性规划中超平面约束 $a_i^T x \le b_i$ 的几何相通。

线性判别　超平面　几何解释

4 感知机及其训练规则

感知机(Perceptron，感知机) 是最早、最朴素的线性分类器，用 $g(x) = w^T x + w_0$ 判定，预测取 $\text{sign}[g(x)]$：$g(x) > 0$ 判 C_1，否则 C_2。

标签约定：$r \in \{+1, -1\}$ 表示 $g(x)$ 应有的符号。给定数据集 $D = \{(x^{(\ell)}, r^{(\ell)})\}_{\ell=1}^{N}$。

训练(错误驱动)：遍历每个样本，

- 若 $r^{(\ell)} g(x^{(\ell)}) \le 0$，说明发生了**误分类**，则更新：

$$w \leftarrow w + \eta\, r^{(\ell)} x^{(\ell)}$$

- 直观上是把超平面朝误分类点的方向"推"，直到训练集全部分类正确。

```
for each (x, r) in D:
    if r * g(x) <= 0:        # 误分类
        w <- w + eta * r * x  # 朝该点修正
repeat until 全部正确
```

感知机依据新点落在超平面的哪一侧来分类。

感知机　错误驱动　训练规则

5 感知机的局限：硬判决难以优化

感知机输出是**硬判决(hard decision)**：只回答 $g(x) > 0$ 吗?——"是/否"的 0-1 决策。

问题在于：

- 模型只能说"x 属于 C_1"，**无法表达"属于的程度"**(到底有多确定?)；
- 0-1 的阶跃决策**不可导、不连续**，难以用基于梯度的方法优化；
- 对靠近边界的样本与远离边界的样本一视同仁。

> 解决思路：引入**带不确定性的线性分类**——直接输出后验概率 $P(C_1 \mid x) = y$、$P(C_2 \mid x) = 1 - y$，用一个**平滑可导**的函数替代阶跃，使其便于优化。这正是逻辑回归的出发点。

感知机　硬判决　局限

6 对数几率(Logit)与 Sigmoid 函数

设后验 $P(C_1 \mid x) = y$，$P(C_2 \mid x) = 1 - y$。定义 y 的**几率(odds)** 为 $\dfrac{y}{1-y}$，其对数即 **对数几率/logit 函数**：

$$\operatorname{logit}(y) = \log \frac{y}{1-y}$$

关键假设与结论：若各类 x 服从高斯分布(同协方差)，可推出后验的对数几率是输入 x 的**线性组合(线性回归)**：

$$\log \frac{y}{1-y} = w^T x + w_0$$

对它求逆，就得到 **Sigmoid(逻辑)函数**，它直接给出后验概率：

$$y = \sigma(z) = \frac{1}{1+e^{-z}}, \qquad z = w^T x + w_0$$

- Sigmoid 是 logit 的**反函数**；
- 它把任意实数 $z \in (-\infty, +\infty)$ 平滑地压到 $(0, 1)$，可解释为概率；
- **平滑可导**，便于梯度优化，克服了感知机硬判决的缺点。

> 一句话："对数几率是 x 的线性函数"⟺"后验概率是 Sigmoid(线性函数)"，这就是 Logistic Regression 名字里 logistic 的由来。

Sigmoid　对数几率　logit

7 逻辑回归模型与预测

逻辑回归(Logistic Regression) 是把决策边界估计为逻辑函数的分类器：

$$y = \sigma(w^T x + w_0) = \frac{1}{1 + \exp[-(w^T x + w_0)]}$$

通常令 $x_0 = 1$ 把偏置 w_0 吸收进权向量，写成 $y = \sigma(w^T x)$。

- **训练(Train)**：从数据估计参数 w 和 w_0。
- **测试(Test)**：对新样本计算 $y = \sigma(w^T x + w_0)$，若 $y > 0.5$ 判 C_1，否则 C_2。y 可解释为后验概率 $P(C_1 \mid x)$。

> 注意：尽管叫"回归"，逻辑回归是**分类**模型。决策边界 $y = 0.5$ 对应 $w^T x + w_0 = 0$，仍是一个**线性超平面**；Sigmoid 只是把线性得分映射为概率，并不改变边界的线性本质。

逻辑回归　模型定义　预测

8 交叉熵/对数似然损失

模型输出 $y \equiv p(C_1 \mid x) = \sigma(w^T x + w_0)$。设真实标签 $r \in \{0, 1\}$（$r = 1$ 表示 $x \in C_1$，$r = 0$ 表示 $x \in C_2$）。

逐情形的目标(最大化对数后验)：

- 若 $r = 1$：最大化 $\log p(C_1 \mid x) = \log y$，代价为 $-\log y$；
- 若 $r = 0$：最大化 $\log p(C_2 \mid x) = \log(1 - y)$，代价为 $-\log(1 - y)$。

合并写成**交叉熵损失(cross-entropy loss)**：

$$\ell(w, w_0 \mid x, r) = -r \log y - (1 - r) \log(1 - y)$$

其中第一项仅当 $r = 1$ 非零，第二项仅当 $r = 0$ 非零。

最大似然等价性：$r \mid x \sim \text{Bernoulli}(y)$，单样本似然

$$p(r \mid x) = y^r (1 - y)^{1 - r} = \begin{cases} y, & r = 1 \\ 1 - y, & r = 0 \end{cases}$$

对其取负对数即得上述交叉熵。因此"最小化交叉熵"与"最大化伯努利似然"是同一件事。

> 交叉熵的一般定义 $CE(p, q) = -\sum_x p(x) \log q(x)$，这里 p 是真实分布(one-hot)，q 是模型预测分布。

交叉熵　对数似然　损失函数

9 梯度推导与训练算法

全体数据的损失：

$$L(w, w_0 \mid D) = -\sum_{\ell=1}^{N} \left[r^{(\ell)} \log y^{(\ell)} + (1 - r^{(\ell)}) \log(1 - y^{(\ell)}) \right]$$

关键导数(链式法则)。记 $a = w^T x + w_0$，$y = \sigma(a)$，利用 Sigmoid 的良好性质：

$$\frac{dy}{da} = y(1 - y), \qquad \frac{\partial a}{\partial w_j} = x_j$$

对 w_j 求偏导：

$$\frac{\partial L}{\partial w_j} = -\sum_{\ell} \left(\frac{r^{(\ell)}}{y^{(\ell)}} - \frac{1 - r^{(\ell)}}{1 - y^{(\ell)}} \right) \frac{\partial y^{(\ell)}}{\partial a^{(\ell)}} \frac{\partial a^{(\ell)}}{\partial w_j}$$

代入 $\frac{\partial y}{\partial a} = y(1 - y)$ 后，分母被巧妙消去，得到极其简洁的结果：

$$\boxed{\frac{\partial L}{\partial w_j} = -\sum_{\ell} \left(r^{(\ell)} - y^{(\ell)} \right) x_j^{(\ell)}}$$

梯度下降更新：

$$w_{t+1} = w_t - \eta_t \frac{\partial L}{\partial w} = w_t + \eta \sum_{\ell} (r^{(\ell)} - y^{(\ell)}) x^{(\ell)}$$

> 直觉：梯度正比于"误差 $(r - y)$ 乘以输入 x"——预测越偏离真实，修正越大；这与线性回归的梯度形式惊人地一致(只是 y 换成了 Sigmoid 输出)。

```
初始化 w_j ← rand(-0.01, 0.01)
repeat
  Δw_j ← 0
  for 每个样本 (x, r):
    a ← Σ w_j x_j;  y ← sigmoid(a)
    Δw_j ← Δw_j + (r - y) x_j
  w_j ← w_j + η Δw_j
until 收敛
```

梯度下降　推导　训练算法

10 Softmax 函数与多分类

二分类用 Sigmoid，多分类(K 类，目标 $r \in \{1, \dots, K\}$)需要把多个得分统一转成概率分布。**Softmax 函数**:

$$y_i = \text{softmax}(a_1, \dots, a_K)_i = \frac{e^{a_i}}{\sum_{j=1}^{K} e^{a_j}}$$

其中 $a_i = W_i x + w_{i0}$ 称为 **logits(得分)**，W_i, w_{i0} 是可训练参数。

性质:

- 把任意实数向量 logits 转成一个**概率分布**($y_i \geq 0$ 且 $\sum_i y_i = 1$);
- 不需要选定参考类，**对各类对称处理**;
- **为什么叫 softmax?** 当某个 a_k 远大于其它时，softmax 近似于 argmax 的平滑版(其实更像"soft-argmax")。"max"是因为它放大最大 logit 的概率，"soft"是因为仍给较小的 a_j 分配一些概率;
- 关键优势是**可微**，便于梯度优化;
- $K = 2$ 时 Softmax 退化为 Sigmoid。

Softmax 多分类 logits

11 Softmax 回归：损失与梯度

Softmax 回归(Softmax Regression) 用 Softmax 估计各类后验：

$$y_i = \frac{\exp[w_i^T x + w_{i0}]}{\sum_{j=1}^{K} \exp[w_j^T x + w_{j0}]}, \quad i = 1, \ldots, K$$

测试时选 $y_i = \max\{y_1, \ldots, y_K\}$ 的类 C_i。

损失：标签为 **one-hot 向量** $r = (r_1, \ldots, r_K)$（$r_i = 1$ 若 $x \in C_i$）。交叉熵损失：

$$L_{CE}(y, r) = -\sum_{i=1}^{K} r_i \log y_i = -r^T (\log y)$$

由于 r 是 one-hot，实际只有真实类那一项 $-\log y_{\text{真实类}}$ 起作用。

梯度。利用 Softmax 导数 $\frac{\partial y_i}{\partial a_k} = y_i(\delta_{ik} - y_k)$ 与约束 $\sum_i r_i = 1$，可推出与逻辑回归同样优美的形式：

$$\frac{\partial L}{\partial w_i} = \sum_{\ell} \left(y_i^{(\ell)} - r_i^{(\ell)}\right) x^{(\ell)}, \qquad \Delta w_{ij} \mathrel{+}= (r_i^{(\ell)} - y_i^{(\ell)}) x_j^{(\ell)}$$

梯度下降算法(伪代码)：与逻辑回归同构，只是把每个样本要更新的权重从一组 w_j 变成 K 组 w_{ij}（$i = 1, \ldots, K$）：

```
初始化 w_ij ← rand(-0.01, 0.01)   # i=1..K, j=0..d
repeat
  所有 Δw_ij ← 0
  for 每个样本 (x, r):
    for i = 1..K:  a_i ← Σ_j w_ij x_j        # 各类 logit
    for i = 1..K:  y_i ← exp(a_i)/Σ_k exp(a_k) # softmax
    for i = 1..K, j = 0..d:
      Δw_ij ← Δw_ij + (r_i - y_i) x_j
  for i = 1..K, j = 0..d:
    w_ij ← w_ij + η Δw_ij
until 收敛
```

> 二分类逻辑回归是 Softmax 回归在 $K = 2$ 时的特例；二者梯度都是"(预测概率 − 真实标签) × 输入"，伪代码也只差一层对类别 i 的循环。

Softmax回归 交叉熵 梯度

12 线性分类器的局限

逻辑回归/Softmax 回归本质上仍是**线性分类器**，存在固有局限：

- **要求数据线性可分(linearly separable)**：决策边界是超平面，对异或(XOR)等非线性可分数据无能为力；
- **解不唯一**：即便数据线性可分，能完美分开的超平面**有很多**，逻辑回归并不保证选到"最好"的那一个(例如离两类都最远的边界)。

> 承上启下：这两个问题分别指向后续课题——用**核技巧/特征映射**处理非线性，用**支持向量机(SVM)** 寻找**最大间隔(maximum margin)** 的唯一最优边界。

局限 线性可分 承上启下

13 📝 例题1：从权重到概率——前向计算 z、Sigmoid 与按 0.5 阈值判类

场景：用逻辑回归判断一名学生这次考试是否「及格(C1)」还是「不及格(C2)」。模型已经训练好，参数为：

- 权重 $w = (w_1, w_2) = (0.8,\ -0.5)$
- 偏置 $b = w_0 = 0.3$

现在来了一条样本 $x = (x_1, x_2) = (2,\ 1)$（比如 x_1=复习小时数、x_2=旷课次数）。问：模型预测该学生属于哪一类？

第1步：算线性得分 $z = w^T x + b$。

逻辑回归的第一步永远是把输入做一个「加权求和」，得到一个实数得分 z：

$$z = w_1 x_1 + w_2 x_2 + b = 0.8 \times 2 + (-0.5) \times 1 + 0.3$$

逐项算：$0.8 \times 2 = 1.6$，$-0.5 \times 1 = -0.5$，再加 0.3：

$$z = 1.6 - 0.5 + 0.3 = 1.4$$

> 为什么要这一步？因为权重 $w_1 = 0.8 > 0$ 表示「复习越多越可能及格」，$w_2 = -0.5 < 0$ 表示「旷课越多越不及格」。z 就是把这些证据加在一起得到的「总分」。z 越大越倾向 C1。

第2步：把 z 用 Sigmoid 压成概率 y。

$z = 1.4$ 还不是概率（概率必须在 0~1 之间）。用 Sigmoid 函数把它压到 $(0, 1)$：

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-1.4}}$$

查 $e^{-1.4} \approx 0.2466$，代入：

$$y = \frac{1}{1 + 0.2466} = \frac{1}{1.2466} \approx 0.802$$

> 这个 $y = 0.802$ 就是模型估计的后验概率 $P(C_1 \mid x)$，意思是「模型有约 80.2% 的把握认为该生及格」。这正是逻辑回归比感知机强的地方——它不仅给出类别，还给出**确定程度**。

第3步：按 0.5 阈值判类。

规则：$y > 0.5$ 判 C1，否则判 C2。

$$y = 0.802 > 0.5 \ \Rightarrow \ \text{判为 C1（及格）}$$

> 小贴士：$y > 0.5$ 完全等价于 $z > 0$。本例 $z = 1.4 > 0$，所以不用算 Sigmoid 也能立刻知道是 C1。Sigmoid 只决定「多确定」，不改变「分到哪边」——决策边界就是 $z = 0$ 这条线。

例题

14 📝 例题2：算这条样本的交叉熵损失（看预测对不对、错多狠）

接着例题1：模型预测 $y = \sigma(1.4) \approx 0.802$。现在我们拿到了**真实标签**，来算这条样本的**交叉熵损失**，衡量预测有多准。

交叉熵公式（单样本）：

$$\ell = -r \log y - (1 - r) \log(1 - y)$$

其中 $r \in \{0, 1\}$ 是真实标签（这里 log 取自然对数 ln）。

情形 A：真实标签 $r = 1$（该生确实及格，模型猜对了）。

当 $r = 1$ 时，公式第二项 $(1 - r) = 0$ 直接消失，只剩第一项：

$$\ell = -1 \cdot \log y - 0 = -\log(0.802)$$

查 $\ln(0.802) \approx -0.2204$，所以：

$$\ell = -(-0.2204) = 0.220$$

> 损失很小（0.220）。因为模型说「80.2% 是 C1」而真相也是 C1——**猜对且自信**，几乎不惩罚。如果模型更自信（比如 $y = 0.99$），损失会更小（$-\ln 0.99 \approx 0.01$）。

情形 B：真实标签 $r = 0$（其实没及格，模型猜错了）。

当 $r = 0$ 时，第一项 $r = 0$ 消失，只剩第二项：

$$\ell = -(1 - 0) \log(1 - y) = -\log(1 - 0.802) = -\log(0.198)$$

查 $\ln(0.198) \approx -1.6204$：

$$\ell = -(-1.6204) = 1.620$$

> 损失大得多（1.620）！因为模型自信地说「80.2% 是 C1」却**猜错了**——自信地犯错要受重罚。这正是交叉熵想要的：**预测越偏离真相、惩罚越大**，从而逼着模型修正。

一句话总结：交叉熵只看「模型给真实那一类分配了多少概率」。给得越高 → $-\log$ 越小 → 损失越小；给得越低 → 损失越大。本例对照：

真实标签	模型给真类的概率	损失 ℓ
$r = 1$（猜对）	$y = 0.802$	0.220（小）
$r = 0$（猜错）	$1 - y = 0.198$	1.620（大）

例题

15 📝 例题3：手算一步梯度下降，更新权重 w 和偏置 b

继续例题1/2 的设置，假设真实标签 $r=1$（该生及格），我们来**手动做一次梯度下降**，看看参数怎么被更新。取学习率 $\eta=0.1$。

已知：$x=(2,1)$，当前 $w=(0.8,-0.5)$，$b=0.3$，预测 $y=0.802$，$r=1$。

第1步：算「误差」$(r-y)$。

逻辑回归梯度有一个极其简洁的形式（课件已推导）：

$$\frac{\partial L}{\partial w_j}=-(r-y)\,x_j$$

所以更新公式（沿负梯度方向）为：

$$w_j \leftarrow w_j-\eta\frac{\partial L}{\partial w_j}=w_j+\eta\,(r-y)\,x_j$$

先算误差这个共用因子：

$$r-y=1-0.802=0.198$$

> 直觉：$(r-y)$ 是「真实 − 预测」。这里为正（0.198），说明模型预测的概率还**偏低**了一点（真相是 100% 的 C1，模型只给了 80.2%），所以要把参数往「更倾向 C1」的方向推一点。

第2步：更新每个权重 w_j。

套公式 $w_j \leftarrow w_j+\eta(r-y)x_j$，其中 $\eta(r-y)=0.1\times 0.198=0.0198$：

- 更新 w_1（对应 $x_1=2$）：

$$w_1=0.8+0.0198\times 2=0.8+0.0396=0.8396$$

- 更新 w_2（对应 $x_2=1$）：

$$w_2=-0.5+0.0198\times 1=-0.5+0.0198=-0.4802$$

第3步：更新偏置 b（相当于 $x_0=1$ 的权重）。

$$b=0.3+0.0198\times 1=0.3198$$

更新后参数： $w=(0.8396,\,-0.4802)$，$b=0.3198$。

验证有没有变好？ 用新参数重算这条样本：

$$z_{新}=0.8396\times 2-0.4802\times 1+0.3198=1.4188$$

$$y_{新}=\sigma(1.4188)\approx 0.805$$

> 概率从 0.802 升到了 0.805，更靠近真实标签 1 了！说明这一步更新方向正确。注意 w_1 增加得最多（因为它对应的 $x_1=2$ 最大）——**输入越大的特征，权重调整越多**，这正是梯度里乘了 x_j 的含义。反复迭代所有样本，损失就会逐步下降直到收敛。

例题

16

Softmax回归　算法

第 7 讲 支持向量机

Support Vector Machine

支持向量机(SVM)通过**最大化间隔(margin)**寻找最优分类超平面：在线性可分情形下，约束 $y(w^\top x + w_0) \geq 1$ 下最大化间隔 $\frac{2}{\|w\|}$ 等价于最小化 $\frac{1}{2}\|w\|^2$，这是一个凸二次规划(QP)。本课进一步用**合页损失(hinge loss)** $\max(0, 1 - y(w^\top x + w_0))$ 加 L_2 正则把它写成无约束目标，并用**梯度下降**直接优化求解。

1 从线性分类器到"哪条边界最好"

分类算法家族回顾：在此之前我们学过若干分类算法——朴素贝叶斯(Naïve Bayes)、决策树(Decision Trees)、最近邻(Nearest Neighbor)、逻辑回归(Logistic Regression)，它们共同构成了**分类算法家族(the family of classification)**。本讲的支持向量机(SVM)是其中又一员，属于**线性分类器**这一支。

给定训练集 $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$，共 N 个样本，其中 $x^{(\ell)} \in \mathbb{R}^d$ 是输入，标签 $y^{(\ell)} \in \{-1, +1\}$ 是目标。若数据**线性可分**，则存在一个线性曲面(超平面) $w^\top x + w_0 = 0$ 把两类分开(2维是直线 line、3维是平面 plane、d维是超平面 hyperplane)。线性分类器的目标就是：**找到一个 $w^\top x + w_0 = 0$ 完美分开两类**。

- 问题在于：能完美分开两类的超平面有**无穷多条**，到底哪条最好?
- SVM 的回答：选择使**间隔(margin)最大**的那条边界。

> 直觉：边界离两类样本都尽量远，则对噪声/扰动更鲁棒，泛化能力更强。统计学习理论表明，**最大间隔超平面具有最小的泛化误差(generalization error)**。"瘦"间隔(skinny margin)模型更灵活、更复杂(容易过拟合)，"胖"间隔(fat margin)模型更简单。

分类算法家族　线性分类器　最大间隔　泛化

2 判别函数与决策规则

SVM 使用线性判别函数(linear discriminant function)对两类建模：

$$g(x) = w^\top x + w_0$$

- 决策超平面：$g(x) = 0$。
- $g(x) > 0 \Rightarrow$ 判为正类 C_1；$g(x) < 0 \Rightarrow$ 判为负类 C_2。
- w 是超平面的**法向量(normal vector)**，决定方向；w_0 是偏置，决定平移。
- 模型结构与感知机(Perceptron)相同：输入 $x_0 = 1, x_1, \ldots, x_d$，加权求和得到 $g(x) = w^\top x + w_0$，再与目标值比较计算损失。

训练：用数据优化参数 w, w_0。

测试：对新输入 x 计算 $g(x) = w^\top x + w_0$，若 $g(x) > 1$ 判为 C_1，否则判为 C_2。

判别函数　决策规则　超平面

3 函数间隔与几何间隔

函数间隔(functional margin)：样本 $(x^{(\ell)}, y^{(\ell)})$ 关于超平面的函数间隔定义为

$$\hat{\gamma}^{(\ell)} = y^{(\ell)}(w^\top x^{(\ell)} + w_0).$$

它 > 0 表示分类正确，其绝对值反映"确信度"。但函数间隔有个缺陷：把 (w, w_0) 同时放大 k 倍，超平面不变，函数间隔却变成 k 倍。

几何间隔(geometric margin)：把函数间隔除以 $\|w\|$ 归一化，得到点到超平面的真实(欧氏)距离

$$\gamma^{(\ell)} = \frac{y^{(\ell)}(w^\top x^{(\ell)} + w_0)}{\|w\|}.$$

几何间隔对参数缩放不敏感，才是"真正的间隔"。

> 正因为函数间隔可任意缩放，SVM 才能不失一般性地把**间隔边界上最近的点**处的函数间隔固定为 1，从而把"最大化间隔"转成对 $\|w\|$ 的优化。

函数间隔　几何间隔　归一化

4 最大化间隔的推导

采用**归一化(规范化)** 约定：令最近样本处满足 $w^\top x + w_0 = \pm 1$，即正负类的边界平面为

$$w^\top x + w_0 = +1 \quad \text{和} \quad w^\top x + w_0 = -1.$$

设 $x^{(1)}, x^{(2)}$ 分别是两侧最近的点，满足 $w^\top x^{(1)} + w_0 = +1$, $w^\top x^{(2)} + w_0 = -1$。两式相减得

$$w^\top (x^{(1)} - x^{(2)}) = 2.$$

两个平面间的距离(即间隔)为

$$\text{margin} = \frac{w^\top (x^{(1)} - x^{(2)})}{\|w\|} = \frac{2}{\|w\|}.$$

- **最大化间隔** $\frac{2}{\|w\|}$ 等价于**最小化** $\frac{1}{2}\|w\|^2$(取平方与系数 $\frac{1}{2}$ 是为了求导方便且不改变最优解)。
- 这就把几何问题转化成一个清晰的优化目标。

最大间隔　推导　几何

5 约束条件与原始优化问题

我们希望所有样本都被正确分类且落在边界之外：

$$w^\top x^{(\ell)} + w_0 \geq +1 \text{ 若 } y^{(\ell)} = +1, \qquad w^\top x^{(\ell)} + w_0 \leq -1 \text{ 若 } y^{(\ell)} = -1.$$

两条可统一写成一个不等式约束：

$$y^{(\ell)}(w^\top x^{(\ell)} + w_0) \geq 1, \quad \ell = 1, \ldots, N.$$

于是 SVM 的原始优化问题(primal problem)为：

$$\min_{w, w_0} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y^{(\ell)}(w^\top x^{(\ell)} + w_0) \geq 1,\ \ell = 1, \ldots, N.$$

- 这是一个二**次规划(Quadratic Programming, QP)** 问题，目标函数二次、约束线性，属于**凸优化**，存在全局最优解。
- 复杂度与输入维度 d 相关。

原始问题 二次规划 凸优化

6 合页损失(Hinge Loss)与无约束目标

对单个样本，模型给出得分 $g(x) = w^\top x + w_0$，设 $y \in \{-1, +1\}$ 为真实标签：

- 若 $y(w^\top x + w_0) < 1$：需要把它推到间隔之外，代价为 $1 - y(w^\top x + w_0)$；
- 若 $y(w^\top x + w_0) \geq 1$(间隔外的安全样本，离群点也包含在内)：无需再优化，代价为 0。

两种情况可统一为**合页损失(hinge loss)**：

$$\ell(w, w_0 \mid x, y) = \max\big(0,\ 1 - y(w^\top x + w_0)\big).$$

整个数据集的目标函数(带 L_2 正则)：

$$L(w, w_0 \mid D) = \frac{1}{N}\sum_{\ell=1}^{N} \max\left(0, 1 - y^{(\ell)}(w^\top x^{(\ell)} + w_0)\right) + \frac{\lambda}{2}\|w\|^2.$$

- 第一项是**经验损失**(惩罚违例样本)，第二项 $\frac{\lambda}{2}\|w\|^2$ 是**正则项**(对应最大化间隔)，λ 是正则系数。
- 合页损失在 $yg(x) = 1$ 处有"折点"，不可导，但它是**凸**函数，可用(次)梯度下降优化。
- 这样就把带约束的 QP 改写成了一个**无约束**的优化目标，便于直接用梯度下降求解。

合页损失 正则化 目标函数

7 梯度下降优化 SVM

把 SVM 写成无约束的合页损失目标后，可用**(次)梯度下降**直接优化。对每个分量 w_j $(j=0,\dots,d)$，子梯度为：

$$\frac{\partial L}{\partial w_j}=\begin{cases}\lambda w_j & \text{若 } y^{(\ell)}(w^\top x^{(\ell)}+w_0)\geq 1\\ \lambda w_j - y^{(\ell)}x_j^{(\ell)} & \text{否则}\end{cases}$$

更新规则：$w^{new}=w^{old}-\eta\dfrac{\partial L}{\partial w}$，其中 η 为学习率。

算法(每个样本逐个更新)：

1. 随机初始化 $w_j\leftarrow \text{rand}(-0.01,0.01)$。
2. 重复直到收敛：对每个样本 ℓ 计算得分 $a=\sum_j w_j x_j^{(\ell)}$。
3. 默认梯度 $\Delta w_j\leftarrow \lambda w_j$，$\Delta w_0\leftarrow 0$。
4. 若 $y^{(\ell)}a<1$(违例)：$\Delta w_j\leftarrow \Delta w_j - y^{(\ell)}x_j^{(\ell)}$，$\Delta w_0\leftarrow \Delta w_0 - y^{(\ell)}$。
5. 更新 $w_j\leftarrow w_j-\eta\,\Delta w_j$。

> 直觉：分对且够远的样本($yg(x)\geq 1$)只受正则项 λw 收缩；分错或落在间隔内的样本($yg(x)<1$)会额外把 w 往正确方向推动。这种"按合页损失更新"的方式正是随机次梯度下降求解 SVM 的核心。注意偏置 w_0 不加正则。

梯度下降　次梯度　算法

8 下一步：从 SVM 到神经网络

本课的 SVM 走的是一条清晰的线性路线：**最大间隔 → 合页损失 → 梯度下降**。它和感知机、逻辑回归一样，本质都是**线性分类器**——决策边界是一个超平面 $w^\top x+w_0=0$。

- 线性模型的局限在于"**非线性的诅咒**(the curse of nonlinearity)"：当两类数据无法用任何一条直线/超平面分开时，纯线性模型就无能为力。
- 解决思路之一是去逼近任意的非线性函数，这正是**神经网络(Neural Networks)** 的出发点——下一讲的主题。

> 一句话承上启下：SVM 把"找最好的线性边界"做到了极致，而要突破线性的边界，我们需要更强的非线性模型。

承上启下　非线性　神经网络

9 📝 例题1：手算函数间隔与几何间隔

我们在二维平面上放 4 个样本，用一个**已经给定的超平面**来体会"间隔"是怎么算的。

数据(共 4 个点)：

点	坐标 $x=(x_1,x_2)$	标签 y
A	$(3,3)$	$+1$
B	$(4,3)$	$+1$
C	$(1,1)$	-1
D	$(0,2)$	-1

给定超平面： $w=(1,1)$，$w_0=-4$，即 $g(x)=x_1+x_2-4=0$。

第 1 步：算每个点的得分 $g(x)=w^\top x+w_0$。

为什么先算它？因为函数间隔、几何间隔、判类全都建立在这个得分上。

- A：$g=3+3-4=2$
- B：$g=4+3-4=3$
- C：$g=1+1-4=-2$
- D：$g=0+2-4=-2$

第 2 步：算函数间隔 $\hat{\gamma}=y\cdot g(x)$。

为什么要乘 y？因为正类希望 $g>0$、负类希望 $g<0$，乘上标签后"分对"就统一表现为**正数**，数值越大越"确信"。

- A：$\hat{\gamma}=(+1)(2)=2$
- B：$\hat{\gamma}=(+1)(3)=3$
- C：$\hat{\gamma}=(-1)(-2)=2$
- D：$\hat{\gamma}=(-1)(-2)=2$

> 4 个全为正 → 4 个点**全部分类正确**。

第 3 步：算几何间隔 $\gamma=\dfrac{\hat{\gamma}}{\|w\|}$**(点到直线的真实距离)**。

为什么要除以 $\|w\|$？因为函数间隔会随 (w,w_0) 整体缩放而变大变小，不是真实距离；除以法向量长度归一化后才是欧氏距离。

这里 $\|w\|=\sqrt{1^2+1^2}=\sqrt{2}\approx 1.414$。

- A：$\gamma=2/\sqrt{2}=\sqrt{2}\approx 1.414$
- B：$\gamma=3/\sqrt{2}\approx 2.121$
- C：$\gamma=2/\sqrt{2}=\sqrt{2}\approx 1.414$
- D：$\gamma=2/\sqrt{2}=\sqrt{2}\approx 1.414$

第 4 步：看哪些点离边界最近。

最小函数间隔是 2，由 **A、C、D** 取得；B 的函数间隔为 3，离边界更远。**间隔边界上最近的这些点**正是决定间隔的关键，下一题就用它们把函数间隔归一化为 1。

例题

10 📝 例题2：归一化函数间隔为 1，算间隔宽度 $2/\|\mathbf{w}\|$

接着例题1的数据。标准 SVM 习惯让**最近点处的函数间隔恰好等于 1**，这样间隔宽度就能写成简洁的 $\frac{2}{\|w\|}$。我们来手动"归一化"一次。

第 1 步：为什么可以随意缩放?

把 (w, w_0) 同时乘以常数 k，超平面 $kw^\top x + kw_0 = 0$ 和原来 $w^\top x + w_0 = 0$ 是**同一条直线**(几何间隔不变)，只有函数间隔被放大 k 倍。所以我们有自由去选一个 k，让最近点的函数间隔正好是 1。

第 2 步：选择缩放系数 k。

例题1 中最近点的函数间隔是 2。要把它变成 1，就把 (w, w_0) 全部**除以 2**(即 $k = \frac{1}{2}$):

$$w' = \left(\tfrac{1}{2}, \tfrac{1}{2}\right), \qquad w_0' = -2.$$

新超平面 $\frac{1}{2}x_1 + \frac{1}{2}x_2 - 2 = 0$，化简还是 $x_1 + x_2 - 4 = 0$，**没变**。

第 3 步：验证最近点函数间隔=1。

- A：$g' = \frac{1}{2}(3) + \frac{1}{2}(3) - 2 = 1$，$\hat{\gamma}' = (+1)(1) = 1$ √
- C：$g' = \frac{1}{2}(1) + \frac{1}{2}(1) - 2 = -1$，$\hat{\gamma}' = (-1)(-1) = 1$ √

最近点恰好落在 $g'(x) = \pm 1$ 这两条"边界线"上。

第 4 步：算间隔宽度 $\frac{2}{\|w'\|}$。

现在 $\|w'\| = \sqrt{(\frac{1}{2})^2 + (\frac{1}{2})^2} = \sqrt{\frac{1}{2}} = \frac{1}{\sqrt{2}} \approx 0.707$。

$$\text{margin} = \frac{2}{\|w'\|} = \frac{2}{1/\sqrt{2}} = 2\sqrt{2} \approx 2.83.$$

> **结论**：归一化后用公式 $\frac{2}{\|w'\|}$ 得到间隔宽度 ≈ 2.83。$\|w'\|$ 越小 → 间隔越宽 → 所以"最大化间隔"就等价于"**最小化** $\frac{1}{2}\|w'\|^2$"。

例题

11 📝 例题3：一步梯度下降更新参数

这道例题把"梯度下降优化 SVM"落到一次具体的手算上，体会**违例样本**是怎样把参数往正确方向推的。

设置：当前参数 $w=(0.5,0.5)$，$w_0=-1$，正则系数 $\lambda=0.1$，学习率 $\eta=0.1$。来一个样本 $x=(1,1)$，标签 $y=+1$。

第 1 步：算得分与函数间隔，判断是否违例。

$$g(x)=0.5(1)+0.5(1)-1=0,\qquad y\,g(x)=(+1)(0)=0.$$

因为 $y\,g(x)=0<1$，这是一个**违例样本**(落在间隔内)，要用违例分支的梯度。

第 2 步：写出各分量的(次)梯度。

违例时 $\dfrac{\partial L}{\partial w_j}=\lambda w_j-y\,x_j$，偏置 $\dfrac{\partial L}{\partial w_0}=-y$(不含正则)。

- w_1：$0.1(0.5)-(+1)(1)=0.05-1=-0.95$
- w_2：$0.1(0.5)-(+1)(1)=0.05-1=-0.95$
- w_0：$-(+1)=-1$

第 3 步：按 $w\leftarrow w-\eta\nabla$ 更新。

- $w_1\leftarrow 0.5-0.1(-0.95)=0.5+0.095=0.595$
- $w_2\leftarrow 0.5-0.1(-0.95)=0.595$
- $w_0\leftarrow -1-0.1(-1)=-1+0.1=-0.9$

第 4 步：验证间隔确实变大了。

用新参数重算该点得分：

$$g'(x)=0.595(1)+0.595(1)-0.9=0.29,\qquad y\,g'(x)=0.29>0.$$

函数间隔从 0 升到 0.29，朝着满足约束 $y\,g(x)\geq 1$ 的方向迈了一步。

> 直觉验证：违例样本把 w 沿 $+y\,x$ 方向推、把 w_0 沿 $+y$ 方向推，从而增大该点的 $y\,g(x)$；若样本本来就满足 $y\,g(x)\geq 1$，则只剩正则项 λw 让 w 缓慢收缩。

例题

第 8 讲　多层感知机

Multi-Layer Perceptron

本讲从单个感知机出发，揭示其无法处理线性不可分（如异或）问题的局限，进而引入隐含层与连续激活函数构造多层感知机（MLP）；核心是用链式法则与反向传播高效计算每个参数的梯度 $\frac{\partial \mathcal{L}}{\partial w}$，并讨论激活函数、损失函数、万能近似定理与深度学习的层次化表征思想。

1 人工神经网络的起源与神经元模型

人工神经网络（Artificial Neural Network, ANN）是**模拟人脑计算原理**而设计的机器学习模型。人脑拥有约 10^{11} 个神经细胞作为基本处理单元、约 10^4 量级的突触作为存储单元，具有并行处理、分布式计算/存储、对噪声和错误高容忍等特点。

神经科学家试图通过建模理解大脑，计算机科学家则借鉴大脑构建更强的计算系统，二者交汇催生了 ANN。

单个人工神经元的计算分两步：

- **输入函数（加权求和）**：$a = \sum_j w_j x_j$
- **激活函数**：$y = \sigma(a)$

其中 w_0 通常作为偏置（bias），对应固定输入 $x_0 = +1$。

神经网络　神经元　起源

2 感知机（Perceptron）

感知机由 Frank Rosenblatt 于 1957 年提出，是模拟生物神经元的最基本单位，也是"大脑仿造计划"的起点。它先对输入做线性加权求和 $a = w^T x$，再经过一个阈值（激活）函数得到输出。

感知机可用于两类任务：

- **分类模型**：$y = \text{sign}(w^T x + w_0)$，需要一个阈值函数作为激活函数

$$\sigma(a) = \begin{cases} +1 & \text{if } a > 0 \\ -1 & \text{otherwise} \end{cases}$$

决策规则为：若 $\sigma(w^T x) = 1$ 则选 C_1，否则选 C_2。

- **回归模型**：$y = w^T x + w_0$（无阈值函数，直接输出线性值）。

感知机本质上是一个**线性模型**，其决策边界 $x_1 w_1 + x_2 w_2 + w_0 = 0$ 是一条直线（超平面）。

感知机　线性分类器　激活函数

3 感知机与布尔函数：异或问题的挑战

布尔函数等价于一个二分类问题。单个感知机可以轻松实现 **AND**、**OR** 等线性可分的布尔函数（例如 AND 可用权重 $w_1 = 0.5, w_2 = 0.5, w_0 = -0.5$ 实现，几何上是一条能把 $(1,1)$ 与其它点分开的直线）。

但 1969 年 Minsky 和 Papert 从理论上证明：感知机**无法学习异或（XOR）**这样的逻辑。

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR 的正例与负例在平面上无法用一条直线分开，即**线性不可分**。这是感知机的根本局限，也使神经网络研究进入了一段"漫长的黑夜"。

感知机　异或　线性不可分

4 破局之道：分而治之与魔力隐含层

单个感知机无法处理非线性数据，但**多个感知机各自处理一部分线性可分数据、再将结果归总**，就能逼近复杂的非线性边界。每个隐含单元（hidden unit）负责一条线性决策边界。

关键洞察：复杂的非线性操作可以**分解为多个线性操作的组合**。例如异或可以写成若干合取式的析取：

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

这恰好对应一个带**隐含层**的多层结构：用两个神经元分别实现两个合取项，再用一个神经元做析取。于是"万事俱备"，只剩一个问题——**离散的阈值函数不可微**，无法用梯度方法训练。

隐含层　分而治之　异或

5 连续激活函数与 Sigmoid

为了让网络可微、可用梯度下降训练，需要把离散阈值函数换成**连续的非线性函数**，称为激活函数。

Sigmoid 函数是经典选择：

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

它把任意实数压缩到 $(0,1)$ 区间，是阈值函数的连续、可微版本。其输出还可解释为输入 x 属于类别 C_1 的**后验概率**。

引入连续激活函数有两层意义：

- **引入非线性**：若没有非线性激活，多层线性变换叠加仍等价于单层线性变换。
- **保证可微**：使反向传播得以进行。

> 提示：激活函数是赋予神经网络表达非线性能力的核心组件，没有它，再深的网络也只是线性模型。

激活函数　Sigmoid　非线性

6 常见激活函数：Sigmoid / Tanh / ReLU

为了给感知机引入非线性，需要引入连续的（阈值）激活函数。PPT 中出现的几种：

- **Sigmoid**：$\sigma(x) = \frac{1}{1+e^{-x}}$，输出 $(0,1)$，是阈值函数的连续、可微版本，输出可解释为后验概率。
- **Tanh**（双曲正切）：$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$，输出 $(-1,1)$。
- **ReLU**（修正线性单元）：$\mathrm{ReLU}(x) = \max(0,x)$，正区间导数恒为 1，常用于深度网络中缓解梯度消失。

激活函数 ReLU Tanh Sigmoid

7 多层感知机（MLP）结构

多层感知机（Multi-Layer Perceptron, MLP）是由多个感知机**逐层复合**而成的前向神经网络，包含三类层：

- **输入层（Input）**：仅接收输入，不做计算（no computation）。
- **隐含层（Hidden）**：实现从 d 维输入空间到 H 维隐空间的**非线性变换**（nonlinear transformation）。
- **输出层（Output）**：在新的 H 维空间中做一个**线性函数**。

以两层 MLP 为例，前向计算为：

$$a_h = \sum_j W_{hj} x_j, \quad z_h = \sigma(a_h), \quad y_i = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

直觉：隐含层先把原始输入"扭曲"到一个新的特征空间，使原本线性不可分的数据在新空间里变得线性可分，输出层再做线性划分。

MLP 前向传播 隐含层

8 训练目标与损失函数

设网络参数集为 $\theta = \{w_0, w_1, \ldots, w_n\}$，训练思路是按**梯度下降**迭代：$\theta^0 \to \theta^1 \to \theta^2 \to \cdots$，核心问题是**如何获得每个参数的梯度**。

给定数据集 $D = \{(x^{(\ell)}, r^{(\ell)})\}_{\ell=1}^{N}$，$x^{(\ell)}$ 为输入、$r^{(\ell)}$ 为目标输出，模型预测为 $y^{(\ell)}$。以回归为例，目标是最小化 **MSE 损失**：

$$\mathcal{L}(\theta \mid D) = \frac{1}{2} \sum_{\ell} \sum_{i} (r_i^{(\ell)} - y_i^{(\ell)})^2$$

由样本独立性，总损失是单样本损失的累加（或平均）：$\mathcal{L}(\theta) = \sum_{\ell=1}^{N} l^{(\ell)}(\theta)$，因此梯度也可分解：

$$\frac{\partial \mathcal{L}(\theta)}{\partial w} = \sum_{\ell=1}^{N} \frac{\partial l^{(\ell)}(\theta)}{\partial w}$$

单样本损失为 $l(x,r) = \frac{1}{2}\sum_i (r_i - y_i)^2$。分类任务常用交叉熵（Cross-Entropy）损失。

损失函数 MSE 梯度下降

9 链式法则（Chain Rule）

反向传播的数学基础是**复合函数求导的链式法则**。

- **情形一（链式串联）**：若 $y = g(x)$，$z = h(y)$，即 $\Delta x \to \Delta y \to \Delta z$，则

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

- **情形二（多路径汇合）**：若 $x = g(s)$，$y = h(s)$，$z = k(x, y)$，即 s 通过两条路径影响 z，则

$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

在 MLP 中，一个权重通过多个下游神经元影响最终损失，对应的正是情形二：**各条路径的贡献要相加**。这是反向传播中误差汇总的核心。

链式法则　求导　反向传播

10 直接求梯度的困难与反向传播思想

对深层 MLP **直接逐个推导**每个参数梯度（如 $\frac{\partial l}{\partial w_L} = \frac{\partial a_L}{\partial w_L}\frac{\partial y}{\partial a_L}\frac{\partial l}{\partial y}$）存在三大问题：

- **极其繁琐**：需要大量矩阵微积分推导。
- **缺乏灵活性**：一旦更换损失函数（如把 Softmax 换成 SVM），就得从头重推。
- **不可行**：对非常复杂的模型无法手算，因为不同层梯度之间存在错综复杂的依赖关系。

反向传播（Backpropagation）是高效计算神经网络梯度的方法。基本思想：把误差从网络最后端**逐层向前传播分配**——类比桌球，根据最终的偏差反推过程中每个环节的偏差。它用**动态规划**复用中间结果，避免重复递归计算。

反向传播　梯度　动态规划

11 反向传播：前向传递（Forward Pass）

对每个权重 w，其对损失的梯度可拆为两部分：

$$\frac{\partial l}{\partial w} = \frac{\partial a}{\partial w} \cdot \frac{\partial l}{\partial a}$$

其中 a 是该神经元的线性输出。

前向传递负责计算 $\frac{\partial a}{\partial w}$ 这一"容易"的部分：因为线性单元 $a = \sum_j w_j x_j$，所以

$$\frac{\partial a}{\partial w_j} = x_j$$

即 $\frac{\partial a}{\partial w}$ **等于与该权重相连的输入**（前一层的输出 z）。由于每个神经元的线性部分简单且固定，可以在一次前向运行中**预先算出并缓存**所有 $\frac{\partial a}{\partial w} = z$。

反向传播　前向传播　链式法则

12 反向传播：反向传递与误差信号 δ

反向传递负责计算"困难"的部分 $\frac{\partial l}{\partial a}$，定义**误差信号** $\delta_j \equiv \frac{\partial l}{\partial a_j}$。

它从输出层向输入层递归计算：

- **输出层**：$\delta^L = \sigma'(a^L) \cdot \nabla_y \mathcal{L}$（例如回归时 $\delta = (r - y) \cdot \sigma'(a)$）。
- **隐含层（非输出层）**：误差由所有下游单元汇总后乘以本层激活导数

$$\delta^l = \sigma'(z^l) \cdot (W^{l+1})^T \delta^{l+1}$$

对单个神经元即 $\delta_j = \sigma'(a_j) \sum_k w_{jk} \delta_k$。

最终每个权重的梯度为"误差 × 输入信号"：$\frac{\partial l}{\partial w_{ij}} = \delta_j z_i$，参数更新为：

$$w'_{ij} = w_{ij} + \eta\, \delta_j z_i$$

（注意 PPT 中按 $-\nabla$ 形式书写，正负取决于误差 δ 的符号约定。）

算法三步：1) 前向传播算出各单元激活；2) 反向传播算出各层误差 δ；3) 用 δ 与输入更新参数。

反向传播　误差信号　参数更新

13 梯度消失

反向传播中，浅层的梯度是各层激活导数的**连乘**：$\delta^l = \sigma'(z^l)(W^{l+1})^T \delta^{l+1}$。

梯度消失（Vanishing Gradient）：当激活导数较小（如 Sigmoid 的导数最大仅 0.25）时，多层连乘后浅层梯度指数级趋近于 0，导致前面的层几乎学不动。这正是前向 DNN "难训练"（too many parameters、gradient vanishing 等）的主要原因之一。

缓解的一个直接思路：把激活函数换成 **ReLU**，其正区间导数恒为 1，连乘时不会衰减。

梯度消失　深度网络　难训练

14 万能近似定理与"深"的优势

万能近似定理（Universal Approximation Theorem）：只要隐含神经元足够多，一个**仅含单隐含层**的神经网络就能以任意精度逼近任意连续函数 $f : \mathbb{R}^N \to \mathbb{R}^M$。也就是说，浅层网络在理论上已经能表示"任何函数"。

那既然浅层够用，为何还要深？在**参数总量相同**的情况下，**深层网络通常更有效**，原因在于**模块化（Modularization）+ 分工协作**：

- 低层学习简单、共享的特征（如"长发/短发"分类器可被多个高层分类器复用）；
- 高层在低层基础上组合出更抽象、更全局、更不变的特征；
- 每个基础模块只需较少数据即可训练好，整体更**数据高效**、更易泛化。

类比：不要把所有功能塞进主函数，而应拆成可复用的子模块，逐层组合。

万能近似定理　深度　模块化

15 深度学习：层次化表征与端到端学习

深度学习（Deep Learning, DL），又称深度表征学习，是采用深层神经网络实现对数据**深层次特征的自动提取与表达**的机器学习技术。三者关系：AI ⊃ 机器学习 ⊃ 深度学习。

核心思想对比：

- **传统机器学习**：人工设计/工程化特征（hand-crafted features）+ 可训练的分类器。
- **深度学习**：**可训练的特征 + 可训练的分类器**，即一个**可训练特征变换的层级结构**。每个模块把输入表征变换为更高层级的表征——低层特征局部且被各类别共享，高层特征更全局、更不变。

层次化表征示例：图像 `像素→边缘→纹理基元→图案→部件→物体`；文本 `字符→词→短语→从句→句子→段落→文档`。

端到端学习（End-to-end Learning）：模型直接学习从原始输入到输出的**所有操作**，省去人工的预处理与特征工程流水线。

典型深度网络包括：全连接网络、卷积神经网络（CNN）、循环神经网络（RNN）、Transformer 等。

深度学习 层次化表征 端到端

16 前向神经网络的缺点与典型深度网络

前向（全连接）深度神经网络的损失函数可选 Square error、Cross-entropy、Hinge loss、Ranking loss 等；**训练**用反向传播；**激活函数**常用 Sigmoid、ReLU。

但前向 DNN 通常**不单独作为模型使用**，其缺点包括：

- **难训练**：参数过多（too many parameters）、梯度消失（gradient vanishing）等；
- **对输入敏感**。

因此它经常用作中间层、作为辅助单元（例如做非线性变换）。

现实中常用的深度网络还有：卷积神经网络（CNN）、循环神经网络（RNN）、Transformer 等。

前向网络 缺点 深度网络

17 📝 例题1：手算一次前向传播（2输入-2隐层-1输出）

我们来搭一个**最小的 MLP**，亲手把每个神经元的数算出来。

网络结构：2 个输入 → 2 个隐层神经元 → 1 个输出神经元，隐层和输出层都用 Sigmoid 激活。

给定参数（权重和偏置）：

- 输入：$x_1 = 1,\ x_2 = 0$
- 隐层神经元 h_1：$w_{11} = 0.5,\ w_{12} = -0.5,\ b_1 = 0$
- 隐层神经元 h_2：$w_{21} = 1.0,\ w_{22} = 1.0,\ b_2 = -1.0$
- 输出神经元 o：$v_1 = 1.0,\ v_2 = -1.0,\ b_o = 0.5$

> 记住两步走：先**加权求和**算 a，再**激活**算 $z = \sigma(a)$。Sigmoid：$\sigma(z) = \dfrac{1}{1+e^{-z}}$。

第1步：算隐层 h_1（先求和，再激活）

$$a_1 = w_{11}x_1 + w_{12}x_2 + b_1 = 0.5 \times 1 + (-0.5) \times 0 + 0 = 0.5$$

激活：$z_1 = \sigma(0.5) = \dfrac{1}{1+e^{-0.5}} = \dfrac{1}{1+0.607} \approx 0.622$。

为什么：a_1 是这条神经元的线性输出，Sigmoid 把它压到 $(0,1)$，得到隐层第一个特征值。

第2步：算隐层 h_2

$$a_2 = 1.0 \times 1 + 1.0 \times 0 + (-1.0) = 0$$

激活：$z_2 = \sigma(0) = \dfrac{1}{1+1} = 0.5$。

为什么：偏置 -1 把求和拉到 0，而 $\sigma(0)$ 恰好是 0.5（Sigmoid 的中点）。

第3步：算输出 o（用上一层的 z_1, z_2 当输入）

$$a_o = v_1 z_1 + v_2 z_2 + b_o = 1.0 \times 0.622 + (-1.0) \times 0.5 + 0.5 = 0.622 - 0.5 + 0.5 = 0.622$$

激活：$y = \sigma(0.622) = \dfrac{1}{1+e^{-0.622}} = \dfrac{1}{1+0.537} \approx 0.651$。

结论：这一次前向传播，信号从输入 $(1,0)$ 一路算到输出 $y \approx 0.651$。

- 隐层值：$z_1 \approx 0.622,\ z_2 = 0.5$
- 输出值：$y \approx 0.651$

> 关键体会：每一层都是"**线性加权 + 非线性激活**"的重复。前一层的输出 z 就是后一层的输入，层层向前传，这就是"前向传播"。

例题

18 📝 例题2：同一组输入，ReLU 和 Sigmoid 对比

激活函数到底有啥不同？我们拿**同样几个输入**，分别送进 ReLU 和 Sigmoid，把数算出来对比。

公式：

- $\mathrm{ReLU}(x) = \max(0, x)$（负的归零，正的原样保留）
- $\sigma(x) = \dfrac{1}{1 + e^{-x}}$（任意实数压到 $(0, 1)$）

逐个计算（取 $x = -2,\ -0.5,\ 0,\ 1,\ 3$）：

输入 x	$\mathrm{ReLU}(x) = \max(0, x)$	$\sigma(x) = 1/(1 + e^{-x})$
-2	$\max(0, -2) = 0$	$1/(1 + e^{2}) = 1/8.39 \approx 0.119$
-0.5	$\max(0, -0.5) = 0$	$1/(1 + e^{0.5}) = 1/2.649 \approx 0.378$
0	$\max(0, 0) = 0$	$1/(1 + 1) = 0.5$
1	$\max(0, 1) = 1$	$1/(1 + e^{-1}) = 1/1.368 \approx 0.731$
3	$\max(0, 3) = 3$	$1/(1 + e^{-3}) = 1/1.050 \approx 0.953$

逐行解读（为什么）：

1. $x = -2$：ReLU 直接输出 0（负区间全归零）；Sigmoid 给出 0.119，仍是个小正数。
2. $x = 0$：ReLU 输出 0，Sigmoid 输出 0.5——这正是 Sigmoid 的"中点"。
3. $x = 3$：ReLU 原样输出 3（不设上限）；Sigmoid 输出 0.953，已接近上界 1，**开始饱和**。

两点对比结论：

- **范围**：Sigmoid 永远落在 $(0, 1)$；ReLU 正区间无上限、可以很大。
- **饱和 vs 不饱和**：Sigmoid 在 x 很大/很小时几乎压平（饱和），导数趋近 0；ReLU 正区间斜率恒为 1，不饱和。

> 这就是为什么隐层默认优先用 ReLU：它在正区间导数为 1，反向传播时梯度不容易被"压扁"，能缓解梯度消失（见例题4）。

例题

19 📝 例题3：用链式法则做一步反向传播并更新权重

我们用一个**极简网络**，亲手用链式法则算出某个权重的梯度，再更新它。

网络：1 输入 → 1 隐层神经元（Sigmoid）→ 1 输出神经元（Sigmoid），回归任务，MSE 损失。

给定：

- 输入 $x = 1$，目标 $r = 1$
- 隐层权重 $w = 0.5$（这就是我们要更新的权重），隐层：$a_1 = wx$，$z_1 = \sigma(a_1)$
- 输出权重 $v = 1.0$，输出：$a_2 = vz_1$，$y = \sigma(a_2)$
- 损失 $l = \frac{1}{2}(r-y)^2$，学习率 $\eta = 0.1$

第1步：前向传播，把各值算出来

- $a_1 = 0.5 \times 1 = 0.5 \Rightarrow z_1 = \sigma(0.5) \approx 0.622$
- $a_2 = 1.0 \times 0.622 = 0.622 \Rightarrow y = \sigma(0.622) \approx 0.651$
- 损失 $l = \frac{1}{2}(1-0.651)^2 = \frac{1}{2}(0.349)^2 \approx 0.0609$

第2步：写出链式法则（w 通过 $a_1 \to z_1 \to a_2 \to y \to l$ 影响损失）

$$\frac{\partial l}{\partial w} = \frac{\partial l}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_1} \cdot \frac{\partial z_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w}$$

第3步：逐项求导（用到 $\sigma'(a) = \sigma(a)(1-\sigma(a)) = z(1-z)$）

1. $\frac{\partial l}{\partial y} = -(r-y) = -(1-0.651) = -0.349$
2. $\frac{\partial y}{\partial a_2} = y(1-y) = 0.651 \times 0.349 \approx 0.227$
3. $\frac{\partial a_2}{\partial z_1} = v = 1.0$
4. $\frac{\partial z_1}{\partial a_1} = z_1(1-z_1) = 0.622 \times 0.378 \approx 0.235$
5. $\frac{\partial a_1}{\partial w} = x = 1$

第4步：连乘得到梯度

$$\frac{\partial l}{\partial w} = (-0.349) \times 0.227 \times 1.0 \times 0.235 \times 1 \approx -0.0186$$

第5步：梯度下降更新权重（沿负梯度方向）

$$w_{\text{new}} = w - \eta \frac{\partial l}{\partial w} = 0.5 - 0.1 \times (-0.0186) = 0.5 + 0.00186 \approx 0.5019$$

结论与解读：

- 梯度是负的 $\Rightarrow$ 增大 w 能减小损失，所以 w 被**调大**了一点点。
- 这一步背后正是"误差信号 δ × 输入"：输出层 $\delta_2 = (r-y)\sigma'(a_2)$，隐层把它乘 v 再乘 $\sigma'(a_1)$ 往回传，最后乘输入 x。

> 注意 PPT 中写成 $w' = w + \eta\,\delta z$，那里的 δ 用了 $(r-y)$ 的符号约定；本例用 $\partial l/\partial w$（含 $-(r-y)$），减去梯度，二者结果一致。

例题

20 📝 例题4：用具体数演示梯度消失（Sigmoid 导数连乘）

为什么深层网络"前面的层学不动"？我们用数字演示**梯度消失**。

核心事实：Sigmoid 的导数 $\sigma'(a) = \sigma(a)(1-\sigma(a))$，**最大值只有 0.25**（在 $a=0$ 处取得，因为 $0.5 \times 0.5 = 0.25$）。

反向传播时，浅层的梯度是各层导数的**连乘**。假设有一个 5 层深的网络，每层激活导数都取理想最大值 0.25，并且为简化假设权重都是 1。那么从输出传到第 1 层，误差信号要连乘 5 个 0.25：

逐层连乘（每往前一层，乘一个 ≤ 0.25）：

- 第 5 层（最靠近输出）：$\delta_5 \propto 1$（基准）
- 第 4 层：$1 \times 0.25 = 0.25$
- 第 3 层：$0.25 \times 0.25 = 0.0625$
- 第 2 层：$0.0625 \times 0.25 = 0.015625$
- 第 1 层：$0.015625 \times 0.25 \approx 0.0039$

也就是：

$$\text{第1层梯度} \propto (0.25)^5 = \frac{1}{4^5} = \frac{1}{1024} \approx 0.00098$$

更现实的情况：实际神经元很少恰好在 $a=0$，导数往往远小于 0.25。若每层导数只有 0.1，则 $(0.1)^5 = 10^{-5}$，第 1 层梯度小到几乎为 0。

结论与解读：

- 即便用"最理想"的 0.25，5 层后梯度也只剩约 $\frac{1}{1000}$；层数越多衰减越狠，呈**指数级**趋近 0。
- 梯度 ≈ 0 意味着 $w_{\text{new}} = w - \eta \cdot (\approx 0) \approx w$，前面的层几乎**不更新**，这就是"梯度消失"。

> 缓解办法：把激活换成 **ReLU**（正区间导数恒为 1，连乘 $1 \times 1 \times \cdots = 1$ 不衰减）。对比一下：$(0.25)^5 \approx 0.001$ vs $(1)^5 = 1$，差距一目了然。

例题

第 9 讲

PyTorch实战

PyTorch

本讲介绍深度学习框架PyTorch的核心机制，包括Tensor张量运算、基于动态计算图的autograd自动求导、用nn.Module定义模型、损失函数与优化器、训练循环(forward/backward/step)、GPU加速以及Dataset/DataLoader数据加载，并以一个MLP回归实例串联完整流程。

1 PyTorch概述与核心特性

PyTorch是一个Python深度学习框架，提供两大高级特性：

1. **Tensor计算**：类似NumPy的多维数组运算，但支持强大的**GPU加速**。
2. **神经网络**：构建在**tape-based autograd（基于磁带的自动求导）**系统之上。

PyTorch相比早期TensorFlow的优势：

- **More Pythonic（命令式/imperative）**：代码即时执行（eager），灵活可控。
- **类NumPy语法、面向对象设计**：代码直观干净。
- **易于调试**：即时错误反馈，可直接用Python调试工具（如`pdb`）。
- **网络搭建直观**：write code as the network works，按forward/backward组织，便于组装CNN、MLP等经典结构。

> 一句话：PyTorch = NumPy + GPU + 自动求导 + 神经网络模块。

PyTorch　概述　特性

2 PyTorch vs TensorFlow

两大框架在多个维度上的对比：

维度	PyTorch	TensorFlow
计算图	**动态计算图**（运行时构建，灵活修改）	默认静态图（预编译优化），2.x也支持Eager
API设计	Pythonic、与NumPy兼容、直观易调试	早期API复杂，2.x用Keras简化
性能	单GPU高效，分布式持续改进（`torch.distributed`）	大规模分布式强，支持TPU/XLA
社区	学术研究主导（约70%顶会论文），NLP活跃（Hugging Face）	工业部署生态完善（Serving/Lite/JS、TFX）
部署	TorchServe/JIT/ONNX	原生端到端部署，生产成熟度高
硬件	CUDA/ROCm，TPU需额外配置	深度集成Google TPU、TensorRT

核心区别：PyTorch采用**动态图（define-by-run）**，TensorFlow早期采用**静态图（define-and-run）**。

对比　TensorFlow　动态图

3 安装与Hello PyTorch

PyTorch可从官网 https://pytorch.org 按系统/CUDA版本选择安装命令，或直接在Colab中使用（无需本地配置）。

装好后，最简单的“Hello PyTorch”就是导入并打印版本号，确认安装成功：

```
>>> import torch
>>> print(torch.__version__)
1.2.0
>>> # Happy!!
```

要点：

- `import torch`：导入PyTorch主包。
- `torch.__version__`：查看当前安装的PyTorch版本，是排查环境问题（如版本不匹配）的第一步。

安装 import version

4 Tensor张量：基本数据结构

Tensor是PyTorch中最基本的数据结构（类似NumPy的`ndarray`，但能在GPU上运算并参与自动求导）。

创建张量：

```
import torch

# 从Python列表创建，可指定dtype
x = torch.tensor([0.1, 0.2, 0.3], dtype=torch.float)
W = torch.tensor([[0.1, 0.2, 0.3],
                  [0.4, 0.1, 0.7],
                  [0.8, 0.1, 0.2]])

# 创建全0张量（指定形状与dtype）
r = torch.zeros([1, 3], dtype=torch.long)
rnd = torch.rand(2, 3)   # [0,1)均匀分布
```

Tensor与NumPy互转（共享内存，零拷贝）：

```
import numpy as np

x_np = np.array([0.1, 0.2, 0.3])
x_torch = torch.from_numpy(x_np)   # numpy -> tensor
back = x_torch.numpy()             # tensor -> numpy
```

> 常用属性：`x.shape` / `x.size()` 查看形状，`x.dtype` 查看类型，`x.view(-1, d)` 改变形状（MLP预测时即用其把数据整理成`(样本数, d)`）。

Tensor 张量 NumPy

5 计算图与requires_grad

PyTorch的Tensor在后台会**隐式构建计算图（computation graph）**，记录张量之间的运算关系，为反向传播做准备。

要让某个张量参与梯度计算，需设置`requires_grad=True`，PyTorch才会为它保留计算图：

```
a = torch.tensor(2.0, requires_grad=True)
b = torch.tensor(1.0, requires_grad=True)
c = a + b      # c = 3
d = b + 1      # d = 2
e = c * d      # e = 6
print(c)  # tensor(3., grad_fn=<AddBackward0>)
print(d)  # tensor(2., grad_fn=<AddBackward0>)
print(e)  # tensor(6., grad_fn=<MulBackward0>)
```

关键点：

- 每个由运算得到的张量都带有`grad_fn`，它记录了产生该张量的运算（如`AddBackward0`、`MulBackward0`），这就是构成计算图的节点。
- **叶子节点（leaf）**是用户直接创建、`requires_grad=True`的张量（如`a`、`b`），梯度最终会累积到它们的`.grad`中。
- PyTorch是**动态图**：图在前向运算时即时构建，每次forward都会重新构建。

计算图 requires_grad grad_fn

6 autograd自动求导与backward()

PyTorch的**autograd**会自动沿计算图反向传播，一次性计算所有梯度。对标量输出的张量调用 `.backward()` 即可。

标量函数求导，例如 $f(x) = (x-2)^2$，求 $f'(1)$：

```
def f(x):
    return (x - 2) ** 2

x = torch.tensor([1.0], requires_grad=True)
y = f(x)
y.backward()          # 对叶子变量反向传播
print(x.grad)         # tensor([-2.])，即 f'(1)=2*(1-2)=-2
```

向量函数同理，例如 $g(w) = 2w_1 w_2 + w_2\cos(w_1)$，求 $\nabla_w g$：

```
def g(w):
    return 2 * w[0] * w[1] + w[1] * torch.cos(w[0])

w = torch.tensor([np.pi, 1.0], requires_grad=True)
z = g(w)
z.backward()
print(w.grad)   # tensor([2.0000, 5.2832])，与解析解一致
```

要点：

- 只能对**标量**直接 `backward()`；若输出是向量，需传入与之同形的 `gradient` 参数。
- 反向传播后梯度存于叶子张量的 `.grad` 属性中。
- 再次 `backward()` 前若不清零，梯度会累加到 `.grad` 上（详见优化器一节）。

autograd　backward　自动求导

7 用nn.Module定义模型

在PyTorch中，机器学习模型通过**继承 `torch.nn.Module`** 来定义。你需要：

1. 在 `__init__` 中定义所用的子模块（层）。
2. **自定义 `forward()` 函数**，描述模型的前向计算图。

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self, d_in, d_hidden, d_out):
        super().__init__()
        self.fc1 = nn.Linear(d_in, d_hidden)
        self.act = nn.Tanh()
        self.fc2 = nn.Linear(d_hidden, d_out)

    def forward(self, x):
        x = self.act(self.fc1(x))
        return self.fc2(x)

model = MyModel(1, 32, 1)
y_hat = model(x)   # 等价于 model.forward(x)
```

搭建模型常用的PyTorch子模块：

- **Basic Modules**：如 `nn.Linear`（全连接层）。
- **Functions**：如各种激活函数 `nn.Tanh`、`nn.ReLU`。
- **Containers**：如 `nn.Sequential`（顺序容器）、concat等。
- **其他现成模型**。

> 注意：调用模型用 `model(x)` 而非 `model.forward(x)`，前者会触发hook等机制。

nn.Module　forward　模型定义

8 nn.Sequential顺序容器

`nn.Sequential` 是一个**顺序容器**，按顺序把各层串起来，无需手写 `forward`，适合简单的前馈结构。

用 `nn.Sequential` 定义一个两隐层的MLP：

```
n_hidden_1 = 32
n_hidden_2 = 32
d = 1
d_out = 1

neural_network = nn.Sequential(
    nn.Linear(d, n_hidden_1),
    nn.Tanh(),
    nn.Linear(n_hidden_1, n_hidden_2),
    nn.Tanh(),
    nn.Linear(n_hidden_2, d_out)
)

y_hat = neural_network(X)    # 数据依次流经每一层
```

- `nn.Linear(in_features, out_features)` 实现 $y = xW^T + b$。
- 容器会自动注册其中各层的参数，可用 `neural_network.parameters()` 取得，交给优化器。

nn.Sequential　MLP　Linear

9 损失函数Loss

PyTorch在 `torch.nn` 中内置了许多常见损失函数，例如 **`MSELoss`（均方误差，用于回归）**和 **`CrossEntropyLoss`（交叉熵，用于分类）**。

```
import torch.nn as nn

mse_loss_fn = nn.MSELoss()
input  = torch.tensor([[0., 0, 0]])
target = torch.tensor([[1., 0, -1]])
loss = mse_loss_fn(input, target)
print(loss)   # tensor(0.6667)
```

计算说明：MSE对各元素差的平方取均值，$\frac{(0-1)^2+(0-0)^2+(0-(-1))^2}{3} = \frac{1+0+1}{3} = 0.6667$。

要点：

- 损失函数是一个可调用对象：`loss = loss_fn(prediction, target)`。
- `loss` 是一个标量张量，对它调用 `loss.backward()` 即可启动反向传播。
- 分类任务常用 `nn.CrossEntropyLoss`（内部含softmax，输入应为logits，target为类别索引）。

损失函数　MSELoss　CrossEntropyLoss

10 优化器Optimizer与zero_grad

`torch.optim` 提供多种基于梯度的优化方法，如**SGD（梯度下降）**、Adam等。优化器至少需要传入**模型参数**和**学习率**。

```
model = nn.Linear(1, 1)
optim = torch.optim.SGD(model.parameters(), lr=1e-2)
```

三条关键规则：

1. **优化器不会替你算梯度**，必须自己调用 `loss.backward()`。
2. **必须在backward()之前调用 `optim.zero_grad()` 清零梯度**——因为PyTorch默认是把新梯度**累加（inplace add）**到 `.grad` 上，而不是覆盖；不清零会导致梯度不断累积。
3. 算完梯度后调用 `optim.step()`，按梯度更新参数。

一次完整的参数更新：

```
X_simple = torch.tensor([[1.]])
y_label  = torch.tensor([[2.]])
loss_fn  = nn.MSELoss()

y_hat = model(X_simple)
loss = loss_fn(y_hat, y_label)
optim.zero_grad()   # 1) 清零梯度
loss.backward()     # 2) 反向传播算梯度
optim.step()        # 3) 更新参数
```

优化器 SGD zero_grad step

11 训练循环：forward, loss, backward, step

PyTorch训练的标准节奏（Rhythm）：

1. 用内置类**设计模型（Model）**；
2. 选定**损失函数与优化器**；
3. 进入**训练循环（forward → backward → update）**。

标准训练循环模板：

```
for epoch in range(n_epochs):
    y_hat = model(X)            # 1) forward 前向
    loss = loss_func(y_hat, y)  # 2) 计算损失
    optim.zero_grad()           # 3) 梯度清零
    loss.backward()             # 4) backward 反向传播
    optim.step()                # 5) step 更新参数
```

若不用优化器，手动更新参数也能说明本质（PPT中的forward/backward写法）：

```
w = torch.Tensor([1.0])   # 任意初值

def forward(x):
    return x * w

def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        l = loss(x_val, y_val)
        l.backward()
        # 手动按梯度更新权重
        w.data = w.data - 0.01 * w.grad.data
        # 更新后手动清零梯度，否则会累加
        w.grad.data.zero_()
```

> 记忆口诀：**zero_grad → forward/loss → backward → step**，四步缺一不可。

训练循环 forward backward step

12 Dataset与DataLoader

`torch.utils.data.Dataset`是表示数据集的抽象类。自定义数据集需继承它并实现两个方法：

- `__len__`：使`len(dataset)`返回样本数量；
- `__getitem__`：使`dataset[i]`返回第i个样本。

```
from torch.utils.data import Dataset
import pandas as pd

class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.sentences = pd.read_csv(csv_file)
    def __len__(self):
        return len(self.sentences)
    def __getitem__(self, idx):
        sample = self.sentences[idx]
        return sample.lower()
```

`torch.utils.data.DataLoader`把Dataset包装成可迭代对象，高效地：

- **batching**：按`batch_size`分批；
- **shuffling**：打乱数据；
- **并行加载**：用`num_workers`个子进程多进程读取。

```
from torch.utils.data import DataLoader

dataloader = DataLoader(my_dataset, batch_size=4,
                        shuffle=True, num_workers=4)
for i_batch, sample_batched in enumerate(dataloader):
    print(i_batch, sample_batched.size())
```

> 用DataLoader迭代取batch，比手写for循环更高效、更易并行。

Dataset　DataLoader　数据加载

13 detach()：从计算图分离张量

`detach()`：从计算图中分离张量，得到一个`requires_grad=False`的副本，常用于可视化或仅取数值、不再求导的场景。

在MLP实例的预测与可视化阶段，PyTorch正是用`detach()`切断梯度后再转`numpy`绘图：

```
# 预测与可视化时切断梯度，再转 numpy
y_hat = neural_network(X_grid)
plt.plot(X_grid.detach().numpy(), y_hat.detach().numpy(), 'r')
plt.show()
```

要点：

- 带`requires_grad=True`的张量参与了计算图，不能直接`.numpy()`，需先`.detach()`分离。
- `detach()`只取数值、不再追踪梯度，适合画图或记录中间结果。

detach　可视化　计算图

14 完整实例：用PyTorch实现MLP回归

把前面的概念串成一个完整流程：拟合 $y = 4\sin(\pi x)\cos(6\pi x^2)$。

1) 构造数据

```
d, n = 1, 200
X = torch.rand(n, d)
y = 4 * torch.sin(np.pi * X) * torch.cos(6 * np.pi * X ** 2)
```

2) 定义模型（两隐层MLP）

```
neural_network = nn.Sequential(
    nn.Linear(d, 32), nn.Tanh(),
    nn.Linear(32, 32), nn.Tanh(),
    nn.Linear(32, 1)
)
```

3) 训练

```
step_size, n_epochs = 0.05, 6000
loss_func = nn.MSELoss()
optim = torch.optim.SGD(neural_network.parameters(), lr=step_size)

for i in range(n_epochs):
    y_hat = neural_network(X)
    loss = loss_func(y_hat, y)
    optim.zero_grad()
    loss.backward()
    optim.step()
    if i % (n_epochs // 10) == 0:
        print(f'{i}, {loss.item()}')   # loss 随训练下降
```

4) 预测与可视化

```
X_grid = torch.from_numpy(np.linspace(0, 1, 50)).float().view(-1, d)
y_hat = neural_network(X_grid)
plt.scatter(X.numpy(), y.numpy())
plt.plot(X_grid.detach().numpy(), y_hat.detach().numpy(), 'r')
plt.show()
```

> 相比纯NumPy手写反向传播，PyTorch代码更少、更快——自动求导承担了梯度计算。

MLP　实例　回归　训练流程

15 📝 例子1：创建张量并与 NumPy 互转

我们从最基础的张量定义练起，复现 PPT 中“如何定义数据”的写法。

```
import torch
import numpy as np

# 1) 从 Python 列表创建张量，可指定 dtype
x = torch.tensor([0.1, 0.2, 0.3], dtype=torch.float)
W = torch.tensor([[0.1, 0.2, 0.3],
                  [0.4, 0.1, 0.7],
                  [0.8, 0.1, 0.2]])
print(x)
print(W)

# 2) 创建全 0 张量（指定形状与 dtype）
r = torch.zeros([1, 3], dtype=torch.long)
print(r)        # tensor([[0, 0, 0]])

# 3) 从 NumPy 数组创建张量
x_np = np.array([0.1, 0.2, 0.3])
x_torch = torch.from_numpy(x_np)   # numpy -> tensor
print(x_torch)

# 4) 张量再转回 NumPy
back = x_torch.numpy()             # tensor -> numpy
print(back)
```

逐步讲解（每步为什么）：

1. **`torch.tensor([...])`**：把 Python 列表变成张量；`dtype=torch.float` 指定浮点类型，因为训练几乎都用浮点。二维列表则得到矩阵 `W`。
2. **`torch.zeros([1,3], dtype=torch.long)`**：按给定形状创建全 0 张量，`long` 表示整型，常用于存放标签/索引。
3. **`torch.from_numpy(x_np)`**：把已有的 NumPy 数组包装成张量，两者通常共享底层内存（零拷贝）。
4. **`.numpy()`**：把张量转回 NumPy 数组，方便和 matplotlib、pandas 等库衔接。

> 小贴士：Tensor 与 ndarray 用法非常相似，所以从 NumPy 迁移到 PyTorch 几乎无缝；区别在于 Tensor 还能在 GPU 上运算并参与自动求导。

例子

16 📝 例子2：用 requires_grad + backward() 演示自动求导并手动核对

自动求导是 PyTorch 的灵魂。我们先用一个**手算也能验证**的简单函数，确认 PyTorch 算的梯度和数学公式一致。

```
import torch

# 目标函数：  y = x^2 + 3x
# 数学求导：  dy/dx = 2x + 3

# 1) 创建需要求导的张量，requires_grad=True 让它被 autograd 追踪
x = torch.tensor(4.0, requires_grad=True)

# 2) 前向计算（PyTorch 在后台悄悄记录计算图）
y = x ** 2 + 3 * x
print(y)          # tensor(28., grad_fn=<AddBackward0>)

# 3) 反向传播：对标量 y 调用 backward()，自动算出所有梯度
y.backward()

# 4) 梯度存放在叶子张量的 .grad 属性里
print(x.grad)     # tensor(11.)

# 5) 手动核对：  dy/dx = 2*x + 3 = 2*4 + 3 = 11  ✅
print(2 * 4 + 3)  # 11
```

再看一个**两个变量**的例子，核对偏导：

```
# z = a*b + a^2,   dz/da = b + 2a,  dz/db = a
a = torch.tensor(2.0, requires_grad=True)
b = torch.tensor(5.0, requires_grad=True)

z = a * b + a ** 2
z.backward()

print(a.grad)   # tensor(9.)   核对：b + 2a = 5 + 4 = 9 ✅
print(b.grad)   # tensor(2.)   核对：a       = 2     ✅
```

逐步讲解（每步为什么）：

1. **requires_grad=True**：只有打开这个开关，PyTorch 才会为这个张量**保留计算图**、记录它参与了哪些运算，将来才能求它的梯度。默认是 `False`（不追踪，省内存）。
2. **前向计算**：写 `y = x**2 + 3*x` 时，PyTorch 一边算数值，一边在后台搭一张“计算图”。打印 `y` 会看到 `grad_fn=<AddBackward0>`，这就是图存在的证据。
3. **backward()**：对**标量** `y` 调用一次，autograd 就沿计算图从后往前，用链式法则把每个变量的梯度都算出来。注意只能对标量直接 `backward()`。
4. **.grad**：算完后梯度累积在**叶子张量**（我们亲手创建、`requires_grad=True` 的 `x`）的 `.grad` 属性里。
5. **手动核对**：`dy/dx = 2x+3`，代入 `x=4` 得 11，和 `x.grad` 完全一致 —— 这说明 PyTorch 的自动求导是可信的，你不用再手推反向传播。

> 易错点：如果连续 `backward()` 两次而没清零，`.grad` 会**累加**（不是覆盖）。所以训练循环里每轮都要 `zero_grad()`。

例子

17 📝 例子3：nn.Linear + nn.MSELoss + 优化器，最小训练循环

现在把零件拼起来，写一个能**真正训练**的最小例子：让模型学会 `y = 2x + 1`。

```
import torch
import torch.nn as nn

# 1) 造一点训练数据：真实规律 y = 2x + 1
X = torch.tensor([[1.0], [2.0], [3.0], [4.0]])   # 形状 (4,1)
y = torch.tensor([[3.0], [5.0], [7.0], [9.0]])   # 2x+1

# 2) 定义模型：一个线性层  y = w*x + b（自动含权重 w 和偏置 b）
model = nn.Linear(in_features=1, out_features=1)

# 3) 损失函数 + 优化器
loss_fn = nn.MSELoss()                                    # 均方误差（回归常用）
optim = torch.optim.SGD(model.parameters(), lr=0.01)    # 随机梯度下降

# 4) 训练循环（核心四步：forward -> loss -> backward -> step）
for epoch in range(1000):
    y_hat = model(X)              # forward：前向算预测
    loss = loss_fn(y_hat, y)      # 算损失（一个标量，衡量预测有多差）

    optim.zero_grad()             # 清零上一轮的梯度（否则会累加）
    loss.backward()               # backward：反向传播，算出每个参数的梯度
    optim.step()                  # step：按梯度更新 w 和 b

    if epoch % 200 == 0:
        print(f'epoch {epoch}, loss {loss.item():.4f}')

# 5) 看看学到的参数，应接近 w=2, b=1
print('weight =', model.weight.item())   # ≈ 2
print('bias   =', model.bias.item())     # ≈ 1
```

逐行讲解（每行为什么这么写）：

1. **造数据**：`X`、`y` 都做成 `(4,1)` 的二维张量，因为 `nn.Linear` 期望输入形状是 `(样本数, 特征数)`。真实规律是 `y=2x+1`，我们希望模型自己把 `w≈2, b≈1` 学出来。
2. **`nn.Linear(1, 1)`**：建一个全连接层，输入 1 维、输出 1 维，内部自动创建可训练参数 `weight` 和 `bias`，实现 `y = x·w + b`。
3. **`nn.MSELoss()`**：均方误差，把预测和真值差的平方求平均，是回归任务的标准损失。
   `SGD(model.parameters(), lr=0.01)`：告诉优化器“要更新哪些参数”和“步子多大（学习率）”。
4. **训练四步**（最重要，**顺序不能乱**）：

- `model(X)` 前向得到预测 `y_hat`；
- `loss_fn(y_hat, y)` 算出当前有多差；
- `optim.zero_grad()` 先把上一轮残留的梯度清零；
- `loss.backward()` 反向传播，把每个参数的梯度写进 `.grad`；
- `optim.step()` 按 `参数 = 参数 - 学习率 × 梯度` 更新一小步。

循环很多轮后，loss 会越来越小，参数越来越准。

1. **查看结果**：训练完打印 `weight`、`bias`，应该非常接近 2 和 1，说明模型确实学到了规律。

> 口诀：**zero_grad → forward/loss → backward → step**。其中 `zero_grad` 容易忘，忘了梯度就会跨轮累加，训练直接跑偏。

例子

18 📝 例子4：用 Dataset + DataLoader 取一个 batch

数据多了就不能一次全塞进模型，要**分批（batch）**喂。`Dataset` 负责“按编号取一条样本”，`DataLoader` 负责“打包成一批、还能打乱”。

```
import torch
from torch.utils.data import Dataset, DataLoader

# 1) 自定义数据集：必须实现 __len__ 和 __getitem__
class MyDataset(Dataset):
    def __init__(self):
        # 10 个样本：特征 x 和标签 y
        self.x = torch.tensor([[float(i)] for i in range(10)])  # 0~9，形状 (10,1)
        self.y = self.x * 2                                     # y = 2x

    def __len__(self):
        return len(self.x)            # 让 len(dataset) 知道有多少样本

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]   # 按下标返回第 idx 条 (特征, 标签)

# 2) 实例化数据集
dataset = MyDataset()
print('样本总数:', len(dataset))     # 10
print('第0条样本:', dataset[0])      # (tensor([0.]), tensor([0.]))

# 3) 用 DataLoader 打包：每批 4 条，并打乱顺序
loader = DataLoader(dataset, batch_size=4, shuffle=True)

# 4) 取出第一个 batch 看看
for xb, yb in loader:
    print('一个 batch 的特征形状:', xb.shape)   # torch.Size([4, 1])
    print('一个 batch 的标签形状:', yb.shape)   # torch.Size([4, 1])
    print(xb)
    break   # 只看第一批就退出
```

逐步讲解（每步为什么）：

1. **继承 `Dataset`**：自定义数据集只需实现两个方法。`__len__` 告诉框架“一共多少条”，`__getitem__(idx)` 告诉框架“第 idx 条长什么样”。有了这两个，`DataLoader` 才知道怎么取数。
2. **`__getitem__` 返回一条样本**：这里返回 `(特征, 标签)` 元组。注意它每次只取一条，分批的活儿交给 DataLoader。
3. **`DataLoader(dataset, batch_size=4, shuffle=True)`**：

- `batch_size=4`：每次自动把 4 条样本**堆叠**成一个批，单条 `(1,)` 堆成 `(4,1)`；
- `shuffle=True`：每个 epoch 打乱顺序，避免模型记住数据排列，提升泛化。

1. **遍历取 batch**：`for xb, yb in loader` 每次拿到一批。`xb.shape` 是 `(4,1)`，正好可以直接喂给 `nn.Linear(1, ...)`。在真实训练里，就是把例子3的循环改成“外层 epoch、内层遍历 loader 取 batch”。

> 为什么不用普通 for 循环手动切片？因为 DataLoader 还能用 `num_workers` 开多进程并行读数据、自动 shuffle 和堆叠，又快又省心。10 条样本、batch=4，会分成 4+4+2 三批。

例子

第 10 讲 语言模型

Language Model

本讲围绕“如何让计算机理解与生成语言”展开：先用词向量/词嵌入(Word2Vec、CBOW、Skip-gram)把离散单词映射为可计算语义的稠密向量，再用语言模型刻画字符串出现概率 $p(w_1,\dots,w_N)=\prod_t p(w_t \mid w_{<t})$，并借助链式法则、马尔可夫假设(N-gram)进行建模；最后引入循环神经网络(RNN)解决变长序列、长距离依赖与参数共享问题，并讨论其序列分类与序列生成的应用。

1 单词的表示：从编号到独热向量

处理语言要先回答两个问题：**单词在计算机中怎么表示**，以及**单词在机器学习模型中怎么表示**。

- **整数编码(数字编码)**：在计算机中给每个词一个 id(ASCII、词典编号等，如 `hotel`:11，`conference`:81，`motel`:1203)。简单，但 id 的数值大小没有语义含义，相邻 id 的词毫无关系。
- **在机器学习模型中**：线性模型按 $g(x)=w^\top x+w_0$ 计算，每个输入特征 x 与权值 w 相乘。若直接把单词的 id 当作特征 x，那么“特征×权值”这个乘积并没有合理的语义涵义——这提示我们需要更好的向量化表示。
- **独热(One-Hot)编码**：用一个长度等于词汇表大小 $|V|$ 的向量表示一个词，只有该词对应的位置为 1，其余全为 0，例如 `hotel = [1,0,0,...]`。向量维度等于词汇表单词数(可达 50 万)。

独热向量的根本缺陷：

1. **维度极高且稀疏**：词表可能有 50 万词，向量就 50 万维。
2. **无法表达相似度**：任意两个不同词的独热向量都是正交的，内积恒为 0，余弦相似度恒为 0。所以从独热向量看不出 `hotel` 和 `motel` 是近义词。

> 这正是引出“词嵌入”的动机：我们需要一种能编码语义相似度的低维表示。

独热编码　词表示　动机

2 词嵌入与分布式假设

词嵌入(Word Embedding) 把单词表示为**低维、稠密**的实数向量，使得**向量间的几何关系刻画单词的语义相似度**：语义相近的词在向量空间中距离更近。词向量也叫词表示，是一种**分布式表示(distributed representation)**——语义被“分布”到向量的多个维度上，而非集中在单一维度。

核心思想来自**分布式假设(Distributional Hypothesis)**：

> “You shall know a word by the company it keeps.” —— J. R. Firth, 1957

即**单词的语义可以由其上下文推断**。例如即使不认识 `tesgüino`，从“A bottle of tesgüino is on the table”“Tesgüino makes you drunk”等上下文也能猜到它是一种酒精饮料。

由此得到算法直觉：

- 文本中**离得越近(共现于同一上下文)的词，语义越相似**；
- 语义相关的词，在相同上下文出现的概率更大。

词嵌入还能编码丰富的类比关系，经典例子是 $\text{vec}(\text{king}) - \text{vec}(\text{man}) + \text{vec}(\text{woman}) \approx \text{vec}(\text{queen})$。

词嵌入 分布式假设 语义相似度

3 Word2Vec：用上下文学习词向量

Word2Vec(Mikolov et al., 2013)是一种**根据单词在文本中的位置自动学习词向量**的技术。它把上述分布式假设变成一个可优化的目标。

基本设定：在一个长度为 $2c$ 的滑动窗口中，对**中心词(center word)** w_t 和它两侧的**上下文词(outside/context words)** w_{t+j} 建模条件概率 $P(w_t \mid w_{t+j})$ 或 $P(w_{t+j} \mid w_t)$。

Word2Vec 提供两种对偶的模型：

- **CBOW**：用上下文词预测中心词(完形填空)。
- **Skip-gram**：用中心词预测上下文词。

训练完成后，模型权重矩阵 W 的每一行就是对应单词的词向量。Word2Vec 的精髓在于**最大化目标函数，让相似的词在空间中彼此靠近**——把语义关系“塑形”进了向量几何中。

Word2Vec Mikolov 上下文窗口

4 CBOW 模型：上下文预测中心词

CBOW(Continuous Bag-of-Words) 根据上下文单词的词向量预测中心词，形如：

$$P(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) = \text{softmax}\big(U\, z\big)$$

用神经网络实现，分三层：

1. **输入层(查表)**：把每个上下文词的独热向量 x_i 通过线性变换 $z_i = Wx_i$ 转成词向量。由于 x_i 是独热的，Wx_i 实际上就是从矩阵 $W \in \mathbb{R}^{|V| \times d}$ 中**查出第 i 行**，因此 W 又称查找表(lookup table)。
2. **投影/映射层(聚合)**：把窗口内所有上下文向量**求和或取平均**聚合成一个向量：$z = \frac{1}{2|c|} \sum_{-c \le j \le c,\, j \ne 0} z_{t+j}$。
3. **输出层(分类)**：$y = Uz$，再经 softmax 得到词表上每个词作为中心词的概率：$p(w_t) = \dfrac{\exp(y_t)}{\sum_{i=1}^{|V|} \exp(y_i)}$。

本质上，预测中心词是一个**在整个词汇表上的多分类问题**。“Bag-of-Words”意味着它**忽略上下文词的顺序**(求和/平均与顺序无关)。

CBOW　查找表　softmax分类

5 Skip-gram 模型与训练目标

Skip-gram 与 CBOW 相反：用**中心词预测上下文词**，建模 $P(w_{t+j} \mid w_t)$。其前向计算为：

$$z = Wx, \quad y = Uz, \quad P(w_{t+j} \mid w_t) = \frac{\exp(y_{t+j})}{\sum_{i=1}^{|V|} \exp(y_i)}$$

训练方法(对 CBOW/Skip-gram 通用)：给定语料 $D = \{w_1, \ldots, w_N\}$，最小化**负对数似然(Negative Log-Likelihood, NLL)** 损失：

$$L(W, U \mid D) = -\frac{1}{N} \sum_{t=1}^{N} \log p(w_t \mid \text{context})$$

并用**梯度下降**优化参数 W(输入词向量)和 U(输出词向量)。

两者对比：

- **CBOW**：用上下文预测中心词，训练更快。
- **Skip-gram**：用中心词预测上下文，对低频/罕见词能学到更细致的表示。

Skip-gram　NLL损失　训练目标

6 从“背单词”到“学语言”：如何衡量语言能力

学会词向量相当于让计算机“背单词”，但语言数据**无处不在**且形式多样——音频(Audio)、单词(Word)、句子(Sentence)、甚至程序代码(Program),都是序列数据。要让计算机真正“学语言”，先要回答一个问题：**怎么衡量计算机(或一个人)的语言能力?**

直觉上，语言能力体现在能分辨“哪句话更标准/更像人话”。例如:

- “It has been a long time since I saw you” 比中式直译 “Long time no see” 更标准;
- “Give you some color see see”“Give me some money buy buy” 这类则明显不符合语言习惯。

衡量这种“标准与否”的工具，正是**语言模型**：它给越自然、越符合语言习惯的句子赋予越高的概率。这就把“语言能力”量化成了一个可计算的概率问题，自然引出下面对语言模型的定义。

语言模型 动机 语言能力

7 语言模型的定义与典型应用

语言模型(Language Model, LM) 是一种**概率模型**，它刻画**一个给定字符串在某种语言中出现的可能性有多大**。

对任意字符/单词序列 $x = (w_1, w_2, \ldots, w_N)$，语言模型定义为该序列出现的概率:

$$p(x) = p(w_1, w_2, \ldots, w_N)$$

直觉上，**符合语言习惯的句子概率高，不通顺的句子概率低**。例如对中文常见说法，模型应给出 $P_1 > P_2 > P_3 > P_4$，即越自然的句子概率越大。语言模型因此可被当作“衡量语言能力(标准与否)”的工具。

典型应用场景:

- 输入法/消息推荐(预测下一个词)
- 文档/文本生成
- 语法纠错
- 机器翻译
- 语音识别

> 在语音识别和机器翻译中，语言模型负责从多个候选输出里挑出“最像人话”的那个。

语言模型 概率模型 应用场景

8 链式法则与马尔可夫假设(N-gram)

如何计算联合概率 $p(w_1, \ldots, w_N)$?

(1) 概率链式法则：把联合概率精确分解为一串条件概率的乘积：

$$p(w_1, \ldots, w_N) = p(w_1)\, p(w_2 \mid w_1) \cdots p(w_N \mid w_1, \ldots, w_{N-1}) = \prod_{t=1}^{N} p(w_t \mid w_{<t})$$

例如 $p(\text{我爱机器学习}) = p(\text{我})p(\text{爱} \mid \text{我})p(\text{机} \mid \text{我爱})\cdots$。

问题：随着序列变长，条件 $w_{<t}$ 越来越长，几乎不可能在语料中统计到完整历史。

(2) 马尔可夫假设：假设下一个词只依赖前面有限的 $n-1$ 个词：

$$p(w_i \mid w_1, \ldots, w_{i-1}) \approx p(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$$

例如 $p(\text{习} \mid \text{我爱机器学}) \approx p(\text{习} \mid \text{机器学}) \approx p(\text{习} \mid \text{学})$。

这就是 **N-gram 模型**：$n=1$ 是 unigram，$n=2$ 是 bigram，$n=3$ 是 trigram。n 越大上下文越长、越准，但参数(可能的 N 元组数)指数增长且数据稀疏越严重。

链式法则　马尔可夫假设　N-gram

9 基于计数的概率估计及其局限

在 N-gram 框架下，条件概率最直接的估计是**频率计数**：

$$p(w_t \mid w_{t-k}, \ldots, w_{t-1}) = \frac{\text{count}(w_{t-k} \ldots w_{t-1} w_t)}{\text{count}(w_{t-k} \ldots w_{t-1})}$$

即“在历史 $w_{t-k} \ldots w_{t-1}$ 后出现 w_t 的次数”除以“历史出现的总次数”。

这种纯计数方法的局限：

- **离散符号、无词义**：词被当成互不相关的离散符号，没有语义。
- **对相似词/句不敏感**：见过“我喜欢自然语言学”不能帮助预测“我喜欢计算机学”，因为它们是不同的离散串。

> 这类问题推动了用**词向量/神经网络**引入语义泛化能力的改进路线。

计数估计　频率　局限

10 神经网络语言模型的设计原则与朴素尝试的失败

把序列建模交给神经网络前，先看几种朴素思路为何不够：

- **思路1 计数**：离散、无语义、对相似句不敏感(见前述计数小节)。
- **思路2 词向量求平均(类 CBOW)**：把历史词向量取平均再预测。问题是**词袋(Bag-of-Words)** 性质——丢失词序，“猫吃鱼”和“鱼吃猫”表示相同。
- **思路3 把若干词向量拼接后输入前馈网络**：能保留局部顺序，但**无法处理变长输入**(拼接维度固定)，且**不同位置不共享参数**，参数量随上下文长度膨胀。

由此提炼出对序列建模网络的**四条设计原则**：

1. 能**处理变长序列**；
2. 能**跟踪长距离依赖**；
3. 能**维持单词的顺序信息**；
4. 能在序列的不同位置/不同序列间**共享模型参数**。

> **循环神经网络(RNN)** 正是同时满足这四条原则的理想方案。

神经网络语言模型 设计原则 词袋问题

11 RNN 的设计动机：复用隐含层做“记忆”

在引入 RNN 之前，先回顾**常规(Vanilla)神经网络**：给定输入 $x \in \mathbb{R}^d$，先算隐含状态 $h = \tanh(W^\top x)$，再算输出 $y = Vh$(也写作 $h = f_W(x)$, $y = g(h)$)。

它的问题在于：**每次只能处理一个单词**——输入向量进、输出向量出，处理完就“忘了”，无法记住一个序列中先前出现过的词。

那怎么才能“记住”整个序列呢？关键观察是：

> 隐含层 h 本身就是一个“记忆体”。既然如此，为什么不在处理下一个词时**复用**上一步算出的隐含向量(记忆)呢？

这正是 **RNN 的核心思想：复用隐含层来记住过去输入的单词**。形象地说，普通网络“总是忘记之前的词”，而 RNN“复用了记忆，所以不会忘记过去的词”。把上一时刻的隐含状态 h_{t-1} 喂给当前时刻，就形成了贯穿时间的“记忆循环”，从而满足前面提出的变长序列、长距离依赖、保留词序、参数共享四条设计原则。

RNN 设计动机 记忆复用

12 循环神经网络(RNN)：原理与计算图

RNN 的核心思想：**复用隐含层来“记住”过去输入的单词**。它把上一时刻的隐状态 h_{t-1} 作为当前时刻的额外输入，形成“记忆循环”。

状态更新与输出：

$$h_t = \tanh\left(W_{hh}\, h_{t-1} + W_{xh}\, x_t\right)$$

$$y_t = W_{hy}\, h_t$$

其中 h_t 是携带历史信息的隐状态(cell state)，x_t 是当前输入向量。

关键性质：

- **参数共享**：W_{hh}, W_{xh}, W_{hy} 在**每个时间步都相同**——“同一个函数、同一套参数反复作用于序列每一步”。这既满足变长输入，又控制了参数量。
- **计算图按时间展开(unrolled across time)**：把循环展开成一条链，可在其上做前向传播得到各步损失 L_t，再反向传播求梯度。

典型使用模式：

- **One-to-One**：退化为普通前馈(“Vanilla”)网络。
- **Many-to-One**：整段序列→一个输出，如**情感分类**。
- **Many-to-Many**：序列→序列，如信息抽取、序列标注。

RNN　隐状态　参数共享　计算图

13 RNN 的应用：序列分类与序列生成

RNN 按输入输出形式可服务于多类任务。

应用一：序列分类(Many-to-One)

- 输入：一段单词序列；输出：序列属于各类别的概率。
- 做法：用 RNN 读完整句，取最后(或聚合)的隐状态送入分类器(如 MLP+Sigmoid/Softmax)。例如情感分类输出 “85% Like / 15% Dislike”。

应用二：序列生成(语言模型本职)

- 输入：前 $t-1$ 个 token；输出：预测下一个 token 的分布(MLP+Softmax)，用**交叉熵**对真实下一个词监督。
- 自回归地“预测一个词→喂回去→再预测”，即可逐字生成文本。典型例子：**AI 作诗**(给“床前明月”预测“光”)、**音乐生成**(预测下一个乐谱符号)。

条件序列生成 → 机器翻译(Encoder-Decoder)

- 用一个 Encoder 读入源语言序列(如中文“我到了”)压成上下文，再用 Decoder 以该上下文为条件**逐词生成**目标语言序列(如英文“I have arrived”)。下一讲展开。

序列分类　序列生成　Encoder-Decoder

14 📝 例题1：用 Bigram 手算一句话的概率

目标：给一个超小语料，用 Bigram(二元语法)统计条件概率 $P(w_i \mid w_{i-1})$，再用链式法则把一句话的概率连乘出来。

语料(3 句话，`<s>` 表示句首，`</s>` 表示句尾)：

1. `<s> 我 爱 学习 </s>`
2. `<s> 我 爱 你 </s>`
3. `<s> 你 爱 学习 </s>`

第 1 步：数单词出现次数(unigram 计数)

为什么先数？因为 Bigram 概率的分母就是“前一个词出现了多少次”。

词	次数
`<s>`	3
我	2
爱	3
学习	2
你	2
`</s>`	3

第 2 步：数相邻词对的次数(bigram 计数)

把每句话里每个“前词→后词”的相邻搭配数一遍：

- `<s>→我` 出现 2 次(句 1、句 2)，`<s>→你` 出现 1 次(句 3)
- `我→爱` 出现 2 次
- `爱→学习` 出现 2 次(句 1、句 3)，`爱→你` 出现 1 次(句 2)
- `学习→</s>` 出现 2 次；`你→爱` 1 次；`你→</s>` 1 次

第 3 步：用公式算条件概率

公式就是 $P(w_i \mid w_{i-1}) = \dfrac{\text{count}(w_{i-1}\, w_i)}{\text{count}(w_{i-1})}$，即“词对次数 ÷ 前词次数”。

- $P(\text{我} \mid \langle s \rangle) = \dfrac{2}{3} \approx 0.667$ (`<s>` 出现 3 次，其中 2 次后面跟 `我`)
- $P(\text{爱} \mid \text{我}) = \dfrac{2}{2} = 1$ (`我` 后面**总是**跟 `爱`)
- $P(\text{学习} \mid \text{爱}) = \dfrac{2}{3} \approx 0.667$
- $P(\langle /s \rangle \mid \text{学习}) = \dfrac{2}{2} = 1$

第 4 步：链式法则连乘，得到整句概率

要算句子 `我 爱 学习` 的概率，把上面 4 个条件概率乘起来(为什么是乘？因为链式法则把联合概率拆成条件概率之积)：

$$P(\langle s \rangle \text{ 我 爱 学习 } \langle /s \rangle) = \tfrac{2}{3} \times 1 \times \tfrac{2}{3} \times 1 = \tfrac{4}{9} \approx 0.444$$

> **小结**：Bigram 只看“前一个词”，把长句子的概率拆成一串好统计的二元条件概率再连乘。注意概率

> 随句子变长会越乘越小，这很正常。

例题

15 📝 例题2：词向量类比 king − man + woman ≈ queen 手算

目标：用一组(简化到 3 维的)假想词向量，亲手算出经典类比 king − man + woman ≈ queen，并用余弦相似度证明结果最接近 `queen`。

假想词向量(真实词向量是几百维，这里压成 3 维方便手算)：

词	向量
`king` (国王)	$[0.9,\ 0.7,\ 0.2]$
`man` (男人)	$[0.8,\ 0.1,\ 0.1]$
`woman` (女人)	$[0.7,\ 0.9,\ 0.1]$
`queen` (女王)	$[0.7,\ 1.6,\ 0.3]$
`banana` (香蕉，干扰项)	$[0.1,\ 0.2,\ 1.5]$

第 1 步：理解为什么要做这个减加法

直觉：`king` → `man` 之间的差向量大致编码了“王室身份去掉、保留男性”的方向；那么 `king - man` 就剥离了“男性”这层语义，再 `+ woman` 就换上“女性”这层语义，理论上应得到“女性版的王”，也就是 `queen`。

第 2 步：逐维做向量加减

$$\text{king} - \text{man} + \text{woman}$$

- 第 1 维：$0.9 - 0.8 + 0.7 = 0.8$
- 第 2 维：$0.7 - 0.1 + 0.9 = 1.5$
- 第 3 维：$0.2 - 0.1 + 0.1 = 0.2$

得到结果向量 $r = [0.8,\ 1.5,\ 0.2]$。

第 3 步：用余弦相似度比较 r 和各候选词

余弦相似度 $\cos(a, b) = \dfrac{a \cdot b}{\|a\|\,\|b\|}$，值越接近 1 越相似(方向越一致)。

以 r 与 `queen` 为例：

- 点积 $r \cdot \text{queen} = 0.8 \times 0.7 + 1.5 \times 1.6 + 0.2 \times 0.3 = 0.56 + 2.40 + 0.06 = 3.02$
- $\|r\| = \sqrt{0.8^2 + 1.5^2 + 0.2^2} = \sqrt{2.93} \approx 1.712$
- $\|\text{queen}\| = \sqrt{0.7^2 + 1.6^2 + 0.3^2} = \sqrt{2.94} \approx 1.715$
- $\cos(r, \text{queen}) = \dfrac{3.02}{1.712 \times 1.715} \approx 0.996$

第 4 步：对比其他词，看谁最近

同样方法可算出：

候选	$\cos(r, \cdot)$
`queen`	≈ 0.996 ✅ 最接近
`king`	≈ 0.914
`banana`	≈ 0.262

r 与 `queen` 的余弦相似度最高(也几乎最近)，所以 $\text{king} - \text{man} + \text{woman}$ 确实**最接近 `queen`**，而离 `banana` 这种无关词很远。

> **小结**：词向量把“性别”“王室”这类语义编码成了向量空间里的**方向**，于是语义类比就变成了简单的向量加减法——这正是独热编码永远做不到的(任意两个独热向量都正交，相似度恒为 0)。

例题

第 11 讲 Transformer

Transformer

本讲从RNN编解码器与注意力机制出发，引出完全基于自注意力(Self-Attention)的Transformer。核心是缩放点积注意力 $\text{Attention}(Q,K,V)=\text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$、多头注意力、位置编码、残差连接与LayerNorm，以及Encoder-Decoder结构和掩码注意力。Transformer用并行的矩阵运算取代RNN的串行递归，更适合长序列建模与GPU加速。

1 条件式序列生成与Seq2Seq

条件式序列生成指在一个源序列 $x=(x_1,\dots,x_{T_x})$ 的条件下，生成另一个目标序列 $y=(y_1,\dots,y_{T_y})$。

典型任务：

- **机器翻译**：中文→英文
- **对话生成**：问句→回复
- **语音识别(ASR)**：语音→文本
- **图像描述(Image Captioning)**：图像→文字

经典做法是**序列到序列学习(Sequence-to-Sequence Learning)**，用两个网络：**编码器(Encoder)** 读入源序列并压缩成中间表征，**解码器(Decoder)** 据此自回归地生成目标序列。Cho等人在2014年提出的RNN编解码器是其代表。

Seq2Seq 编解码器 序列生成

2 RNN编码器-解码器模型

RNN编解码器用两个RNN分别作编码器和解码器。

编码器：逐步读入源序列，更新隐含状态

$$h_t=\text{RNN}_{in}(x_t,h_{t-1})$$

并取最后一个隐含状态作为**上下文向量(context vector)** $c=h_{T_x}$，作为整个源序列 x 的表征。

解码器：以 c 为条件，自回归地预测下一个词

$$s_i=\text{RNN}_{out}(y_{i-1},s_{i-1},c)$$

$$p(y_i\mid y_{<i},c)=\text{softmax}(\text{MLP}(s_i))$$

> 直觉：编码器把源句子"读完并记在脑子里"压成一个向量 c，解码器再"看着这个记忆"一个词一个词地写出译文。

RNN 编码器 解码器 上下文向量

3 Seq2Seq的应用场景

Seq2Seq(编码器-解码器)框架适用于各类"输入一个序列、输出另一个序列"的任务：

- **机器翻译(Translation)**：中文→英文，编码器、解码器都是RNN。
- **对话生成(Conversation)**：上文→回复，例如开放域闲聊机器人(Open-Domain Dialog Generation)。
- **图像描述(Image Captioning)**：图像→文字描述。**注意此时编码器是CNN**(用CNN提取图像特征作为上下文)，解码器仍是RNN生成文字。

> 关键观察：编码器的具体网络类型取决于**输入模态**——文本/语音用RNN，图像用CNN；而解码器负责自回归生成目标序列。这说明Seq2Seq是一个通用的"序列(或表征)→序列"范式，并不局限于纯文本。

Seq2Seq　应用　图像描述

4 Seq2Seq的训练目标

给定数据集 $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$，训练通过**最小化交叉熵损失**(即最大化目标序列的对数似然)进行。

单个样本损失(对每个时间步的预测取负对数似然求和)：

$$\ell(\theta \mid x, y) = -\sum_{t=1}^{T_y} \log p_\theta(y_t \mid y_1, \ldots, y_{t-1}, x)$$

整体损失：

$$L(\theta \mid D) = -\frac{1}{N} \sum_{l=1}^{N} \ell(\theta \mid x^{(l)}, y^{(l)})$$

- 每个时间步输出词表上的概率分布，与真实词的one-hot标签算交叉熵。
- 优化算法用**梯度下降**(及其变体)。

交叉熵　训练　极大似然

5 长序列瓶颈与注意力的动机

RNN编解码器的**核心问题**：把一个超长序列压缩成**单一固定**的上下文向量 c，没有考虑序列中每个位置的不同重要性。

后果：

- 信息瓶颈——长序列的早期信息容易被遗忘。
- 实验观察：随着输入序列长度增大，翻译质量(性能)下降，即所谓"**长度诅咒**"。

解决思路：动态上下文向量。让解码器在生成每个词 y_i 时使用一个专属的、动态的上下文向量 c_i，它由源序列各位置隐含状态的**加权线性组合**得到：

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

权重 α_{ij} 表示生成 y_i 时应关注源位置 j 的程度。

长序列　瓶颈　注意力动机

6 基于注意力的Seq2Seq

在解码的每一步，模型**动态地**关注源序列的不同单词，而非依赖单一固定编码。

注意力分数 $a_{i,j}$ 衡量解码器状态 s_{i-1} 与编码器状态 h_j 的匹配程度，可用一个小神经网络估计：

$$a_{i,j} = \mathrm{NN}(s_{i-1}, h_j)$$

归一化为概率(对齐权重)：

$$\alpha_i = \mathrm{softmax}(a_{i,1}, a_{i,2}, \ldots, a_{i,T_x})$$

动态上下文：

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

解码器更新改为 $h_i = \mathrm{RNN}_{out}(h_{i-1}, y_{i-1}, c_i)$。

> 可视化：在英法翻译中，注意力会自动学到"European Economic Area"对应法语"zone économique européenne"的逆序对齐，无需人工标注。这种权重也叫**全局对齐权重(global alignment weights)**。

注意力 对齐 动态上下文

7 Query-Key-Value范式

注意力可统一抽象为 **Query-Key-Value(QKV)** 检索范式：

- **Query(查询 q)**："我想找什么样的内容"——发起检索的一方。
- **Key(键 k)**："我在别人眼中是什么"——用来与query匹配。
- **Value(值 v)**："我到底携带什么内容"——被检索、被聚合的实际信息。

计算流程：用 q 与每个 k 算相似度得分→softmax归一化为权重→对相应的 v 加权求和得到输出。

在传统Seq2Seq注意力里，q 来自解码器状态 s_{i-1}，k 和 v 来自编码器状态 h_j。把这套机制改成同一序列内部互相检索，就得到了**自注意力**。

QKV 注意力 检索

8 RNN的缺陷：无法并行

RNN语言模型的根本缺陷在于**串行依赖**：

- 一个位置的隐含状态计算依赖前一位置隐含状态的计算结果，即 h_t 必须等 h_{t-1} 算完才能算。
- 因此同一层内部**无法并行**，序列越长，前向/反向传播的时间步链越长(大量for循环)。
- 长程依赖还会带来梯度消失/爆炸，远距离词之间的信息要经过很多步才能传递。

Self-Attention的突破：每个新状态 h_i' 直接基于**整个输入序列**计算，各位置可以**同步(并行)计算**，且任意两个位置之间是"一步可达"。这就是"再见RNN：凡是用RNN的地方都可替换为自注意力"的底气。

RNN缺陷 并行 串行依赖

9 自注意力的直觉

自注意力(Self-Attention) 让句子中的每个单词都去关注(attend to)序列里的所有其他单词，把相关词的"理解"融合进当前词的新表征中。

- RNN：维持一个隐含状态，**逐个**把前面的词注入到正在处理的当前词。
- Self-Attention：**同时**把其他相关词的语义"烘焙"进当前词的表征。

经典例子：句子 "The animal didn't cross the street because **it** was too tired." 中的 "it" 指代谁？自注意力能让 "it" 对 "animal" 给出高权重，从而解析指代。

直觉总结：对每个query词，用它与各词的相关性分数对各词的Value加权求和，得到该词"刷新后的记忆"(new state)。

自注意力 直觉 指代消解

10 自注意力的逐步计算

以一个序列为例，自注意力分5步(忽略多头时)：

1. **词嵌入**：把每个词元(整数id)转成词向量，作为初始隐含状态 $h_i = W x_i$。
2. **生成Q/K/V**：对每个 h_i 做线性投影

$$q_i = W_q h_i, \quad k_i = W_k h_i, \quad v_i = W_v h_i$$

即每个词有"三个分身"。

1. **算注意力分数**：对每个 $\langle q, k \rangle$ 对算**缩放点积**

$$a_{i,j} = \frac{q_i^\top k_j}{\sqrt{d}}$$

其中 d 是 q, k 的维度。

1. **归一化**：$\alpha_{i,1}, \ldots, \alpha_{i,T} = \mathrm{softmax}(a_{i,1}, \ldots, a_{i,T})$。
2. **聚合Value**：$h'_i = \sum_j \alpha_{i,j} v_j$。

例：query "robot" 与各key点积得 $q_1 \cdot k_1 = 112$, $q_1 \cdot k_2 = 15$, $q_1 \cdot k_3 = 96$, $q_1 \cdot k_4 = 41$，除以 $\sqrt{d}$ 再softmax得 $\alpha = [0.66, 0.02, 0.28, 0.09]$，最后对各Value加权求和得到 z_1。

自注意力 计算步骤 缩放点积

11 缩放点积注意力与矩阵化

把所有 h 堆成矩阵 H，自注意力可整体写成矩阵乘法，实现高度并行：

$$Q = W_q H, \quad K = W_k H, \quad V = W_v H$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

要点解读：

- $QK^\top$ 一次性算出所有 query-key 对的相似度矩阵 A(行为不同query)。
- 除以 $\sqrt{d_k}$ 做**缩放**：当 d_k 较大时点积的方差会很大，使softmax进入梯度极小的饱和区；除以 $\sqrt{d_k}$ 把方差控制回合理范围，稳定训练。
- softmax**沿行**归一化得权重矩阵 $\hat{A}$，再与 V 相乘聚合。
- 全程是矩阵乘法，可用**GPU加速**，所有输出 $h'_1, \ldots, h'_n$ 并行得出。

缩放点积 矩阵运算 GPU

12 位置编码

自注意力对输入**顺序不敏感**(置换不变)：打乱词序，每个词看到的key/value集合不变，输出只是相应换位，丢失了语序信息。而语言中顺序至关重要。

位置编码(Positional Encoding) 为每个位置赋予一个唯一的向量 e_i，**加到**词嵌入上再送入自注意力层：

$$\tilde{h}_i = h_i + e_i$$

PPT给出的**可学习**方案：令 p_i 为指示 x_i 在序列中位置的one-hot向量，从可学习的位置嵌入矩阵 W_p 中查得

$$e_i = W_p p_i$$

W_p 的列数即最大序列长度(如1024)。

位置编码 顺序 Positional Encoding

13 多头注意力

在Transformer的结构图中，自注意力以**多头注意力(Multi-Head Attention)** 的形式出现：用多组独立的注意力"头"**并行**做注意力，再把各头的结果合并。

直觉：不同的"头"可以关注**不同类型的关系**(如指代、邻接、句法等)，类似CNN里用多个卷积核各管一类特征，从而得到比单头更丰富的表征。注意力可视化(如论文中第5层、某个头)显示，不同头会对不同的词给出各自"尖锐"或"分散"的关注模式。

多头注意力 Multi-Head 子空间

14 Transformer整体结构：Encoder-Decoder

Transformer(Vaswani等"Attention Is All You Need")是堆叠 N_x 层的Encoder-Decoder。

编码器层(重复 N_x 次)：

1. 多头自注意力 + Add&Norm
2. 前馈网络(FFN) + Add&Norm

输入是"词嵌入 + 位置编码"。

解码器层(重复 N_x 次)：

1. **掩码多头自注意力**(Masked) + Add&Norm —— 关注已生成序列
2. **交叉注意力(Cross-Attention)** + Add&Norm —— Query来自解码器，Key/Value来自编码器输出，使解码器关注源序列
3. 前馈网络 + Add&Norm

输出：解码器顶层经Linear+Softmax得到下一个词的概率分布。解码器输入是"输出序列右移一位(shifted right)"，即用 `<BOS>`、已生成词去预测下一个词。

Transformer　Encoder-Decoder　结构

15 注意力掩码：填充掩码与因果掩码

注意力掩码(Attention Mask) 是一个 $L \times L$ 的矩阵，加在注意力分数上(被屏蔽位置置为 $-\infty$，softmax后权重≈ 0)，避免模型注意到指定位置。三类常见掩码：

- **填充掩码(Padding Mask)**：用于编码器和解码器，屏蔽 `<pad>` 占位符，使其不被关注(对应列全为$-\infty$)。
- **交叉注意力掩码(Cross-Attention Mask)**：仅解码器，屏蔽源序列中的padding。
- **因果/前瞻掩码(Causal/Look-ahead Mask)**：**仅解码器自注意力**，下三角为0、上三角为 $-\infty$，保证位置 i 只能看到 $\leq i$ 的已生成词，**不能偷看未来**。这是自回归生成的关键，使训练(并行)与推理(逐词)行为一致。

> 因果掩码示意：第1个词只能看自己，第2个词能看前2个，以此类推，形成阶梯状的0/$-\infty$ 矩阵。

掩码　因果掩码　Padding

16 Transformer的训练与应用

训练：与RNN编解码器一致，最小化交叉熵

$$\ell(\theta \mid x, y) = -\sum_{t=1}^{T_y} \log p_\theta(y_t \mid y_1, \ldots, y_{t-1}, x)$$

$$L(\theta \mid D) = -\frac{1}{N} \sum_{l=1}^{N} \ell(\theta \mid x^{(l)}, y^{(l)})$$

用**梯度下降**优化；因解码器使用因果掩码，目标序列各位置的损失可在一次前向中**并行**计算。

优势回顾：自注意力是矩阵运算，可用GPU加速、所有位置并行计算；序列内任意两个位置可"一步"直接交互(相比RNN需逐步传递)，更利于建模长程依赖。

应用：Transformer广泛用于序列到序列任务——机器翻译、文本摘要、对话生成、AI诊断等，是现代深度语言模型的基础架构。

训练 应用 并行

17 📝 例题1：2维自注意力全过程手算（3个token）

我们用一个**最小的可手算例子**，把缩放点积注意力 $\text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$ 从头算一遍。只用 $d_k=2$ 维、3个token。

已知：经过 W_q, W_k, W_v 投影后，三个词的 Q/K/V 向量为（每个都是2维行向量）：

token	q	k	v
词1	$(1,0)$	$(1,0)$	$(1,0)$
词2	$(1,0)$	$(0,1)$	$(0,1)$
词3	$(0,1)$	$(1,1)$	$(1,1)$

我们只算**词1的输出** z_1（即以 $q_1=(1,0)$ 作为query）。

第1步：算原始点积分数 $q_1\cdot k_j$。 为什么用点积？点积越大表示两个向量方向越一致、越"相关"。

- $q_1\cdot k_1=(1,0)\cdot(1,0)=1\cdot1+0\cdot0=1$
- $q_1\cdot k_2=(1,0)\cdot(0,1)=1\cdot0+0\cdot1=0$
- $q_1\cdot k_3=(1,0)\cdot(1,1)=1\cdot1+0\cdot1=1$

第2步：除以 $\sqrt{d_k}=\sqrt{2}\approx1.414$ **做缩放**。 为什么？防止分数太大让softmax饱和（例题2细讲）。

- $a_1=1/1.414\approx0.707$
- $a_2=0/1.414=0$
- $a_3=1/1.414\approx0.707$

第3步：softmax归一化成权重。 公式 $\alpha_j=\frac{e^{a_j}}{\sum_i e^{a_i}}$，作用是把分数变成**和为1的概率**。

- $e^{0.707}\approx2.028,\quad e^0=1,\quad e^{0.707}\approx2.028$
- 分母 $=2.028+1+2.028=5.056$
- $\alpha=(2.028/5.056,\ 1/5.056,\ 2.028/5.056)\approx(0.401,\ 0.198,\ 0.401)$

> 解读：词1和词3的key都与 q_1 对齐（点积=1），所以权重都是0.401；词2不相关（点积=0），权重最低0.198。三者相加 $0.401+0.198+0.401=1$，验证无误。

第4步：用权重对Value加权求和，得到输出 $z_1=\sum_j \alpha_j v_j$。

$$z_1=0.401\cdot(1,0)+0.198\cdot(0,1)+0.401\cdot(1,1)$$

- 第一维：$0.401\cdot1+0.198\cdot0+0.401\cdot1=0.802$
- 第二维：$0.401\cdot0+0.198\cdot1+0.401\cdot1=0.599$
- 所以 $z_1\approx(0.802,\ 0.599)$

结论：词1"刷新后的表征" z_1 是三个Value的加权平均，主要吸收了词1和词3的信息。整个过程没有任何递归，纯矩阵运算，可以对所有token同时并行做。

例题

18 📝 例题2：为什么要除以 $\sqrt{d_k}$？数值对比softmax饱和

这是初学者最容易忽略的细节。我们用**具体数字**看看不缩放会发生什么。

背景：当 q, k 各分量是均值0、方差1的独立随机数时，点积 $q^\top k = \sum_{i=1}^{d_k} q_i k_i$ 的方差正比于 d_k。维度越高，点积数值波动越大。

对比实验：设两个query分别与3个key打分。假设 $d_k = 64$，不缩放时点积容易达到几十的量级。看两组分数：

A组（不缩放，分数偏大）：原始分数 $= [20,\ 10,\ 0]$

- $e^{20} \approx 4.85 \times 10^8$，$e^{10} \approx 2.20 \times 10^4$，$e^0 = 1$
- softmax $\approx [0.99995,\ 0.00005,\ 0.0000002]$
- 几乎变成 one-hot：**第一个词独吞几乎全部权重**，其余被压成0。

B组（缩放后，除以 $\sqrt{64} = 8$）：分数变成 $[20/8, 10/8, 0/8] = [2.5,\ 1.25,\ 0]$

- $e^{2.5} \approx 12.18$，$e^{1.25} \approx 3.49$，$e^0 = 1$
- 分母 $= 12.18 + 3.49 + 1 = 16.67$
- softmax $\approx [0.731,\ 0.209,\ 0.060]$
- 权重**分布更平滑**，第二、三个词仍有非零关注。

为什么饱和不好？

1. **梯度消失**：softmax在接近one-hot时，输出对输入几乎不变化，梯度 ≈ 0，反向传播时这一层学不动。
2. **信息丢失**：本该参考多个词，结果只盯死一个词，注意力退化。

> 一句话记忆：d_k **越大点积越大** → **不缩放softmax越尖锐越饱和** → **梯度越小越难训练**。除以 $\sqrt{d_k}$ 正好把点积方差从 d_k 拉回到 1 的量级，让softmax工作在"敏感区"。

例题

19 📝 例题3：自注意力如何让 "it" 关注 "animal"（指代消解）

经典句子："The animal didn't cross the street because **it** was too tired." 这里 "it" 指代谁？人一看就知道是 animal（不是 street，因为"累了"的是动物）。自注意力靠**数字**自动学到这一点。

为了能手算，我们只取4个关键词，用 $d_k = 2$ 维。设 "it" 作为query，它要和其他词的key算相关性：

词	key向量	value向量（携带的语义）
animal	$k = (2, 0)$	$v = (1, 0)$ "动物"
street	$k = (0, 2)$	$v = (0, 1)$ "街道"
tired	$k = (1, 0)$	$v = (1, 1)$ "疲倦"
it	$k = (0, 0)$	$v = (0, 0)$

训练后，"it" 的query向量学成了 $q_{it} = (3, 0)$（在第一维上很强，正好和 animal、tired 这类"会累的实体"对齐）。

第1步：算 $q_{it} \cdot k_j$（先不缩放看趋势）

- 对 animal：$(3, 0) \cdot (2, 0) = 6$
- 对 street：$(3, 0) \cdot (0, 2) = 0$
- 对 tired：$(3, 0) \cdot (1, 0) = 3$
- 对 it 自己：$(3, 0) \cdot (0, 0) = 0$

第2步：缩放 $\div\sqrt{2} \approx 1.414 \rightarrow [4.243,\ 0,\ 2.121,\ 0]$

第3步：softmax

- $e^{4.243} \approx 69.7,\ e^{0} = 1,\ e^{2.121} \approx 8.34,\ e^{0} = 1$
- 分母 $= 69.7 + 1 + 8.34 + 1 = 80.04$
- $\alpha \approx [0.871,\ 0.012,\ 0.104,\ 0.012]$

> 解读：**"it" 把87.1%的注意力放在 animal 上**，只有1.2%给 street。模型成功"指认"了 it=animal！这正是 PPT 里那张 "it" 高亮 "animal" 的注意力图背后的计算。

第4步：聚合Value，"it" 的新表征 $\approx 0.871 \cdot (1, 0) + 0.104 \cdot (1, 1) + \cdots \approx (0.975,\ 0.104)$，主体是"动物"的语义。于是 "it" 这个本来空洞的代词，被"烘焙"进了 animal 的含义。

关键启示：所谓"理解指代"，本质就是 q, k 经训练后让相关词点积大、softmax权重高，**不需要任何人工规则**。

例题

第 12 讲 大语言模型

Large Language Model

本讲从"小数据任务难以充分训练 Transformer"出发，介绍**预训练语言模型**（BERT 掩码语言建模、GPT 自回归语言建模）与"预训练—微调"两阶段范式；进而讲解**大语言模型（LLM）**随规模扩大出现的**涌现能力**，以及如何利用 LLM——微调（全量/参数高效 LoRA）、提示工程（零样本/小样本上下文学习/思维链 CoT/ReAct）、检索增强生成（RAG）与智能体（Agent），并以 DeepSeek 的 MoE 与思维链强化学习作为前沿案例。

1 动机：当 Transformer 遇上数据不足

Transformer 等强大模型参数众多，需要海量数据才能训练充分。但现实中许多具体任务（如垃圾邮件分类、特定领域问答）只有**很少的标注数据**，从头训练一个大模型既容易过拟合，也无法学到通用的语言知识。

核心思想是提出一个问题：**我们能否为所有文本学习一个统一的向量空间?**

- 先在**超大规模、无标注**的文本语料（电子书、维基百科等）上学习通用的语言表征，这个表征被**所有下游任务共享**；
- 再针对每个**小数据任务**做轻量适配。

这正是"预训练—微调"范式的出发点：把"读了一百万本书"的通用能力迁移到"小菜一碟"的具体任务上。

预训练　迁移学习　动机

2 预训练—微调两阶段范式

预训练语言模型（Pre-trained Language Model, PLM）的训练分为两个阶段：

1. **Phase 1 预训练（Pre-training）**：在超大规模文本语料上进行**无监督/自监督**训练，让模型学习通用的语言规律与世界知识。这一阶段不需要人工标注，监督信号直接来自文本本身（如预测被遮罩的词或下一个词）。
2. **Phase 2 微调（Fine-tuning）**：在特定下游任务的**少量标注数据**上进行**有监督**训练，使模型适配具体任务（如分类、问答）。

这种范式的优势在于：

- **数据效率高**：通用知识在预训练阶段一次性学好，下游任务只需少量数据即可收敛。
- **效果好**：预训练表征往往显著优于从零开始训练。

> 直觉：先"博览群书"打好基础（预训练），再"专项突击"应对具体考试（微调）。

预训练　微调　两阶段

3 BERT：双向 Transformer 编码器

BERT（Bidirectional Encoder Representations from Transformers） 是一种基于 **Transformer 编码器** 的预训练模型，用来学习**单词与句子的向量表征**。

结构要点：

- 与 Transformer 编码器一致，输入一个单词（词元）序列，输出每个位置的**上下文相关词向量（Contextualized Word Vector）**。
- 首位放一个特殊符号 **[CLS]**，其对应的输出向量用作整个**句子的表征（Sentence Vector）**，常用于句子级分类。
- 句子之间用 **[SEP]** 分隔。

"双向"是指 BERT 在编码每个词时能同时看到它**左右两侧**的全部上下文（编码器内部是全连接自注意力），这与只能看左侧的自回归模型（如 GPT）形成对比。BERT 适合**理解类任务**（分类、序列标注、阅读理解），不直接擅长文本生成。

BERT　编码器　双向

4 BERT 预训练任务一：掩码语言建模（MLM）

掩码语言建模（Masked Language Model, MLM） 是 BERT 的核心预训练任务：

- 随机遮罩输入中约 **15%** 的词元（替换为特殊符号 **[MASK]**）；
- 让模型根据**上下文（左右两侧）**预测这些被遮罩的词；
- 在被遮罩位置接一个分类器（MLP + Softmax），在整个词表上输出概率分布，用交叉熵训练。

例如输入"机器 [MASK] 退了 很 有趣"，模型需在 [MASK] 处预测出"学习"等合适的词。

要点与直觉：

- 因为要同时利用左右上下文，MLM 天然鼓励模型学到**双向**语义表征。
- 这类似"完形填空"，迫使模型理解词与上下文的关系，而非简单背诵。

BERT　掩码语言建模　MLM

5 BERT 预训练任务二：相邻句预测（NSP）

相邻句预测（Next Sentence Prediction, NSP） 让 BERT 学习**句子之间的关系**：

- 输入两句话 A 和 B（用 [SEP] 分隔），模型预测 **B 是否紧接在 A 之后出现**；
- 训练时约 85% 的样本 B 确实是 A 的下一句（Is Next），约 15% 是从语料中随机抽取的句子（Not Next）；
- 用 **[CLS]** 位置的输出接分类器（MLP + Softmax）做二分类。

意义：NSP 让模型获得**句间连贯性/蕴含关系**的建模能力，对自然语言推理、问答、句对匹配等下游任务有帮助。

> 注意：MLM 关注"词"，NSP 关注"句"，二者联合预训练让 BERT 同时具备词级与句级的理解能力。

BERT　相邻句预测　NSP

6 BERT 的四类微调任务

预训练好的 BERT 通过"在顶部接一个从头训练的小分类器"即可适配多种下游任务，常见四类：

1. **句子分类**：输入单句，输出句子类别。用 [CLS] 的输出向量接分类器，如情感分析、文档分类。
2. **单词标注（序列标注）**：输入单句，对**每个词**输出一个类别。如槽位填充（slot filling），例如把"Arrive Shanghai on Nov 2nd"中各词标为目的地/时间等。
3. **句子对关系分类**：输入两句话，输出二者关系。如自然语言推理（NLI）：给定"前提"判断"假设"为真/假/未知。
4. **抽取式问答（机器阅读理解）**：输入文档 D 与问题 Q，输出答案在文档中的**起止位置** (s, e)，答案即 $A = \{d_s, \ldots, d_e\}$。对每个位置分别用 Softmax 预测它是起点/终点的概率。BERT 在 SQuAD 等基准上一度"屠榜"。

关键：BERT 主体被微调，顶部分类器从随机初始化"从头学习（learned from scratch）"。

BERT　微调　下游任务

7 GPT：基于解码器的自回归语言模型

GPT（Generative Pre-trained Transformer） 是基于 **Transformer 解码器** 的预训练语言模型，与 BERT 相对。

预训练任务就是经典的**自回归语言模型（Autoregressive LM）**：输入前 $n-1$ 个词，让模型预测第 n 个词。训练目标是最大化序列的似然，即最小化负对数似然：

$$\mathcal{L}(\theta) = -\sum_{t} \log p_\theta(x_t \mid x_1, \ldots, x_{t-1})$$

GPT 与 BERT 的对比：

- **方向性**：GPT 是**单向（从左到右）**的，预测时只能看到前文；BERT 是**双向**的。
- **任务**：GPT 天生擅长**生成**；BERT 擅长**理解**。
- **规模演进**：BERT 约 340M 参数，GPT-2 约 1.5B（1542M），GPT-3 达 175B，参数量级不断跃升。

这种"预测下一个词"看似简单，却让模型在海量文本上学到了语法、知识乃至推理的雏形。

GPT　解码器　自回归

8 从预训练语言模型到大语言模型（LLM）

大语言模型（Large Language Model, LLM） 是"可适配多种任务的超大规模预训练语言模型"。沿着 GPT 的发展史，模型规模一路飙升：GPT-2（约 3.45 亿）→ GPT-3（约 15 亿/百亿千亿级）→ ChatGPT（约 1750 亿）→ GPT-4（约 1.7 万亿）等，参数量级不断跃升。

关键直觉：

- LLM 仍然是"预训练语言模型"，只是把规模（参数量、数据量、算力）推到了极大，从而获得通用、可迁移的能力。
- 规模足够大之后，模型表现出小模型所没有的新能力（见"涌现能力"），可以**不经任务专门训练**，仅靠自然语言提示就完成翻译、推理、写代码等任务。

这正是为什么近年要不断把模型"做大、喂更多数据、算得更久"——但单纯堆规模成本极高，DeepSeek 的意义正在于以"低成本"达到媲美顶尖模型的性能。

LLM　规模　预训练

9 涌现能力（Emergent Ability）

涌现能力指：随着 LLM 规模增大，模型**突然出现了训练目标（文本补全）之外的新行为/新能力**——这些能力在小模型上几乎不存在，到某个规模后"顿悟"般出现，呈高度**非线性**增长。

核心特点：

- 涌现能力让 LLM 能在**没有任务专门训练**的情况下完成任务，只需用自然语言把任务"描述"出来（如直接用提示让模型做翻译、推理、写代码）。
- 这是"上下文学习（In-Context Learning）""思维链推理"等能力的基础。

> 直觉：就像大脑规模稳步增长，到一定程度会涌现出质变的智能。涌现现象解释了为何"大"模型与"小"模型不仅是量的差别，更可能带来能力上的跃迁。

涌现能力　LLM　规模

10 利用 LLM 的两大范式：微调 vs 提示工程

如何使用一个预训练好的 LLM？主要有两大思路：

- **微调（Fine-tuning）**：在下游任务数据上**继续训练、更新参数**。包括**全量微调（Full fine-tuning）**和**参数高效微调（PEFT）**。优点是适配充分、效果上限高；缺点是

1. **计算开销大**（大模型全量更新代价高）；

2. **容易过拟合**（下游数据少时尤甚）。

- **提示工程（Prompt Engineering）**：**冻结参数、不更新**，仅通过设计输入提示来引导模型。包括零样本、小样本（上下文学习）、思维链、ReAct 等。优点是无需训练、灵活快速。

此外还有进阶用法：**检索增强生成（RAG）** 为模型补充外部知识，**Agent（智能体）** 让模型自主规划与调用工具。实际系统往往是这些方法的组合。

LLM　微调　提示工程

11 全量微调与参数高效微调（PEFT）

全量微调（Full fine-tuning） 更新模型的**全部参数**，把预训练 LLM 适配到具体任务（如把通用模型微调成专门的问答/摘要/分类模型）。但它存在两大问题：**计算昂贵**与**过拟合风险**。

由此引出问题：**能否只优化一小部分参数？** —— 这就是**参数高效微调（Parameter-Efficient Fine-Tuning, PEFT）** 的目标：

- **冻结**预训练模型的绝大部分参数，只训练**少量新增或选定**的参数；
- 大幅降低显存与计算成本，同时缓解过拟合，且便于为不同任务保存多个轻量"插件"。

Huggingface 等生态提供了便捷的 PEFT 工具：安装 PEFT 库后，给模型"添加 adapter"即可开始高效微调。LoRA 是最具代表性的 PEFT 方法之一。

PEFT　微调　效率

12 LoRA：低秩适配——给大模型"贴膜"

LoRA（Low-Rank Adaptation） 的核心思想：用**低秩分解**来表示权重更新量，而非直接更新庞大的原权重矩阵。

设预训练权重为 $W \in \mathbb{R}^{d\times d}$，微调要学习的更新量 ΔW 被分解为两个**小矩阵**之积：

$$\Delta W = W_A W_B, \quad W_A \in \mathbb{R}^{d\times r},\ W_B \in \mathbb{R}^{r\times d},\ r \ll d$$

前向计算变为 $h = Wx + \Delta W x = Wx + W_A W_B x$。

要点：

- **冻结** W，只训练 W_A, W_B；由于秩 $r \ll d$，需训练的参数量极小。
- 形象比喻：原模型像"名画"，LoRA 像在画上贴一层"薄膜"，**只在薄膜上编辑**而不动原作，既高效又可随时拆换。

优点：显存/算力开销小、训练快、可为多任务维护多套小 adapter；推理时可将 ΔW 合并回 W，几乎不增加推理延迟。

LoRA　PEFT　低秩分解

13 提示工程与零样本/小样本提示

提示工程（Prompt Engineering）：用任务指令"提示"LLM，**不更新任何参数**。一个好的提示常包含角色（Role）、背景（Context）、指令（Instruction）等结构化部分。

按是否给示例分为：

- **零样本提示（Zero-shot）**：只给任务描述，不给任何示例，直接让模型作答。
- **小样本提示（Few-shot）**：在任务描述之外，再给**几个示例（k 个 demonstration）**，模型模仿这些示例来回答新问题。例如给若干"评论 ⇒ 情感"的例子，再让模型判断新评论的情感，**全程不做梯度更新**。

> 小样本提示是一种"现学现用"：示例只出现在输入上下文里，模型权重始终冻结。

提示工程　零样本　小样本

14 上下文学习（In-Context Learning, ICL）

上下文学习（In-Context Learning, ICL） 是小样本提示背后的能力：模型通过**模仿输入上下文中给出的示例**来完成新任务，而**不更新任何参数**。

关键特征：

- "学习"发生在**前向推理过程**中，靠注意力机制"读懂"示例的输入—输出模式，而非传统意义上的训练（无梯度、无权重更新）。
- 它是 LLM **涌现能力**的典型体现：足够大的模型才表现出可靠的 ICL。

意义：ICL 让同一个冻结的模型仅通过改变提示就能切换任务，极大降低了使用门槛——无需为每个任务单独训练模型，"现学现用"。

> 对比：微调改变的是模型"权重"，ICL 改变的只是模型当下的"上下文"。

上下文学习 ICL 小样本

15 思维链提示（Chain-of-Thought, CoT）

思维链提示（Chain-of-Thought, CoT）：引导模型**逐步输出中间推理过程**，再给出最终答案，从而显著提升复杂推理（数学、逻辑、多步问题）的准确率。

两种常见做法：

- **Few-shot CoT**：在示例中不仅给答案，还把"得到答案的具体推理步骤"写清楚，让模型模仿这种"分步思考"。
- **Zero-shot CoT**：在问题后加一句"让我们一步步思考（Let's think step by step）"，即可诱导模型展开推理。

例如算"5 行×4 棵玫瑰 + 3 行×6 棵向日葵"，模型会先算 $5 \times 4 = 20$、再算 $3 \times 6 = 18$、最后 $20 + 18 = 38$，分步得出共 38 棵植物。

价值：CoT 让推理过程**可见、可验证**（如金融计息可逐步审计），减少"跳步出错"，是激发大模型推理能力的关键提示技术，也是 o1/DeepSeek-R1 等推理模型的核心思想之一。

思维链 CoT 推理

16 ReAct：推理 + 行动

ReAct（Reasoning + Acting） 让 LLM **交错地输出推理（Reasoning）步骤和具体的动作（Action）**：

- **Reasoning（思考）**：模型用语言推断当前该做什么；
- **Action（行动）**：模型生成调用**外部工具**的动作（如搜索、查数据库、调用计算器），获取额外信息后再继续推理。

核心价值：ReAct 框架让 LLM 能够**与外部工具/环境交互**，检索到模型自身不掌握的额外信息，从而给出**更可靠、更符合事实**的回答，减少凭空编造。

它把"想—查—再想—再答"串成一个循环，是连接"纯文本推理"与"工具调用/智能体"的桥梁。

ReAct 工具调用 推理

17 使用 LLM API 进行大模型编程

除了在本地加载模型，还可以直接通过**大模型厂商提供的 API** 来调用大模型，把它当作一项云端服务来用，无需自己部署庞大的模型。

基本步骤：

1. **找到 API 平台**：在大模型官网通常都有 API 平台的入口链接。
2. **充值并获取 API Key**：注册账号、充值后，平台会发放一个 **API Key（密钥）**，用于身份认证与计费。
3. **在程序中调用**：在自己的代码里使用这个 API Key，按照官方接口直接发起请求调用大模型（传入提示，拿到模型回复）。

要点：

- 这种方式**无需下载/部署模型、无需本地显卡**，开发门槛低、上手快，适合快速搭建应用。
- API Key 相当于"通行证 + 账单"，要妥善保管、避免泄露。

> 直觉：本地微调/部署是"自己买设备开工厂"，调用 API 则是"按需向云端工厂下单"，付费即用。

API　LLM　编程

18 检索增强生成（RAG）

检索增强生成（Retrieval-Augmented Generation, RAG） 解决的问题是：LLM 可能**缺乏最新知识或专用领域知识**（例如问"某门课的授课教师是谁"，模型只会说"不知道"或编造）。RAG 通过**检索外部文档**为模型补充知识。

通常包含三大核心组件：

1. **向量嵌入模型（Embedding Model）**：把文档/查询转成向量，以便快速检索；
2. **向量数据库（Vector Database）**：以向量形式存储知识并建立索引；
3. **大模型（LLM）**：基于检索到的知识生成答案。

基本流程分三阶段：

- **索引（Indexing）**：把文档切成小块（chunk）并嵌入成向量存入数据库；
- **检索（Retrieval）**：对新查询计算查询向量，与库中向量比相似度，取 **Top-K** 最相关的 chunk；
- **生成（Generation）**：把检索到的 chunk **拼接到原始提示**中一起喂给 LLM，生成有依据的回答。

优点：无需重新训练即可注入新知识/私有知识，显著**减少模型凭空编造**、提升时效性与可溯源性。

RAG　检索　向量数据库

19 LLM 智能体（Agent）

LLM Agent（智能体） 以大模型为"大脑"，是具有**自主思考、决策与执行任务能力**的 AI 系统，区别于被动应答的普通 LLM。其能力包括：

- **主动交互**：理解用户的高层目标并主动采取行动；
- **任务分解与规划**：把复杂问题拆成可执行子任务并规划求解路径；
- **调用工具**：灵活使用检索、数据库、计算器等外部工具，而不仅靠模型自身；
- **反思与进化**：评估自身行为效果并调整改进。

核心组件：① 大模型（大脑，负责理解/推理/决策）；② 工具集 Tools（通过生成函数调用/JSON 等结构化指令触发执行）；③ 记忆 Memory（短期上下文 + 长期经验，常用向量数据库以"索引—内容"条目存储）；④ 控制循环。

工作流程通常是"感知 → 思考/规划 → 行动 → 反思"的循环：任务理解（Perception）→ 任务规划（Planning，拆解子任务）→ 工具调用（Action）→ 评估优化（Evaluation）→ 循环与终止（Iteration），直到任务完成。

Agent　智能体　工具调用

20 常用的 LLM 工具

围绕大模型的使用与开发，课程介绍了几款常用工具，覆盖"本地部署 → 知识管理 → 微调 → 应用开发"全链路：

- **Ollama**：用于在**本地部署**大语言模型。下载安装包后，用命令行即可快速运行模型，例如 `ollama run llama3.2:1b`。让大模型可以离线、私有地跑在自己的机器上。
- **AnythingLLM**：一个**本地知识管理 / 文档问答**框架，支持把本地文档与 LLM 结合（可对接 Ollama 中下载的模型）。通过上传文档构建本地**向量数据库（embedding）**，即可对私有文档做问答——本质上是一套开箱即用的本地 RAG 系统。
- **LLaMA-Factory**：一站式**大模型微调平台**，提供命令行与 Web 两种方式，可指定不同微调方式（如 LoRA 等），并配套数据格式（如 alpaca 格式）来便捷地微调模型。
- **LangChain**：用于**开发由 LLM 支持的应用程序**的框架，便于把提示、工具调用、检索、链式流程等组件组合起来构建复杂应用。

> 记忆线索：Ollama 跑模型、AnythingLLM 管文档/做本地问答、LLaMA-Factory 做微调、LangChain 搭应用。

工具　Ollama　微调

21 前沿案例：DeepSeek 的架构与训练创新

DeepSeek 以**显著更低的训练成本与推理价格**，在主流评测基准上达到媲美顶尖模型的性能，塑造了"低成本、高效能"的新范式。其突破更多是站在前人肩膀上的**高壁垒工程创新**：

- **架构：混合专家（Mixture of Experts, MoE）**。MoE 概念早在 1991 年就提出，DeepSeek 在其上做了关键改进——用**多头潜在注意力**让不同注意力头**动态选择专家**、采用**无辅助损失的负载均衡**、**多标记预测**目标、并在 6710 亿总参数中**仅激活约 370 亿参数**，从而兼顾能力与推理效率（更省显存、更低成本）、训练更稳定。
- **训练方法：思维链 + 强化学习**。在"现存数据收敛、单纯预训练边际收益下降"的背景下，转向"CoT + RL"的新范式。DeepSeek-R1 采用**多阶段训练**（冷启动 SFT → 推理 RL → 通用 SFT → 全场景 RL，先专精再泛化）、**群体相对策略优化（GRPO）**（每次生成多个答案、选组内最优作为改进方向，省去单独的奖励/评判模型）、以及**拒绝采样**自动筛选高质量数据形成"越练越强"的闭环。

这反映了 LLM 训练范式的演进：从"堆规模做预训练"走向"后训练 + 强化学习"提升推理能力。

DeepSeek　MoE　强化学习

22 📝 例子1：同一道题，三种提示写法（Zero-shot / Few-shot / CoT）

假设我们要让一个**冻结参数**的大模型解这道小学应用题：

> 食堂有 23 个苹果，中午用掉了 20 个，下午又买了 6 个，现在还有几个?

下面用三**种提示工程**写法，看看为什么思维链（CoT）最稳。

写法一：Zero-shot（零样本，只给问题）

```
Q：食堂有23个苹果，用掉20个，又买6个，现在有几个?
A：
```

- 模型可能直接蹦出一个数字，比如 `9`（错的，正确是 23−20+6=9… 这题恰好对，但换成多步数字时模型常"跳步"算错）。
- **为什么容易错**：模型被训练成"补全下一个词"，零样本时它倾向一**步给答案**，中间运算全在"脑子里"，没有展开，错了也看不出来。

写法二：Few-shot（小样本，给几个示范）

```
Q：树上有5只鸟，飞走2只，又来3只，现在几只？  A：6
Q：盘子里有8块饼干，吃掉5块，再放4块，现在几块？  A：7
Q：食堂有23个苹果，用掉20个，又买6个，现在有几个？  A：
```

- 我们给了 2 个"问题⇒答案"的示范（demonstration），模型**模仿格式**作答。
- **为什么有用**：示范让模型明白"输出一个数字即可"，统一了格式；但它**仍然只给最终数字**，多步运算依旧靠"心算"，复杂题还是会错。

写法三：Chain-of-Thought（思维链，要求分步思考）

```
Q：食堂有23个苹果，用掉20个，又买6个，现在有几个?
A：让我们一步步思考。
```

模型输出：

```
第1步：原有 23 个。
第2步：用掉 20 个，剩 23 − 20 = 3 个。
第3步：又买 6 个，3 + 6 = 9 个。
所以现在有 9 个。
```

- **为什么 CoT 让推理题更准**：

1. **拆解难度**：把"一步算出答案"拆成几个**简单子步骤**，每步只做一次加减，出错概率大大降低。

2. **更多"思考算力"**：每多写一个中间步骤，模型就多用一些 token（即多一些前向计算）去推理，相当于"把草稿纸写出来"。

3. **可见可验证**：中间过程白纸黑字，错在哪一步一眼能看出来，便于纠正。

> 一句话记忆：Zero-shot=直接问；Few-shot=给几个例子让它照着答；CoT=逼它把草稿写出来。题目越需要多步推理，CoT 的优势越明显。

例子

23 📝 例子2：打地基→学专业——预训练与微调两阶段

很多同学分不清"预训练"和"微调"，我们用**培养一名医生**来类比，走一遍"预训练—微调"两阶段范式。

阶段一：预训练（Pre-training）—— 打地基，博览群书

- 做法：在**海量无标注文本**（整个互联网、电子书、维基）上做"预测下一个词"（GPT）或"完形填空"（BERT 的掩码语言建模）。没有人工标注，监督信号来自文本本身。
- 类比：让一个孩子**读完了一百万本书**，认识了字、懂了语法、积累了常识——但他还**不知道**该怎么针对具体任务好好回答。
- 产物：一个"什么都略懂、但不针对具体任务"的基座模型（base model）。比如给它"中国的首都是"，它会接"北京"；但你直接问它问题，它可能继续"补全"出更多问题而不是回答你。

阶段二：微调（Fine-tuning）—— 学专业，专项突击

- 做法：用**少量高质量的标注数据**继续训练、更新参数，把通用基座适配到具体任务（如问答、摘要、分类，或"看到指令就给出对应回答"）。
- 类比：孩子去读**医学院**，老师给他成千上万对"病例→标准诊断"，他学会了**针对问题给专业回答**。
- 例子：给模型大量 `指令：把这句话翻成英文 / 输入：我爱机器学习 / 输出：I love machine learning`，它就学会了听从翻译指令。
- 注意：**全量微调**更新全部参数，**计算贵、易过拟合**；所以常用 **LoRA** 这类参数高效微调（PEFT），冻结原权重、只训练"薄膜"小矩阵，既省算力又缓解过拟合。

> 两阶段口诀：**预训练打地基（在海量无标注文本上学通用语言能力）→ 微调学专业（用少量标注数据适配具体任务）**。越往后，数据越少越精，越贴近具体任务需求。

例子

24 📝 例子4：一个词一个词地往外蹦——自回归生成全过程

GPT 类模型是**自回归（autoregressive）**的：每次只预测**下一个 token**，把它接到输入末尾，再预测下下一个，循环往复。我们用提示 `机器学习很` 走一遍。

前提：模型的输入是"前面所有词"，输出是"词表上每个词作为下一个词的概率分布"。

第 1 步：输入 `机器学习很`

- 模型对整个词表打分，得到概率分布，例如：

- `有趣` → 0.55

- `重要` → 0.20

- `难` → 0.15

- 其他 → …

- 选概率最高的 `有趣`（贪心解码），**接到末尾**。
- 当前序列变成：`机器学习很有趣`

第 2 步：把**新序列**整体 `机器学习很有趣` 再喂回模型

- 注意：这里**输入变长了**，模型每一步都能看到**之前生成的全部内容**（这就是"自回归"——用自己之前的输出当输入）。
- 新的概率分布，例如：

- `，` → 0.40

- `。` → 0.30

- `而且` → 0.10

- 选 `，`，序列变成：`机器学习很有趣，`

第 3 步：输入 `机器学习很有趣，`

- 预测出 `值` → 0.35，接上 → `机器学习很有趣，值`

第 4 步：输入 `机器学习很有趣，值`

- 预测出 `得` → 0.6，接上 → `机器学习很有趣，值得`

…… **一直循环** ……

- 直到模型生成一个特殊的**结束符** `<eos>`（**end of sentence**），或达到设定的最大长度，生成才停止。
- 最终可能得到：`机器学习很有趣，值得好好学习。`

几个要点解释为什么：

1. **为什么叫"自回归"**：因为第 t 步的输入 = 原始提示 + **前面所有自己生成的词**，模型"回过头"用自己的产出，公式是 $p_\theta(x_t \mid x_1, \ldots, x_{t-1})$。
2. **为什么是"一个一个"出**：聊天界面上文字"一个个蹦出来"的打字机效果，正是这个逐 token 生成过程的直接体现。
3. **解码策略影响结果**：每步选概率最高的叫**贪心**；若按概率"抽签"（采样，带 temperature）则更多样、也更可能跑偏。

> 一句话：大模型写句子不是"一次想好整句"，而是**预测一个词→接上→再预测下一个**，循环到结束符为止。

例子

第 13 讲 卷积神经网络

CNN

卷积神经网络(Convolutional Neural Network, CNN)针对图像数据设计，用**局部连接(local connectivity)**、**权值共享(weight sharing)** 的卷积核(filter)在空间上滑动提取特征，保留图像的空间结构；配合步长(stride)、填充(padding)、池化(pooling)逐层降采样，最终堆叠成 LeNet、AlexNet、VGG、ResNet 等经典深层网络。卷积输出尺寸由公式 $\lfloor \frac{N+2P-F}{S} \rfloor + 1$ 决定。

1 图像建模的挑战：为什么 MLP 不够用

计算机看到的图像就是一个数字矩阵：一张 RGB 图像是 $W \times H \times 3$ 的张量，每个像素取值 $[0, 255]$。最朴素的想法(Idea #1)是把所有像素**拉直成一维向量**(如 $32 \times 32 \times 3 = 3072$ 维)送进全连接网络(MLP)。但这会带来一系列问题：

- **空间结构被破坏**：拉成 Seq-of-Pixels 后，相邻像素的二维邻接关系丢失。
- **对局部不敏感**：每个隐藏单元都关注整张图，而非局部特征。
- **平移不变性差(translation variant)**：物体在图中位置稍微移动，输入向量就完全改变，模型很敏感。
- **参数量爆炸**：全连接层把 3072 维接到一个隐藏层就需要海量参数。

此外图像还有视角变化(viewpoint)、光照(illumination)、形变(deformation)、遮挡(occlusion)、背景干扰(background clutter)、类内差异(intra-class variation)等天然困难。手工设计特征(Idea #2，如"轮子+车牌+车灯")又难以穷举且不通用。

> 结论：我们需要一种能**自动学习特征**(Idea #3, Deep Learning)、又能保留空间结构的网络结构——这就是 CNN。

图像建模 MLP局限 动机

2 图像建模的四条设计准则

要为图像设计好的网络，PPT 总结了四条准则：

1. **保留空间结构(keep the spatial structure)**：不要一上来就把图拉直。
2. **对局部敏感(sensitive to locality)**：先关注小的局部区域，扫描感兴趣的物体。
3. **处理输入变化(handle variants)**：对平移、形变等具有一定鲁棒性。
4. **跨空间区域共享参数(share parameters)**：同一个特征检测器在全图复用。

这四条准则直接对应 CNN 的三大核心思想：

- 局部连接 → 准则 2；
- 权值共享 → 准则 4；
- 卷积+池化保留并逐步抽象空间信息 → 准则 1、3。

> 生物学启发：Hubel & Wiesel (1959) 发现猫的视觉皮层(striate cortex)中每个神经元都有特定形状的**感受野(receptive field)**，且皮层中相邻细胞对应视野中相邻区域。CNN 正是模仿了这种"局部感受 + 层级组织"的机制。

设计准则　生物启发　感受野

3 卷积核与卷积运算

卷积核(convolutional kernel / filter) 就是一组赋给输入局部区域的权重，由一个隐藏单元承担。CNN 的基本思想是：把输入层的一个**局部patch**连接到下一层的一个神经元，让它在图上滑动扫描某个特定特征("这里有没有嘴巴?")。

以 1-D 卷积、滤波器大小为 2 为例，第一个输出：

$$h_1 = w_1 x_1 + w_2 x_2 + w_0.$$

滑动到下一位置(stride=1)复用**同一组权重**：

$$h_2 = w_1 x_2 + w_2 x_3 + w_0, \quad h_3 = w_1 x_3 + w_2 x_4 + w_0, \dots$$

2-D 卷积是元素逐位相乘再求和(elementwise multiplication and sum)：把 $F \times F$ 的滤波器盖在图上对应区域，对应元素相乘后累加得到一个输出值，例如

$$h_{11} = \sum_{i,j} w_{ij}\, x_{ij}.$$

滤波器扫遍所有空间位置，得到一张**特征图(feature map)**。

> 之所以叫"卷积"，是因为该运算与两个信号的卷积相关(严格说深度学习里实现的是互相关，但习惯统称卷积)。卷积保留了输入的二维空间信息。

卷积核　卷积运算　feature map

4 局部连接与权值共享

对比全连接网络(MLP)与卷积网络(CNN)，差别在两点：

- **局部连接(local connectivity)**：MLP 中每个隐藏单元连接**全部**输入单元("我总是看整张图")；CNN 中每个隐藏单元只连接输入的一个**局部区域**("我只看我感兴趣的小区域")。这把隐藏单元的视野限制在局部特征上。
- **权值共享(weight sharing)**：同一个滤波器在全图滑动时，所有位置**复用同一组权重**("Parameters are shared over the whole picture for one filter")。

这两点带来的好处：

- **参数量大幅减少**：一个 $F \times F$ 滤波器只有 F^2(加偏置)个参数，与图像大小无关。
- **平移等变性(translation equivariance)**：同一特征出现在不同位置都能被同一滤波器检测到。
- **保留空间结构**：输出本身仍是二维的特征图。

> 直觉：检测"竖直边缘"这个能力，在图像左上角和右下角是一样的，没必要为每个位置单独学一套权重。

局部连接　权值共享　参数量

5 步长(Stride)与多滤波器

步长(stride, S) 控制滤波器每次滑动跳过多少像素：

- $S = 1$ 时逐像素滑动，输出最密、尺寸最大；
- $S = 2$ 时每隔一个像素扫一次，输出尺寸约减半，相当于一种降采样。

大步长 = 更激进的降采样、更少计算、更小输出；小步长 = 更精细、保留更多空间细节。

多滤波器(multiple filters)：一个滤波器只能检测一种特征，要同时检测多种特征(边缘、纹理、角点……)就用多个滤波器，每个滤波器配一套独立权重 w, w'：

$$h_t = w_1 x_t + w_2 x_{t+1} + w_0, \qquad h'_t = w'_1 x_t + w'_2 x_{t+1} + w'_0.$$

有几个滤波器，就得到几张特征图，它们沿通道方向堆叠(stack)成输出。

> 记忆：**一个滤波器 ⇒ 一张特征图(one filter ⇒ one feature map)**。滤波器个数 = 输出通道数。

stride　多滤波器　降采样

6 通道、3-D 卷积与特征图堆叠

彩色图像有多个**输入通道(input channels)**(R、G、B)，所以二维卷积实际上是 **3-D 卷积**：

- 滤波器**总是贯穿输入体的整个深度**(filters always extend the full depth of the input volume)。例如对 $32 \times 32 \times 3$ 的图像，用 $5 \times 5 \times 3$ 的滤波器(深度必须等于输入通道数 3)。
- 滤波器在空间上滑动，每个位置做一次跨所有通道的点积，三个通道结果相加(再加偏置)得到一**个**输出值。因此一个 $5 \times 5 \times 3$ 滤波器扫过 $32 \times 32 \times 3$ 图像，得到一张 $28 \times 28 \times 1$ 的特征图。
- 用 K 个滤波器，就得到 K 张特征图，堆叠成 $28 \times 28 \times K$ 的输出体。

例：$3 \times 32 \times 32$ 输入图，用 6 个 5×5 滤波器(参数张量 $6 \times 3 \times 5 \times 5$)→ 6 张 28×28 激活图 → 输出 $6 \times 28 \times 28$。

> 关键规则：**滤波器深度 = 输入通道数；滤波器个数 = 输出通道数**。

通道　3D卷积　特征图堆叠

7 特征图(Feature/Activation Map)与多层堆叠

特征图(feature map / activation map) 记录了某个滤波器在各空间位置被激活的程度——"在哪里检测到了这个特征"。比如一个检测"嘴巴"的滤波器，在嘴巴所在位置输出值很高。

CNN 把若干**卷积层堆叠**起来，层与层之间插入**激活函数**(如 ReLU)：

$$\text{CONV} \to \text{ReLU} \to \text{CONV} \to \text{ReLU} \to \cdots$$

例如：$32 \times 32 \times 3 \to (6\text{ 个 }5 \times 5 \times 3) \to 28 \times 28 \times 6 \to (10\text{ 个 }5 \times 5 \times 6) \to 24 \times 24 \times 10 \to \cdots\cdots$

注意第二层滤波器的深度(6)必须等于上一层输出通道数。

层级特征(hierarchical features)：可视化表明，浅层滤波器学到的是边缘、颜色、简单纹理等**低级特征**；越深的层组合出眼睛、轮子等**高级、语义化的特征**。这种由简到繁的层级抽象正是深度卷积网络的威力所在。

特征图　多层堆叠　层级特征

8 填充(Padding)

卷积时**边缘神经元**会遇到问题：滤波器盖到图像边界外侧没有输入。同时每做一次卷积，特征图尺寸都会缩小，深层网络会让空间维度迅速消失。

(零)填充(Zero-Padding) 解决之：在输入矩阵四周**对称地补零**，补 P 圈零。作用：

- 让滤波器能覆盖到边缘像素，边缘信息不被丢弃；
- **控制输出尺寸**，使其满足我们的需求(常用来让输出和输入同尺寸，即 "same" 卷积)。

对 $N = 7, F = 3$:

- 不填充、$S = 1$：输出 $\frac{7-3}{1} + 1 = 5$；
- 填充 $P = 1$、$S = 1$：输出 $\frac{7+2-3}{1} + 1 = 7$，与输入同尺寸。

> 经验法则：对 $F \times F$、$S = 1$ 的卷积，取 $P = \frac{F-1}{2}$ 可保持空间尺寸不变(如 $F = 3 \Rightarrow P = 1$，$F = 5 \Rightarrow P = 2$)。

padding　零填充　边缘

9 输出尺寸计算公式

这是 CNN 最常考的计算。设输入空间尺寸 N、滤波器尺寸 F、步长 S、单边填充 P，则卷积后每个空间维的输出尺寸为：

$$\text{输出} = \left\lfloor \frac{N + 2P - F}{S} \right\rfloor + 1.$$

无填充时退化为 $\frac{N-F}{S} + 1$。

几个例子($N = 7, F = 3$)：

- 无填充 $S = 1$：$(7 - 3)/1 + 1 = 5$；
- 无填充 $S = 2$：$(7 - 3)/2 + 1 = 3$；
- 无填充 $S = 3$：$(7 - 3)/3 + 1 = 2.33$ ✗(不能整除，需调整 P 或 S)；
- $P = 1, S = 1$：$(7 + 2 - 3)/1 + 1 = 7$；$P = 1, S = 2$：$(7 + 2 - 3)/2 + 1 = 4$。

经典 Quiz：输入 $32 \times 32 \times 3$，用 10 个 5×5 滤波器、$S = 1$、$P = 2$：

$$\frac{32 + 2 \times 2 - 5}{1} + 1 = 32,$$

故输出体为 $32 \times 32 \times 10$(空间尺寸不变，通道数 = 滤波器个数 10)。

> 输出通道数永远等于滤波器个数，与 N, F, S, P 无关；后者只决定空间尺寸。

输出尺寸　公式　计算

10 池化(Pooling)

池化(pooling, 池化) 对特征图做**降采样(downsampling)**，让表示更小、更易管理，减少内存与计算，并引入一定的空间不变性。常见配置是 2×2 窗口、$S = 2$，使每个空间维减半(如 $224 \times 224 \times 64 \to 112 \times 112 \times 64$，**通道数不变**)。

两种常见池化：

- **最大池化(Max Pooling)**：取窗口内**最大值**，保留最显著(最亮)的响应，突出强特征、对小位移更鲁棒。
- **平均/均值池化(Mean/Average Pooling)**：取窗口内**平均值**，起平滑作用，但会模糊掉尖锐特征。

池化的关键性质：

- **没有可学习参数(no learnable parameters)**——只是固定的取最大/取平均操作；
- **引入空间不变性(spatial invariance)**——物体轻微平移后输出基本不变；
- 池化**不改变通道数**，也"不改变物体本身"。

池化　max/mean　降采样

11 CNN 总体架构(The Big Picture)

一个典型 CNN 是若干"卷积 + 激活 + 池化"模块的堆叠，最后接全连接层做分类：

$$[\text{CONV} \to \text{ReLU} \to \text{POOL}] \times n \to \text{FC} \to \text{输出(类别分数)}.$$

- **卷积层**：提取局部特征，参数为滤波器权重(局部连接 + 权值共享)。
- **激活(ReLU)**：引入非线性。
- **池化层**：降采样，缩小空间尺寸、增强不变性、无参数。
- **全连接层(FC)**：把高层特征整合为类别得分。

两个思考题(PPT 留白)：

- **CNN 能否退化成 MLP?** 能——当滤波器尺寸等于整张特征图($F = N$、$P = 0$、每个输出只连一个全局感受野)时，卷积层就等价于全连接层。
- **CNN 能否用于文本?** 能——把文本看作 1-D 序列(词向量序列)，用 1-D 卷积在序列上滑动即可(text-CNN)。

总体架构 CONV-POOL-FC 通用性

12 经典网络:LeNet-5 / AlexNet / VGG

LeNet-5 [LeCun et al., 1998]：最早的实用 CNN，结构 [CONV1-POOL1-CONV2-POOL2-FC1-FC2]。卷积核 5×5、$S = 1$；下采样(池化)2×2、$S = 2$。用于手写数字识别。

AlexNet [Krizhevsky et al., NIPS 2012]：首个深度学习 ImageNet 冠军(8 层)，开启深度学习浪潮。结构要点：CONV1 用 96 个 11×11、$S = 4$ 大卷积核；随后 3×3 MaxPool($S = 2$)、归一化层(NORM)、CONV2(256 个 5×5)、CONV3-5(3×3)，最后 FC6/FC7 各 4096 神经元、FC8 输出 1000 类。它是首个深度学习在 ILSVRC 上夺冠的网络，标志着深度学习时代的开端。

VGGNet [Simonyan & Zisserman, 2015]：只用 3×3 **小卷积核**($S = 1$)和 2×2 MaxPool($S = 2$)，把网络做得更深(16/19 层)。核心洞见：**堆叠 3 个 3×3 卷积层的有效感受野等于 1 个 7×7 卷积**，但更深、非线性更多、参数更少——3 个 3×3 的参数 $3 \times (3^2C^2) = 27C^2$ 远小于 7×7 的 $49C^2$(C 为每层通道数)。

LeNet AlexNet VGG

13 感受野与 ResNet 残差连接

感受野(receptive field)：输出特征图上一个像素"看到"的输入区域大小。卷积越深、池化越多，感受野越大。VGG 的关键论点正是用小核**堆叠**来扩大感受野：k 个 3×3($S=1$)层的有效感受野为 $(2k+1)\times(2k+1)$，例如 3 层 $\rightarrow 7\times7$。

深度的困境：在"普通"CNN 上一味加深，更深的模型反而**训练误差更高**——且这不是过拟合，而是**优化变难**(梯度难以传播)。

ResNet [He et al., 2015] 用**残差连接(residual / skip connection)** 解决之。每个残差块含两层 3×3 卷积，学习的是残差 $F(x)$，输出为

$$y = F(x) + x.$$

好处：

1. 梯度可通过"高速通道(highway)"快速回传(类比 LSTM 的门控通路)，缓解梯度消失；
2. 每块只需学习**小的残差**，更易优化；
3. 多余层可学成**恒等映射(identity mapping,** $F(x)=0$)，保证深模型至少不差于浅模型。

凭此 ResNet 训出 152 层模型，横扫 ILSVRC'15 与 COCO'15，被称为"深度的革命(The Revolution of Depth)"。网络规模演进：AlexNet(5 conv)→ VGG(16)→ GoogLeNet(22)→ ResNet(>100)。

感受野　ResNet　残差连接

14 展望：计算机视觉的世界(What's Next?)

CNN 学到的特征不止能做"这是猫还是狗"的**图像分类**，它是整个计算机视觉世界的基石。PPT 最后展望了 CNN 之上更丰富的视觉任务：

- **图像分类(Classification)**：判断整张图属于哪个类别(如"cat"/"dog")，是最基本的任务，也是本讲 CNN 主要解决的问题。
- **目标检测(Object Detection)**：不仅识别图中有哪些物体，还要用边界框**定位**它们的位置(如同时框出图中的猫和狗)。
- **语义分割(Semantic Segmentation)**：对**每个像素**做分类，给整张图打上逐像素的标签(如把图分成 GRASS、CAT、TREE、SKY……区域)。
- **图像描述(Image Captioning)**：为图像生成一句自然语言描述(如"A cat standing on the grass")，是视觉与语言的结合。
- **图像生成(Image Generation)**：反过来由模型**生成**新的图像。

> 这些都是在 CNN 提取的层级特征之上构建的下游任务，体现了卷积网络作为视觉"特征提取器"的通用价值——把图像编码成有意义的表示后，可以接不同的"任务头"去做分类、定位、分割、描述或生成。

计算机视觉　下游任务　展望

15 📝 例题1：手动滑动算卷积，得出特征图

任务：给定一张 5×5 的输入图像和一个 3×3 的卷积核(滤波器)，步长 $S=1$、不填充 $P=0$，手动把卷积输出特征图算出来。

输入图像 X（5×5）：

$$X=\begin{bmatrix}1&2&3&0&1\\0&1&2&3&1\\1&0&1&2&0\\2&1&0&1&3\\1&2&1&0&2\end{bmatrix}$$

卷积核 K（3×3，这是一个常见的边缘检测核）：

$$K=\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$

第 0 步：先算输出有多大。 用公式 $\lfloor\frac{N+2P-F}{S}\rfloor+1=\frac{5+0-3}{1}+1=3$。所以输出是一张 3×3 的特征图，一共要算 9 个数。

> 卷积运算 = 把核盖在图上对应的 3×3 区域，**对应位置相乘再全部加起来**（elementwise multiply and sum）。

第 1 步：算左上角 h_{11}。 把核盖在 X 左上角的 3×3 块上：

$$\begin{bmatrix}1&2&3\\0&1&2\\1&0&1\end{bmatrix}\odot\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$

逐位相乘再求和：

- 第一行：$1\times1+2\times0+3\times(-1)=1+0-3=-2$
- 第二行：$0\times1+1\times0+2\times(-1)=0+0-2=-2$
- 第三行：$1\times1+0\times0+1\times(-1)=1+0-1=0$
- 合计：$h_{11}=-2-2+0=\boxed{-4}$

第 2 步：核向右滑一格（stride=1），算 h_{12}。 盖住第 1~3 行、第 2~4 列：

$$\begin{bmatrix}2&3&0\\1&2&3\\0&1&2\end{bmatrix}\odot K$$

- 行1：$2-0=2$（$2\times1+3\times0+0\times(-1)$）
- 行2：$1-3=-2$
- 行3：$0-2=-2$
- $h_{12}=2-2-2=\boxed{-2}$

第 3 步：再右滑一格，算 h_{13}。 盖住第 1~3 行、第 3~5 列：

$$\begin{bmatrix}3&0&1\\2&3&1\\1&2&0\end{bmatrix}\odot K$$

- 行1：$3-1=2$；行2：$2-1=1$；行3：$1-0=1$
- $h_{13}=2+1+1=\boxed{4}$

第 4 步：回到最左，核向下滑一格，算 h_{21}。 盖住第 2~4 行、第 1~3 列：

$$\begin{bmatrix}0&1&2\\1&0&1\\2&1&0\end{bmatrix}\odot K$$

- 行1：$0-2=-2$；行2：$1-1=0$；行3：$2-0=2$
- $h_{21}=-2+0+2=\boxed{0}$

第 5 步：按同样方法把剩下 4 个也算完（例如 h_{22}：盖住第 2~4 行、第 2~4 列 $\begin{bmatrix}1&2&3\\0&1&2\\1&0&1\end{bmatrix}$，左列和 − 右列和 $=(1+0+1)-(3+2+1)=2-6=-4$），最终得到完整特征图：

$$H=\begin{bmatrix}-4&-2&4\\0&-4&-1\\2&0&-3\end{bmatrix}$$

> **为什么这么算？** 这个核左列是 $+1$、右列是 -1，相当于「左边亮度减右边亮度」，所以它在**竖直边缘**处响应最大——这就是卷积核「检测特定特征」的含义。同一个核扫遍全图（权值共享），在哪里有竖直边缘，那里的输出值绝对值就大。

例题

16 📝 例题2：用输出尺寸公式连算 4 种配置

任务：牢记这个公式并反复套用——它是 CNN 最常考的计算。

$$\text{输出尺寸} = \left\lfloor \frac{N + 2P - F}{S} \right\rfloor + 1$$

其中 N=输入空间尺寸，F=卷积核尺寸，P=单边填充圈数，S=步长，$\lfloor\ \rfloor$ 表示向下取整。

例 (a)：$N = 32, F = 5, S = 1, P = 2$（经典 Quiz）

1. 先算分子：$N + 2P - F = 32 + 2 \times 2 - 5 = 32 + 4 - 5 = 31$
2. 除以步长：$31 \div 1 = 31$
3. 加 1：$31 + 1 = \boxed{32}$

> 输出空间尺寸 32×32，和输入一样大——这正是「same 卷积」。若用 10 个滤波器，输出体就是 $32 \times 32 \times 10$（**通道数=滤波器个数 10**，和 N, F, S, P 无关）。

例 (b)：$N = 7, F = 3, S = 2, P = 0$（增大步长）

1. 分子：$7 + 0 - 3 = 4$
2. 除以步长：$4 \div 2 = 2$
3. 加 1：$2 + 1 = \boxed{3}$

> 输出 3×3。对比 $S = 1$ 时输出是 5×5——步长翻倍，输出尺寸明显变小，这就是用步长做降采样。

例 (c)：$N = 224, F = 11, S = 4, P = 0$（AlexNet 第一层）

1. 分子：$224 + 0 - 11 = 213$
2. 除以步长：$213 \div 4 = 53.25$
3. 向下取整再加 1：$\lfloor 53.25 \rfloor + 1 = 53 + 1 = \boxed{54}$

> 注意这里出现了小数，需要**向下取整**。框架对放不下的边缘窗口直接丢弃。（AlexNet 论文实际用 227×227 输入正好整除得 55，这里演示取整规则。）

例 (d)：要让 $N = 32$、$F = 3$ 卷积后尺寸不变，P 取多少？（反推填充）

用经验公式 $P = \frac{F-1}{2} = \frac{3-1}{2} = 1$。验证：$\frac{32+2\times1-3}{1} + 1 = \frac{31}{1} + 1 = 32$ √

> **口诀**：$S = 1$ 时取 $P = \frac{F-1}{2}$ 就能保持尺寸（$F = 3 \Rightarrow P = 1$，$F = 5 \Rightarrow P = 2$，$F = 7 \Rightarrow P = 3$）。

例题

17 📝 例题3：对同一个 4×4 区域做最大池化和平均池化

任务：给定一张 4×4 特征图，用 2×2 窗口、步长 $S=2$ 分别做**最大池化**和**平均池化**，把结果手算出来。这正是 PPT 第 44 页的例子。

输入特征图 X（4×4）：

$$X=\begin{bmatrix}12&7&13&14\\8&7&5&3\\12&9&5&7\\13&2&10&3\end{bmatrix}$$

第 0 步：先算输出尺寸。 池化也用同一个公式（$P=0$）：$\frac{4-2}{2}+1=2$。所以输出是 2×2，要算 4 个数。2×2、$S=2$ 的窗口**互不重叠**，正好把图切成 4 个角块。

第 1 步：把 4×4 切成 4 个 2×2 块。

- 左上块：$\begin{bmatrix}12&7\\8&7\end{bmatrix}$
- 右上块：$\begin{bmatrix}13&14\\5&3\end{bmatrix}$
- 左下块：$\begin{bmatrix}12&9\\13&2\end{bmatrix}$
- 右下块：$\begin{bmatrix}5&7\\10&3\end{bmatrix}$

第 2 步：最大池化 = 每个块取最大值。

- 左上：$\max(12,7,8,7)=12$
- 右上：$\max(13,14,5,3)=14$
- 左下：$\max(12,9,13,2)=13$
- 右下：$\max(5,7,10,3)=10$

$$\text{MaxPool}(X)=\begin{bmatrix}12&14\\13&10\end{bmatrix}$$

第 3 步：平均池化 = 每个块取平均（4 个数相加除以 4）。

- 左上：$(12+7+8+7)/4=34/4=8.5$
- 右上：$(13+14+5+3)/4=35/4=8.75$
- 左下：$(12+9+13+2)/4=36/4=9$
- 右下：$(5+7+10+3)/4=25/4=6.25$

$$\text{MeanPool}(X)=\begin{bmatrix}8.5&8.75\\9&6.25\end{bmatrix}$$

> **对比与理解**：最大池化保留每个块里**最亮（最强）的响应**，突出显著特征、对小位移更鲁棒；平均池化把所有值揉成平均，起**平滑**作用，尖锐特征会被弱化。两者都**没有可学习参数**（只是固定的取大/取平均），且**不改变通道数**——若输入是 $4\times4\times64$，输出就是 $2\times2\times64$。

例题

18 📝 例题4：算卷积层参数量，和全连接对比省了多少

任务：同样要把一张 $32 \times 32 \times 3$ 的输入「变成」一张 32×32 的特征图，分别用①卷积层 ②全连接层，各需要多少参数？看看权值共享到底省了多少。

记住一条规则：一个卷积核的参数 = $F \times F \times C_{in} + 1$（最后的 $+1$ 是偏置 bias），其中 C_{in} 是输入通道数。整层参数 = 单核参数 × 核个数 K。

① 卷积层（1 个 5×5 核，$C_{in} = 3$，输出 1 张特征图）

1. 单个核的权重：$5 \times 5 \times 3 = 75$
2. 加上 1 个偏置：$75 + 1 = 76$
3. 只有 1 个核，所以整层就是 $\boxed{76}$ 个参数。

> 关键：这 76 个参数在 32×32 共 1024 个输出位置上**反复复用**（权值共享）——核滑到哪里都用同一套权重，参数量和图像大小**无关**。

② 全连接层（把输入全部像素接到同样多的输出神经元）

1. 输入拉直后维度：$32 \times 32 \times 3 = 3072$
2. 输出也要 $32 \times 32 = 1024$ 个神经元
3. 全连接 = 每个输出都连到每个输入：$3072 \times 1024 = 3\,145\,728$ 个权重
4. 再加 1024 个偏置：$3\,145\,728 + 1024 = \boxed{3\,146\,752}$ 个参数。

③ 对比

$$\frac{3\,146\,752}{76} \approx 41\,404 \text{ 倍}$$

> 也就是说，卷积层用 76 个参数干了全连接层 300 **多万**个参数才能干的事，省了约 **4 万倍**！这就是「局部连接 + 权值共享」的威力：参数少 → 不容易过拟合、训练快、还保留了空间结构。

④ 进阶：多核多层的例子。 若用 6 个 5×5 核（$C_{in} = 3$）：单核 $5 \times 5 \times 3 + 1 = 76$，6 个核共 $76 \times 6 = \boxed{456}$ 个参数，输出 $32 \times 32 \times 6$（这里用了 $P = 2$ 保持尺寸）。下一层若用 10 个 5×5 核，注意此时 $C_{in} = 6$（上层输出通道数），单核 $5 \times 5 \times 6 + 1 = 151$，10 个核共 $151 \times 10 = 1510$ 个参数。

> **核的深度必须等于输入通道数**——这是写参数量时最容易出错的地方，务必看清当前层的输入有几个通道。

例题

第 14 讲 计算机视觉

Computer Vision

本讲系统介绍计算机视觉的核心任务与代表方法：图像分类、语义分割(全卷积网络、转置卷积上采样、U-Net 编解码结构)、目标检测(R-CNN 系两阶段方法、Faster R-CNN 的 RPN 与锚框、YOLO 单阶段回归)、图像描述(Encoder-Decoder 与注意力)，以及视觉 Transformer(ViT)、CLIP 对比式图文预训练与 LLaVA 等视觉大模型(VLM)。

1 计算机视觉的核心任务谱系

计算机视觉旨在让机器理解图像内容。按输出粒度与形式，常见任务可分为：

- **图像分类(Classification)**：给整张图一个类别标签，如 "cat"。典型流程是 `卷积 + MLP + Softmax`，用交叉熵训练。
- **语义分割(Semantic Segmentation)**：给**每个像素**打类别标签(GRASS/CAT/TREE/SKY...)。
- **目标检测(Object Detection)**：定位图中每个物体并给出**边界框** + **类别**，如 cat/dog 各自的框。
- **图像描述(Image Captioning)**：为图像生成自然语言句子，如 "A cat standing on the grass"。
- **图像生成(Image Generation)** 等。

> 直觉：从"整图一个标签"到"每像素一个标签"再到"每物体一个框"，输出空间逐渐变得结构化、密集，对模型的空间定位能力要求越来越高。

本讲(Today)重点讲三类任务：**图像分割、目标检测、图像描述**。

任务概览 分类 分割 检测

2 语义分割：从朴素逐像素到滑窗的困境

语义分割的目标：**将图像切分为语义独立的若干区域，并给每个像素打类别标签**。输入是图片，输出是与输入同尺寸的像素类别图。

两种朴素思路及其缺陷：

1. **直接分类单个像素**：单个像素脱离上下文(context)几乎无法判断类别——一个绿色像素既可能是草也可能是衣服。**问题：缺乏上下文信息**。
2. **滑动窗口(Sliding Window)**：围绕每个像素抠出一个 patch，用 CNN 分类其中心像素。这引入了局部上下文，但**重叠 patch 之间的卷积特征被反复重算，极其低效，没有复用共享特征**。

> 这两个失败的尝试引出了关键改进方向：用一个网络一次性、共享地为所有像素做预测——即全卷积网络。

语义分割 滑动窗口 上下文

3 全卷积网络(FCN)与下采样-上采样结构

全卷积网络(Fully Convolutional Network, FCN) 的核心思想：网络只由卷积层构成(无全连接)，一次性为所有像素输出预测。

- 朴素全卷积(始终保持原分辨率)：输入 `3×H×W` → 多层卷积 → 得分图 `C×H×W` → 对通道取 `argmax` → 预测 `H×W`。**问题：在原始分辨率上做卷积计算/显存开销极大。**
- 实用做法：网络内部先**下采样(Downsampling)**(池化/带步长卷积)压缩分辨率、提取高层语义，再**上采样(Upsampling)** 恢复到原分辨率输出逐像素预测。

这样既能在低分辨率特征图上廉价地捕获大范围上下文，又能最终输出密集预测。下采样好理解(池化、strided conv)，难点在"如何在网络内上采样"。

代表工作：Long et al. (FCN, CVPR 2015)、Noh et al. (Deconvolution Network, ICCV 2015)。

FCN　全卷积　下采样　上采样

4 网络内上采样：Unpooling 与转置卷积

把低分辨率特征图放大到高分辨率，有几类方法：

1. 固定规则的 Unpooling(无参数)

- **Nearest Neighbor(最近邻)**：把每个值复制填满对应区域，如 `2→[[2,2],[2,2]]`。
- **Bed of Nails(钉床)**：只把值放到块的某个固定位置(如左上角)，其余补 0。

2. 转置卷积(Transposed Convolution / Deconvolution，可学习)

普通卷积可写成矩阵乘法：把核重排成卷积矩阵 C (如 4×16)，输入展平为 16×1 列向量，输出 $C \cdot x$ 得 4×1。

转置卷积则用**转置矩阵** $C^\top$ (16×4) 去乘一个 4×1 的输入，得到 16×1 的输出再 reshape。直觉上它"把 1 个输入值散布(connect)到输出中的多个值"，实现可学习的上采样。

> 注意：转置卷积常被叫"反卷积(deconvolution)"，但它并非数学上卷积的逆运算，只是把卷积的连接关系反向使用。它的参数是可学习的，比固定 unpooling 更灵活。

转置卷积　上采样　Unpooling　反卷积

5 编解码结构与 U-Net

Encoder-Decoder(编码器-解码器) 是分割网络的主流骨架：

- **Encoder(编码器)**：用卷积 + 下采样逐步压缩空间分辨率、增大感受野、提取高层语义。
- **Decoder(解码器)**：用反卷积/上采样逐步恢复分辨率，输出逐像素预测。

U-Net (Ronneberger et al.，最初用于生物医学图像分割)在此基础上加入了关键的 **跳跃连接(Skip Connections)**：将编码器对应层的高分辨率特征直接拼接到解码器同级，**把浅层精细的空间/边缘细节补回解码器**。

优点：

- 跳跃连接缓解了下采样导致的细节丢失，分割边界更精确；
- 在小数据集(如医学图像)上也能训练得很好。

> U-Net 形如字母 "U"：左半边收缩(下采样)、右半边扩张(上采样)，横向的跳跃连接把两侧对称层连起来。

U-Net 编解码 跳跃连接 语义分割

6 目标检测=分类+定位(单物体)

对**单个物体**而言，目标检测 = **分类(Classification)** + **定位(Localization)**。

一个 CNN 提取特征(如 4096 维向量)后接两个头(head)：

- **分类头**：FC `4096→1000`(类别数)，输出类别分数(Cat:0.9, Dog:0.05...)，用**交叉熵(Cross-Entropy)**损失。
- **定位头**：FC `4096→4`，回归边界框坐标 (x, y, w, h)，用 **L2 损失**对比预测框与真值框。

总损失是两者的加权和(多任务学习)：

$$L = L_{cls} + \lambda L_{loc}$$

> 这是检测的最简形式，但它假设图中**恰好一个**物体。一旦物体数量不定，固定数目的输出头就行不通了。

目标检测 分类 定位 多任务损失

7 多物体检测的难点与区域提议

多物体检测的核心困难：每张图的物体数量不同，CNN 需要的输出数目就不固定(2 个物体要 8 个坐标，3 个要 12 个……)，无法用固定输出网络直接解决。

朴素解法：枚举图像中各种 **裁剪框(crop)**，对每个框分类为某物体或背景。问题：需要在**海量位置、尺度、长宽比**上跑 CNN，计算量爆炸。

区域提议(Region Proposals) 缓解了这一点：

- 先用启发式算法找出图中"看起来像物体(blobby / objectness)"的少量候选区域，再只对这些区域分类。
- 代表算法 **Selective Search**：几秒内(CPU)给出约 **2000** 个区域提议，覆盖率高、相对快速。

> 思路转变：与其在所有可能位置盲目搜索，不如先"聪明地"提出少数可能含物体的候选框，把搜索空间从百万级降到千级。

多物体 区域提议 Selective Search objectness

8 R-CNN 与 Fast R-CNN

R-CNN (Girshick et al., CVPR 2014) 流程：

1. 用 Selective Search 提取约 2k 个 ROI(候选区域)；
2. 每个 ROI 缩放到 224×224，分别送入 ImageNet 预训练 CNN 提特征；
3. 用 **SVM** 对每个区域特征分类；
4. 用**边界框回归(Bbox regression)** 修正坐标 (dx, dy, dw, dh)。

> R-CNN 的致命问题：每张图要做约 **2000 次独立的 CNN 前向传播**，极慢(训练/推理都慢)。

Fast R-CNN 的关键改进：**先对整图跑一次 CNN，再在卷积特征图上裁剪 ROI**(ROI Pooling)，而非在原图上裁剪后各自跑 CNN。

- 所有 ROI **共享同一次主干(backbone)卷积计算**，大幅提速；
- 每个区域只需一个轻量的 Per-Region 网络做 `Linear+softmax` 分类和坐标回归；
- 端到端训练，用神经网络头替代了 SVM。

主干网络(Backbone)可用 VGG、ResNet 等。

R-CNN Fast R-CNN ROI Pooling 两阶段

9 Faster R-CNN 与区域提议网络(RPN)

Fast R-CNN 仍依赖 Selective Search 在 CPU 上生成提议，成为新瓶颈。**Faster R-CNN (Ren et al., NIPS 2015)** 的突破：**让 CNN 自己生成提议**——插入 **区域提议网络(Region Proposal Network, RPN)**。

RPN 工作方式：

- 在特征图的**每个点**放置固定大小的 **锚框(anchor box，锚盒)**；实际用 K 个不同尺寸/长宽比的锚框；
- 对每个锚框做二**分类**(是否含物体 / objectness)，输出 `K×H'×W'`；
- 对判定含物体的锚框**回归 4 个坐标**修正框位置，输出 `4K×H'×W'`；
- 按 objectness 分数排序所有 `K×H'×W'` 个框，取 **top ~300** 作为提议。

Faster R-CNN 联合训练 **4 个损失**：

1. RPN 物体/非物体分类；
2. RPN 框坐标回归；
3. 最终类别分类；
4. 最终框坐标回归。

这是经典的**两阶段(two-stage)检测器**：RPN 先提议，再对提议做精细分类与回归。整个流程几乎全在 GPU 上，速度比 Fast R-CNN 大幅提升。

Faster R-CNN　RPN　锚框　两阶段

10 锚框(Anchor Box)机制

锚框(Anchor Box) 是密集检测的关键设计：在特征图每个空间位置预设一组**参考框**，模型不直接预测绝对坐标，而是预测每个锚框"是否含物体"以及相对锚框的**偏移量(box corrections)**。

要点：

- 每个位置放 K 个不同**尺度(scale)与长宽比(aspect ratio)** 的锚框，以覆盖大小不一、形状各异的物体；
- 锚框使"不定数目物体检测"转化为"对固定数目锚框做固定输出的分类+回归"，解决了输出数目不定的难题。

> 直觉：与其让网络从零猜框，不如给它一堆"模板框"作为起点，只需学习微调即可。锚框是 Faster R-CNN 等检测器的基石。

锚框　anchor　尺度　长宽比

11 YOLO：单阶段实时检测

YOLO (You Only Look Once, Redmon et al., CVPR 2016) 把目标检测**直接当作一个回归问题**，一次前向就输出所有框，实现**实时**检测。

流程：

1. 将图片划分为 $S \times S$ 个网格(grid cell，原文 $S = 7$)；
2. **每个网格预测 B 个带置信度的边界框** $(x, y, h, w, \text{conf})$；
3. **每个网格同时预测 C 个类别(含背景)的概率分布**；
4. 输出张量尺寸为 $S \times S \times (5B + C)$；
5. 综合各框回归出最终预测框。

单阶段(one-stage) vs 两阶段(two-stage)对比：

- 单阶段(YOLO)：无独立提议步骤，分类与定位一气呵成，**速度快、可实时**；
- 两阶段(Faster R-CNN)：先提议再精修，**精度高但较慢**。

> 名字 "You Only Look Once" 点明其本质：整张图只看一次就完成检测，无需反复抠图分类。

YOLO　单阶段　实时检测　网格

12 图像描述：Encoder-Decoder 与注意力

图像描述(Image Captioning)：输入图像 I，输出自然语言序列 $y = y_1, y_2, \ldots, y_T$。

基础 Encoder-Decoder：

- Encoder：用预训练 CNN 提取空间特征 z，经 MLP 得上下文向量 $c = f_W(z)$；
- Decoder：RNN 逐词生成 $y_t = g_V(y_{t-1}, h_{t-1}, c)$，常令初始状态 $h_0 = c$。
- **问题**：整张图的信息被压缩(bottleneck)进单一向量 c，长描述时信息不足。

带注意力的 Encoder-Decoder (Show, Attend and Tell, ICML 2015)：

- 保留 CNN 的**空间特征网格** $z_{i,j}$；
- 解码每个词时计算一组**注意力权重** $a_{i,j}$，生成与该词相关的上下文向量 c_t，即 $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$；
- 这样生成"hat"时注意力聚焦帽子区域，生成"person"时聚焦人——**每个词关注图像不同区域**，缓解了瓶颈问题、可解释性更强。

> 进一步可对 CNN 特征施加**自注意力(self-attention)**，乃至完全用 Transformer 取代卷积。

图像描述　Encoder-Decoder　注意力　瓶颈

13 从注意力到纯 Transformer：自注意力与无卷积图像描述

在带注意力的 Encoder-Decoder 基础上，PPT 进一步推进了"用注意力取代卷积"的思路，作为通向 Vision Transformer 的过渡。

1. CNN 特征上的自注意力(Self-Attention)

注意力不仅能用在"解码词"对"图像区域"上，**也可以直接施加在 CNN 提取的特征上**：把特征图的各个空间位置当作一组元素，让它们彼此做自注意力，从而在特征内部建立任意两个位置之间的长程依赖关系，而不再局限于卷积核的局部感受野。

2. 完全用 Transformer 做图像描述(Image Captioning using ONLY Transformers)

既然自注意力如此有效，自然引出一个大胆的问题:

> "也许我们根本不需要卷积？(Perhaps we don't need convolutions at all?)"

于是图像描述可以做成**纯 Transformer**:不再用 CNN 提特征，而是把图像本身也表示成一串 token 送入 Transformer，与文本统一处理。这一步直接催生了下一节的 **Vision Transformer (ViT)**。

> 主线脉络:单一上下文向量(瓶颈)→ 解码时对图像区域做注意力 → 对 CNN 特征做自注意力 → 干脆抛弃卷积、全用 Transformer。

自注意力 图像描述 Transformer 过渡

14 视觉 Transformer(ViT)

ViT (Vision Transformer, Dosovitskiy et al., 2020)——"An Image is Worth 16×16 Words"——主张**"也许根本不需要卷积"**，直接用 Transformer 做图像识别。

核心流程:

1. 把图像切成固定大小的 **patch**(如 16×16)，每个 patch 展平并线性投影成一个 token("图像版的词");
2. 加上**位置编码(positional encoding)** 保留空间信息;
3. 通常前置一个可学习的 `[CLS]` token;
4. 送入标准 **Transformer Encoder**(多头自注意力 + MLP);
5. 用 `[CLS]` 的输出做分类。

ViT vs ResNet 的关键经验:

- ViT 缺少 CNN 的**归纳偏置(局部性、平移不变性)，在中小数据集上不如 CNN**;
- 但在**超大规模数据**预训练时，ViT 能超越 ResNet，且扩展性(scaling)更好。

> ViT 把"图像"重新表述为"patch 序列"，使图像与语言能共用同一套 Transformer 范式，为后续多模态大模型铺平道路。检测领域亦有纯 Transformer 方案 **DETR**(Carion et al., ECCV 2020)，端到端直接预测框集合。

ViT Transformer patch DETR

15 CLIP：对比式图文预训练

CLIP (Contrastive Language-Image Pre-training)——"Learning Transferable Visual Models From Natural Language Supervision"——用**自然语言监督**学习可迁移的视觉表示。

训练阶段(图文对比学习)：

- 一个 batch 的图片送入**图片编码器**、对应文本送入**文本编码器**；
- 计算两两编码向量的**余弦相似度**，构成相似度矩阵；
- 用**对比损失(Contrastive Loss)** 拉近**匹配对(similar pairs)**、推远**不匹配对(dissimilar pairs)**——即对角线上的真实图文对相似度最大化。

推理阶段(零样本图文检索/分类)：

- 根据类别标签构造文字描述(如 "a photo of a {label}")；
- 将图片与各候选文本分别编码，**取相似度最高者**作为预测，无需在目标数据上重新训练即可分类(zero-shot)。

> CLIP 的双编码器(dual-encoder)架构与海量网络图文对训练，使其学到对齐的图文语义空间，成为后续众多 VLM 的视觉编码器基础。

CLIP　对比学习　图文对齐　零样本

16 视觉大模型(VLM)与 LLaVA

视觉大模型(Vision-Language Models, VLM)：面向多模态(text-image)数据的大模型。常见架构：

- **双编码器(dual-encoder)**：图、文各一个编码器，对齐表示(如 CLIP)；
- **Encoder-Decoder / 融合(fusion)**：跨模态融合后解码；
- **适配 LLM(Adapted LLM)**：把视觉特征接入现成大语言模型。

VLM with Adapted LLM 的典型结构：图片经**图片编码器** → 经**多模态对齐**映射成与词向量同空间的视觉 token → 与输入文本(问题)的词向量**拼接** → 送入**文本解码器(LLM)** 生成回答。

LLaVA (Visual Instruction Tuning)：用视觉输入扩展 LLaMA 等大语言模型。

- 结构：**预训练 CLIP 视觉编码器** 提取 Z_V → 经**映射** W 对齐为 H_V → 与文本指令 H_q 一起送入语言模型 f_θ 生成应答；
- 数据构造：利用 **ChatGPT** 把图片注解转写成丰富的(指令-应答)对话数据，解决现有图文数据仅有简单注释、缺乏多样任务的问题；
- 训练分两步:先用图文数据微调映射 W 做对齐，再用生成的指令对话微调大模型(**指令微调**)。

更广义还可推广到更多模态，如 **PaLM-E**(结合 PaLM-540B 与 ViT-22B，把文本、图像、机器人状态编码进同一空间做下一词预测，用于具身智能)。

VLM　LLaVA　指令微调　多模态

17 📝 例题1：把卷积写成矩阵乘法，再理解转置卷积

PPT 用一个具体例子说明：**普通卷积可以重写成矩阵乘法**，而转置卷积就是把这个矩阵转置后使用。我们顺着走一遍。

第 1 步：普通卷积 = 卷积矩阵 × 展平输入

设输入是 4×4、卷积核是 3×3、输出是 2×2(无 padding、步长 1)。

- 把输入按行展平成一个 16×1 的列向量；
- 把卷积核"铺"到对应位置、其余补 0，重排成一个 **卷积矩阵** C，尺寸为 4×16(输出有 4 个元素，输入有 16 个元素)；
- 于是卷积结果 $=C\cdot x$，得到一个 4×1 向量，reshape 回 2×2 就是输出。

> 关键观察：每个输出元素只是输入中 9 个值的加权和，所以 C 的每一行最多有 9 个非零项，其余为 0。

第 2 步：转置卷积 = 用 $C^\top$ 反向相乘

上采样要做的是"小图变大图"：输入 2×2(4×1 向量)→ 输出 4×4(16×1 向量)。

- 把第 1 步的卷积矩阵 C(4×16)**转置**成 $C^\top$(16×4)；
- 用 $C^\top\cdot z$ 把 4×1 的输入映射成 16×1 的输出，再 reshape 成 4×4。

第 3 步：理解它的连接关系

- 普通卷积：把输入里的 9 个值"汇聚"成输出里的 1 个值(多对一)；
- 转置卷积：把输入里的 1 个值"散布"到输出里的多个值(一对多)。

结论：转置卷积(又叫"反卷积/deconvolution")并不是卷积的数学逆运算，它只是把卷积的连接关系反过来用，且 $C^\top$ 里的权重是**可学习**的——这正是网络内可学习上采样的来源。

例题

18 📝 例题2：算一算 RPN 与 YOLO 的输出张量尺寸

检测网络的输出张量尺寸是"按定义直接数出来"的，PPT 给了具体数字，我们核对一遍。

(一) Faster R-CNN 的 RPN 输出

设主干输出的特征图为 $512 \times 20 \times 15$(通道×高×宽)，每个空间位置放 K 个锚框。

- **锚框是否含物体(objectness 二分类)**：每个位置对 K 个锚框各出 1 个分数，输出尺寸 = $K \times 20 \times 15$。
- **框坐标修正(box corrections)**：每个锚框回归 4 个坐标数字，输出尺寸 = $4K \times 20 \times 15$。

> 若 $K = 1$(每点一个锚框)，两者分别是 $1 \times 20 \times 15$ 和 $4 \times 20 \times 15$，正好对应 PPT 的简化示意图。

最后把全部 $K \times 20 \times 15$ 个候选框按 objectness 分数排序，取 **top ~300** 作为提议。

(二) YOLO 的输出

YOLO 把图划分成 $S \times S$ 个网格(原文 $S = 7$)：

- **每个网格预测 B 个带置信度的边界框**，每个框有 $(x, y, h, w, \text{conf})$ 共 **5** 个数 $\rightarrow 5B$；
- **每个网格预测 C 个类别(含背景)的概率** $\rightarrow C$；
- 所以每个网格输出 $5B + C$ 个数，整张图的输出张量是 $S \times S \times (5B + C)$。

代入数字：取 $S = 7$、$B = 2$、$C = 20$，则每格 $5 \times 2 + 20 = 30$，输出张量 = $7 \times 7 \times 30$。

> 记忆要点：RPN 的两个分支是 "K 个分类 + $4K$ 个回归"；YOLO 一个网格是 "$5B$ 个框参数 + C 个类别概率"。把定义里的每一项乘起来即可，不用记结论。

例题

第 15 讲 K-means聚类

K-means Clustering

K-means是最经典的无监督聚类算法，通过最小化簇内平方和(SSE) $J = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - \mu_k\|^2$，交替执行"分配样本到最近质心"与"更新质心为簇均值"两步直至收敛。

1 无监督学习与聚类(Unsupervised Learning & Clustering)

前面几讲(回归、分类、生成)都依赖**标签**，属于**监督学习(Supervised Learning)**：目标是发现把数据属性与目标(类别)属性关联起来的模式。

当数据**没有标签**时，我们仍可从中学习结构，这就是**无监督学习(Unsupervised Learning)**：探索数据以发现某种内在结构(intrinsic structure)。常见任务：

- **聚类(Clustering)**：学习相似性，把样本分组，如 K-means、高斯混合模型(GMM)。
- **降维(Dimensional Reduction)**：数据压缩，如 PCA、自编码器(Auto-encoder)。
- **密度估计/生成(Density Estimation/Generative)**：如 GAN、VAE、Diffusion。

聚类(Clustering) 的定义：将无标签数据组织成若干**相似性组(簇, cluster)**。一个**簇(cluster)** 是一组彼此"相似(similar)"、且与其它簇中样本"不相似(dissimilar)"的数据项。

> 核心问题：如何刻画"相似"与"不相似"？——通常用**距离**(如欧氏距离)度量，距离越小越相似。

无监督学习 聚类 概念

2 如何表示簇：质心与隶属关系(Centroid & Membership)

假设我们已经(比如靠肉眼)分好了簇，如何**表示**它们?

- 为每个簇选一个**中心点(center)**，让它最好地代表整个簇；
- 为每个样本赋一个**标签(label)**，表示它属于哪个簇(membership/隶属关系)。

好的中心(质心)应满足：

- 质心(centroid)与本簇内样本的相似度，高于与其它样本的相似度；
- 簇内样本到**本簇质心**的距离，小于到**其它簇质心**的距离。

这就构成了"中心-隶属(center-membership)"框架，其中可被优化的有两件事：

1. **更新中心(center update)**：质心应由同簇样本的位置(均值)决定；
2. **更新隶属(label update)**：给定质心后，重新决定每个样本的归属。

这两步**交替反复**执行——这正是 K-means 的核心思想，本质上是一种 EM 式(期望-最大化)的交替优化。

质心 隶属关系 交替优化

3 K-means的问题形式化(Problem Formulation)

给定数据集 $D = \{x_1, \ldots, x_N\}$，其中每个 $x_i \in \mathbb{R}^d$ 是 d 维向量。K-means(MacQueen, 1967)要做的是：

- 把数据划分成 **K 个簇**，每个簇的中心定义为簇内所有样本的**均值(mean)**，称为**质心(centroid)**；
- 找到 K 个参考向量(质心) $\mu_1, \ldots, \mu_K$ 来最好地代表各簇；
- 给每个样本 x_i 赋一个变量 $m_i \in \{1, \ldots, K\}$，标记它的簇隶属。

注意 K 是**用户预先指定**的超参数。"reference vector(参考向量)"即质心，可看作该簇的原型(prototype)。

形式化 质心 K值

4 目标函数：簇内平方和SSE/WSS

K-means 优化的目标函数是**簇内平方和误差(Sum of Squared Error, SSE)**，又称**簇内平方和(Within-Sum-of-Squares, WSS)**：

$$J = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - \mu_k\|^2$$

其中 C_k 是第 k 个簇，μ_k 是其质心。J 衡量所有样本到各自质心的平方距离之和，越小说明簇越紧凑。

为什么质心是均值? 固定隶属关系，对单个簇求 $\sum_{x \in C_k} \|x - \mu_k\|^2$ 关于 μ_k 的最小值：令导数为零

$$\frac{\partial}{\partial \mu_k} \sum_{x \in C_k} \|x - \mu_k\|^2 = -2 \sum_{x \in C_k} (x - \mu_k) = 0 \Rightarrow \mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$$

即让平方误差最小的中心点恰好是簇内样本的**算术平均**。这从数学上解释了"更新质心=取均值"的合理性。

> 因为采用平方欧氏距离，K-means 隐含假设簇是**各向同性的球形**。

目标函数 SSE 推导

5 K-means算法流程(分配/更新两步)

标准 K-means 流程：

1. **初始化**：随机选 K 个数据点作为初始质心(seeds/种子)。
2. **分配步(Assignment)**：把每个样本分配到**最近的质心**所在的簇("赢者通吃 winner-takes-all")：$m_i = \arg\min_k \|x_i - \mu_k\|^2$。
3. **更新步(Update)**：用当前的簇隶属重新计算各簇质心(取均值)：$\mu_k = \frac{1}{|C_k|}\sum_{x \in C_k} x$。
4. **重复**第 2、3 步，直到满足收敛准则。

这两步交替进行，每次迭代都不会增大目标函数 J，因此算法单调下降。可以把分配步类比 EM 的 E 步(估计隐变量/归属)，更新步类比 M 步(优化参数/质心)。

```
repeat:
    # 分配步
    for each x_i:  m_i = argmin_k ||x_i - mu_k||^2
    # 更新步
    for each k:    mu_k = mean(x in cluster k)
until 收敛
```

算法流程 EM 迭代

6 收敛准则与时间复杂度(Convergence & Complexity)

收敛准则(Convergence Criterion)，满足任一即可停止：

- 没有(或极少)样本被重新分配到不同的簇；
- 质心没有(或极小)变化；
- SSE 的下降量低于某个阈值(minimum decrease in SSE)。

时间复杂度：设两个样本间算距离为 $O(d)$(d 为维度)。

- 分配步(重分配)：需要 $O(knd)$ 次距离计算(n 为样本数，k 为簇数)；
- 更新步(算质心)：每个样本加到某个质心一次，$O(nd)$；
- 设两步各执行 I 轮迭代，总复杂度 $O(Iknd)$。

因为对样本数 n 是线性的，K-means 被视为**线性算法**，这是它高效、流行的重要原因。

收敛 复杂度 线性

7 局部最优与对初始种子的敏感性(Local Optimum)

- 寻找 K-means 目标的**全局最优是 NP-hard** 的;
- K-means 算法只保证收敛到**局部最优(local optimum)**，不保证全局最优。

不确定性来自哪里? 主要来自**初始质心(种子)的随机选择**:

- 不同随机种子会导致不同的聚类结果;
- 某些种子会导致收敛很慢，或陷入次优(sub-optimal)的簇划分。

实践缓解办法:

- **多次重启(multiple restarts)**：用不同随机种子跑多次，取 SSE 最小的结果;
- 使用更聪明的初始化(如 K-means++);
- 用启发式或其它方法的结果挑选"好种子"。

局部最优 初始化敏感 NP-hard

8 如何选择K值：肘部法(Choosing K by Elbow Method)

K 是用户必须指定的超参数。PPT 指出**肘部法(Elbow Method)是寻找 K 的最佳方法**:

- 对每个 K，计算簇内平方和 **WSS(within-sum-of-squares)**——每个簇内成员到其质心的平方距离之和;
- 随 K 增大，WSS 单调下降(K=N 时降为 0)，所以不能简单取"WSS 最小的 K";
- 画出 WSS 随 K 变化的曲线，曲线出现明显"**拐点(肘部)**"——即再增大 K 带来的 WSS 下降明显放缓的位置——对应的 K 即为较优选择。

> 注意：PPT 中"K with the least WSS is taken as the optimum"的字面表述需谨慎理解。WSS 随 K 单调下降，真正要找的是**边际收益急剧减小的肘部**，而非绝对最小值。

K值选择 肘部法 WSS

9 K-means++初始化

K-means++ 是一种更好的种子选择方法，让初始质心彼此尽量分散，从而减少陷入次优的风险、加快收敛:

1. 从数据点中**均匀随机**选取第一个中心;
2. 对每个数据点 x，计算 $D(x)$ —— x 到**已选中心中最近者**的距离;
3. 以与 $D(x)^2$ **成正比**的加权概率，随机选取一个新数据点作为新中心;
4. 重复第 2、3 步，直到选出 K 个中心;
5. 之后照常运行标准 K-means。

直觉：离已有中心越远的点，被选为下一个中心的概率越大($\propto D(x)^2$)，使初始中心"撒得开"，覆盖不同簇区域，避免多个种子挤在一起。这通常显著改善最终聚类质量与收敛速度。

K-means++ 初始化 加权概率

10 应用与优缺点(Applications, Pros & Cons)

典型应用:

- **图像分割(Image Segmentation)**：按像素颜色聚类;
- **图像压缩(Image Compression)**：用每个像素邻近的质心颜色替代原值，减少颜色数。

优点(Advantages):

- **简单**：易于理解和实现;
- **高效**：对样本数线性($O(Iknd)$)，是最流行的聚类算法之一。

缺点(Disadvantages):

- 假设每个簇是**球形(sphere)**，对非球形的簇效果差;
- 只在**均值有定义**时适用(如纯类别型数据均值难定义);
- 需要用户**指定 K**;
- 对**异常点(outliers)** 敏感(均值易被极端值拉偏)。

> PPT 留下一个开放问题：如果我们**只知道样本间的两两距离**(没有坐标、均值难定义)，又该怎么办?

应用 优缺点

11 📝 例题1：一维6个点，K=2，手把手跑到收敛

题目：6 个一维数据点 {1, 2, 3, 8, 9, 10}，取 $K = 2$。初始质心选 $\mu_1 = 2$, $\mu_2 = 9$。一步步跑 K-means 直到收敛，并算最终 SSE。

> 一维下，点 x 到质心 μ 的（平方）距离就是 $(x-\mu)^2$；比大小时直接比 $|x-\mu|$ 即可。

第 1 轮 —— 分配步：把每个点分给更近的质心（$\mu_1 = 2, \mu_2 = 9$）。

点	到 $\mu_1 = 2$ 的距离	到 $\mu_2 = 9$ 的距离	归入
1	1	8	簇1
2	0	7	簇1
3	1	6	簇1
8	6	1	簇2
9	7	0	簇2
10	8	1	簇2

于是 $C_1 = \{1, 2, 3\}$，$C_2 = \{8, 9, 10\}$。

第 1 轮 —— 更新步：质心取簇内**均值**。

- $\mu_1 = \dfrac{1+2+3}{3} = 2$
- $\mu_2 = \dfrac{8+9+10}{3} = 9$

为什么取均值? 因为目标是最小化簇内平方和，固定簇成员后，使 $\sum(x-\mu)^2$ 最小的 μ 正是算术平均（对 μ 求导置零即得）。

第 2 轮 —— 分配步：新质心还是 2 和 9，所以每个点的归属**和上一轮完全一样**，没有任何点被重新分配。

> 收敛准则之一就是"没有样本被重新分配"或"质心不再变化"。这里两者都满足，算法**停止**。

最终结果：$C_1 = \{1, 2, 3\}$, $\mu_1 = 2$；$C_2 = \{8, 9, 10\}$, $\mu_2 = 9$。

算 SSE（簇内平方和）：

$$J = \underbrace{(1-2)^2 + (2-2)^2 + (3-2)^2}_{C_1} + \underbrace{(8-9)^2 + (9-9)^2 + (10-9)^2}_{C_2} = （1+0+1）+（1+0+1）= 4$$

所以收敛后 $J = 4$。这就是一次完整的 K-means：**分配 → 更新 → 再分配 → 发现不变 → 停**。

例题

12 📝 例题2：二维4个点，K=2，逐点算距离跑2轮

题目：平面上 4 个点 $A(1,1)$，$B(2,1)$，$C(4,3)$，$D(5,4)$，$K=2$。初始质心选两个数据点：$\mu_1=A=(1,1)$，$\mu_2=D=(5,4)$。用**平方欧氏距离** $d^2=(x_1-x_2)^2+(y_1-y_2)^2$ 跑 K-means。

第 1 轮 —— 分配步：算每个点到两个质心的平方距离。

点	到 $\mu_1(1,1)$	到 $\mu_2(5,4)$	归入
$A(1,1)$	$0+0=0$	$16+9=25$	簇1
$B(2,1)$	$1+0=1$	$9+9=18$	簇1
$C(4,3)$	$9+4=13$	$1+1=2$	簇2
$D(5,4)$	$16+9=25$	$0+0=0$	簇2

得 $C_1=\{A,B\}$，$C_2=\{C,D\}$。

第 1 轮 —— 更新步：分别对 x、y 坐标取均值。

- $\mu_1=\left(\frac{1+2}{2},\frac{1+1}{2}\right)=(1.5,\ 1)$
- $\mu_2=\left(\frac{4+5}{2},\frac{3+4}{2}\right)=(4.5,\ 3.5)$

第 2 轮 —— 分配步：用新质心 $\mu_1=(1.5,1),\mu_2=(4.5,3.5)$ 再算一遍。

点	到 $\mu_1(1.5,1)$	到 $\mu_2(4.5,3.5)$	归入
$A(1,1)$	0.25	18.5	簇1
$B(2,1)$	0.25	12.5	簇1
$C(4,3)$	10.25	0.5	簇2
$D(5,4)$	21.25	0.5	簇2

归属仍是 $C_1=\{A,B\}$，$C_2=\{C,D\}$，**没有变化 → 收敛，停止**。

最终 SSE（用收敛质心 $\mu_1=(1.5,1),\mu_2=(4.5,3.5)$）：

$$J=(0.25+0.25)_{C_1}+(0.5+0.5)_{C_2}=0.5+1.0=1.5$$

> 小技巧：比较"谁更近"时用平方距离即可，**不用开根号**——开根号不改变大小顺序，还省计算。

例题

13 📝 例题3：同样的点，换个起点跑——掉进更差的局部最优

题目：4 个点摆成一个矩形：$A(0,0)$，$B(0,4)$，$C(10,0)$，$D(10,4)$，$K=2$。这两簇"天生"应该按**左右**分（左列 A,B vs 右列 C,D，因为左右相距 10、上下只差 4）。我们用两种初始质心各跑一次，看结果差多少。

情形①（好的起点）：$\mu_1=(0,2)$（偏左），$\mu_2=(10,2)$（偏右）。

第 1 轮分配：

点	到 $\mu_1(0,2)$	到 $\mu_2(10,2)$	归入
$A(0,0)$	$0+4=4$	$100+4=104$	簇1
$B(0,4)$	$0+4=4$	$100+4=104$	簇1
$C(10,0)$	$100+4$	4	簇2
$D(10,4)$	$100+4$	4	簇2

得 $C_1=\{A,B\}, C_2=\{C,D\}$。更新质心：$\mu_1=(0,2), \mu_2=(10,2)$，没变 → **收敛**。

SSE：$(0+4)+(0+4)\,[C_1]+(0+4)+(0+4)\,[C_2]=\boxed{16}$。这是**全局最优**。

情形②（坏的起点）：两个种子选在**上下**——$\mu_1=A=(0,0)$，$\mu_2=B=(0,4)$。

第 1 轮分配：

点	到 $\mu_1(0,0)$	到 $\mu_2(0,4)$	归入
$A(0,0)$	0	16	簇1
$B(0,4)$	16	0	簇2
$C(10,0)$	100	116	簇1
$D(10,4)$	116	100	簇2

糟糕！按"上下"分了：$C_1=\{A,C\}$（下排），$C_2=\{B,D\}$（上排）。更新质心：$\mu_1=(5,0), \mu_2=(5,4)$。

第 2 轮：A、C 仍离 $(5,0)$ 近，B、D 仍离 $(5,4)$ 近 → 归属不变 → **收敛（但卡死了）**。

SSE：A,C 到 $(5,0)$：$25+25=50$；B,D 到 $(5,4)$：$25+25=50$；合计 $\boxed{100}$。

对比与结论：

起点	最终簇	SSE
好（左右）	$\{A,B\}$ / $\{C,D\}$	**16**
坏（上下）	$\{A,C\}$ / $\{B,D\}$	**100**

> 同样的数据，只因**初始质心不同**，K-means 收敛到了完全不同的结果，坏起点的 SSE 是好起点的 **6 倍多**！这就是"K-means 只保证**局部最优**、对初始种子敏感"的真实写照。

> **怎么办？** 多用几组随机种子各跑一次，**留 SSE 最小的那个**（multiple restarts）；或者用 **K-means++** 让初始质心尽量分散，避免两个种子挤在一起。

例题

14 📝 例题4：用肘部法挑 K——看 SSE 随 K 怎么降

题目：还是一维 6 个点 $\{1,2,3,8,9,10\}$。我们分别试 $K=1,2,3$，算出各自收敛后的 SSE（WSS），用**肘部法**判断该选哪个 K。

$K=1$：全部一个簇，质心 $\mu=\dfrac{1+2+3+8+9+10}{6}=5.5$。

$$\begin{aligned}\text{SSE}&=(1-5.5)^2+(2-5.5)^2+(3-5.5)^2+(8-5.5)^2+(9-5.5)^2+(10-5.5)^2\\&=20.25+12.25+6.25+6.25+12.25+20.25=77.5\end{aligned}$$

$K=2$：由例题1，分成 $\{1,2,3\}$ 和 $\{8,9,10\}$，$\text{SSE}=4$（一下子从 77.5 暴跌到 4）。

$K=3$：自然分成 $\{1,2,3\},\{8,9\},\{10\}$ 之类。取 $\{1,2,3\}(\mu=2),\{8,9\}(\mu=8.5),\{10\}(\mu=10)$：

$$\text{SSE}=(1+0+1)+\left((8-8.5)^2+(9-8.5)^2\right)+0=2+0.5+0=2.5$$

画成表看趋势：

K	SSE(WSS)	比上一档下降
1	77.5	—
2	4	**−73.5（巨大）**
3	2.5	−1.5（很小）

怎么读这张表?

1. SSE 随 K 增大**一定单调下降**（K 越多，每个点离质心越近；极端地 $K=N$ 时 SSE=0）。所以**不能简单取 SSE 最小的 K**——那永远会选最大的 K。
2. 找**"肘部"**：从 $K=1\to2$ 下降 73.5 非常剧烈，$K=2\to3$ 只降 1.5。曲线在 $K=2$ **处明显拐弯变平**。
3. 结论：**选 $K=2$**——再加簇带来的收益已经很小了。

> 肘部法的精髓不是"SSE 最小"，而是"再增大 K，SSE 下降明显放缓"的那个**拐点**。这和数据本身有两个清晰的簇（左 3 个、右 3 个）正好吻合。

例题

第 16 讲 降维

Dimensionality Reduction

降维(Dimensionality Reduction)是在尽量保留数据内在结构的前提下，把高维数据映射到低维空间的无监督技术，用于对抗维度灾难、可视化、压缩与去噪。核心方法是主成分分析(PCA)：通过对协方差矩阵 $S = \frac{1}{N} X X^\top$ 做特征值分解，选取方差最大的前 k 个正交方向，即求解 $\max_u u^\top S u$ s.t. $u^\top u = 1$。本讲还从最小重构误差视角理解 PCA，并将其推广为单隐层线性自编码器，进而引出(深度/去噪)自编码器。

1 维度灾难 (The Curse of Dimensionality)

核心论断：随着维度增长，为了准确泛化所需要的数据量呈**指数级**增长 (Charles Isbell)。

直觉举例(课件)：给定带二值属性的样本 $(x_1, \ldots, x_N)$，要估计联合分布 $p(x_1, \ldots, x_N)$。如属性为 性别、专业、年龄…，列联表需要 2^d 行；只要 d 稍大，表格规模就远超我们能采集的样本数，导致绝大多数格子都没有样本支撑。

后果：

- 需要的样本量随维度爆炸式增长，“特征多、样本少”时模型极易过拟合。
- 计算与存储开销大。

> 降维正是缓解维度灾难的手段之一。

维度灾难 动机

2 为什么要降维：动机与目标

降维定义：减少数据的维度(属性)数量，同时**保留数据的内在特性(intrinsic characteristics)**，本质是寻找控制数据的若干“隐藏因素”。

主要动机：

1. **数据本质上活在低维空间**：表面上是高维(如 3D)，真实自由度可能只有 2D（数据位于一个低维流形/子空间上）。
2. **特征太多、样本太少**：降维可降低过拟合风险。
3. **降低计算开销**：减少内存与时间消耗。
4. **可视化**：把数据投到 2D/3D 便于观察。
5. **统一维度**：让不同维度的数据可以喂给同一模型。

典型应用：数据可视化、预处理、压缩、为新数据建模(先验)、图像检索、人脸识别(Eigenfaces)等。

降维动机 应用

3 降维方法概览：线性 vs 非线性

线性方法(用一个线性投影把 x 映到低维)：

- **PCA**(主成分分析)：无监督，最大化投影方差。
- **LDA**(线性判别分析)：有监督，最大化类间/类内方差比。
- **因子分析(Factor Analysis)**等。

非线性方法(可刻画弯曲的流形结构)：

- **KPCA / KLDA**：用核技巧把 PCA/LDA 推广到非线性。
- **MDS**(多维标度法)：保持样本间距离。
- **t-SNE**：保持局部邻域结构，擅长可视化。
- **Auto-encoder**(自编码器)：用神经网络学习非线性压缩表示。

> 一句话区分：线性方法假设数据落在一个低维**子空间**(平面)上；非线性/流形方法允许数据落在一个弯曲的**流形**上。

方法分类 线性 非线性

4 本讲定位：无监督学习的三大任务

继聚类之后，本讲继续在**无监督学习**框架下讨论数据中“聚类之外”的模式。课件把无监督学习分为三大类任务：

1. **聚类(Clustering)**——学习数据间的相似性。代表方法：K-means、高斯混合模型(GMM)。
2. **生成(Generative / 密度估计)**——学习数据分布以生成新样本。代表方法：GAN、VAE、Diffusion。
3. **降维(Dimensional Reduction / 数据压缩)**——本讲主题。代表方法：**主成分分析(PCA)** 与 **自编码器(Auto-encoder)**。

问题引子：大数据时代特征非常多，但它们**真的都有用吗**？很多维度其实是冗余或相互关联的，数据的“有效自由度”往往远低于名义维度——这正是降维的出发点。

> 本讲聚焦降维的两条主线：线性的 **PCA** 和它向非线性的推广 **自编码器**；课件结尾预告下一讲进入**深度生成模型**(GAN / VAE / Diffusion)。

无监督学习 本讲概览

5 PCA 的核心思想与问题设定

主成分(Principal Component) = 数据散布趋势最大的方向 = **方差最大的维度**。PCA 寻找一组**正交**方向，按所解释的方差从大到小排列。

问题设定(先中心化，使 $\sum_{i=1}^{N} x_i = 0$，即减去样本均值)：

- 数据矩阵 $X = [x_1, \dots, x_N]$，$x_i \in \mathbb{R}^d$。
- 投影到方向 u：$z_j = u_j^\top x$，整体 $Z = U^\top X$。
- 投影后方差：

$$\text{var}(u^\top X) = \tfrac{1}{N}(u^\top X)(u^\top X)^\top = \tfrac{1}{N} u^\top X X^\top u = u^\top S u$$

其中协方差矩阵 $S = \text{cov}(X) = \frac{1}{N} X X^\top$。

> 因为已中心化，$u^\top S u$ 就是投影坐标的方差。PCA 要找让这个方差最大的单位方向 u。

PCA　协方差矩阵　最大方差

6 PCA 作为约束优化问题及其求解

优化目标(最大方差视角)：

$$\max_{u} \; u^\top S u \quad \text{s.t.} \quad u^\top u = 1$$

约束 $u^\top u = 1$ 防止简单放大 u 的模长把目标做大。这是一个**二次规划/凸优化**问题；S 是**半正定**的，故 $u^\top S u \geq 0$。

用拉格朗日乘子求解：

$$L = u^\top S u - \lambda(u^\top u - 1), \qquad \nabla_u L = 0 \;\Rightarrow\; S u = \lambda u$$

这正是**特征值问题**：u 是 S 的特征向量，λ 是对应特征值。

关键结论：把 $S u = \lambda u$ 代回目标得 $u^\top S u = \lambda u^\top u = \lambda$。所以

- **特征值 λ = 数据投影到对应特征向量方向上的方差**；
- 要让方差最大，就取**最大特征值**对应的特征向量。

由于 S 半正定且对称，可正交对角化：

$$S = U \Sigma U^\top = [u_1, \dots, u_d] \, \text{diag}(\lambda_1, \dots, \lambda_d) \, [u_1, \dots, u_d]^\top$$

PCA　特征值分解　拉格朗日

7 选取前 k 个主成分与确定 k

做法：找 k 个正交主成分，对应 S 的 k 个**最大特征值**的特征向量，丢弃小特征值方向：

$$S \approx U_{1:k}\Sigma_{1:k}U_{1:k}^{\top}, \qquad z = U_{1:k}^{\top}x$$

每个输入被投到这个 k 维子空间，得到压缩表示 $z \in \mathbb{R}^k$。

直觉：特征值是该方向上的方差；特征向量彼此正交，互不相关。

如何确定 k——控制信息损失在可接受范围(通常 5%)：

$$\text{info loss} = \frac{\lambda_{k+1} + \cdots + \lambda_d}{\lambda_1 + \lambda_2 + \cdots + \lambda_d}$$

等价地，**方差解释比(累计)** $= \dfrac{\lambda_1 + \cdots + \lambda_k}{\lambda_1 + \cdots + \lambda_d} \geq 95\%$。

> 即：把要丢弃的小特征值之和占总特征值之和的比例控制在 5% 以内。

主成分选择　方差解释比　信息损失

8 PCA 算法流程

算法五步：

1. **中心化**：数据集减去均值，使其零均值。
2. **计算协方差矩阵** $S = \frac{1}{N}XX^{\top}$。
3. 对 S 做**特征值分解**，并按特征值大小对特征向量排序。
4. 根据可接受的信息损失**确定** k。
5. 取前 k 个最大特征值对应的特征向量作为新空间的基，投影得到 z。

关于中心化：中心化是 PCA 的**必要**步骤——只有零均值时 $u^{\top}Su$ 才正好等于投影坐标的方差，否则第一主成分会偏向均值方向。课件给出的代码也是先“shifting data set to have zero mean”，再算协方差、做特征分解。

PCA算法　中心化

9 PCA 的应用：人脸识别与“特征脸 (Eigenfaces)”

PCA 的典型应用：数据可视化、预处理、建模(为新数据提供先验)、压缩。课件以**人脸识别**作为重点示例。

问题背景：人脸图像所在的空间维度极高。例如 24×24 的灰度图就有 $24 \times 24 = 576$ 维，直接处理既慢又费存储。但关键观察是：**在全部 576 维向量中，真正构成“有效人脸”的只占极小一部分**——合法人脸只分布在一个低维子空间里。

Eigenfaces 做法：对一批人脸图像(如 Viola-Jones 数据集的 24×24 人脸)做 PCA，取前 K 个主成分。

- 这些主成分本身可视化出来就像一张张“模糊的脸”，称为**特征脸(Eigenfaces)**，它们是张成人脸子空间的正交基。
- 任意一张人脸都可近似表示为样本均值脸加上若干特征脸的加权组合，只需保存这 K 个权重 $z = (z_1, \ldots, z_K)$ 即可，实现压缩与高效识别。

> 核心思想：用 PCA 找到人脸图像所在的低维子空间，把高维像素表示替换为紧凑的特征脸系数。

PCA应用 人脸识别 特征脸

10 PCA 的另一视角：最小重构误差

PCA 有两个等价视角：**最大化投影方差** 与 **最小化重构误差**。

重构视角：用前 k 个主成分把 x 表示为基的线性组合(再加上样本均值)：

$$x \approx \bar{x} + z_1 u_1 + z_2 u_2 + \cdots + z_k u_k, \qquad z_j = u_j^\top (x - \bar{x})$$

手写数字 x 可近似为若干“基础笔画分量” $u_1, u_2, \ldots$ 的加权叠加，权重 $z = (z_1, \ldots, z_k)^\top$ 就是压缩表示。

等价性：在所有 k 维正交投影中，PCA 选的子空间使**平均重构误差** $\frac{1}{N}\sum_i \|x_i - \hat{x}_i\|^2$ **最小**。直觉上，保留方差最大的方向 = 丢弃信息最少的方向 = 重构误差最小。两个视角通过“总方差 = 保留方差 + 丢失方差(=重构误差)”联系起来。

最小重构误差 PCA双视角

11 PCA = 单隐层线性自编码器 → 自编码器

PCA 可看作一个隐层、线性激活的神经网络：

- **编码(投影)**：$z = U^\top x$（朝主成分方向投影）。
- **解码(重构)**：$\hat{x} = \sum_j z_j u_j = Uz$（用 U 重构 x）。
- **目标**：最小化重构误差 $\|x - \hat{x}\|^2$。当激活为线性、权重共享时，最优解张成的子空间与 PCA 相同。

自编码器(Auto-encoder)：编码器-解码器结构的神经网络，输出尽量重构输入，从**无标签**数据中学习低维特征 z。

- **瓶颈隐层**：维度小于输入，迫使网络学习压缩编码；重构损失迫使编码尽量保留原始信息。
- **深度自编码器**：堆叠多层、用非线性激活，能学到比 PCA 更强的**非线性**压缩 (Hinton & Salakhutdinov, 2006)。

变体：

- **去噪自编码器(Denoising AE)**：输入加噪 x'，训练目标是预测原始 x，用于去噪并学到鲁棒特征。
- **卷积自编码器 / U-Net**：用于图像压缩。

> PCA 是线性自编码器的特例；自编码器是 PCA 向非线性的推广。

自编码器　PCA神经网络视角　去噪

12 📝 例题1：手把手做一遍 PCA（5 个二维点）

我们有 5 个二维样本（比如 5 个人的「身高指标 x」和「体重指标 y」，已无量纲化）：

样本	x	y
1	1	3
2	3	1
3	5	5
4	7	9
5	9	7

目标：把这 2 维数据压成 **1 维**。跟着下面五步走。

第 1 步：中心化（减去均值）

> 为什么？只有零均值时，$u^\top S u$ 才正好等于「投影后的方差」，否则第一主成分会偏向均值方向。

均值 $\bar{x} = \frac{1+3+5+7+9}{5} = 5$，$\bar{y} = \frac{3+1+5+9+7}{5} = 5$，即 $\bar{\mathbf{x}} = (5, 5)$。

每个样本减去 $(5, 5)$，得到中心化后的数据：

$$(-4, -2),\ (-2, -4),\ (0, 0),\ (2, 4),\ (4, 2)$$

第 2 步：计算协方差矩阵 $S = \frac{1}{N} X_c X_c^\top$（$N = 5$）

> 为什么？协方差矩阵刻画了「各方向上数据有多分散、两维之间有多相关」。

逐项累加（用中心化后的坐标 (x', y')）：

- $\sum x'^2 = (-4)^2 + (-2)^2 + 0^2 + 2^2 + 4^2 = 16 + 4 + 0 + 4 + 16 = 40$
- $\sum y'^2 = (-2)^2 + (-4)^2 + 0^2 + 4^2 + 2^2 = 4 + 16 + 0 + 16 + 4 = 40$
- $\sum x'y' = (-4)(-2) + (-2)(-4) + 0 + 2 \cdot 4 + 4 \cdot 2 = 8 + 8 + 0 + 8 + 8 = 32$

除以 $N = 5$：

$$S = \begin{bmatrix} 40/5 & 32/5 \\ 32/5 & 40/5 \end{bmatrix} = \begin{bmatrix} 8 & 6.4 \\ 6.4 & 8 \end{bmatrix}$$

对角线 $8, 8$ 是 x, y 各自的方差；非对角 6.4 是协方差（正值说明 x 大时 y 也偏大）。

第 3 步：求特征值与特征向量（这里直接给出结果）

> 为什么求这个？解 $Su = \lambda u$ 就是 PCA 的核心：特征向量是主成分方向，特征值是该方向上的方差。

对 $S = \begin{bmatrix} 8 & 6.4 \\ 6.4 & 8 \end{bmatrix}$ 解特征方程 $\det(S - \lambda I) = 0$：

$$(8 - \lambda)^2 - 6.4^2 = 0 \ \Rightarrow\ 8 - \lambda = \pm 6.4$$

得到两个特征值：

$$\lambda_1 = 8 + 6.4 = 14.4, \qquad \lambda_2 = 8 - 6.4 = 1.6$$

对应的单位特征向量（这种「对角相等」的矩阵，方向必沿 $\pm 45°$）：

$$u_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}, \qquad u_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \approx \begin{bmatrix} 0.707 \\ -0.707 \end{bmatrix}$$

第 4 步：选最大特征值对应的主成分

$\lambda_1 = 14.4 > \lambda_2 = 1.6$，所以第一主成分就是 $u_1 = \frac{1}{\sqrt{2}}(1,1)$。它正好指向数据散得最开的「左下—右上」对角线方向，符合直觉。

第 5 步：投影，得到一维坐标 $z = u_1^\top x'$

用公式 $z = \frac{x'+y'}{\sqrt{2}}$（因为 $u_1 = \frac{1}{\sqrt{2}}(1,1)$）：

- $(-4,-2) \to \frac{-6}{\sqrt{2}} \approx -4.24$
- $(-2,-4) \to \frac{-6}{\sqrt{2}} \approx -4.24$
- $(0,0) \to 0$
- $(2,4) \to \frac{6}{\sqrt{2}} \approx 4.24$
- $(4,2) \to \frac{6}{\sqrt{2}} \approx 4.24$

于是 5 个二维点被压成了 5 个一维数 $[-4.24, -4.24, 0, 4.24, 4.24]$。

> 验算：这串 z 的方差恰好是 $14.4 = \lambda_1$ —— 特征值就是投影坐标的方差，完美对上！

例题

13 📝 例题2：方差解释比——这一维到底保住了多少信息？

接着例题1 的结果：两个特征值是 $\lambda_1 = 14.4$、$\lambda_2 = 1.6$。

第 1 步：算总方差

> 所有特征值之和 = 数据的总方差（也等于协方差矩阵对角线之和 $8 + 8 = 16$）。

$$\lambda_1 + \lambda_2 = 14.4 + 1.6 = 16$$

第 2 步：算每个主成分的方差解释比

$$\text{PC1 占比} = \frac{\lambda_1}{\lambda_1 + \lambda_2} = \frac{14.4}{16} = 0.9 = 90\%$$

$$\text{PC2 占比} = \frac{\lambda_2}{\lambda_1 + \lambda_2} = \frac{1.6}{16} = 0.1 = 10\%$$

第 3 步：决定保留几维

> 准则：累计方差解释比 $\geq 95\%$（或信息损失 $\leq 5\%$）。

- 只保留 PC1（降到 1 维）：保留 90% 的信息，**信息损失** $= \frac{\lambda_2}{\lambda_1+\lambda_2} = 10\%$。
- 这 10% 超过了 5% 的阈值，说明丢掉第二维会损失稍多。但若我们能接受 10% 的损失（换来维度减半），降到 1 维就足够。

结论：是否降到 1 维，取决于你能容忍多少损失。本例 PC1 已占 90%，用 1 维表示通常很划算；若要求很严格（损失 $\leq 5\%$），就得保留两维（不降维）。

> 一句话记忆：**特征值大小 = 信息多少**；把特征值从大到小累加，加到「够 95%」就停，前面用到几个特征向量，就保留几维。

例题

14 📝 例题3：直观感受「维度灾难」——联合分布表格爆炸

维度一高，估计联合分布所需的样本量会爆炸式增长。我们用具体数字感受一下（即课件给出的例子）。

假设每个特征是二值的（如 性别∈{男,女}、是否CS专业∈{是,否}、年龄∈{<25,≥25}⋯）。要估计 d 个二值特征的联合分布 $p(x_1,\dots,x_d)$，列联表需要 2^d 行：

- $d=3$：$2^3=8$ 行，几十个样本就能填满。
- $d=10$：$2^{10}=1024$ 行。
- $d=20$：$2^{20}\approx 10^6$ 行。
- $d=50$：$2^{50}\approx 1.13\times 10^{15}$ 行 —— 比地球上所有人还多！

> 结论：维度每 $+1$，需要的样本量大致 **翻倍**（指数增长）。特征一多，绝大多数格子根本没有样本，模型无从学起。

这跟 PCA 有什么关系？ 真实数据往往「名义维度很高，但本质维度很低」（例题1 里数据其实就排在一条对角线上）。PCA 就是去找到那几个真正有方差的方向，把数据从高维拉回紧凑的低维，从而缓解维度灾难。

例题

第 17 讲 生成模型

Generative Models

本讲介绍深度生成模型：目标是学习一个 $p_{model}(x)$ 去逼近真实数据分布 $p_{data}(x)$，从而采样生成新样本。重点覆盖判别式与生成式之分、自编码器(AE)、变分自编码器(VAE，重建+KL 正则、重参数化)、生成对抗网络(GAN，$\min_G \max_D V(D,G)$)及其变体(DCGAN/CGAN/CycleGAN)、扩散模型(前向加噪/反向去噪)。

1 判别式模型 vs 生成式模型

机器学习模型按建模目标可分两大类：

- **判别式模型 (Discriminative)**：学习条件分布 $p(y \mid x)$ 或直接学习决策边界，回答“这张图是男是女”，关注**如何把输入分到正确类别**。例如逻辑回归、SVM、普通分类 CNN。
- **生成式模型 (Generative)**：学习数据本身的分布 $p(x)$（或联合分布 $p(x,y)$），关注**数据是怎么产生的**，因此可以**采样合成新样本**。例如 GAN、VAE、Diffusion。

> 类比语言模型：$p(x) = p(w_1, w_2, \ldots, w_N)$ 给一句话打概率分数。“他明天去上课”比“依明朝得返课啦”在普通话中概率更高。生成 = 学习训练数据的概率分布。

直觉：判别式只需在类别间画线；生成式要刻画整个数据空间的“地形高低”——真实样本处概率密度高，奇怪样本处几乎为 0。

生成式vs判别式 概念

2 图像的概率分布与生成目标

把一张 $64 \times 64 \times 3$ 的图像看作高维向量 x，则存在一个真实数据分布 $p_{data}(x)$：

- 像“戴眼镜的男人”“黑发女人”这类真实人脸，$p_{data}(x)$ **密度很高**；
- 随机噪声、扭曲怪图，$p_{data}(x) \approx 0$。

生成模型的目标：找到一个可参数化的 $p_{model}(x)$ 使其尽量逼近 $p_{data}(x)$。这引出两个核心问题：

1. **如何表示并采样 $p_{model}(x)$?** —— 用一个生成器神经网络 G，把简单分布（如高斯噪声 $z \sim N(0, I)$）映射成复杂的图像分布 P_G。
2. **如何让 $p_{model}(x) \approx p_{data}(x)$?** —— 不同模型给出不同答案：GAN 用判别器对抗，VAE 用重建+KL 正则，Diffusion 用去噪。

> 关键直觉：直接写出高维图像的显式概率密度极难，所以现代生成模型大多采用“**隐式**采样器 + 某种分布匹配准则”的思路。

数据分布 生成目标

3 自编码器 (Autoencoder, AE)

自编码器是一种无监督的**降维/压缩**模型，由编码器和解码器组成：

$$x \xrightarrow{\text{Encoder}} z \xrightarrow{\text{Decoder}} \hat{x}$$

- **编码器** 把输入压成低维隐编码 z（瓶颈，bottleneck）；
- **解码器** 从 z 重建 $\hat{x}$；
- 训练目标是**最小化重建误差**，如 $\|x - \hat{x}\|^2$。

它本质上是 PCA 的非线性推广，能学到数据的压缩表示。

为什么 AE 不能直接做生成？ 训练后 z 的分布是“数据决定的”、不可知（intractable）的：随便采一个 z 喂给解码器，往往落在训练时从未出现过的区域，解码出无意义内容。也就是说隐空间**不连续、不完整**。这正是 VAE 要解决的问题。

自编码器 降维

4 变分自编码器 (VAE) — 核心思想

VAE 是自编码器的**概率化改造**：让隐编码 z 既是某张真实图像的压缩码，又**被强制服从一个先验分布**（通常是单位高斯 $N(0, I)$）。

实现方式：编码器不再输出一个确定的 z，而是输出一个高斯分布的**均值向量** μ 和**标准差向量** σ，再从中采样：

$$z \sim N(\mu, \sigma^2)$$

- **编码器** 学习近似后验 $q_\phi(z \mid x)$；
- **解码器** 学习似然 $p_\theta(x \mid z)$；
- 因为 z 被约束接近标准高斯，**生成时只需从 $N(0, I)$ 采样 z，再解码即可得到新图像**，解决了 AE 隐空间不可用的问题。

> 直觉：VAE 在隐空间“填洞铺路”，让相邻的 z 解码出相似内容、任意采的 z 都解码出有意义内容。

VAE 概率生成

5 VAE 的损失函数：重建 + KL 正则

VAE 的训练目标是**最大化对数似然**。等价地，最小化损失：

$$L(\theta, \phi \mid x) = -\mathbb{E}_{z \sim q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] + \mathrm{KL}\big(q_\phi(z \mid x) \,\|\, p(z)\big)$$

其中编码器计算近似后验 $q_\phi(z \mid x)$，解码器计算似然 $p_\theta(x \mid z)$。两项含义：

1. **重建损失 (Reconstruction Loss)**：$-\mathbb{E}_{q_\phi}[\log p_\theta(x \mid z)]$，要求由 z 解码出的 $\hat{x}$ 像原图 x，保证编码不丢信息。
2. **正则项 (Regularization / KL 项)**：$\mathrm{KL}(q_\phi(z \mid x) \,\|\, p(z))$，把近似后验拉向先验 $p(z) = N(0, I)$（常用单位高斯），让隐空间规整。

> KL 散度衡量两个分布的差异，恒 ≥ 0，当两分布相同时为 0。

VAE 损失函数 KL散度

6 重参数化技巧 (Reparameterization Trick)

> 讲义标注此为**拓展内容、不作要求**，了解思路即可。

问题：损失里要对 $z \sim N(\mu, \sigma^2)$ 的采样求期望，但**采样操作不可微**，梯度无法从解码器回传到编码器的 μ, σ。

解决：把随机性“移出”计算路径——

$$z = \mu + \sigma \odot \varepsilon, \qquad \varepsilon \sim N(0, I)$$

其中 $\odot$ 为逐元素乘。这样：

- 随机性集中在与参数无关的噪声 ε 上；
- z 对 μ, σ 是**确定的可微函数**，梯度可正常反向传播；
- 采样得到的 z 仍服从 $N(\mu, \sigma^2)$，数学等价。

> 这是 VAE 能端到端用梯度下降训练的关键工程技巧。

VAE 重参数化

7 VAE 正则化的作用：连续性与完整性

KL 正则项让隐空间具备两个期望性质，使其真正可用于生成：

1. **连续性 (Continuity)**：隐空间中**相近的两点解码后内容也相似**，不会一点点移动 z 就跳变出完全不同的图。
2. **完整性 (Completeness)**：从隐空间**任意采样都能解码出“有意义”的内容**，没有空洞/无效区域。

对比：

- **无正则化 (普通 AE)**：相近的点可能解码差异巨大；某些点解码出无意义内容。
- **有正则化 (VAE)**：相近点相似解码，且都有意义。

隐空间扰动 (Latent Perturbation)：缓慢增减 z 的**单个维度**，输出会沿某个可解释属性平滑变化，例如“微笑程度”“头部姿态”。说明不同维度编码了不同的可解释潜在特征——这是 VAE 表示学习的价值。

VAE 正则化 隐空间

8 GAN：生成器与判别器的对抗框架

生成对抗网络 (Generative Adversarial Network) 让两个神经网络互相博弈：

- **生成器 G**：输入隐编码/噪声 $z \sim N(0, I)$，输出假图 $G(z)$，目标是把噪声变成以假乱真的图像来**骗过判别器**。
- **判别器 D**：是个二分类器，输入一张图输出它是真(p_{data})还是假(p_G)的概率，目标是**分辨真假**。

训练像“造假者 vs 鉴定师”不断对抗升级。**直觉结论**：当判别器再也无法区分真假时（$D \approx 0.5$），说明 $p_{model}(x) \approx p_{data}(x)$，生成器已复现真实数据分布。

- D 希望对真图输出 $D(x) \to 1$，对假图输出 $D(G(z)) \to 0$；
- G 希望让 $D(G(z)) \to 1$，与 D 目标完全相反（adversarial）。

GAN 对抗训练

9 GAN 的目标函数：minimax 博弈

基于二元交叉熵 $\ell(h(x), y) = -y \log h(x) - (1-y)\log(1-h(x))$，GAN 的价值函数为：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

分解理解：

- **判别器目标（最大化 V）**：$D^\star = \arg\max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_G}[\log(1-D(x))]$，即让真图得分高、假图得分低。
- **生成器目标（最小化 V）**：G 只与第二项有关，$G^\star = \arg\min_G \mathbb{E}_{z\sim p_z}[\log(1-D(G(z)))]$，即让 $D(G(z))$ 变大。

这是一个 **minimax 极小极大博弈**。理论上**全局最优解**是 $p_G = p_{data}$，即生成器复现真实数据分布，此时 $D \equiv 1/2$。

GAN　minimax　目标函数

10 GAN 的训练算法

交替优化（外层迭代 N 次）：

1. **更新判别器（重复 K 次）**：从噪声先验 $p_z(z)$ 采 m 个 z、从数据分布 $p_{data}(x)$ 采 m 个 x，沿**随机梯度上升**方向最大化

$$\frac{1}{m}\sum_{i=1}^{m}\left[\log D(x_i) + \log(1-D(G(z_i)))\right]$$

1. **更新生成器**：再采 m 个 z，沿**随机梯度下降**方向最小化 $\frac{1}{m}\sum_{i=1}^{m}\log(1-D(G(z_i)))$（更新 G 时冻结 D）。

如此判别器与生成器交替更新、轮流变强，最终收敛到 G 复现真实数据分布。

GAN　训练算法

11 GAN 变体：DCGAN / 条件 GAN / CycleGAN

- **DCGAN (Deep Convolutional GAN)**：生成器用**反卷积/转置卷积 (deconvolution)** 上采样，判别器用**卷积**下采样，是把 GAN 做稳的经典工程实践。还能做**隐向量算术 (latent vector arithmetic)**，如“戴眼镜男 − 男 + 女 ≈ 戴眼镜女”，说明隐空间语义线性可组合。
- **条件 GAN (Conditional GAN, CGAN)**：给 G 和 D 都额外输入一个**条件**（如类别标签）来**控制输出内容**；D 判断的是“图像与条件是否为真实配对”。可用于图像到图像翻译，但需要**成对 (parallel) 监督数据**。
- **CycleGAN**：在**无配对样本**下做域转换（如斑马↔马、夏↔冬）。用两个生成器 G_{AB}, G_{BA} 与**循环一致性损失 (cycle-consistency loss)**：$x \to G_{AB}(x) \to G_{BA}(G_{AB}(x))$ 应重建回 x（L2/L1 损失），从而在没有配对标签时保持图像结构内容不变。

GAN　DCGAN　CycleGAN

12 VAE 与 GAN 的局限：一次性生成

VAE 和 GAN 虽然都能生成图像，但讲义指出它们有两点共同**局限**：

1. **隐编码 z 的尺寸远小于原图**：z 是一个压缩得很小的向量（如 100 维），要一步映射出 $64 \times 64 \times 3$ 的完整图像，信息跨度很大。
2. **内容一次性 (all at once) 生成**：无论 VAE 的解码器还是 GAN 的生成器，都是把噪声/隐码**一口气**变成整张图：

$$z \xrightarrow{G} G(z)$$

这带来一个自然的问题：**能否“从粗到细 (coarse-to-fine)”、逐步地编码与生成图像，而不是一步到位?**

> 这正是扩散模型的出发点——把“一步生成”这个困难任务，拆成许多步“每次只改一点点”的简单任务，从而更稳定、质量更高。

VAE　GAN　局限　Diffusion动机

13 扩散模型 (Diffusion) — 前向加噪与反向去噪

去噪扩散模型 (Denoising Diffusion Model) 的核心思想：**通过去噪来学会生成**。它包含两个相反的过程：

1. **前向扩散过程 (Forward Diffusion)，固定不可训练**：分 T 步，每步给上一步图像 x_{t-1} 加一点固定的高斯噪声（噪声强度 β_t）：

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon, \quad \varepsilon \sim N(0, I)$$

经过足够多步后，x_T 变成近乎纯高斯噪声。

1. **反向去噪过程 (Reverse Denoising)，可训练**：从噪声 x_T 出发，用神经网络逐步去噪 $x_t \to x_{t-1}$，最终还原/生成出清晰图像 x_0。

> 直觉：把“破坏数据”的过程一步步逆转，反向过程看起来就像在**从随机噪声里生成新数据**。相比 VAE/GAN 一次性生成，扩散是**从粗到细、逐步**生成的。

Diffusion　前向加噪　反向去噪

14 扩散模型的网络、损失与训练

反向过程用神经网络估计每步 x_{t-1} 的分布（均值 μ_θ 与方差），网络通常是 **U-Net**（含 ResNet 块与自注意力层）。

预测噪声而非图像：实践中并不直接预测去噪图像，而是让网络 $\varepsilon_\theta(x_t, t)$ **预测当初加入的噪声** ε，再由它反推 μ_θ。

损失函数：本质是个**去噪自编码器**，最小化预测与目标的均方误差，简化形式为

$$\mathcal{L} = \mathbb{E}_{x_0,\varepsilon,t}\left[\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2\right]$$

（讲义写作 $\mathrm{MSE}(\mu_\theta(x_t, t),\, x_{t-1})$）。

训练流程：① 从数据集采图 x_0，用前向过程加噪得 x_t；② 网络据 x_t 和时间步 t 预测噪声/去噪；③ 对 t 从 T 到 0 反复估计。

优缺点：生成质量高、多样性好、训练稳定（在 ImageNet 上 **Diffusion 击败 GAN**），是 Stable Diffusion 等的基础；缺点是**采样需多步迭代、速度慢**。

Diffusion　U-Net　损失函数

15 三类模型对比：GAN vs VAE vs Diffusion

模型	训练准则	采样	质量/多样性	主要缺点
GAN	对抗博弈 $\min_G \max_D V$	一步生成，快	图像锐利	训练不稳定、无显式似然
VAE	最大化对数似然（重建+KL）	一步生成，快	隐空间规整可解释，但偏模糊	重建/正则权衡致图像糊
Diffusion	去噪 MSE（预测噪声）	多步迭代，慢	质量高、多样性好、训练稳	采样慢、计算量大

> 选择直觉：要快且锐用 GAN；要可解释隐空间用 VAE；要最高保真度可接受慢用 Diffusion。实验表明在 ImageNet 等基准上 **Diffusion 的图像质量可超过 GAN**。

对比　总结

16 📝 例子1：学会"画猫" vs 学会"判断是不是猫"

判别式和生成式的区别，用小朋友学画画来体会最直观。

场景：给班里两个小朋友看了一万张猫的照片。

- **小明（判别式模型）**：他只学了一件事——"给我一张图，我告诉你这是不是猫"。考试时给他猫图他说"是"，给他狗图他说"不是"。但你让他**凭空画一只猫**，他画不出来，因为他从没学过"猫长什么样、该怎么生成"，他只学了一条"分界线"。
- **小红（生成式模型）**：她学的是"猫到底长什么样"——有毛、有胡须、有尖耳朵、眼睛的位置……她把"猫的样子"这个**分布**记在脑子里。所以你让她闭眼**画一只全新的、世界上不存在的猫**，她也能画出来。

分步骤理解为什么不同：

1. 判别式学的是 $p(y \mid x)$：已知图 x，问它属于哪类 y。它只关心"类与类之间的边界在哪"，不关心猫本身的细节。
2. 生成式学的是 $p(x)$（数据本身的分布）：它要刻画"什么样的 x 才像真猫"。学到了 $p(x)$，就能从里面**采样**出新猫图。
3. 所以判别式只能**回答问题**，生成式能**创造新东西**。GAN、VAE、扩散都属于生成式。

> 一句话记忆：判别式 = 鉴别师（认得出），生成式 = 画家（画得出）。能画出来的，一定也认得出；但只会认的，不一定画得出。

例子

17 📝 例子2：造假币的小偷 vs 验钞的警察（GAN 一轮轮博弈）

GAN 训练像一场"小偷造假币、警察验钞"的猫鼠游戏，一轮轮升级。**生成器 G = 造假币的小偷，判别器 D = 验钞的警察**。

第 1 轮：

1. 小偷刚入行，造的假币是一张白纸涂点颜色，非常假。
2. 警察拿真钞和这张假币一对比，轻松识破："真钞我给 0.99 分，你这假币 0.01 分"。
3. 小偷被抓后**反思**："原来要有水印和头像"，于是改进工艺。

第 2 轮：

1. 小偷加上了水印和头像，假币更像了。
2. 警察这次有点犹豫，但还是看出纸张手感不对，又识破了。
3. 警察也**升级**：开始检查纸张材质。
4. 小偷再反思，换了纸……

第 N 轮（理想终点）：

- 小偷的假币和真钞**一模一样**，警察怎么看都只能瞎猜，正确率掉到 50%（$D \approx 0.5$）。
- 此时说明小偷已经完全掌握了真钞的"分布"，即 $p_G = p_{data}$，**这正是 GAN 的全局最优**。

关键点：两人目标完全相反（对抗）——警察想让"真币判真、假币判假"（最大化 V）；小偷想让"假币也被判真"（最小化 V），合起来就是 $\min_G \max_D V(D, G)$。

例子

18 📝 例子3：VAE 为什么"编码成一个分布"还能反向传播（重参数化）

普通自编码器把图压成一**个点** z；VAE 把图压成一**团云（一个高斯分布）**。为什么要这么麻烦？用"投递地址"的比喻来理解。

比喻：普通 AE 像精确门牌，VAE 像"小区范围"

- 普通 AE：每张猫图对应隐空间里**精确一个点**（门牌号 8 号楼 3 单元）。两个门牌之间的空地没人住，随便戳一个坐标解码出来是一堆乱码——所以 AE 不能随机采样生成。
- VAE：每张猫图对应隐空间里一**小片区域**（"这一带都是这只猫附近"），用均值 μ（中心在哪）和标准差 σ（范围多大）描述。训练时强迫所有图的"小区"都挤在标准高斯 $N(0, I)$ 这个大城里、互相重叠铺满，于是**整片隐空间都住满了人**，随便采一个点都能解码出有意义的图。

第二个难点：采样这一步怎么求导?

1. VAE 要从 $z \sim N(\mu, \sigma^2)$ 里**随机抽**一个 z 去解码。
2. 但"随机抽"是个**掷骰子**动作，骰子没法求导——梯度流到这里就断了，编码器的 μ, σ 没法更新。
3. **重参数化技巧**：把掷骰子"挪到旁边"。先在与参数无关的地方掷一个标准骰子 $\varepsilon \sim N(0, I)$，再用公式 $z = \mu + \sigma \odot \varepsilon$ 把它"搬运"到目标分布。
4. 这样 z 对 μ, σ 就是**确定的加法和乘法**（可求导！），随机性全甩给了和参数无关的 ε。

类比：原来是"在 μ 这个位置直接掷一个范围为 σ 的骰子"（位置和骰子缠在一起，没法对位置求导）；现在是"先掷一个固定的标准骰子拿到结果 ε，再把它平移 μ、缩放 σ"。平移和缩放当然能求导，骰子被隔离在外面，于是梯度能一路传回编码器。

> 记忆：$z = \mu + \sigma \odot \varepsilon$ —— 把"随机"挤到 ε 里，剩下的全是可导运算。

例子

19 📝 例子4：扩散模型——把照片一步步"撕碎成雪花"，再学着"拼回来"

扩散模型的思路特别像"把一杯清水一滴滴滴入墨水，最后变全黑；再训练一个机器人**倒着**把墨水一点点抽出来还原清水"。用一张猫照片具体走一遍。

前向加噪过程（固定，不用学）：

1. 拿一张清晰猫照片 x_0。
2. 第 1 步：撒一把很细的雪花（高斯噪声），得到 x_1，猫还看得清，只是有点麻点。
3. 第 2 步：再撒一把，得到 x_2，更糊了……
4. 一直撒到第 T 步（比如 1000 步），x_T 变成**纯雪花噪声**，完全看不出原来是猫。
5. 公式：$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon$ —— 每步把旧图缩一点、掺一点新噪声。这个过程**没有任何参数要学**，照公式撒就行。

反向去噪过程（要训练的神经网络）：

1. 现在反过来：从一团**纯雪花** x_T 出发，目标是一步步擦掉雪花、还原出清晰图。
2. 训练时给网络看"第 t 步的糊图 x_t"，让它**猜：这一步当初撒进去的雪花 ε 长什么样？**
3. 网络猜出噪声后，把它**减掉一点**，就得到稍微清晰一点的 x_{t-1}。
4. 损失就是"猜的噪声"和"真撒的噪声"的均方误差：$\mathcal{L} = \|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$。
5. 推理生成时：随手抓一团新雪花 x_T，让训练好的网络从 T 到 0 **去噪 1000 步**，最后蹦出一只全新的猫——因为每一步都"擦掉一点噪声"，看起来就像在"从噪声里慢慢长出图来"。

为什么这样更好（对比 GAN/VAE 的一步生成）：

- GAN/VAE 是"一口气"把噪声变成图，难度大、容易翻车（不稳定/模糊）。
- 扩散把任务**拆成 1000 个简单小步**，每步只需"去掉一点点噪声"，简单又稳定，所以质量高、多样性好。
- 代价：生成要跑很多步，**慢**。

> 一句话：加噪是"把猫慢慢揉成雪花"（固定），去噪是"学会把雪花慢慢捏回猫"（神经网络）。

例子

第 18 讲 强化学习

Reinforcement Learning

强化学习研究智能体(Agent)如何在与环境交互的动态过程中，通过试错学习一个策略 π 以最大化长期累积奖励 $R_t = \sum_{i \geq t} \gamma^{i-t} r_i$。本讲从马尔可夫决策过程(MDP)建模出发，依次讲解值函数与Q函数、Q 值的递归与迭代估算、蒙特卡洛与时序差分(TD)、Q-Learning、DQN、策略梯度(REINFORCE)、Actor-Critic与优势函数，并以AlphaGo、PPO/GRPO等案例收束到大模型对齐。

1 第三类学习范式：强化学习概览

机器学习通常分为三大范式：

- **监督学习**：数据是 (x, y)，目标是学习映射 $x \to y$，如分类、回归。
- **无监督学习**：数据只有 x 无标签，目标是挖掘隐含结构，如聚类、降维。
- **强化学习(Reinforcement Learning, RL)**：数据是 **状态-动作对 + 奖励(state-action pairs + rewards)**，目标是 **从环境中获得最大化的累积奖励**。

RL 的核心特征是 **在动态环境中通过交互试错学习**：没有现成的标准答案标签，只有环境给出的奖励/惩罚(reward/penalty)信号。典型应用包括下围棋(AlphaGo)、星际争霸(AlphaStar)、Atari 游戏、机器人控制、自动驾驶等。这些场景共同的问题是：**在每个状态下应该采取哪个动作，才能最终取得成功(赢)?**

强化学习 学习范式 概念

2 强化学习的基本要素：Agent / Environment / State / Action / Reward

强化学习系统由若干基本要素构成：

- **智能体(Agent)**：执行动作并从中学习的扮演者。
- **环境(Environment)**：智能体所处的外部“世界”，智能体可对其采取动作。
- **状态(State s_t)**：智能体对环境的观测(observation)。
- **动作(Action a_t)**：智能体在环境中可执行的一个操作；所有可能动作的集合称为 **动作空间(Action space)** A(如 $\leftarrow \uparrow \rightarrow \downarrow$)。
- **奖励(Reward r_t)**：环境对智能体某个动作打分的反馈(feedback)信号，可正可负。
- **策略(Policy π)**：从状态到动作的映射 $\pi : S \to A$，决定在每个状态下做什么。

交互循环为：智能体在状态 s_t 下根据策略选动作 a_t → 环境给出奖励 r_t 并转移到新状态 s_{t+1} → 重复。这一闭环是所有 RL 算法的基础。

基本要素 Agent 策略

3 马尔可夫决策过程 (MDP)

RL 本质上是一个 **随机过程**，更确切地说是 **马尔可夫决策过程(Markov Decision Process, MDP)**——它在马尔可夫过程的基础上引入了 **动作** 和 **奖励**。

马尔可夫性(Markov property)：下一状态只依赖于当前状态和动作，与更早的历史无关。

MDP 用 **五元组** $(S, A, \{P_{sa}\}, \gamma, R)$ 表示：

- S：状态集合(a set of states)。
- A：动作集合(a set of actions)。
- $\{P_{sa}\}$：状态转移概率(probabilities of state transition)，即在状态 s 执行动作 a 后转移到各状态的概率分布。
- $R : S \times A \to \mathbb{R}$：奖励函数(reward function)。
- $\gamma \in [0, 1]$：未来奖励的 **折扣因子(discount factor)**。

MDP 的动态过程(Dynamics)：① 从状态 s_0 开始；② Agent 选动作 $a_0 \in A$；③ 得到奖励 $r(s_0, a_0)$；④ 按 $s_1 \sim P_{s_0 a_0}$ 随机转移；⑤ 重复。MDP 为强化学习建立了一种统一的数学框架。

MDP　马尔可夫　五元组

4 长期回报与折扣因子 γ

RL 的目标不是最大化单步奖励，而是 **长期回报(long-term return)**。

- **实际长期回报**：从时刻 t 起所有即时奖励之和

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + r_{t+2} + \cdots$$

- **折扣长期回报(Discounted return)**：用折扣因子 γ 对未来奖励加权

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

折扣因子 $\gamma \in [0, 1]$ 的含义与作用：

- $\gamma \to 0$：智能体“目光短浅”，只看重眼前奖励。
- $\gamma \to 1$：智能体“高瞻远瞩”，几乎同等看重遥远的未来奖励。
- 数学上，$\gamma < 1$ 能保证无限时域(infinite horizon)下回报级数收敛，避免发散；同时反映“未来不确定、当下奖励更可靠”的直觉。

> 学习目标可概括为：找到一个策略 π，使期望折扣回报 $\mathbb{E}[R_t]$ 最大化。

长期回报　折扣因子　目标函数

5 动作值函数 Q(s,a) 与最优策略

Q 函数(动作值函数, action-value function) 度量在状态 s 执行动作 a 后，能获得的 **期望总未来回报**：

$$Q(s_t, a_t) = \mathbb{E}[R_t \mid s_t, a_t]$$

它相当于给每个 `(状态, 动作)` 对打分 `score(state, action)`。

Q 函数 **隐式定义了最优策略**：在任意状态 s 下，选择使 Q 值最大的动作即为最优动作：

$$\pi^\star(s) = \arg\max_a Q(s, a)$$

直觉：好的策略应当在每个状态选择“未来回报最大”的动作。

(后续 Actor-Critic 还会用到 **状态值函数(state-value function)** $V^\pi(s) = \mathbb{E}[R_t \mid s]$，它只评估“处在状态 s 有多好”，是对动作按策略求期望后的标量。)

Q函数 值函数 最优策略

6 Q 值的递归结构与迭代估算

Q 值具有 **递归(recursive)结构**——当前状态的 Q 值依赖于下一状态的 Q 值。

推导(对最优 Q)：

$$Q(s_t, a_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] = r_t + \gamma\, \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots]$$

$$= r_t + \gamma \max_{a'} Q(s_{t+1}, a')$$

即把无穷级数“折叠”成 **即时奖励 + 折扣的未来最优 Q 值**。这给出了两类估计 Q 的思路：

- **简单更新(Q 定义直接代入)**：$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s', a')$。
- **更稳健的增量更新(时序差分)**：$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$，其中 α 是学习率，括号内是 TD 误差。

这一递归思想(基于对下一状态的估计来更新当前 Q)是 Q-Learning、DQN 等值方法的理论基石。

递归 Q值 迭代估算

7 估计 Q 值的两条路：蒙特卡洛 vs 时序差分

如何计算 Q 值？讲义给出两大方案，区别在于 **用什么来近似未来回报**：

方案一：蒙特卡洛预演 (Monte-Carlo Rollout)

- 思想：依赖重复随机采样获得数值结果(如用随机投点估计圆面积)。最朴素地认为 **值(value) = 平均累计奖励(mean return)**。
- 做法：用策略 π 从 s_t 采样 N 个完整片段(episode)，计算每个片段的真实累计回报 R_t，再取平均更新 Q：

$$Q(s_t, a_t) = \mathbb{E}[R_t \mid s_t, a_t] \approx \frac{1}{N} \sum_{i=1}^{N} R_t^{(i)}$$

- 特点：使用 **真实回报(use real return)**，无偏(unbiased)，但 **必须等整局结束** 才能更新，且方差大。

方案二：时序差分 (Temporal-Difference, TD)

- 思想：用 **下一状态的估计值** 代替剩余真实回报(use the value from the next state)，即自举(bootstrapping)。
- 更新：$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$。
- 特点：**单步即可更新**，方差小、效率高，但因依赖自身估计而 **可能有偏(biased)**。

蒙特卡洛 时序差分 TD 估计方法

8 Q-Learning 算法与网格世界示例

Q-Learning 用一张查找表(lookup table)记录所有 `(s,a)` 对的 Q 分数，并基于 TD 思想反复更新，直到收敛。

核心更新式：

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

(简化版直接令 $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$。)

网格世界(Grid World)示例：

- 设非终止状态每步奖励 $r = -0.04$(鼓励尽快到达目标)，终止格子给 ± 1。
- 初始化 Q 表全为 0，从起点出发；每走一步用更新式回填 Q 值，例如 $Q(s_0, a_\uparrow) \leftarrow -0.04 + \gamma \max(0, 0, 0) = -0.04$。
- 到达终止状态后重新开始(start over)，多轮迭代后 Q 表收敛。
- 最终从 Q 表中读出每个格子的最优动作 $\arg\max_a Q(s, a)$，即得到一张 **策略地图**。

关键点：Q-Learning 的更新目标里用 $\max_{a'}$(贪婪)，因此学到的是 **最优策略对应的 Q 值**。多轮迭代后，算法就为每个状态(格子)总结出一套动作策略。

Q-Learning 网格世界 查找表

9 探索与利用：ε-greedy 采样

在训练 RL 智能体(如 DQN)时，需要平衡 **探索(Exploration)** 与 **利用(Exploitation)**：一味选当前最优动作(利用)可能错过更好的动作；一味随机(探索)又学不到好策略。

ε-greedy 采样：以概率 ε 随机选一个动作(探索)，以概率 $1-\varepsilon$ 选 $\arg\max_a Q(s,a)$(利用)。

这正是 DQN 算法采样步骤所用的策略——讲义的 Deep Q-learning 算法中写道：

> With probability ε select a random action a_t, otherwise select $a_t = \max_a Q^\star(s_t, a; \theta)$.

直觉：万一某个当前 Q 值偏低的动作其实更好，但因没试过而分数虚低，只“利用”就永远发现不了；探索就是给这些动作一个被尝试的机会。实践中常在训练初期把 ε 设大(多探索)，随训练逐渐衰减(多利用)。

探索利用 | ε-greedy | DQN采样

10 Q-Learning 的局限：维度灾难

表格型 Q-Learning 的致命缺陷是 **维度灾难(Curse of Dimensionality)**：`(s,a)` 对的空间太大，无法枚举存表。

- 若状态由 N 个二值变量描述，则状态数为 2^N；若每维有 k 个取值，则为 k^N——随维度 **指数爆炸**。
- 当状态是连续的(如车的位置/速度、屏幕像素)时，状态数 **无限**，根本无法用离散表格表示。

> 结论：真实问题(Atari 画面、围棋棋盘、机器人传感器)中状态往往是不可枚举的(intractable)。

破解之道：不再为每个 `(s,a)` 存一个数，而是 **用一个带参数 θ 的函数(神经网络)去近似 Q 值** $Q_\theta(s,a)$——这就引出了深度 Q 网络(DQN)。

维度灾难 | 局限 | 函数近似

11 深度 Q 网络 (DQN) 与经验回放、目标网络

DQN(Deep Q-Network) 用深度神经网络(如 CNN)近似 Q 函数 $Q_\theta(s,a)$，把状态(如游戏画面图像)作为输入。常见输出形式：输入状态 s，一次性输出所有动作的 Q 值 $Q_\theta(s,a_1),\ldots$。

学习目标(回归到 TD 目标)：以 $y = r_t + \gamma \max_a Q_\theta(s_{t+1}, a)$ 为目标，最小化均方误差(MSE)：

$$\ell(\theta) = \left\| \left(r_t + \gamma \max_a Q_\theta(s_{t+1}, a)\right) - Q_\theta(s_t, a_t) \right\|^2$$

预测过程：对新状态 s 计算所有 $Q_\theta(s,a)$，选 $a = \arg\max_a Q_\theta(s,a)$。

两大稳定化技巧：

- **经验回放(Experience Replay)**：把多回合的转移 (s,a,r,s') 存入回放缓冲区 D，训练时从中 **随机采样小批量(mini-batch)**。这打破样本间时间相关性，提高数据利用率。
- **目标网络(Target Network)**：直接用同一个 θ 既算目标又算预测会导致 **目标非平稳(non-stationary target)**、训练震荡。解决办法是用一套 **固定的旧参数** θ_{old} 计算目标，每隔 C 步才同步一次：

$$\nabla_{\theta_t} L = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{old}) - Q(s,a;\theta_t)\right) \nabla_{\theta_t} Q(s,a;\theta_t)\right]$$

DQN | 经验回放 | 目标网络

12 值学习 vs 策略学习：Policy Gradient 思想

RL 算法可分为两大类：

- **值学习(Value Learning)**：先学 $Q(s,a)$，再 **间接** 推出策略 $a = \arg\max_a Q(s,a)$。DQN 属此类。
- **策略学习(Policy Learning)**：**直接** 学一个策略 $\pi(s)$，按 $a \sim \pi(s)$ 采样动作。

策略梯度(Policy Gradient)的关键思想：用神经网络 $\pi_\theta(a \mid s)$ 直接输出动作的概率分布(如 $P(a_1 \mid s) = 0.9$)，并 **直接对策略本身做优化**，无需经过 Q 函数中转。

相比 DQN(从 Q 值取 argmax 得确定性动作)，策略网络天然输出 **随机策略**，更适合动作空间巨大或连续、最优动作需要随机性的场景。训练时用智能体的对局经验估计参数 θ；测试时按 $a \sim \pi_\theta(a \mid s)$ 选动作。

策略学习 值学习 策略梯度

13 策略梯度推导与 REINFORCE 算法

目标：最大化期望累计回报 $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)] = \sum_\tau R(\tau) p_\theta(\tau)$，用梯度上升 $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$。

对数似然技巧(log-derivative trick) 推导梯度：

$$\nabla_\theta J(\theta) = \sum_\tau R(\tau) \nabla_\theta p_\theta(\tau) = \sum_\tau R(\tau) p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = \mathbb{E}_\tau[R(\tau) \nabla_\theta \log p_\theta(\tau)]$$

由于轨迹概率 $p_\theta(\tau) = p(s_1) \prod_t \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$ 中，环境转移项与 θ 无关，求对数梯度后只剩策略项：

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau^{(i)}) \sum_t \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$$

REINFORCE 算法(Williams 1992)：

1. 按当前策略 π_θ 采样若干完整轨迹 $\{\tau^{(1)}, \dots, \tau^{(N)}\}$；
2. 对每条轨迹计算各 (s_t, a_t) 的对数概率 $\log \pi_\theta(a_t \mid s_t)$ 及其梯度，并 **乘以该轨迹总回报** $R(\tau)$ 作为权重；
3. 用平均策略梯度更新 $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$，重复至收敛。

直觉：与分类的交叉熵 $\nabla L = -\sum_t \nabla \log \pi_\theta(a_t \mid s_t)$ 相比，策略梯度多了权重 R_τ——**回报高的轨迹提高其动作概率，回报低/为负的降低概率**。

策略梯度 REINFORCE 对数似然技巧

14 策略梯度的局限与 Actor-Critic 框架

REINFORCE 的局限：

- 回报 $R(\tau) = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} r_T$ **必须等整局结束** 才能算出；
- 不同轨迹的 $\{R_\tau\}$ **方差很大**，导致梯度估计噪声大、训练不稳。

能否在每个 s_t 处就估出 $\mathbb{E}(R_\tau)$？这引出 **演员-评论家(Actor-Critic)** 框架，融合策略学习与值学习：

- **演员(Actor)**：策略网络 $\pi_\theta(a_t \mid s_t)$，负责在每个状态预测/采样动作 —— policy-based。
- **评论家(Critic)**：值网络，估计长期回报 $\mathbb{E}(R_t \mid s_t, a_t) = Q^\pi(s_t, a_t)$ 或 $V^\pi(s_t)$，对动作好坏打分 —— value-based。

把 REINFORCE 中的 R_τ 换成 Critic 给出的 $Q(s_t, a_t)$，就得到 Q Actor-Critic：

$$J_{act}(\theta) = \sum_t Q(s_t, a_t) \log \pi_\theta(a_t \mid s_t)$$

Critic 提供逐步的、低方差的价值估计，使 Actor 不必等到整局结束即可更新。

Actor-Critic　策略梯度局限　方差

15 优势函数与 TD 误差：A2C 算法

Q 值往往数值很大，直接用作权重会引入大方差。做法是 **减去一个基线(baseline)** $V(s_t)$，得到 **优势函数(Advantage Function)**：

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

含义：动作 a 相对“平均动作”好多少。$A > 0$ 说明该动作是“惊喜”，值得 **提高其概率**；$A < 0$ 说明比平均差，应 **降低其概率**。目标函数变为

$$\nabla_\theta J(\theta) = \mathbb{E}\Big[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A^\pi(s_t, a_t)\Big]$$

用 TD 误差替代优势函数：状态值函数满足 $V^\pi(s_t) \approx r + \gamma V^\pi(s_{t+1})$，定义 **时序差分误差(TD error)**

$$\delta_t = r + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

可以证明 δ_t 恰好是优势函数 $A^\pi(s, a)$ 的一个 **无偏估计**。于是：

- **Actor 更新**(策略梯度，用 δ_t 当权重)：$\nabla J(\theta) = \sum_t \delta_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$。
- **Critic 更新**(类似 Q-Learning 的值回归)：$L(\varphi) = \mathbb{E}[\|r + \gamma V(s'; \varphi) - V(s; \varphi)\|^2]$，同样可用固定旧目标 φ_{old} 稳定训练。

这就是 **基于优势的 Actor-Critic 算法(A2C)**：每步采样动作 $a_t \sim \pi_\theta$，算 TD 目标 $y_t = r + \gamma V_\varphi(s_{t+1})$ 与误差 $\delta_t = y_t - V_\varphi(s_t)$，用 δ_t^2 更新 Critic、用 $\theta \leftarrow \theta + \eta \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot \delta_t$ 更新 Actor。

优势函数　TD误差　A2C

16 案例 AlphaGo：策略网络、值网络与蒙特卡洛树搜索

围棋对计算机极难：棋盘配置约 10^{700} 种，博弈树复杂度 b^d(围棋分支 $b \approx 250$、深度 $d \approx 150$，远超国际象棋的 35/80)，无法暴力穷举。

RL 建模：状态=棋盘配置(可看作图像，用 CNN 表示)；动作=落子位置；奖励=非终局 $r_t = 0$、终局胜负 ± 1；目标=最大化期望总收益。

三大网络/搜索组件：

- **策略网络(Policy Network)**：输入棋盘输出落子概率，**缩小搜索宽度(breadth)**。先用 3000 万人类棋谱做监督学习(最大化 $\log p_\sigma(a \mid s)$，达 57% 预测准确率)，再用 **自我对弈 + 策略梯度** $\sigma \leftarrow \sigma + \eta \nabla_\sigma \log p_\sigma(a \mid s) z$ 强化。
- **值网络(Value Network)**：评估某局面有多好 $V(s_t) = \mathbb{E}[z_t]$，用蒙特卡洛得到的真实胜负 z 做回归 $L = \|z_t - V_\theta(s_t)\|^2$，**缩小搜索深度(depth)**。
- **蒙特卡洛树搜索(MCTS)**：结合采样进行决策，四阶段循环 —— **Selection**(按 $Q + u(P)$、$u(P) \propto P/N$ 选叶节点)、**Expansion**(用策略网络展开下一层并赋先验概率 P)、**Evaluation/Rollout**(用值网络+随机预演估计该节点价值)、**Backup/回溯**(把评估值沿树向上更新访问计数 N 与动作值 Q)。

AlphaGo　MCTS　策略网络　值网络

17 强化学习与大语言模型：RLHF、PPO 与 GRPO

RL 是大语言模型(LLM)对齐的核心工具。**LLM Agent 的 MDP 设定**：状态 s_t = {对话历史, 环境上下文, 智能体记忆}；动作 a_t=文本生成(或调用工具/API)；奖励=即时反馈(如生成正确文本 $r = +1$、错误 $r = -1$)；由策略 $\pi(a \mid s)$ 驱动。

RLHF 两阶段：① 训练 **奖励模型(Reward Model)** 对回答打分；② 用 RL(常用 PPO) 以奖励分数微调 SFT 模型。

PPO (Proximal Policy Optimization, OpenAI 2017)：在 Actor-Critic 基础上支持 **离线/旧数据学习**，并通过 **截断(clip)机制** 限制新旧策略差距防止策略跑偏。引入策略比率 $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$，截断目标函数：

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

其中 A_t 为优势函数。流程：旧策略 θ_{old} 采集轨迹 → Critic 估计目标价值与 A_t → 多个 epoch 用 mini-batch 梯度上升更新 Actor、梯度下降更新 Critic。

GRPO (Group Relative Policy Optimization)：相对 PPO 的改进，**砍掉庞大的 Critic 价值网络**，改用 **群组内部相对比较** 计算优势。对同一输入生成多个候选 $\{o_1, \ldots, o_G\}$ 及奖励 $\{r_1, \ldots, r_G\}$，相对优势 $A_i = \dfrac{r_i - \text{mean}(r_{1:G})}{\text{std}(r_{1:G})}$，再用类 PPO 的截断目标并加 KL 正则与参考策略对齐，显著降低训练开销。

RLHF　PPO　GRPO　大语言模型

18 📝 例题1：手把手算一条轨迹的累积折扣回报 G

目标：搞懂 $R_t = \sum_{i \geq t} \gamma^{i-t} r_i$ 这个公式到底怎么代数字。

题目设定：一个小机器人在迷宫里走，每走一步环境给一个奖励 r。它一局(episode)走了 4 步，依次拿到的即时奖励是：

时刻	r_0	r_1	r_2	r_3
奖励	-1	-1	-1	$+10$

含义：前 3 步每步罚 -1(鼓励它别磨蹭)，最后一步走到出口奖励 $+10$。折扣因子取 $\gamma = 0.9$。

问：从 $t = 0$ 算起，这条轨迹的累积折扣回报 $G = R_0$ 是多少?

第一步：写出公式，明确每一项要乘几次 γ

$$R_0 = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3$$

> 关键直觉：**离现在越远的奖励，打折越狠**。第 k 步的奖励要乘 γ^k。$\gamma = 0.9 < 1$，所以未来奖励“缩水”。

第二步：算出每个 γ 的幂

- $\gamma^0 = 1$
- $\gamma^1 = 0.9$
- $\gamma^2 = 0.9 \times 0.9 = 0.81$
- $\gamma^3 = 0.81 \times 0.9 = 0.729$

第三步：逐项代入相乘

- 第0步：$1 \times (-1) = -1$
- 第1步：$0.9 \times (-1) = -0.9$
- 第2步：$0.81 \times (-1) = -0.81$
- 第3步：$0.729 \times (+10) = 7.29$

第四步：把四项加起来

$$R_0 = -1 - 0.9 - 0.81 + 7.29 = \mathbf{4.58}$$

对比一下：如果换成 $\gamma = 1$(不打折)，$R_0 = -1 - 1 - 1 + 10 = 7$；如果 $\gamma = 0$(只看眼前)，$R_0 = r_0 = -1$。

> **结论**：γ 越大，机器人越“高瞻远瞩”，越愿意忍受前面的小惩罚去换最后的大奖励；γ 越小越“目光短浅”。这就是折扣因子的物理意义。

例题

19 📝 例题2：手把手做一次 Q-learning 更新

目标：把更新公式 $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ 里的每个符号都换成数字算一遍。

题目设定：智能体在状态 s 选了动作 a("向右")，环境给奖励 $r = 5$ 并把它带到新状态 s'。学习率 $\alpha = 0.5$，折扣 $\gamma = 0.9$。当前 Q 表里的几个数已知：

- 旧值 $Q(s,a) = 2$
- 新状态 s' 有三个可选动作，它们当前的 Q 值是：$Q(s', \uparrow) = 10$，$Q(s', \downarrow) = 3$，$Q(s', \rightarrow) = 6$

问：更新后的 $Q(s,a)$ 变成多少?

第一步：先算 $\max_{a'} Q(s',a')$ —— 在新状态里挑最大的那个 Q

$$\max_{a'} Q(s',a') = \max(10,\ 3,\ 6) = 10$$

> 为什么取 max? 因为 Q-learning 假设“到了 s' 之后我会走最优动作”，所以更新目标里用未来最好的那个 Q 值。

第二步：算 TD 目标(target) = 即时奖励 + 折扣后的未来最优 Q

$$\text{target} = r + \gamma \max_{a'} Q(s',a') = 5 + 0.9 \times 10 = 5 + 9 = 14$$

第三步：算 TD 误差(error) = 目标 − 当前估计

$$\delta = \text{target} - Q(s,a) = 14 - 2 = 12$$

> $\delta = 12$ 是正的，说明“实际比我原来想的好”，所以这个动作的 Q 值应该往上调。

第四步：按学习率 α 把旧值往目标方向挪一点

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,\delta = 2 + 0.5 \times 12 = 2 + 6 = \mathbf{8}$$

> **理解 α 的作用**：$\alpha = 0.5$ 表示“只朝目标走一半”。如果 $\alpha = 1$，直接跳到 $Q = 14$；如果 α 很小(如 0.1)，则更新很慢、更稳。本例里 Q 从 2 更新到了 8，正好是 2 和目标 14 的“中点”。

例题

20 📝 例题3：ε-greedy 探索 vs 利用怎么选动作

目标：用骰子的思路搞懂 ε-greedy 到底是怎么在“探索”和“利用”之间做选择的。

题目设定：智能体在某状态 s 下有 4 个动作，当前 Q 表为：

动作	←	↑	→	↓
Q 值	2	**8**	5	1

探索率 $\varepsilon = 0.1$。

问：(1) 哪个动作是“利用(贪婪)”动作？(2) 选到它的总概率是多少？(3) 每个动作各自被选中的概率？

第一步：找出贪婪动作 = Q 值最大的那个

$$a^\star = \arg\max_a Q(s,a) = \uparrow \quad (\text{因为 } Q=8 \text{ 最大})$$

这就是“**利用(Exploitation)**”要选的动作——用已学到的知识选当前最好的。

第二步：理解 ε-greedy 的两种情况

- 以概率 $1-\varepsilon = 0.9$ → **利用**：直接选贪婪动作 ↑。
- 以概率 $\varepsilon = 0.1$ → **探索**：从全部 4 个动作里 **均匀随机** 挑一个(每个 $0.1/4 = 0.025$)。

> 为什么要探索？万一某个现在 Q 值低的动作其实更好，但因为没试过所以分数虚低，只“利用”就永远发现不了。探索就是给冷门动作一个被尝试的机会。

第三步：算每个动作的总概率

非贪婪动作(只能靠探索被选中)：

$$P(\leftarrow) = P(\rightarrow) = P(\downarrow) = \frac{\varepsilon}{4} = \frac{0.1}{4} = 0.025$$

贪婪动作 ↑(利用时一定选它 + 探索时也可能随机抽到它)：

$$P(\uparrow) = \underbrace{(1-\varepsilon)}_{\text{利用}} + \underbrace{\frac{\varepsilon}{4}}_{\text{探索抽中}} = 0.9 + 0.025 = \mathbf{0.925}$$

第四步：验算总概率是否为 1

$$0.925 + 0.025 + 0.025 + 0.025 = 1.000 \ \checkmark$$

> **结论**：贪婪动作 ↑ 有 92.5% 的概率被选中(主要靠利用)，其余动作各 2.5%(纯靠探索)。训练初期常把 ε 设大(多探索)，后期逐渐衰减到很小(多利用)。

例题

第 19 讲 期末复习

Final Review

本讲是全课程要点串讲，串联监督学习、神经网络与深度学习、无监督与生成模型、强化学习四大板块，梳理高频考点、核心公式与易混辨析。

1 机器学习总览：四要素与考试形式

机器学习是用统计方法、从数据中改进以达成目标、无需人工显式编程的 AI 子领域。

四要素 (Key Elements):

1. **数据 Data**（经验 Experience）
2. **模型 Model**（假设 Hypothesis）
3. **损失函数 Loss Function**（目标 Objective）
4. **优化 Optimization**（改进 Improvement）

> 考试为**开卷**（可带任何纸质资料）、中文作答、可带计算器；题型含判断、选择、简答、计算、系统设计。

学习记忆框架：拿到任何一个模型（如 SVM、逻辑回归、DQN）都可以按「假设/模型形式 → 损失函数 → 优化方法」三段式去对比记忆。

总览 四要素 考试

2 过拟合与欠拟合：诊断与防治

- **欠拟合 Underfitting**：模型不够复杂，无法捕捉真实规律，假设可能不成立 → 训练误差和测试误差都高。
- **过拟合 Overfitting**：模型太复杂，把数据细节（如随机噪声）也学进来，而非底层规律 → 训练误差很低但测试误差高。

随复杂度增大，训练误差单调下降，而测试误差先降后升，呈 U 形，最优点在中间（**合适复杂度**）。

防止过拟合的常用手段:

1. 增加训练数据
2. **正则化**（惩罚模型复杂度，如 L_1/L_2）
3. **留出法 / 交叉验证**（用未见数据评估泛化）
4. **早停 Early Stopping**
5. 引入先验知识（如贝叶斯先验）

> 易混点：增加模型复杂度、减小正则化系数会**加重**过拟合；它们是欠拟合时才用的手段。

过拟合 正则化 泛化

3 决策树与 ID3：信息增益计算

决策树自顶向下、分而治之：构造含全部数据的根 → 按某准则选最有利属性 → 按属性取值划分子集 → 为每个子集建子节点 → 递归直到停止。

ID3 用信息增益选属性。数据集熵：

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$$

条件熵（按属性 A 划分后）：

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i)$$

信息增益：

$$\text{Gain}(D, A) = H(D) - H(D|A)$$

计算示例（PPT 样例）：根节点 $H(D) \approx 0.91$，划分后 $H(D|A=1) = 0$、$H(D|A=2) \approx 1$，信息增益即用 $H(D)$ 减去加权条件熵。

> 易混：信息增益越大越好（选它划分）；纯节点熵为 0。ID3 偏向取值多的属性，C4.5 用信息增益率纠正。

决策树　ID3　信息增益

4 朴素贝叶斯：条件独立假设

贝叶斯分类目标（MAP）：

$$c_{MAP} = \arg\max_{c} P(x_1, \ldots, x_d|c)\, P(c)$$

直接估计联合概率 $P(x_1, \ldots, x_d|c)$ 参数量巨大、不可行。

朴素贝叶斯的核心：条件独立假设——给定类别 c，各特征相互独立：

$$P(x_1, \ldots, x_d|c) \approx \prod_{j=1}^{d} P(x_j|c)$$

于是

$$c_{NB} = \arg\max_{c} P(c) \prod_{j=1}^{d} P(x_j|c)$$

训练即从数据估计 $P(c)$ 与各 $P(x_j|c)$：

$$\hat{P}(c) = \frac{\text{count}(C=c)}{N}, \quad \hat{P}(x_i|c) = \frac{\text{count}(x_i, c)}{\sum_{x} \text{count}(x, c)}$$

> 朴素贝叶斯把指数级参数降为线性级，是其「朴素」的代价与好处；实践中常配合拉普拉斯平滑避免零概率。

朴素贝叶斯　条件独立　生成式

5 K-NN：决策边界与 K 的作用

K 近邻按「**多数投票**」分类：用距离函数找 k 个最近邻，取其中最常见类别。属于**惰性学习**（无显式训练，全部计算在预测时）。

决策边界：

- $K = 1$ 时由 **Voronoi 分割（维诺图）** 给出，每个区域归属最近的训练点；边界是到两个不同类训练点等距的点。
- 数据多且标签有噪声时，$K = 1$ 边界会非常「破碎」。

K 的影响：

- K 越大，边界越**平滑**（抗噪声、偏差增大）。
- $K = N$ 时永远预测**多数类**。

> 易混：K 太小→过拟合（对噪声敏感）；K 太大→欠拟合（边界过平滑）。K-NN 无参数但预测开销大、依赖距离度量。

KNN　Voronoi　决策边界

6 线性判别 / 逻辑回归 / Softmax 回归对比

线性判别函数：$g_i(x) = w_i^T x + w_{i0}$，若 $g_1(x) > g_2(x)$ 判 C_1。优点：$O(d)$ 时空复杂度、可解释（加权和）、假设满足时准确。

Sigmoid / 逻辑回归（二分类）：从 log-odds（logit）线性出发

$$\log \frac{y}{1-y} = w^T x + w_0 \;\Rightarrow\; y = \frac{1}{1+\exp[-(w^T x + w_0)]}$$

$y = P(C_1|x)$ 是后验概率，Sigmoid 平滑、便于优化。

Softmax 回归（多分类）：

$$y_i = \frac{\exp(w_i^T x + w_{i0})}{\sum_{j=1}^{K} \exp(w_j^T x + w_{j0})}$$

输出归一化的类别概率，选概率最大类。

> 对比：逻辑回归是 Softmax 在 $K = 2$ 时的特例；两者输出都可解释为后验概率，区别在二分类 vs 多分类。

逻辑回归　Softmax　线性分类

7 各损失函数汇总：交叉熵的二分类与多分类形式

二分类交叉熵（逻辑回归），$r \in \{0,1\}$：

$$\ell(w, w_0|x, r) = -r \log y - (1-r)\log(1-y)$$

来源：$r=1$ 时最大化 $\log y$，$r=0$ 时最大化 $\log(1-y)$，等价于最大化伯努利似然 $p(r|x) = y^r(1-y)^{1-r}$。

多分类交叉熵（Softmax），r 为 one-hot：

$$L_{CE}(y, r) = -\sum_{i=1}^{K} r_i \log y_i = -r^T(\log y)$$

回归常用：均方误差 $\frac{1}{2}\|\hat{y} - r\|^2$（如 DQN/Critic 的值回归）。

> 关键认识：**交叉熵 = 最大似然 = 最小化 KL 散度**。$CE(p,q) = \mathbb{E}_{x\sim p}\log\frac{1}{q(x)} = -\sum_x p(x)\log q(x)$。

损失函数 交叉熵 最大似然

8 SVM：最大间隔与约束优化

SVM 寻找**最大间隔**超平面。规范化使边界两侧最近的样本点满足 $w^Tx + w_0 = \pm 1$。两个最近异类点 $x^{(1)}, x^{(2)}$ 满足 $w^T(x^{(1)} - x^{(2)}) = 2$，间隔为

$$\text{margin} = \frac{w^T(x^{(1)} - x^{(2)})}{\|w\|} = \frac{2}{\|w\|}$$

最大化间隔 $\Leftrightarrow$ 最小化 $\frac{1}{2}\|w\|^2$。

约束优化形式：

$$\min_w \ \frac{1}{2}\|w\|^2 \quad \text{s.t. } y^{(\ell)}(w^Tx^{(\ell)} + w_0) \geq 1,\ \ell = 1, \ldots, N$$

这是一个**二次规划 (QP)**，属于凸优化，复杂度依赖输入维度 d。

> 易混：约束 $y^{(\ell)}(w^Tx^{(\ell)} + w_0) \geq 1$ 统一了两类（$y = \pm 1$）。最大化间隔对应最小化 $\|w\|$（不是最大化）。

SVM 最大间隔 凸优化

9 神经网络与反向传播：前向缓存 + 误差回传

感知机是神经元的基本建模：输入函数 $a = \sum_j w_j x_j$，激活函数 $y = \sigma(a)$。单层只能线性可分；**加隐藏层**可学习非线性映射，每个隐单元负责一条线性边界。

反向传播 = 链式法则，分两步：

- **前向 Forward Pass**：对每个权重预先算出 $\partial a/\partial w =$ 连到该权重的输入 z（线性单元梯度固定、易算）。
- **反向 Backward Pass**：递归计算每个线性输出的 $\partial l/\partial a$。

$$\frac{\partial l}{\partial w} = \frac{\partial a}{\partial w} \cdot \frac{\partial l}{\partial a} = z \cdot \frac{\partial l}{\partial a}$$

误差传播视角（误差信号 δ）：

$$\delta^L = \sigma'(z^L) \odot \nabla_y \text{Loss}, \qquad \delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$

> 记忆：$\partial l/\partial w =$（前向缓存的输入 z）×（反向传来的误差 δ）。

神经网络　反向传播　链式法则

10 深度学习核心思想：可训练特征与层次表示

- **传统 ML**：人工设计/固定特征 + 可训练分类器。
- **深度学习**：**可训练特征提取器 + 可训练分类器**（端到端）。

层次表示 (Hierarchical Representations)：低层特征 → 中层特征 → 高层特征，逐层抽象、模块化（modularization），且模块自动从数据中学到。

深 vs 浅（同参数量下，深网络通常更优）：深网络通过模块复用，能用更少数据/参数表达复杂函数。

词嵌入 Word Embedding：把词表示为低维稠密向量，编码语义相似性（相比 one-hot 稀疏正交）。Word2Vec（CBOW / Skip-gram）通过「中心词 ↔ 上下文词」的向量相似度预测来学习，是分布式表示。

> 易混：one-hot 各词正交（无相似性）；嵌入向量距离反映语义近似。

深度学习　表示学习　词嵌入

11 序列建模：RNN → Attention → Transformer

语言模型：给序列打概率 $p(x) = p(w_1, \ldots, w_N)$，应用于补全、纠错、翻译、语音识别。纯计数法离散、无语义、对相似句不敏感。

RNN：复用隐藏状态存历史，$h_t = f_W(h_{t-1}, x_t)$，**每个时间步共享同一组参数**。可做序列生成、Encoder-Decoder（seq2seq）。

注意力 Attention：解决固定编码向量的瓶颈，解码时动态生成上下文 $c_i = \sum_j \alpha_{ij} h_j$，权重

$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

自注意力 Self-Attention：每个词关注所有词，$Q = W_q H$, $K = W_k H$, $V = W_v H$：

$$H' = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V$$

Transformer：多头自注意力 + 前馈 + Add&Norm + 位置编码；解码器用 **Masked** 多头注意力（只看已生成部分）。预训练语言模型 = 大规模无监督预训练 + 下游有监督微调。

RNN Attention Transformer

12 CNN：卷积动机、输出尺寸与分割检测

MLP 处理图像的问题：破坏空间结构、对局部不敏感、平移不变性差、参数过多。

卷积的核心：局部连接（每个隐单元只看局部 patch）+ **权值共享**，保留空间信息、减少参数。3-D 卷积核**贯穿输入全部通道**；n 个滤波器产生 n 张特征图（activation map）。

输出尺寸公式（务必会算）：

$$\text{output} = \frac{N + 2P - F}{\text{stride}} + 1$$

例：$N = 7, F = 3$，stride 1 → 7；stride 2 → 4；stride 3 → 3。

应用：语义分割（逐像素分类，下采样+上采样，转置卷积/反卷积上采样，U-Net 用跳跃连接）；目标检测（Fast R-CNN 先过卷积再裁剪 ROI）；ViT（图像切 16×16 patch 喂 Transformer）。

> 高频计算题：给定 N, F, P, stride 求输出特征图尺寸。

CNN 卷积 输出尺寸

13 无监督学习：K-means、PCA、自编码器

无监督三大方向：**聚类**（K-means、GMM）、**降维**（PCA、自编码器）、**生成**（GAN、VAE、Diffusion）。

K-means 步骤：①随机选 k 个种子为初始质心 ②每点分到最近质心 ③用当前成员重算质心（均值）④重复②③直至收敛。

PCA：找方差最大的方向。投影方差 $\text{var}(u^TX) = u^TSu$，$S = \frac{1}{N}XX^T$ 为协方差。优化：

$$\max_u u^TSu \quad \text{s.t. } u^Tu = 1$$

拉格朗日求解得 $Su = \lambda u$ → **特征分解**：取 k 个最大特征值对应的特征向量为主成分；特征值 = 投影方差，特征向量正交。

自编码器 Auto-encoder：encoder→瓶颈隐层（latent code）→decoder，用**重构损失**强迫学到压缩表示。

> 易混：PCA 是线性降维、解析解（特征分解）；自编码器可非线性、靠梯度训练。K-means 对初始化敏感、可能陷局部最优。

无监督 PCA K-means 自编码器

14 生成模型：GAN 与扩散模型

生成目标：学到 $p_{model}(x)$ 逼近 $p_{data}(x)$。

GAN：生成器 G 把噪声 $z \sim N(0, I)$ 变为复杂分布 P_G；判别器 D 区分真假。极小极大博弈：

$$\min_G \max_D V(G, D) = \mathbb{E}_{x\sim P_{data}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]$$

D 最大化（分辨真假），G 最小化（骗过 D）。全局最优时 G 重现真实分布。CycleGAN 用循环一致（重构 L_2）损失保持内容。

扩散模型 Diffusion：

- **前向加噪**（固定）：逐步给 x_0 加高斯噪声 β_t，$x_0 \to x_T$（噪声）。
- **反向去噪**（可训练）：用神经网络（U-Net / 去噪自编码器）逐步从 x_t 预测 x_{t-1} 的 μ, σ^2，$x_T \to x_0$。

> 易混：GAN 是对抗训练、易模式坍塌；扩散稳定但采样慢。前向过程无参数，反向过程才训练。

生成模型 GAN 扩散模型

15 强化学习基础：回报、Q、V 与 DQN

RL 从与环境交互中学习：状态 S、动作 A、转移规则、奖励规则，目标是学策略（状态→动作）使**长期回报**最大。

折扣回报：$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i = r_t + \gamma r_{t+1} + \dots$

Q 函数（状态-动作价值）满足贝尔曼递归：

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t] = r_t + \gamma\, Q(s_{t+1}, a_{t+1})$$

状态值函数：$V^{\pi}(s) = \mathbb{E}[R_t | s]$，TD 误差 $\delta = r + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$。

DQN：用神经网络近似 Q_θ，**经验回放**打散相关性，目标值 $y = r + \gamma \max_{a'} Q_\theta(s', a')$：

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}\left[\left\| (r + \gamma \max_{a'} Q_\theta(s', a')) - Q_\theta(s, a) \right\|^2\right]$$

> 易混：Q 评估「状态+动作」，V 只评估「状态」；DQN 用 max 取目标动作，属离策略（off-policy）。

强化学习 Q函数 DQN

16 策略梯度家族：PG → Actor-Critic → PPO → GRPO

策略梯度 (REINFORCE)：直接优化策略 π_θ，

$$\nabla_\theta R_\theta = \mathbb{E}_{\tau \sim p(\tau)}[R(\tau) \nabla_\theta \log p_\theta(\tau)] \approx \frac{1}{N} \sum\sum R^{(l)} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

用整条轨迹回报 $R(\tau)$ 加权，方差大。

Actor-Critic：Actor 出动作（policy-based），Critic 估值（value-based）。用 **优势函数** $A(s,a) = Q(s,a) - V(s)$ 替代 R 降方差；**TD 误差** $\delta = r + \gamma V(s') - V(s)$ **是优势的无偏估计**：

$$\nabla J(\theta) = \sum_t \delta_t \nabla \log \pi_\theta(a_t | s_t)$$

Critic 做值回归最小化 δ^2。

PPO（OpenAI 2017）：离线学习，用旧策略采样，引入策略比率 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$，**截断**目标防止策略跑偏：

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t A_t,\ \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t)\right]$$

GRPO（群组相对策略优化）：**砍掉 Critic 价值网络**，用同一输入生成一组候选 $\{o_1, \dots, o_G\}$，以**组内相对**计算优势：

$$A_i = \frac{r_i - \text{mean}(r_{1:G})}{\text{std}(r_{1:G})}$$

> 演进主线：降方差（引入 Critic/优势）→ 稳定（PPO 截断）→ 省算力（GRPO 去 Critic）。

策略梯度 Actor-Critic PPO GRPO

17 📝 综合例题1：同一个小数据集，KNN / 朴素贝叶斯 / 决策树三种思路对比

我们用**同一个小数据集**，分别用三种分类器去预测，体会它们「怎么想问题」的根本差异。

> 数据集（判断「是否买伞」，标签 Buy∈{是,否}）：

ID	天气Weather	湿度Humidity	Buy
1	阴	高	是
2	晴	高	否
3	阴	低	是
4	雨	高	是
5	晴	低	否

待预测样本 `x* = （天气=阴，湿度=高）`。

思路一：KNN（惰性、看邻居）

1. KNN 不训练，预测时才算距离。把类别特征用「相同记 0、不同记 1」的汉明距离来比。
2. 算 `x*` 到每个样本的距离：到 ID1=(0,0)→**0**；ID2=(1,0)→1；ID3=(0,1)→1；ID4=(1,0)→1；ID5=(1,1)→2。
3. 取 `K=1`：最近的是 ID1（距离 0），它的标签是「是」→ **预测「是」**。
4. 为什么：KNN 完全靠「谁离我最近」，不抽象任何规律，本质是 Voronoi 划分。K 太小对噪声敏感。

思路二：朴素贝叶斯（生成式、比概率）

1. 先验：$P(\text{是}) = 3/5$，$P(\text{否}) = 2/5$。
2. 条件概率（用计数，给定类别下逐特征独立）：

- $P(\text{天气}=\text{阴}|\text{是}) = 2/3$，$P(\text{湿度}=\text{高}|\text{是}) = 2/3$
- $P(\text{天气}=\text{阴}|\text{否}) = 0/2 = 0$，$P(\text{湿度}=\text{高}|\text{否}) = 1/2$

1. 比后验（取对数前先看乘积）：

- 是：$3/5 \times 2/3 \times 2/3 = 0.267$
- 否：$2/5 \times 0 \times 1/2 = 0$

1. **预测「是」**。注意「否」类出现了 0 概率——实践中要用**拉普拉斯平滑**避免直接清零。
2. 为什么：朴素贝叶斯假设「给定类别后各特征独立」，把联合概率拆成连乘，参数从指数级降到线性级。

思路三：决策树 ID3（按信息增益分裂）

1. 根节点熵：5 个里 3 个「是」2 个「否」，$H(D) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \approx 0.971$。
2. 看「天气」分裂：阴(2 个全「是」,熵0)、晴(2 个全「否」,熵0)、雨(1 个「是」,熵0)→ 加权条件熵 $H(D|\text{天气}) = 0$，**信息增益 = 0.971**（最大！）。
3. 「天气」一刀就把数据切纯了，直接当根属性。`x*` 天气=阴 → 落到「全是」叶子 → **预测「是」**。
4. 为什么：决策树主动**寻找最能降熵的属性**，构造可读的 if-else 规则。

> **三者殊途同归都预测「是」，但思路完全不同**：KNN 记忆样本、靠距离；朴素贝叶斯估计概率、靠独立假设连乘；决策树抽取规则、靠信息增益分裂。考试常考「给场景判断各自优缺点」。

综合例题

18 📝 综合例题2：给场景选模型（线性回归 / 逻辑回归 / SVM / 决策树）

考试常考「给一个业务场景，该选哪个模型并说理由」。记住选型口诀：**先看输出是连续还是离散 → 再看要不要概率/可解释 → 最后看数据是否线性可分**。

场景A：预测某城市明天的用电量（千瓦时）

- 输出是**连续数值** → 这是**回归**问题。
- 选 **线性回归**：模型 $\hat{y} = w^T x + w_0$，损失用 **MSE** $\frac{1}{2}\|\hat{y} - r\|^2$。
- 为什么不选逻辑回归/SVM 分类器？因为它们输出的是类别或概率，不是连续量。

场景B：判断一封邮件是否为垃圾邮件，并且想知道「有多大把握」

- 输出是二**分类** + 需要**概率**（置信度）→ 选 **逻辑回归**。
- 为什么：逻辑回归 $y = \text{sigmoid}(w^T x + w_0)$ 直接输出后验概率 $P(C_1|x)$，可设阈值（如 0.9 才判垃圾）。
- 为什么不选 SVM？SVM 默认只给「在边界哪一侧」，不直接输出概率。

场景C：高维特征（如文本上万维）、样本不算多、要追求强泛化的二分类

- 选 **SVM**：目标 $\min \frac{1}{2}\|w\|^2$ s.t. $y^{(\ell)}(w^T x^{(\ell)} + w_0) \geq 1$，**最大间隔**带来好的泛化，复杂度依赖维度 d、对高维友好。
- 为什么不选决策树？高维稀疏下单棵树容易过拟合。

场景D：银行要把「是否放贷」的决策规则给业务员看，必须能解释、能画成流程图

- 选 **决策树**：天然是 if-else 规则、可视化、可解释；能直接处理类别特征、不需要标准化。
- 为什么不选 SVM/逻辑回归？它们是「加权和」式黑箱权重，业务员不易读懂分裂规则。

分步选型流程（背下来）：

1. 输出连续？→ **线性回归**。
2. 输出离散（分类）→ 进入第 3 步。
3. 要概率/简单可解释的线性边界？→ **逻辑回归**。
4. 高维、要最大间隔强泛化？→ **SVM**。
5. 要规则可视化、混合特征、强可解释？→ **决策树**。

> 易混提醒：逻辑回归名字里有「回归」，但它是**分类**模型（输出概率再阈值化），不要被名字骗了。

综合例题

19 📝 综合例题3：判断过拟合还是欠拟合，并给对策

这是高频综合题：给训练/测试误差，判断病症并开药方。核心口诀——**看「训练误差」高不高，再看「训练与测试的差距」大不大**。

给定三个模型在同一任务上的表现：

模型	训练误差	测试误差
A	30%	32%
B	2%	28%
C	8%	10%

第1步：逐个诊断。

- **模型A**：训练误差就很高(30%)，测试也高 → 模型根本没学会 → **欠拟合 (Underfitting)**。说明假设太简单、容量不够。
- **模型B**：训练误差极低(2%)，但测试误差高(28%)，二者**差距巨大** → 把训练集噪声背下来了 → **过拟合 (Overfitting)**。
- **模型C**：训练误差不高、测试误差也不高、差距小 → **复杂度合适**，是 U 形曲线的最优点附近，最佳。

第2步：对症下药。

欠拟合(模型A)对策——增加模型容量：

1. 用更复杂的模型（如线性→加多项式特征/换非线性模型/加隐藏层）。
2. 增加特征、减少正则化强度（**减小** λ）。
3. 训练更久（别太早停）。

过拟合(模型B)对策——抑制复杂度/增强泛化：

1. **增加训练数据**（最根本）。
2. **正则化**：加 L_1/L_2 惩罚，**增大** λ。
3. **交叉验证 / 留出法**选超参数。
4. **早停 Early Stopping**。
5. 引入先验知识。

第3步：避坑（易混点）。

> ⚠️ 同样是调正则化系数 λ，方向相反：过拟合要**调大** λ、欠拟合要**调小** λ。

> ⚠️ 「增加模型复杂度」「减小正则化」是治**欠拟合**的，对过拟合是火上浇油。

> ⚠️ 别只看训练误差！只看训练误差会把过拟合(模型B)误判为「最好」。

随复杂度增大：训练误差单调降，测试误差先降后升（U 形），最优点在中间。

综合例题

20 📝 综合例题4：MSE / 交叉熵 / 合页损失——用一个例子看何时用哪个

三种损失不是随便挑的，**任务类型决定损失函数**。我们用一句话场景把它们串起来。

场景：一个模型对某个样本给出输出，真实标签已知。我们分三种任务看用哪个损失。

① 回归任务 → 均方误差 MSE

- 场景：预测房价，真实 $r = 300$（万），模型预测 $\hat{y} = 280$。
- 损失：$\frac{1}{2}(\hat{y} - r)^2 = \frac{1}{2}(280 - 300)^2 = 200$。
- 为什么用它：输出是**连续值**，MSE 惩罚「预测与真值的欧氏距离」，可导、好优化。
- 课程中 DQN/Critic 的**值回归**也用 MSE：$\|(r + \gamma \max_{a'} Q(s', a')) - Q(s, a)\|^2$。

② 分类任务（要概率）→ 交叉熵 Cross-Entropy

- 场景：二分类，真实 $r = 1$，模型输出概率 $y = 0.8$。
- 损失：$-r \log y - (1 - r) \log(1 - y) = -\log 0.8 \approx 0.22$；若 $y = 0.2$ 则 $-\log 0.2 \approx 1.61$（错得离谱罚得重）。
- 为什么用它：分类输出是**概率分布**，交叉熵 = 最大似然 = 最小化 KL 散度，配 Sigmoid/Softmax 梯度漂亮。
- 多分类用 $L_{CE} = -\sum_i r_i \log y_i = -r^T \log y$（$r$ 为 one-hot）。

③ 最大间隔分类（SVM）→ 合页损失 Hinge Loss

- 场景：SVM 二分类，$y \in \{-1, +1\}$，定义间隔分 $m = y \cdot (w^T x + w_0)$。
- 损失：$\max(0, 1 - m)$。若 $m \geq 1$（点在间隔外、分对且够远）→ 损失 **0**，不再管它；若 $m < 1$ → 线性惩罚。
- 为什么用它：SVM 追求**最大间隔**，分得「又对又远」（$m \geq 1$）的点直接零损失、不再惩罚，只对落在间隔内/分错的点线性惩罚；合页损失正是「最大间隔」思想的损失形式，课程中用**梯度下降**直接优化它。

一句话对比记忆：

- 连续输出 → **MSE**（罚距离）。
- 要概率的分类 → **交叉熵**（罚似然，错得越自信罚越狠）。
- 要最大间隔的分类 → **合页**（分对且够远就零损失，只盯边界）。

> 易混：分类问题用 MSE 也能跑，但配 Sigmoid 时梯度容易饱和、收敛慢，所以分类**首选交叉熵**。

综合例题

21 📝 综合例题5：同一个问题，有监督 / 无监督 / 强化学习三种提法

这三类学习的区别不在「数据」，而在「**有没有标签、反馈是什么形式**」。我们用**同一个业务场景——电商用户**，换三种提法来区分。

背景：一家电商平台有大量用户的浏览、购买行为数据。

① 有监督学习（Supervised）——有明确标签

- 提法：「**已知**哪些用户是流失用户（标签 churn=是/否），训练模型**预测**新用户会不会流失。」
- 特点：每条样本都带**正确答案**（标签）。学的是「输入 → 标签」的映射。
- 用什么：逻辑回归 / SVM / 决策树（分类）；损失用**交叉熵**，最小化「预测与真实标签的差」。
- 关键词：**标签、预测、分类/回归**。

② 无监督学习（Unsupervised）——没有标签

- 提法：「我们**不知道**用户该分几类，请把行为相似的用户**自动分组**，挖掘潜在人群。」
- 特点：**没有标签**，只有输入 x。学的是数据的**内在结构**。
- 用什么：**K-means**（聚类，按距离把用户分到最近质心）；或 **PCA / 自编码器** 把高维行为压缩成低维表示。
- 关键词：**无标签、聚类、降维、找结构**。

③ 强化学习（Reinforcement）——靠奖励反馈试错

- 提法：「让一个推荐**智能体**给用户推商品，用户**点击/下单就给奖励**，没反应就没奖励，让它**通过不断交互学会**最大化长期收益的推荐策略。」
- 特点：没有「标准答案」，只有**延迟的标量奖励** r；要在**与环境交互**中学**策略**（状态→动作），目标是最大化**长期折扣回报** $R_t = \sum_i \gamma^{i-t} r_i$。
- 用什么：DQN（学 $Q(s,a)$）/ 策略梯度 / Actor-Critic / PPO。
- 关键词：**奖励、策略、状态-动作、长期回报、试错**。

一表对比（务必能区分）：

维度	有监督	无监督	强化
数据	有标签 (x, y)	只有 x	状态/动作/奖励
学什么	输入→标签	数据结构	策略(状态→动作)
反馈	正确答案	无	标量奖励(可延迟)
代表	逻辑回归/SVM	K-means/PCA	DQN/PPO

> 易混：聚类(无监督)和分类(有监督)都「分组」，区别在**分类有现成标签、聚类没有**。强化学习的「奖励」不是标签——它不告诉你「正确动作是什么」，只告诉你「这步好不好」。

综合例题