Introduction to Artificial Intelligence

邢添瑆

2025 年 12 月 30 日

目录

Chapter 1

模拟思维：搜索与博弈

1.1 引言：什么是搜索？

人类解决问题的过程往往可以抽象为在可能的状态集合中寻找目标状态的过程。在人工智能领域，这一过程被称为**搜索（Search）**。如果环境中有其他竞争主体，这种搜索就演变为**博弈（Game Playing）**。

本章我们将构建智能体（Agent）的决策大脑，从最基础的盲目游走，到具备远见卓识的 A* 算法，再到与对手博弈时的剪枝策略，最终抵达基于模拟的蒙特卡洛树搜索。

1.2 搜索问题的形式化定义

在讨论具体算法之前，我们需要用数学语言严格定义什么是“搜索”。

定义：搜索问题五元组

一个标准的搜索问题形式化为 (S, s_0, A, T, C)：

1. **状态空间 (State Space, S)**：所有可能状态的集合（例如：地图上所有路口）。
2. **初始状态 (Initial State, s_0)**：智能体的起始点，$s_0 \in S$。
3. **动作 (Actions, $A(s)$)**：在状态 s 下合法动作的集合。
4. **转移模型 (Transition Model, $T(s, a)$)**：状态转移函数 $s' = T(s, a)$。
5. **目标测试 (Goal Test)**：判断当前状态是否为目标。
6. **路径代价 (Path Cost)**：$C(s, a, s')$，通常假设每一步代价非负。

1.3 无信息搜索 (Uninformed Search)

无信息搜索（Blind Search）指智能体没有任何关于“目标在哪里”的线索，只能按固定顺序遍历。

1.3.1 宽度优先 (BFS) 与深度优先 (DFS)

这是考试中最基础的对比点，请务必背诵下表：

维度	宽度优先搜索 (BFS)	深度优先搜索 (DFS)
策略	层层推进 (Layer-wise)	一条路走到黑 (Backtracking)
数据结构	**队列 (Queue)** - 先进先出	**栈 (Stack)** - 后进先出
完备性	是 (只要有解，一定能找到)	否 (可能陷入无限深分支)
最优性	是 (仅当每步代价相等时)	否
时间复杂度	$O(b^d)$	$O(b^m)$
空间复杂度	$O(b^d)$ (指数爆炸，**极高**)	$O(bm)$ (线性增长，**极低**)

表 1.1: BFS 与 DFS 对比 (b: 分支因子, d: 解深度, m: 最大深度)

注记：核心理解

为什么 BFS 容易内存溢出？] 假设分支因子 $b = 10$（每个路口有 10 条路）。

- **DFS**：像一个孤独的旅行者，只记得回家的路。深度为 10 时，只存 10 个节点，空间占用非常低。
- **BFS**：像地毯式轰炸。深度为 10 时，需要同时记住第 10 层的所有节点（10^{10} 个），内存瞬间爆炸。

通俗讲解：DFS 和 BFS 的使用场景

- **BFS**：适合寻找最短路径问题，例如迷宫求最少步数、社交网络最短距离等。
- **DFS**：适合需要探索所有可能解的问题，如组合问题、图的拓扑排序、连通性检测。
- **空间敏感场景**：当内存有限时，优先考虑 DFS。
- **可控深度**：DFS 可以通过限制最大深度避免陷入无限分支。

1.4 启发式搜索 (Heuristic Search)

为了解决盲目搜索效率低下的问题，我们引入**启发函数** $h(n)$：估计从节点 n 到目标的最小代价。启发式搜索不仅考虑历史代价，还结合未来预估，智能选择搜索方向，从而大幅减少搜索空间。

1.4.1 A* 算法 (A-Star)

A* 算法是一种结合了**历史代价**与**启发式估计**的最优路径搜索算法，其核心思想是：在每一步搜索中，选择综合代价最小的节点进行扩展，从而既保证搜索效率，又能保证找到最优解（前提是启发函数可采纳）。

$$f(n) = g(n) + h(n) \tag{1.1}$$

其中：

- $g(n)$：从起点到节点 n 的已知代价。
- $h(n)$：从节点 n 到目标节点的预估代价（启发函数）。
- $f(n)$：综合代价，用于优先选择扩展节点。

注记：核心理解

可以把 A* 想象成一个旅行者：

- $g(n)$ 是他已经走过的路程；
- $h(n)$ 是他对剩余路程的乐观估计；
- $f(n)$ 就是他每次选择的决策依据——既考虑过去，也考虑未来。

A* 算法伪代码

Algorithm 1: A* 搜索伪代码

Input: 起点 $start$, 目标 $goal$, 启发函数 $h(n)$

Output: 从起点到目标的最优路径

初始化优先队列 $Open$，将 $start$ 加入 $Open$，$g(start) = 0$;
初始化空集合 $Closed$; // 已扩展节点集合
while $Open$ 非空 **do**
    从 $Open$ 中取出 $f(n)$ 最小的节点 $current$;
    if $current = goal$ **then**
        return 通过回溯父节点构建的最优路径;
    end
    将 $current$ 加入 $Closed$;
    for 每个 $current$ 的邻居 $neighbor$ **do**
        if $neighbor \in Closed$ **then**
            continue;
        end
        计算 $tentative\_g = g(current) + cost(current, neighbor)$;
        if $neighbor \notin Open$ 或 $tentative\_g < g(neighbor)$ **then**
            $parent(neighbor) = current$;
            $g(neighbor) = tentative\_g$;
            $f(neighbor) = g(neighbor) + h(neighbor)$;
            if $neighbor \notin Open$ **then**
                将 $neighbor$ 加入 $Open$;
            end
        end
    end
end
return 失败，目标不可达;

关键点详解

- **优先队列 (Open)**：按 $f(n)$ 排序，保证每次扩展的是最有希望的节点。
- **Closed 集合**：记录已经扩展过的节点，避免重复扩展。
- **可采纳性 (Admissibility)**：保证 $h(n) \leq h^*(n)$，使 A* 不会错过最优解。
- **一致性 (Consistency / Monotonicity)**：若 h 满足一致性，则 A* 在图搜索中不会重新扩展节点。

- **路径回溯**：通过每个节点记录父节点，最终从目标节点回溯得到完整路径。

注记：核心理解

- 启发函数越接近真实代价，搜索空间越小，效率越高。
- $h(n) = 0$ 时，A* 退化为 Dijkstra 算法。
- 启发函数过低估，仍保证最优但搜索节点多。
- 启发函数过高估，搜索节点少，但无法保证最优。

1.5 对抗搜索 (Adversarial Search)

当环境中存在具有**自主决策能力和相互冲突目标**的多个智能体时，搜索问题将不再是单一智能体的路径规划问题，而演化为**博弈问题 (Game Problem)**。典型例子包括象棋、围棋、国际象棋以及各种双人对抗类游戏。在这类环境中，智能体不仅要考虑自身的行动结果，还必须显式建模对手可能采取的策略与反应。

在人工智能中，这类问题通常抽象为**完全信息博弈**，即双方对当前状态、可行动作以及收益函数都有完全认知。同时，许多经典博弈（如棋类游戏）可建模为**零和博弈 (Zero-Sum Game)**，即一方的收益完全等于另一方的损失，用数学形式表示为：

$$u_{\text{Max}}(s) = -u_{\text{Min}}(s)$$

在零和博弈假设下，双方的目标完全对立，使得问题结构更加清晰，便于理论分析。

1.5.1 Minimax 算法

核心思想：

Minimax 算法基于一个关键假设：**对手是完全理性的**。也就是说：

- 我方（Max）总是试图最大化自己的最终收益；
- 对手（Min）总是试图最小化我的收益（等价于最大化自己的收益）。

因此，在决策时，智能体不能假设对手会犯错，而必须在**最坏情况下仍能保证自身收益最大**。这一思想在博弈论中称为**极小化最大收益原则 (Minimize the Maximum Loss)**。

在博弈树中：

- Max 节点表示轮到我方行动；
- Min 节点表示轮到对手行动；

- 叶节点表示终局状态，其效用值由效用函数直接给出。

定义：

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{若 } s \text{ 为叶节点} \\ \max_{a \in A(s)} \text{Minimax}(\text{Result}(s,a)), & \text{若 } s \text{ 为 Max 节点} \\ \min_{a \in A(s)} \text{Minimax}(\text{Result}(s,a)), & \text{若 } s \text{ 为 Min 节点} \end{cases} \tag{1.2}$$

其中：

- s 表示博弈中的某一状态；
- $A(s)$ 表示状态 s 下所有合法动作的集合；
- $\text{Result}(s,a)$ 表示在状态 s 执行动作 a 后到达的新状态；
- $\text{Utility}(s)$ 是终局状态的效用值，用于衡量该状态对 Max 的好坏程度。

算法流程：

1. 从根节点（当前局面）出发，递归展开博弈树；
2. 对所有可能的行动序列，计算其最终终局效用；
3. 自底向上传播效用值：
    - Max 节点选择其子节点中的最大值；
    - Min 节点选择其子节点中的最小值；
4. 在根节点选择对应最大 Minimax 值的动作作为最优决策。

理论性质：

- **最优性**：在对手完全理性、博弈树完整展开的前提下，Minimax 保证找到最优策略；
- **完备性**：若博弈树是有限的，则 Minimax 一定能够终止；
- **时间复杂度**：$O(b^d)$，其中 b 为分支因子，d 为搜索深度；
- **空间复杂度**：$O(bd)$（深度优先实现）。

伪代码：

```
function Minimax(node, player):
    if node is terminal:
        return Utility(node)
    if player == Max:
        value = -∞
        for each action in node.actions:
            value = max(value, Minimax(Result(node, action), Min))
        return value
    else:
        value = +∞
        for each action in node.actions:
            value = min(value, Minimax(Result(node, action), Max))
        return value
```

1.5.2 Alpha-Beta 剪枝

尽管 Minimax 在理论上是最优的，但其计算复杂度随博弈深度呈指数级增长。在实际问题中（如国际象棋或围棋），完全展开博弈树是不可行的。Alpha-Beta 剪枝通过提前排除**不可能影响最终决策的分支**，显著减少搜索节点数量。

Alpha-Beta 剪枝的核心思想是：在搜索过程中维护一个当前已知的**收益区间** $[\alpha, \beta]$，并在该区间无法被改进时终止搜索。

- $\boldsymbol{\alpha}$：当前路径上，Max 已经可以保证的最小收益（下界）；
- $\boldsymbol{\beta}$：当前路径上，Min 可以强加给 Max 的最大收益（上界）。

随着搜索深入：

- α 只会单调递增；
- β 只会单调递减。

剪枝条件：

- 在 Max 节点，若发现某一子节点的值 $\geq \beta$，则可以立即停止搜索该节点的其余子节点；
- 在 Min 节点，若发现某一子节点的值 $\leq \alpha$，同样可以停止后续搜索。

这是因为在上述情况下，对手一定不会允许该分支被选中，其结果不可能影响根节点的最终决策。

Alpha-Beta 伪代码：

```
function AlphaBeta(node, alpha, beta, player):
    if node is terminal:
        return Utility(node)
    if player == Max:
        value = -∞
        for each action in node.actions:
            value = max(value, AlphaBeta(Result(node, action), alpha, beta, Min))
            if value >= beta:
                break  // Beta 剪枝
            alpha = max(alpha, value)
        return value
    else:
        value = +∞
        for each action in node.actions:
            value = min(value, AlphaBeta(Result(node, action), alpha, beta, Max))
            if value <= alpha:
                break  // Alpha 剪枝
            beta = min(beta, value)
        return value
```

性能分析：

- 在最优节点顺序下，Alpha-Beta 的时间复杂度可降至 $O(b^{d/2})$；
- 剪枝不会改变 Minimax 的最终决策结果，仅减少计算量；
- 实际性能高度依赖于**节点展开顺序**。

1.5.3 小结

- 对抗搜索通过显式建模对手行为，使智能体能够在竞争环境中进行理性决策；
- Minimax 算法在零和完全信息博弈中保证找到最优策略；
- Alpha-Beta 剪枝通过区间界限显著减少搜索规模，而不影响最优性；
- 在实际复杂博弈中，通常结合**深度限制搜索**与**启发式评估函数**，在效率与决策质量之间取得平衡。

1.6 蒙特卡洛树搜索 (MCTS)

蒙特卡洛树搜索（Monte Carlo Tree Search, MCTS）是一种基于随机模拟的决策搜索算法，尤其适合应对高分支因子和无法完全评估的复杂博弈，如围棋（Go）、将棋等。其核心思想是通过不断模拟对局来逐步估计每个动作的价值，而不是像 Minimax/Alpha-Beta 剪枝那样尝试完全搜索整个博弈树。

1.6.1 MCTS 的核心思想

MCTS 的核心在于统计模拟 + 局部搜索：

- 不需要完全搜索所有可能的状态，只在最有希望的节点上投入更多模拟。
- 利用历史模拟结果动态调整搜索策略，使搜索集中在“既有高胜率又未探索充分”的节点上。
- 四个核心阶段：**选择 (Selection)**、**扩展 (Expansion)**、**模拟 (Simulation)**、**回溯 (Backpropagation)**，形成闭环。

1.6.2 探索与利用的平衡：UCB 公式

MCTS 面临的经典问题是：**应该多尝试新动作（探索），还是多走已经表现好的动作（利用）?**

为解决这个问题，MCTS 使用 UCB1（Upper Confidence Bound）公式：

$$\mathrm{UCB}_i = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}} \tag{1.3}$$

其中：

- w_i：节点 i 的累计胜利次数。
- n_i：节点 i 被访问的次数。
- N：父节点被访问的总次数。
- c：探索系数，控制探索与利用的权衡，通常经验取 $\sqrt{2}$。

公式解释：

- 第一项 $\frac{w_i}{n_i}$：**利用**，优先选择胜率高的节点。
- 第二项 $c\sqrt{\frac{\ln N}{n_i}}$：**探索**，优先选择访问次数少的节点。

1.6.3 MCTS 的四个阶段详解

1. **选择 (Selection)** 从根节点开始，根据 UCB 公式选择子节点，沿着当前最优策略向下走，直到遇到尚未完全扩展的节点或叶节点。

2. **扩展 (Expansion)** 对于未完全展开的节点，随机选择一个尚未尝试的合法动作生成新节点，并加入树中。

3. **模拟 (Simulation)** 从新生成的节点开始，进行一次快速模拟（Rollout）。

   - 通常使用随机策略或简单启发式策略完成一局游戏。
   - 得到最终胜负结果 r，例如赢为 1，输为 0。

4. **回溯 (Backpropagation)** 将模拟结果沿路径向上更新每个节点：

$$w_i \leftarrow w_i + r$$
$$n_i \leftarrow n_i + 1$$

   更新胜率和访问次数，以便下次选择时使用 UCB 公式。

1.6.4 MCTS 算法流程总结

可以用伪代码描述 MCTS：

```
function MCTS(root, iterations):
    for i = 1 to iterations:
        node = Selection(root)
        if node not fully expanded:
            node = Expansion(node)
        result = Simulation(node)
        Backpropagation(node, result)
    return action with highest visit count from root
```

特点：

- 迭代次数越多，估计越准确。
- 可以随时停止（适合实时决策），是一种任何时刻可中断的算法。
- 对高分支因子游戏表现优异，因为不会尝试完全遍历整个树。

1.6.5 实践优化与变种

- **启发式模拟：**模拟阶段不完全随机，可以使用简单评估函数提高效率。
- **树策略改进：**可引入 RAVE（Rapid Action Value Estimation）或 PUCB（Polynomial UCB）提升探索效率。
- **并行化：**MCTS 易于多线程模拟，提高搜索速度。
- **神经网络结合：**如 AlphaGo 使用策略网络 + 价值网络引导 MCTS，提升棋力。

1.6.6 MCTS 的优缺点

优点：

- 不需要完整的状态评估函数。
- 自适应搜索，越有希望的区域越多模拟。
- 可中断，随时得到决策。

缺点：

- 对模拟策略依赖较大，纯随机模拟可能不够准确。
- 高分支因子下仍需大量迭代。
- 不保证绝对最优，只保证统计上近似最优。

Chapter 2

知识表示与推理：逻辑、规则系统与知识图谱

知识表示与推理（Knowledge Representation and Reasoning, KR&R）是人工智能的核心基础之一，其目标是将现实世界中的事实、规则与关系以形式化方式表示出来，并在此基础上进行自动推理。一个良好的知识表示体系应当同时具备**表达能力**与**可推理性**，既能描述复杂知识，又能支持高效推断。

2.1 命题逻辑与谓词逻辑（最小必备）

逻辑是最经典、最严格的知识表示形式，在符号主义人工智能中占据核心地位。根据表达能力的不同，逻辑系统主要分为命题逻辑和谓词逻辑。

2.1.1 命题逻辑

命题逻辑（Propositional Logic）是最基础的逻辑形式，其基本单位是**命题**，即非真即假的陈述句。

基本元素：

- 命题符号：$P, Q, R, \ldots$
- 逻辑联结词：
  - 否定：$\neg P$
  - 合取：$P \wedge Q$
  - 析取：$P \vee Q$
  - 蕴含：$P \rightarrow Q$
  - 等价：$P \leftrightarrow Q$

语义：命题逻辑通过**真值表**定义语义，推理的本质是判断在所有可能赋值下公式是否为真。

优点与局限：

- 优点：形式简单、推理清晰、可判定；
- 局限：无法表示对象、属性及其关系，表达能力有限。

2.1.2 谓词逻辑

为克服命题逻辑表达能力不足的问题，引入了**谓词逻辑（Predicate Logic，也称一阶逻辑）**。

核心扩展：

- 个体（对象）：x, y, z
- 谓词：$P(x), Loves(x, y)$
- 量词：
  - 全称量词：$\forall$
  - 存在量词：$\exists$

示例：

$$\forall x \ (Human(x) \rightarrow Mortal(x))$$

特点：

- 表达能力强，可描述复杂关系；
- 推理复杂度显著高于命题逻辑；
- 是专家系统、知识图谱逻辑基础。

2.2 归结原理（Resolution）的推理要点

归结原理是一种**基于反证法的自动推理规则**，广泛用于命题逻辑与一阶逻辑中的定理证明。

核心思想：

若知识库 KB 与目标命题的否定 $\neg\alpha$ 推导出矛盾，则 $\text{KB} \models \alpha$。

推理步骤：

1. 将所有公式转化为**合取范式（CNF）**；

2. 将每个子句视为一个子句集合；

3. 对含互补文字的子句应用归结规则；

4. 若推导出空子句 $\square$，则证明成立。

归结规则（命题逻辑）：

$$(P \vee Q), (\neg P \vee R) \Rightarrow (Q \vee R)$$

特点：

- 完备（在命题逻辑中）；
- 易于自动化；
- 是逻辑推理引擎的理论核心。

2.3 规则系统与专家系统：前向 / 后向链

规则系统是将知识表示为**“如果-那么”**形式的推理框架，是专家系统的基础。

2.3.1 规则表示

规则的一般形式为：

$$\text{IF } \text{Condition}_1 \wedge \cdots \wedge \text{Condition}_n \text{ THEN Conclusion}$$

示例：

$$\text{IF } Fever \wedge Cough \text{ THEN } Flu$$

2.3.2 前向链与后向链

前向链（Forward Chaining）：

- 数据驱动；
- 从已知事实出发，不断触发规则；
- 适合监控、诊断类任务。

后向链（Backward Chaining）：

- 目标驱动；
- 从目标出发，反向寻找支持条件；
- 适合问答与推理证明。

2.4 知识图谱（KG）：三元组与嵌入（常用公式）

知识图谱通过图结构表示现实世界中的实体与关系，是连接符号 AI 与统计学习的重要桥梁。

2.4.1 三元组表示

知识图谱的基本单位为**三元组**：

$$(h, r, t)$$

其中：

- h：头实体（Head）
- r：关系（Relation）
- t：尾实体（Tail）

示例：

$$(\text{Paris}, \text{capital\_of}, \text{France})$$

2.4.2 链接预测：嵌入打分函数（举例）

知识图谱嵌入将实体和关系映射到低维向量空间。

TransE 模型：

$$\mathbf{h} + \mathbf{r} \approx \mathbf{t}$$

对应打分函数：

$$f(h, r, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|$$

得分越高，三元组越可能成立。

2.5 结构化知识与大模型：检索增强与迭代推理（思想框架）

随着大语言模型（LLM）的发展，符号知识与神经模型逐渐融合。

检索增强生成（RAG）：

- 从知识库 / 知识图谱中检索相关事实；
- 将检索结果注入模型上下文；
- 提升事实准确性与可解释性。

迭代推理：

- 多步调用模型进行逻辑推演；
- 显式构造中间推理状态；
- 缓解“一步到位幻觉”问题。

这一方向代表了当前 AI 从“纯统计”走向“符号 + 神经协同”的重要趋势。

Chapter 3

模拟思维：隐马尔可夫模型 (HMM)

3.1 应用背景：序列标注与命名实体识别 (NER)

3.1.1 从文本到知识 (From Text to Knowledge)

在构建知识图谱时，我们需要从非结构化的文本中提取出结构化的信息（如实体和关系）。其中最基础的任务就是命名实体识别 (Named Entity Recognition, NER)。

定义：命名实体识别 (NER)

NER 的目标是识别文本中具有特定意义的实体，并将其归类。常见的实体类型包括：

- PER (Person)：人名（如：鲁迅、乔布斯）
- LOC (Location)：地名（如：北京、瓦坎达）
- ORG (Organization)：组织机构名（如：中国人民大学、微软）

例子：

[谢顿]$_{PER}$ 离开了 [川陀]$_{LOC}$。

3.1.2 如何让计算机理解：序列标注问题

计算机无法直接“理解”概念，我们需要将 NER 转化为一个序列标注 (Sequence Labeling) 问题。最常用的方法是 BIO 标注法：

- B-XXX (Begin)：实体的开头。
- I-XXX (Inside)：实体的中间或结尾。
- O (Outside)：非实体字符。

输入序列 (观测)	我	在	人	大	学	习
输出标签 (隐藏)	O	O	B-ORG	I-ORG	O	O

表 3.1: NER 的序列标注示例

3.1.3 序列标注的评价指标：精确率、召回率与 F1 分数

在 NER 或其他序列标注任务中，我们不仅关心模型是否能预测正确，还需要用量化指标评估模型性能。常用指标有：

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

表 3.2: 分类任务的混淆矩阵示意

- **精确率 (Precision)**：预测为正例的样本中有多少是真正的正例

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **召回率 (Recall)**：所有实际正例中被正确预测的比例

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 分数 (F-score)**：精确率和召回率的调和平均

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

在 NER 的序列标注任务中，我们通常把一个完整实体预测正确才算 TP，否则算 FP 或 FN。这种评估方法比单字或单标签精确度更严格。

3.1.4 为什么需要 HMM？

在这个问题中，我们可以看到明显的“双层结构”，这正是隐马尔可夫模型 (HMM) 的用武之地：

1. 我们能看到的：句子的文字序列（“我”, “在”, “人”, ...）。→ 对应 HMM 中的 **观测状态**。
2. 我们想知道的：每个字背后的词性或实体标签（O, B-ORG, ...）。→ 对应 HMM 中的 **隐藏状态**。
3. 上下文依赖：一个字是 B-ORG 还是 O，不仅取决于这个字本身，还取决于它前面的字（比如“人”在“人大”里是 B-ORG，在“好人”里是 O）。→ 对应 HMM 的 **马尔可夫假设**。

3.1.5 HMM 与 CRF 在序列标注中的应用

隐马尔可夫模型 (HMM)

HMM 是早期处理序列标注问题的经典模型，其基本思想如下：

- 序列中的每个观测值（文字、词）对应一个隐藏状态（标签）。
- 假设当前隐藏状态只依赖于前一个隐藏状态（**一阶马尔可夫假设**）。
- 每个隐藏状态产生一个观测值的概率（发射概率）。

对于 NER：

- **观测序列**：句子中的词或字
- **隐藏序列**：对应的实体标签

模型通过最大化整个序列的联合概率来预测最可能的隐藏状态序列，即最短路径或 Viterbi 算法。

条件随机场 (CRF)

CRF 是一种判别式模型，相比 HMM 有几个优势：

- HMM 是生成式模型，建模 $P(\text{observations, states})$
- CRF 直接建模条件概率 $P(\text{states}|\text{observations})$。
- CRF 可以灵活地使用丰富的特征函数（如上下文、词性、字形特征），而 HMM 主要受限于状态转移和发射概率。
- CRF 可以避免独立假设限制，更好地捕捉序列中标签之间的全局依赖。

在 NER 或 sequence labeling 中：

- HMM 通过发射和转移概率来计算最可能的标签序列。
- CRF 通过条件概率最大化整个标签序列的正确性，并结合特征函数进行判定。

比较 HMM 与 CRF

在实际 NER 任务中，CRF 往往比 HMM 精度更高，尤其是面对复杂上下文和多类别实体时。同时，评价指标（Precision、Recall、F1）也能帮助我们量化模型性能。

	HMM	CRF
模型类型	生成式	判别式
依赖假设	马尔可夫假设（状态依赖）	无需独立假设，可利用全局特征
特征灵活性	受限	高度灵活，可使用上下文、词形、词性等
序列预测	Viterbi	最大化条件概率（解码可用 Viterbi）

表 3.3: HMM 与 CRF 在序列标注任务中的对比

3.2 HMM 的直观理解与定义

3.2.1 直观场景：天气与冰淇淋

想象你被关在一个没有窗户的房间里（你看不到外面的天气），你唯一的信息来源是你的朋友每天是否吃冰淇淋。

- 隐藏状态 (Hidden States, Q)：真实的天气情况（晴天、雨天）。这是我们想推断但看不见的。
- 观测状态 (Observations, V)：朋友的行为（吃、不吃）。这是我们直接可见的。

我们希望通过观察一串“吃、吃、不吃”的序列，来推断那几天的天气或者是预测下一天他会不会吃。

3.2.2 HMM 的两个核心假设

HMM 之所以能计算，是因为它对现实世界做了极大的简化（假设）：

定义：1. 齐次马尔可夫假设 (Homogeneous Markov Assumption)

任意时刻的状态 q_t 只依赖于前一时刻的状态 q_{t-1}，与更早的状态无关。

$$P(q_t|q_{t-1}, q_{t-2}, \dots, q_1) = P(q_t|q_{t-1}) \tag{3.1}$$

人话：明天的天气只取决于今天，跟昨天无关。

定义：2. 观测独立性假设 (Observation Independence Assumption)

任意时刻的观测 o_t 只依赖于该时刻的状态 q_t，与其他时刻的状态或观测无关。

$$P(o_t|q_t, q_{t-1}, o_{t-1}, \dots) = P(o_t|q_t) \tag{3.2}$$

人话：我今天吃不吃冰淇淋，只取决于今天是不是晴天，跟昨天我吃没吃没关系。

3.2.3 HMM 的五元组模型 $\lambda = (N, M, \pi, A, B)$

虽然常简写为 $\lambda = (\pi, A, B)$，但完整定义如下：

1. N：隐藏状态的数量（如：晴、雨，N=2）。

2. M：观测值的数量（如：吃、不吃，M=2）。

3. π (初始状态概率向量)：

$$\pi_i = P(q_1 = i)$$

即：模型刚启动时（第一天），处于状态 i 的概率。

4. A (状态转移矩阵)：描述隐藏状态之间的转化。

$$a_{ij} = P(q_{t+1} = j | q_t = i)$$

即：如果今天是状态 i，明天变成状态 j 的概率。满足 $\sum_{j=1}^{N} a_{ij} = 1$。

5. B (观测发射概率矩阵)：描述由状态生成观测的过程。

$$b_j(k) = P(o_t = k | q_t = j)$$

即：在状态 j 下，观测到符号 k 的概率。

3.3 问题一：概率评估 (Evaluation) ——前向算法

问题描述：已知模型参数 $\lambda = (\pi, A, B)$ 和一个观测序列 $O = (o_1, o_2, \ldots, o_T)$。**目标：**计算这个观测序列出现的概率 $P(O|\lambda)$。

3.3.1 暴力解法 vs. 动态规划

- 暴力解法：列举所有可能的隐藏状态序列 Q，算出 $P(O, Q|\lambda)$，然后求和。

$$P(O|\lambda) = \sum_{Q} P(O|Q, \lambda)P(Q|\lambda)$$

  复杂度是 $O(N^T)$，指数级爆炸，完全不可行。

- 前向算法 (Forward Algorithm)：利用动态规划，复杂度降为 $O(N^2T)$。

3.3.2 前向变量 $\alpha_t(i)$ 的定义

定义：前向变量 $\alpha_t(i)$

定义 $\alpha_t(i)$ 为：在时刻 t，观测序列为 $(o_1, \ldots, o_t)$，且当前隐藏状态恰好为 i 的联合概率。

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\lambda)$$

3.3.3 算法流程与公式推导

步骤 1：初始化 (时刻 $t = 1$) 第一天处于状态 i 且观测到 o_1 的概率：

$$\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \le i \le N \tag{3.3}$$

步骤 2：递推 (时刻 $t = 1 \to T - 1$) 这是核心步骤。我们要计算 $\alpha_{t+1}(j)$，即明天处于状态 j 的概率。它来自今天所有可能的状态 i（$1 \ldots N$）：

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(o_{t+1}) \tag{3.4}$$

通俗讲解：公式解读

- $\alpha_t(i)$：今天在状态 i 且前面观测都对上了的概率积攒。
- a_{ij}：从今天状态 i 跳到明天状态 j 的概率。
- $\sum(\ldots)$：明天到达状态 j 有 N 条路（从今天的晴天来、从今天的雨天来...），把它们全加起来。
- $b_j(o_{t+1})$：到了明天状态 j 后，必须还能“发射”出明天的观测值 o_{t+1}。

步骤 3：终止最终的概率是 T 时刻所有可能状态的概率之和：

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{3.5}$$

3.3.4 前向算法伪代码

```
def forward_algorithm(O, pi, A, B):
    T = len(O) # 观测序列长度
    N = len(pi) # 状态数
    # alpha 表格: T x N
    alpha = zeros(T, N)

    # 1. 初始化
    for i in range(N):
        alpha[0][i] = pi[i] * B[i][O[0]]

    # 2. 递推
    for t in range(0, T-1):
        for j in range(N):
            sum_prob = 0
            for i in range(N):
                sum_prob += alpha[t][i] * A[i][j]
            alpha[t+1][j] = sum_prob * B[j][O[t+1]]

    # 3. 终止
    return sum(alpha[T-1])
```

3.3.5 前向算法 (Forward Algorithm) ——生动理解

例子：天气和冰淇淋（详细版）

假设我们有三天的观测记录，表示每天是否吃冰淇淋：

$$O = (\text{冰淇淋}, \text{冰淇淋}, \text{没吃})$$

隐藏状态表示每天的天气：

$$S = \{\text{晴天 (Sunny)}, \text{雨天 (Rainy)}\}$$

模型参数：

- 初始状态概率：

$$\pi = (\pi_{\text{晴}} = 0.6, \pi_{\text{雨}} = 0.4)$$

意思是第一天是晴天的概率是 0.6，雨天的概率是 0.4。

- 状态转移概率矩阵：

$$A = \begin{bmatrix} a_{\text{晴, 晴}} & a_{\text{晴, 雨}} \\ a_{\text{雨, 晴}} & a_{\text{雨, 雨}} \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

含义：

  - $a_{\text{晴, 晴}} = 0.7$：如果今天是晴天，明天继续晴天的概率。
  - $a_{\text{晴, 雨}} = 0.3$：如果今天是晴天，明天变雨天的概率。
  - $a_{\text{雨, 晴}} = 0.4$：如果今天是雨天，明天晴天的概率。
  - $a_{\text{雨, 雨}} = 0.6$：如果今天是雨天，明天还是雨天的概率。

- 发射概率矩阵：

$$B = \begin{bmatrix} b_{\text{晴}}(\text{冰淇淋}) & b_{\text{晴}}(\text{没吃}) \\ b_{\text{雨}}(\text{冰淇淋}) & b_{\text{雨}}(\text{没吃}) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

含义：

  - $b_{\text{晴}}(\text{冰淇淋}) = 0.9$：晴天吃冰淇淋的概率。
  - $b_{\text{晴}}(\text{没吃}) = 0.1$：晴天没吃冰淇淋的概率。
  - $b_{\text{雨}}(\text{冰淇淋}) = 0.2$：雨天吃冰淇淋的概率。
  - $b_{\text{雨}}(\text{没吃}) = 0.8$：雨天没吃冰淇淋的概率。

前向算法计算步骤详解

前向变量定义：

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\lambda)$$

意思是：前 t 天已经观察到的冰淇淋情况，并且第 t 天的天气是 i 状态的联合概率。

—

第一天 (t=1)：

$$\alpha_1(\text{晴}) = \pi_{\text{晴}} \cdot b_{\text{晴}}(o_1) = 0.6 \times 0.9 = 0.54$$

$$\alpha_1(\text{雨}) = \pi_{\text{雨}} \cdot b_{\text{雨}}(o_1) = 0.4 \times 0.2 = 0.08$$

解释：

- $\alpha_1(\text{晴}) = 0.54$：第一天是晴天，并且吃了冰淇淋的概率。

- $\alpha_1(\text{雨}) = 0.08$：第一天是雨天，并且吃了冰淇淋的概率。

—

第二天 (t=2)：

$$\alpha_2(\text{晴}) = [\alpha_1(\text{晴}) \cdot a_{\text{晴, 晴}} + \alpha_1(\text{雨}) \cdot a_{\text{雨, 晴}}] \cdot b_{\text{晴}}(o_2)$$

代入数值：

$$\alpha_2(\text{晴}) = [0.54 \cdot 0.7 + 0.08 \cdot 0.4] \cdot 0.9 = [0.378 + 0.032] \cdot 0.9 = 0.37$$

$$\alpha_2(\text{雨}) = [0.54 \cdot 0.3 + 0.08 \cdot 0.6] \cdot 0.2 = [0.162 + 0.048] \cdot 0.2 = 0.042$$

解释：

- $\alpha_2(\text{晴}) = 0.37$：第二天是晴天，并且前两天的观测都是冰淇淋的联合概率。
- $\alpha_2(\text{雨}) = 0.042$：第二天是雨天，并且前两天的观测都是冰淇淋的联合概率。
- 每一步都是把“昨天可能状态的概率 $\times$ 转移概率”求和，再乘上今天观察的发射概率。

—

第三天 (t=3)：
观察 $o_3 =$ 没吃

$$\alpha_3(\text{晴}) = [\alpha_2(\text{晴}) \cdot a_{\text{晴, 晴}} + \alpha_2(\text{雨}) \cdot a_{\text{雨, 晴}}] \cdot b_{\text{晴}}(o_3)$$

$$\alpha_3(\text{晴}) = [0.37 \cdot 0.7 + 0.042 \cdot 0.4] \cdot 0.1 = [0.259 + 0.0168] \cdot 0.1 = 0.0276$$

$$\alpha_3(\text{雨}) = [0.37 \cdot 0.3 + 0.042 \cdot 0.6] \cdot 0.8 = [0.111 + 0.0252] \cdot 0.8 = 0.1098$$

—

终止 (概率求和)：
观测序列 $O = (\quad , \quad , \quad)$ 的总概率：

$$P(O|\lambda) = \alpha_3(\text{晴}) + \alpha_3(\text{雨}) = 0.0276 + 0.1098 = 0.1374$$

—

每一步意义总结：

- $\alpha_t(i)$：到第 t 天，观测到前 t 天的数据，同时今天是状态 i 的概率。

- 每天的 α 都利用前一天的 α，通过转移矩阵和发射矩阵更新，避免了枚举所有状态序列。
- 最后把所有可能的第 T 天状态的 $\alpha_T(i)$ 相加，就得到观测序列出现的总概率。

直观理解：

- 前向算法就像一条概率树，每天的每个状态都是树的一个节点，每条路径的概率积攒起来。- 动态规划把每一天的节点概率记下来，不用重复计算。- 最后所有路径的概率加起来，就是观测序列出现的概率。

3.4 问题二：解码问题 (Decoding) ——Viterbi 算法

问题描述：已知模型参数 λ 和观测序列 O。

目标：找到一条最有可能的隐藏状态序列 $Q^* = (q_1^*, \ldots, q_T^*)$。

例子：通过连续 3 天吃冰淇淋，推断这 3 天最可能是“晴-晴-雨”。

3.4.1 Viterbi 变量 $\delta_t(i)$ 的定义

与前向算法类似，但我们不再求和，而是求最大值。

定义：最大概率变量 $\delta_t(i)$

在时刻 t，沿着某条路径到达状态 i，且观测序列吻合的最大概率。

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1, \ldots, q_{t-1}, q_t = i, o_1, \ldots, o_t|\lambda)$$

为了能找回路径，我们还需要一个记录变量 $\psi_t(i)$：记录是哪一个前驱状态让我们到达了当前的最优状态。

3.4.2 算法流程

步骤 1：初始化

$$\delta_1(i) = \pi_i b_i(o_1) \tag{3.6}$$

$$\psi_1(i) = 0 \quad \text{(起始点没有前驱)} \tag{3.7}$$

步骤 2：递推 (求 Max) 对于 $t = 2, \ldots, T$ 和 $j = 1, \ldots, N$：

$$\delta_t(j) = \max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \cdot b_j(o_t) \tag{3.8}$$

同时记录路径（注意：记录路径时不需要乘 $b_j(o_t)$，因为只关心谁跳过来的概率大）：

$$\psi_t(j) = \arg\max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \tag{3.9}$$

步骤 3：终止与回溯 (Backtracking) 先找到终点最大的那个概率：

$$P^* = \max_{1\le i\le N} \delta_T(i)$$

$$q_T^* = \arg\max_{1\le i\le N} \delta_T(i)$$

然后沿着 ψ 数组倒着往回找：

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \ldots, 1$$

3.4.3 Viterbi 算法伪代码

```
def viterbi_algorithm(O, pi, A, B):
    # delta 存概率值, psi 存路径索引
    delta = zeros(T, N)
    psi = zeros(T, N)

    # 1. 初始化
    for i in range(N):
        delta[0][i] = pi[i] * B[i][O[0]]
        psi[0][i] = 0

    # 2. 递推
    for t in range(1, T):
        for j in range(N):
            # 找前一时刻哪个 i 跳过来概率最大
            max_val = -1
            max_idx = -1
            for i in range(N):
                p = delta[t-1][i] * A[i][j]
                if p > max_val:
                    max_val = p
                    max_idx = i

            delta[t][j] = max_val * B[j][O[t]]
            psi[t][j] = max_idx # 记住: 我是从 i 来的

    # 3. 回溯
    path = zeros(T)
    path[T-1] = argmax(delta[T-1]) # 终点最佳状态

    for t in range(T-2, -1, -1):
        # 既然下一时刻是 path[t+1], 那一时刻是从哪来的? 查 psi
        path[t] = psi[t+1][ path[t+1] ]

    return path
```

3.4.4 Viterbi 算法——找最可能的天气路径

Viterbi 和前向算法类似，但我们不累加概率，而是 **取最大值**，找到最可能的隐藏状态序列。

例子：Viterbi 算法——吃冰淇淋问题详解

延续前面的例子，观测序列：

$$O = (\text{冰淇淋}, \text{冰淇淋}, \text{没吃})$$

隐藏状态：

$$S = \{\text{晴天 (Sunny)}, \text{雨天 (Rainy)}\}$$

模型参数同前向算法：

$$\pi = (0.6, 0.4), \quad A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}, \quad B = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

—

Viterbi 变量定义：

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1, \ldots, q_{t-1}, q_t = i, o_1, \ldots, o_t|\lambda)$$

意思：前 t 天观测到 $O_1, \ldots, O_t$，且第 t 天是状态 i 的最大概率。
同时记录路径：

$$\psi_t(i) = \arg\max_j[\delta_{t-1}(j) \cdot a_{j,i}]$$

—

第一天 (t=1)：初始化

$$\delta_1(\text{晴}) = \pi_{\text{晴}} \cdot b_{\text{晴}}(o_1) = 0.6 \cdot 0.9 = 0.54$$

$$\delta_1(\text{雨}) = \pi_{\text{雨}} \cdot b_{\text{雨}}(o_1) = 0.4 \cdot 0.2 = 0.08$$

路径：

$$\psi_1(\text{晴}) = 0, \quad \psi_1(\text{雨}) = 0 \quad (\text{第一天没有前驱})$$

—

第二天 (t=2)：
计算每个状态的最大概率：

$$\delta_2(\text{晴}) = \max(\delta_1(\text{晴}) \cdot a_{\text{晴, 晴}},\ \delta_1(\text{雨}) \cdot a_{\text{雨, 晴}}) \cdot b_{\text{晴}}(o_2)$$

代入数值：

$$\delta_2(\text{晴}) = \max(0.54 \cdot 0.7,\ 0.08 \cdot 0.4) \cdot 0.9 = \max(0.378, 0.032) \cdot 0.9 = 0.3402$$

$$\psi_2(\text{晴}) = \text{从晴天来的} \quad (\text{因为 } 0.378 > 0.032)$$

—

$$\delta_2(\text{雨}) = \max(0.54 \cdot 0.3,\ 0.08 \cdot 0.6) \cdot 0.2 = \max(0.162, 0.048) \cdot 0.2 = 0.0324$$

$$\psi_2(\text{雨}) = \text{从晴天来的} \quad (\text{因为 } 0.162 > 0.048)$$

解释：- $\delta_2(\text{晴}) = 0.3402$：前两天观测到冰淇淋，第二天是晴天的最大概率路径。- $\delta_2(\text{雨}) = 0.0324$：前两天观测到冰淇淋，第二天是雨天的最大概率路径。- $\psi_2(i)$：记录哪一个前一天的状态让我们到达今天状态概率最大。

—

第三天 (t=3)：观察 $o_3 =$ 没吃

$$\delta_3(\text{晴}) = \max(\delta_2(\text{晴}) \cdot a_{\text{晴, 晴}},\ \delta_2(\text{雨}) \cdot a_{\text{雨, 晴}}) \cdot b_{\text{晴}}(o_3)$$

$$\delta_3(\text{晴}) = \max(0.3402 \cdot 0.7,\ 0.0324 \cdot 0.4) \cdot 0.1 = \max(0.23814, 0.01296) \cdot 0.1 = 0.023814$$

$$\psi_3(\text{晴}) = \text{从晴天来的} \quad (\text{因为 } 0.23814 > 0.01296)$$

—

$$\delta_3(\text{雨}) = \max(0.3402 \cdot 0.3,\ 0.0324 \cdot 0.6) \cdot 0.8 = \max(0.10206, 0.01944) \cdot 0.8 = 0.081648$$

$$\psi_3(\text{雨}) = \text{从晴天来的} \quad (\text{因为 } 0.10206 > 0.01944)$$

—

终止：找到最大概率的最后一天状态

$$P^* = \max(\delta_3(\text{晴}), \delta_3(\text{雨})) = \max(0.023814, 0.081648) = 0.081648$$

$$q_3^* = \text{雨天} \quad (\text{概率最大})$$

—

回溯找最优路径：

$$q_3^* = \text{雨天}$$

$$q_2^* = \psi_3(q_3^*) = \psi_3(\text{雨}) = \text{晴天}$$

$$q_1^* = \psi_2(q_2^*) = \psi_2(\text{晴}) = \text{晴天}$$

—

最优隐藏状态序列：

$$Q^* = (\text{晴天}, \text{晴天}, \text{雨天})$$

—

小结：

- Viterbi 算法求的是观测序列最可能对应的隐藏状态序列，而不是所有路径的总概率。
- $\delta_t(i)$ 表示到第 t 天，状态为 i 的最大概率路径。
- $\psi_t(i)$ 记录了每一步从哪来的路径，便于最后回溯。
- 前向算法是“概率求和”，Viterbi 是“概率取最大”，两者核心思想类似。

3.5 Forward-Backward 算法：上帝视角下的概率推断

3.5.1 为什么要引入 Forward-Backward？

在前两节中，我们已经拥有了两个强大的工具：

1. **前向算法 (Forward)**：算出观测序列出现的总概率 $P(O|\lambda)$。
2. **维特比算法 (Viterbi)**：算出一条**最可能**的隐藏路径。

痛点：维特比算法太“武断”了。它只给出一条硬路径（Hard Decoding）。但如果我们想问：“在第 3 天，虽然最可能的路径说是‘雨天’，但其实是‘晴天’的概率有多大?”维特比回答不了这个问题。

解决：我们需要结合**过去**的信息和**未来**的信息，站在“上帝视角”（全序列已知）来评估每一个时刻的状态概率。这就是 Forward-Backward 算法。

3.5.2 两个核心变量：α 与 β

为了评估时刻 t 的状态，我们需要把时间轴一分为二：

定义：1. 前向变量 $\alpha_t(i)$ (The Past)

定义：在时刻 t，观测到了序列 $(o_1, \dots, o_t)$，且当前状态恰好是 i 的概率。

$$\alpha_t(i) = P(o_1, \dots, o_t, q_t = i|\lambda)$$

人话：从开始走到现在，经历了前面的事，最终停在状态 i 的概率。

定义：2. 后向变量 $\beta_t(i)$ (The Future)

定义：在时刻 t 状态为 i 的**前提下**，未来能够观测到序列 $(o_{t+1}, \dots, o_T)$ 的概率。

$$\beta_t(i) = P(o_{t+1}, \dots, o_T|q_t = i, \lambda)$$

人话：既然现在处于状态 i，那么要想看到未来那一串特定的结局，概率有多大？

递推逻辑（反向计算）：

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j) \tag{3.10}$$

解释：从时刻 t 的状态 i，转移到下一时刻所有可能的状态 j (a_{ij})，发射出下一时刻的观测 o_{t+1} (b_j)，然后接上 j 之后的未来 (β_{t+1})。

3.5.3 融合：平滑概率 $\gamma_t(i)$

现在，我们在时刻 t 将“历史”(α) 和“未来”(β) 在状态 i 处汇合。

$$P(q_t = i, O|\lambda) = \alpha_t(i) \times \beta_t(i)$$

但这算出来的是联合概率。我们需要的是条件概率（在已知观测 O 的情况下，时刻 t 是状态 i 的概率）：

$$\gamma_t(i) = P(q_t = i|O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{3.11}$$

- **分子**：经过状态 i 生成整个观测序列的概率路径总和。
- **分母**：生成整个观测序列的总概率（归一化因子，等于 $P(O|\lambda)$）。

3.5.4 算法应用与小结

- **软解码 (Soft Decoding)**：与维特比的“非黑即白”不同，$\gamma_t(i)$ 告诉我们“这一天有 70% 概率是晴，30% 是雨”。这在很多需要置信度的场景下非常重要。
- **模型学习 (Baum-Welch 算法)**：这是 Forward-Backward 最大的用途。

  > 如果我们只有观测数据（没有标注），如何训练 HMM 的参数？我们可以先随机猜一组参数，算出 $\gamma_t(i)$（即“这一天是晴天的期望”），然后用这个期望值去重新估计参数（比如如果 γ 显示某天大概率是晴天，而那天朋友吃了冰淇淋，我们就增加“晴天 $\rightarrow$ 吃”的概率）。这就是 EM 算法在 HMM 中的应用。

3.5.5 生动理解

- Forward：从左到右，把概率积攒起来。- Backward：从右到左，把后面概率积攒起来。- Forward-Backward：两边结合，告诉我们每个隐藏状态在序列中的概率。

Chapter 4

K-means 聚类与 EM 算法

4.1 K-means 聚类：基于原型的硬聚类

4.1.1 核心思想与目标函数

K-means 是一种原型聚类方法。它假设聚类结构可以通过一组原型（Prototypes，即簇中心）$\{\mu_1, \ldots, \mu_K\}$ 来刻画。算法的核心目标是最小化所有样本点到其所属簇中心的平方欧几里得距离之和。这个目标函数被称为畸变函数 (Distortion Function)：

$$J(r, \mu) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 \tag{4.1}$$

其中 r_{nk} 是指示变量 (Indicator Variable)，表示第 n 个样本是否属于第 k 个簇：

$$r_{nk} = \begin{cases} 1, & \text{如果} x_n \text{ 被分配给簇} k \\ 0, & \text{否则} \end{cases}$$

4.1.2 坐标下降法视角 (Coordinate Descent)

K-means 算法的迭代过程实际上是使用坐标下降法优化 J 的过程：

1. **固定 μ，优化 r (分配步)：**

   为了使 J 最小，对于每一个 x_n，我们必须选择能让 $\|x_n - \mu_k\|^2$ 最小的那个 k：

   $$r_{nk} = \begin{cases} 1, & \text{如果} k = \arg\min_j \|x_n - \mu_j\|^2 \\ 0, & \text{否则} \end{cases}$$

   这被称为硬分配 (Hard Assignment)，因为一个点要么完全属于簇 A，要么完全不属于。

2. **固定 r，优化 μ (更新步)：**

   J 是关于 μ_k 的二次函数。对 μ_k 求导并令其为 0：

$$\frac{\partial J}{\partial \mu_k} = 2\sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0$$

   解得：

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

   物理意义：新的簇中心 μ_k 恰好是簇内所有点的几何重心 (Mean)。

4.2 无监督学习：EM 算法详解

4.2.1 问题背景：含隐变量的参数估计

在现实世界中，很多数据生成过程包含我们无法直接观测到的变量，称为隐变量 (Latent Variables) Z。

- 观测数据：X（如：大家的身高）
- 隐变量：Z（如：性别。如果你只知道身高数据，不知道这人是男是女，性别就是隐变量）
- 参数：θ（如：男生身高的分布参数 μ_1, σ_1，女生的 μ_2, σ_2）

如果 Z 已知，我们可以直接用极大似然估计 (MLE) 求 θ。但在 Z 未知的情况下，对数似然函数

$$L(\theta) = \log P(X|\theta) = \log \sum_{Z} P(X, Z|\theta)$$

变得非常复杂（对数里面有求和号，很难求导）。

4.2.2 通俗理解：K-Means 的升级版

EM 算法可以看作是 K-Means 聚类算法的软化 (Soft) 版本。

- K-Means (硬聚类)：一个点要么属于 A 类，要么属于 B 类（0 或 1）。
- EM (软聚类)：一个点有 70% 的概率属于 A 类，30% 的概率属于 B 类。

4.2.3 EM 算法的核心流程

目标：最大化对数似然函数 $\mathcal{L}(\theta) = \log P(X|\theta)$。通过引入隐变量 Z 的分布 $Q(Z)$，不断优化下界（ELBO）。

通俗讲解：EM 算法的两步迭代

1. E 步 (Expectation) —— “猜标签”：固定当前的参数 θ_{old}，计算每个数据点属于各个隐状态的后验概率（即隐变量 Z 的期望）。

$$Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)};\theta_{old})$$

直觉：根据现在的身高参数，算出小明是男生的概率是 *0.8*。

2. M 步 (Maximization) —— “更参数”：利用 E 步算出的概率（权重），重新计算参数 θ_{new}，使得期望似然最大化。

$$\theta_{new} = \arg\max_{\theta} \sum_{i} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}$$

直觉：把小明当成 *0.8* 个男生和 *0.2* 个女生，重新计算男女生的平均身高。

4.2.4 算法核心：Q 函数

EM 算法不直接最大化复杂的 $L(\theta)$，而是通过迭代构建一个下界（Lower Bound）来逼近。

E-step (期望步)：我们来“猜”隐变量基于当前的参数估计 $\theta^{(t)}$，计算隐变量 Z 的后验概率 $P(Z|X,\theta^{(t)})$。然后计算完整数据对数似然的期望，记为 Q 函数：

$$Q(\theta,\theta^{(t)}) = \sum_{Z} P(Z|X,\theta^{(t)}) \log P(X,Z|\theta) \tag{4.2}$$

直观理解：既然不知道 Z 到底是多少，我们就用在当前参数下 Z 出现的概率，对“假如已知 Z 时的似然函数”进行加权平均。

M-step (最大化步)：我们来“修”参数找到能让这个期望似然 Q 最大的参数 θ，作为下一轮的参数：

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta,\theta^{(t)})$$

4.2.5 典型应用：高斯混合模型 (GMM)

考试中常以 GMM 为例考察 EM。假设数据是由 K 个高斯分布混合而成的。

- 隐变量：第 i 个数据具体属于哪一个高斯分布。
- 参数 θ：每个高斯分布的均值 μ_k、协方差 Σ_k 以及混合系数 π_k。

GMM 的 M 步公式（背诵）：新的均值 μ_k 等于所有数据点的加权平均，权重就是该点属于第 k 类的概率：

$$\mu_k^{new} = \frac{\sum_{i=1}^{N} \gamma(z_{ik}) x_i}{\sum_{i=1}^{N} \gamma(z_{ik})} \tag{4.3}$$

4.3 深度解析：K-means 是 EM 的特例

这是理解聚类算法本质的关键。K-means 实际上是高斯混合模型 (GMM) 在极限条件下的 EM 算法实现。

4.3.1 设定：从 GMM 退化

我们定义一个受限的高斯混合模型 (GMM)：

- 数据生成：有 K 个高斯分布组件。
- 协方差矩阵固定:假设所有高斯组件的协方差矩阵都是球面状且相等的,即 $\Sigma_k = \epsilon I$（ϵ 是方差）。
- 先验概率相等：每个簇被选中的概率相等，即 $\pi_k = \frac{1}{K}$。

此时，混合模型中观测到数据点 x 的概率密度为：

$$P(x|\mu, \epsilon) = \sum_{k=1}^{K} \frac{1}{K} \frac{1}{(2\pi\epsilon)^{D/2}} \exp\left(-\frac{1}{2\epsilon}\|x - \mu_k\|^2\right)$$

4.3.2 推导 E-step：从软分配到硬分配

在 EM 的 E-step 中，我们需要计算样本 x_n 属于簇 k 的后验概率（即责任度 γ_{nk}）：

$$\gamma_{nk} = P(z_{nk} = 1|x_n) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \epsilon I)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \epsilon I)}$$

代入我们简化的假设，约去常数项，得到：

$$\gamma_{nk} = \frac{\exp\left(-\frac{1}{2\epsilon}\|x_n - \mu_k\|^2\right)}{\sum_{j=1}^{K} \exp\left(-\frac{1}{2\epsilon}\|x_n - \mu_j\|^2\right)}$$

关键极限：当 $\epsilon \to 0$ 时会发生什么？

- 对于距离 x_n 最近的那个中心 μ_k，$\|x_n - \mu_k\|^2$ 最小，其指数项在分母中占绝对主导地位。

- 对于其他中心，指数项相对于最近的那个中心会趋近于 0。

因此，当 $\epsilon \to 0$：

$$\gamma_{nk} \to \begin{cases} 1, & \text{如果} k = \arg\min_j \|x_n - \mu_j\|^2 \\ 0, & \text{其他} \end{cases}$$

结论：GMM 的后验概率（Soft Assignment）退化成了 K-means 的指示变量 r_{nk}（Hard Assignment）。

4.3.3 推导 M-step：加权平均的简化

在 EM 的 M-step 中，GMM 更新均值的公式是加权平均：

$$\mu_k^{\text{new}} = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}$$

由于在 $\epsilon \to 0$ 的极限下，γ_{nk} 变成了非 0 即 1 的 r_{nk}，公式变为：

$$\mu_k^{\text{new}} = \frac{\sum_{n \in C_k} x_n}{|C_k|} = \frac{1}{|C_k|} \sum_{x_n \in C_k} x_n$$

结论：这正是 K-means 的“更新中心”步骤（算术平均）。

4.3.4 总结：两者的区别与联系表

维度	K-means	EM (用于 GMM)
分配方式	硬分配 (Hard)：非此即彼	软分配 (Soft)：概率属于
模型假设	数据呈球状分布，方差相等且极小	可适应椭圆分布，方差可不同
隐变量	离散指示变量 (0/1)	连续概率 (0 到 1 之间)
优化目标	最小化欧氏距离平方和 (Distortion)	最大化对数似然 (Log-Likelihood)
计算复杂度	较轻，收敛快	较重，计算指数和协方差矩阵

表 4.1: K-means 与 EM 的区别与联系

—

4.4 EM 算法一般推导

4.4.1 问题形式：为什么很难直接求导？

假设我们要优化模型参数 θ,以最大化观测数据 X 的对数似然函数 $L(\theta) = \log P(X|\theta)$。

- 完整数据情况：如果数据包含观测变量 X 和隐变量 Z，即 (X, Z)，则目标是 $\log P(X, Z|\theta)$。这是很容易优化的。
- 不完整数据情况：我们只有 X，隐变量 Z 未知。我们需要对 Z 进行边缘化（求和或积分）：

$$L(\theta) = \log P(X|\theta) = \log \sum_Z P(X, Z|\theta)$$

困难点：对数函数 log 里面包含了一个求和符号 $\sum$。这导致求导时，各项参数紧密耦合在一起，无法得到解析解。EM 算法的核心思想就是：想办法把求和号 $\sum$ 挪到 log 的外面去。

4.5 EM 算法一般推导

4.5.1 问题形式：为什么很难直接求导？

假设我们要优化模型参数 θ,以最大化观测数据 X 的对数似然函数 $L(\theta) = \log P(X|\theta)$。

- 完整数据情况：如果数据包含观测变量 X 和隐变量 Z，即 (X, Z)，则目标是 $\log P(X, Z|\theta)$。这是很容易优化的。
- 不完整数据情况：我们只有 X，隐变量 Z 未知。我们需要对 Z 进行边缘化（求和或积分）：

$$L(\theta) = \log P(X|\theta) = \log \sum_Z P(X, Z|\theta)$$

困难点：对数函数 log 里面包含了一个求和符号 $\sum$。这导致求导时，各项参数紧密耦合在一起，无法得到解析解。EM 算法的核心思想就是：想办法把求和号 $\sum$ 挪到 log 的外面去。

4.5.2 ELBO 与 Jensen 不等式

为了交换 log 和 $\sum$ 的顺序，我们需要引入 Jensen 不等式。

1. Jensen 不等式 (Jensen's Inequality)

如果函数 $f(x)$ 是凹函数 (Concave)（例如 $f(x) = \log x$，二阶导数 < 0），那么：**“期望的函数大于等于函数的期望”**

$$f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$$

即：

$$\log\left(\sum_i p_i x_i\right) \geq \sum_i p_i \log(x_i)$$

其中 p_i 是概率分布（和为 1）。

2. 推导证据下界 (ELBO)

为了凑出 Jensen 不等式的形式，我们引入一个关于隐变量 Z 的任意分布 $Q(Z)$（满足 $\sum_Z Q(Z) = 1$）。对目标函数进行变换：

$$L(\theta) = \log \sum_Z P(X, Z|\theta) \tag{4.4}$$

$$= \log \sum_Z Q(Z) \frac{P(X, Z|\theta)}{Q(Z)} \quad \text{（分子分母同乘 } Q(Z)\text{）} \tag{4.5}$$

$$= \log \mathbb{E}_{Z\sim Q} \left[\frac{P(X, Z|\theta)}{Q(Z)} \right] \tag{4.6}$$

根据 Jensen 不等式（把 log 塞进期望里面）：

$$\log \sum_Z Q(Z) \frac{P(X, Z|\theta)}{Q(Z)} \geq \sum_Z Q(Z) \log \frac{P(X, Z|\theta)}{Q(Z)} \tag{4.7}$$

公式右边被称为 ELBO (Evidence Lower Bound)。

$$\text{ELBO}(Q, \theta) = \sum_Z Q(Z) \log P(X, Z|\theta) - \sum_Z Q(Z) \log Q(Z)$$

EM 算法的逻辑是：既然无法直接最大化 $L(\theta)$，我们就不断最大化它的下界 ELBO。

4.5.3 EM 两步的数学解释

EM 算法本质上是坐标上升法 (Coordinate Ascent)：

- E 步：固定 θ，调整 $Q(Z)$ 使得下界 ELBO 尽可能贴近 $L(\theta)$（即使等号成立）。
- M 步：固定 $Q(Z)$，调整 θ 使得 ELBO 最大化。

1. E 步：如何使等号成立？

Jensen 不等式取等号的条件是：随机变量是常数。即 $\frac{P(X,Z|\theta)}{Q(Z)} = c$。这推导出 $Q(Z)$ 必须正比于 $P(X, Z|\theta)$。考虑到 $Q(Z)$ 是概率分布，其最优解正是后验概率：

$$Q^*(Z) = P(Z|X, \theta^{old})$$

此时 ELBO 就等于 $L(\theta)$。

2. M 步：最大化期望

将 $Q^*(Z)$ 代入 ELBO，去掉对 θ 求导为 0 的项（即含 $Q(Z) \log Q(Z)$ 的项），M 步的目标函数变为：

$$\theta^{new} = \arg\max_\theta \sum_Z P(Z|X, \theta^{old}) \log P(X, Z|\theta)$$

这就是我们在讲义前面看到的 $Q(\theta, \theta^{old})$ 函数。

4.6 HMM 的 EM: Baum-Welch 更新公式

隐马尔可夫模型 (HMM) 的参数学习通常使用 Baum-Welch 算法，它实际上就是 EM 算法在 HMM 中的具体实现。

4.6.1 HMM 参数定义

我们需要学习 HMM 的三组参数 $\lambda = (A, B, \pi)$：

- π：初始状态概率向量，$\pi_i = P(z_1 = i)$。
- A：状态转移矩阵，$a_{ij} = P(z_{t+1} = j | z_t = i)$。
- B：发射概率矩阵，$b_j(k) = P(x_t = k | z_t = j)$（状态 j 产生观测 k 的概率）。

4.6.2 E 步：计算两个关键统计量

为了进行 M 步更新，我们需要先利用前向-后向算法 (Forward-Backward) 算出两个中间变量：γ 和 ξ。

1. $\gamma_t(i)$：时刻 t 处于状态 i 的概率

给定观测序列 O 和参数 λ，在时刻 t 隐状态为 i 的概率：

$$\gamma_t(i) = P(z_t = i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

其中 α 是前向概率，β 是后向概率。

2. $\xi_t(i, j)$：时刻 t 在 i 且 $t+1$ 在 j 的概率

这是为了更新转移矩阵 A 用的。

$$\xi_t(i, j) = P(z_t = i, z_{t+1} = j | O, \lambda) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}$$

4.6.3 M 步：参数更新公式 (背诵)

本质是“加权平均”（频数估计概率）。

1. 更新初始状态 π

直接看 $t = 1$ 时刻处于各状态的概率：

$$\bar{\pi}_i = \gamma_1(i)$$

2. 更新转移矩阵 A

$$\bar{a}_{ij} = \frac{\text{从}i\text{ 转移到}j\text{ 的期望次数}}{\text{从}i\text{ 出发的总期望次数}}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

3. 更新发射矩阵 B

$$\bar{b}_j(k) = \frac{\text{状态为}j\text{ 且观测为}k\text{ 的期望次数}}{\text{状态为}j\text{ 的总期望次数}}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1,o_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

注意：分子中的求和 $\sum_{t=1,o_t=k}^{T}$ 意思是只累加那些观测值确实等于 k 的时刻。

Chapter 5

机器学习基础：线性回归与逻辑回归

5.1 机器学习概览

5.1.1 机器学习的定义

- **Arthur Samuel (1959) 定义**：机器学习是“在不直接编程的情况下，赋予计算机学习能力的研究领域”。
- **Tom Mitchell (1998) 定义**：一个计算机程序从经验 E 中学习任务 T，并用性能度量 P 来衡量。如果随着经验 E 的增加，在任务 T 上的性能 P 提高了，就说它“学习”了。

5.1.2 机器学习的核心任务

机器学习的核心任务是从数据中寻找规律（函数 f）。根据反馈信号的不同，主要分为三类：

- **监督学习 (Supervised Learning)**：老师教你。数据有标签 (x, y)。目标是学映射 $y = f(x)$。（如：分类、回归）
- **无监督学习 (Unsupervised Learning)**：自学。数据无标签 x。目标是发现数据内部结构。（如：聚类、降维）
- **强化学习 (Reinforcement Learning)**：从教训中学。数据是“状态-动作-奖励”序列。目标是长期收益最大化。

5.1.3 基本术语

- **样本 (Instance/Sample)**：一条数据记录，例如“这只西瓜的记录”。
- **特征 (Feature/Attribute)**：描述样本的属性，例如“色泽、根蒂、敲声”。

- **标记 (Label)**：我们想预测的结果，例如“好瓜/坏瓜”或“价格”。
- **特征空间 (Feature Space)**：所有样本存在的空间。
- **假设空间 (Hypothesis Space)**：模型可能学到的所有规则的集合。学习过程就是在假设空间里搜索最符合数据的那个假设。

5.1.4 机器学习三要素

1. **模型 (Model)**：我们要学习的函数形式，例如线性模型 $f(x) = wx + b$。
2. **策略 (Strategy)**：评判模型好坏的方法，通过损失函数 (Loss Function)。目标是最小化损失（经验风险最小化 + 结构风险最小化/正则化）。
3. **算法 (Algorithm)**：如何找到最优参数，通常使用梯度下降 (Gradient Descent)。

5.2 线性回归 (Linear Regression)

线性回归是最经典的回归模型之一，用于预测连续值。它假设预测变量 x 与输出变量 y 之间存在线性关系。

5.2.1 模型定义

定义：线性回归模型

- **目标**：学习一个函数 $f : \mathbb{R}^D \to \mathbb{R}$，使得给定输入特征向量 $x \in \mathbb{R}^D$，预测输出 $\hat{y}$ 尽可能接近真实值 y。
- **模型形式**：

$$f(x) = w^T x + b = \sum_{j=1}^{D} w_j x_j + b$$

- **参数说明**：
  - $w = (w_1, \ldots, w_D)^T$ 是权重向量，表示各特征的重要性。
  - $b \in \mathbb{R}$ 是偏置项（Intercept），允许回归直线不经过原点。
- **假设**：输出变量与特征之间是线性关系。

在线性回归中，我们的任务就是找到最优的参数 w 和 b，使模型预测尽量准确。

5.2.2 损失函数：均方误差 (MSE)

定义：均方误差 (Mean Squared Error)

$$L(w,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w^T x_i + b))^2$$

- N：样本总数。
- y_i：第 i 个样本的真实值。
- $f(x_i)$：第 i 个样本的预测值。

损失函数衡量了预测值与真实值的差距，我们的目标是最小化 $L(w,b)$。

5.2.3 求解参数的方法

线性回归主要有两种求解方法：解析解和梯度下降。

解析解 (Normal Equation)

对于小规模数据，可以直接通过矩阵运算求出最优参数：

定义：解析解公式

$$w^* = (X^T X)^{-1} X^T y$$

其中 $X \in \mathbb{R}^{N\times D}$ 是样本特征矩阵，每行是样本特征向量；$y \in \mathbb{R}^N$ 是目标向量。
偏置 b 可以通过在 X 中添加一列全 1 来一并求解。
优缺点：

- 优点：直接求解，无需迭代，精确。
- 缺点：当 D 或 N 很大时，$X^T X$ 求逆计算量大，且可能不可逆（需要正则化）。

梯度下降 (Gradient Descent)

当数据量较大或者无法直接求逆时，可以使用迭代优化方法：

$$w \leftarrow w - \eta\frac{\partial L}{\partial w}, \quad b \leftarrow b - \eta\frac{\partial L}{\partial b}$$

其中 η 是学习率。
损失函数对参数的梯度为：

$$\frac{\partial L}{\partial w} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - w^T x_i - b)x_i, \quad \frac{\partial L}{\partial b} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - w^T x_i - b)$$

梯度下降通过不断沿负梯度方向更新参数，使损失函数逐步减小，最终收敛到最优值或局部最优。

5.2.4 线性回归小结

- 假设输入与输出之间存在线性关系。
- 均方误差是最常用的损失函数。
- 参数求解可以通过解析解或梯度下降。
- 偏置 b 和权重 w 是模型学习的核心。

5.3 逻辑回归 (Logistic Regression)

逻辑回归是用于分类任务的基础模型，特别是二分类（0/1，正类/负类，或好/坏）。它通过对线性组合的特征进行非线性映射，将输出压缩到概率范围 $[0, 1]$，从而实现分类。

5.3.1 从回归到分类

线性回归输出连续数值，不适合直接用于分类。分类任务需要输出一个概率：

定义：逻辑回归模型

- **模型公式：**

$$P(y=1|x)=\sigma(z)=\sigma(w^Tx+b)=\frac{1}{1+e^{-(w^Tx+b)}}$$

- **符号说明：**
  - $x\in\mathbb{R}^D$：输入特征向量。
  - $w\in\mathbb{R}^D$：权重向量。
  - $b\in\mathbb{R}$：偏置项。
  - $z=w^Tx+b$：线性组合。
  - $\sigma(z)$：Sigmoid 激活函数，将线性输出映射到 $(0,1)$。
  - $y\in\{0,1\}$：二分类标签。

输出 $P(y=1|x)$ 可以解释为样本属于正类的概率，预测规则通常是：

$$\hat{y}=\begin{cases}1, & P(y=1|x)\geq 0.5\\ 0, & P(y=1|x)<0.5\end{cases}$$

5.3.2 常用激活函数

激活函数在分类模型中用于引入非线性映射，将模型输出压缩到指定区间：

定义：常用激活函数

- **Sigmoid：**

$$\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1)$$

将线性输出映射到概率。

- **Tanh (双曲正切)：**

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \in (-1,1)$$

对输入进行中心化映射，常用于隐藏层。

- **ReLU (Rectified Linear Unit)：**

$$\mathrm{ReLU}(z) = \max(0,z)$$

简单有效，解决梯度消失问题，常用于深度神经网络。

- **Softmax (多分类)：**

$$\mathrm{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1,\dots,K$$

将多维输出转化为概率分布，$\sum_i \mathrm{Softmax}(z_i) = 1$。

5.3.3 损失函数：对数似然 / 交叉熵

逻辑回归采用对数似然损失函数，又称二分类交叉熵损失：

定义：逻辑回归损失函数

$$L(w,b) = -\sum_{i=1}^{N} \left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i) \right], \quad \hat{y}_i = \sigma(w^T x_i + b)$$

- 最大化似然等价于最小化负对数似然。
- N：样本数。
- y_i：真实标签。
- $\hat{y}_i$：预测概率。

梯度公式

逻辑回归损失函数可用梯度下降求解参数：

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{N} (\hat{y}_i - y_i) x_i, \quad \frac{\partial L}{\partial b} = \sum_{i=1}^{N} (\hat{y}_i - y_i)$$

参数更新公式：

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

其中 η 是学习率。

5.3.4 正则化：防止过拟合

为了避免模型过拟合，可以在损失函数中加入正则化项：

定义：逻辑回归正则化

- **L2 正则化 (Ridge)：**

  $$L_{\text{reg}} = L + \lambda \|w\|_2^2$$

  惩罚权重过大，使模型更平滑。

- **L1 正则化 (Lasso)：**

  $$L_{\text{reg}} = L + \lambda \|w\|_1$$

  可产生稀疏权重，有特征选择效果。

- **符号说明：**

  - $\lambda > 0$：正则化强度，λ 越大，惩罚越强。
  - $\|w\|_1 = \sum_j |w_j|$, $\|w\|_2^2 = \sum_j w_j^2$

5.3.5 逻辑回归小结

- 将线性输出通过 Sigmoid 映射为概率，适合二分类任务。
- 损失函数采用对数似然或交叉熵。
- 参数优化可使用梯度下降。
- 可通过正则化防止过拟合，提升泛化能力。
- 激活函数（Sigmoid/Tanh/ReLU/Softmax）提供非线性映射，可扩展到多分类或深度网络。

Chapter 6

强化学习

6.0.1 基本概念与 MDP 五元组

强化学习描述的是智能体 (Agent) 在环境 (Environment) 中通过试错来学习策略的过程。数学上描述为马尔可夫决策过程 (MDP)：$\langle S, A, P, R, \gamma \rangle$。

定义：MDP 核心要素

1. S (State)：状态空间（机器人在哪）。
2. A (Action)：动作空间（往哪走）。
3. P (Transition Probability)：$P(s'|s,a)$。在 s 做 a，有概率变成 s'。（环境的不确定性，如路滑摔倒）。
4. R (Reward)：$R(s,a)$。环境给的立即反馈（糖果或鞭子）。
5. γ (Discount Factor)：$\gamma \in [0,1]$。折扣因子。体现远视程度，$\gamma = 0$ 鼠目寸光，$\gamma \to 1$ 高瞻远瞩。

定义：策略 π

策略（Policy）π 定义为在状态空间 $\mathcal{S}$ 上，对动作空间 $\mathcal{A}$ 的一个条件概率分布：

$$\pi(a \mid s) \triangleq \Pr(A_t = a \mid S_t = s), \quad \forall s \in \mathcal{S},\ a \in \mathcal{A}.$$

并且满足概率分布的基本性质：

$$\pi(a \mid s) \geq 0, \quad \sum_{a \in \mathcal{A}} \pi(a \mid s) = 1, \quad \forall s \in \mathcal{S}.$$

其中，$\pi(a \mid s)$ 表示在状态 s 下选择动作 a 的概率。

6.0.2 回报 (Return) 与价值函数 (Value Function)

目标：最大化长期累积回报 G_t：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

为了评估“现在的状态好不好”，我们定义了两个核心函数：

1. 状态价值函数 $V_\pi(s)$

在状态 s，如果你按照策略 π 继续玩下去，平均能拿多少分？

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \tag{6.1}$$

2. 动作价值函数 $Q_\pi(s,a)$ ——(Q-Learning 的核心)

在状态 s，你非要执行动作 a（哪怕它不是策略推荐的），然后之后再按策略 π 玩，平均能拿多少分？

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{6.2}$$

6.0.3 贝尔曼方程 (Bellman Equation)

这是 RL 的递归灵魂：**当前价值 = 立即奖励 + 折扣后的未来价值**。

1. 贝尔曼期望方程（评估现有策略）

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \underbrace{\left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_\pi(s') \right]}_{\text{执行动作}a\text{ 的期望回报}}$$

在上述形式中，奖励函数 $R(s,a)$ 表示在状态 s 下执行动作 a 所获得的即时奖励。

Bellman 期望方程的另一种等价形式是将奖励显式地定义为与状态转移相关的函数，此时方程可写为：

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s' \mid s,a) \left[R(s,a,s') + \gamma V_\pi(s') \right]$$

其中，$R(s,a,s')$ 表示在状态 s 下执行动作 a 并转移到下一状态 s' 时获得的即时奖励。

两种写法的等价性说明：若定义

$$R(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \big[R(s,a,s') \big] = \sum_{s' \in S} P(s'|s,a) R(s,a,s'),$$

则第二种形式可以化简为第一种形式。

因此，这两种 Bellman 期望方程在数学上是等价的，差异仅来源于奖励函数的建模方式：前者将奖励视为 (s, a) 的函数，奖励先对 s' 求了期望，得到求期望后的奖励；后者将奖励视为 (s, a, s') 的函数，表示奖励依赖转移。

2. 贝尔曼最优方程 (寻找最佳策略)

如果我们总是选最好的动作（Greedy），那么最优价值 $V^*(s)$ 满足：

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right] \tag{6.3}$$

这实际上就是说：在这个状态下，我只选那个能让我未来收益最大的动作。

其中

6.1 求解 MDP：有模型 vs. 无模型

- 有模型 (Model-based)：已知 P 和 R（有上帝视角，知道地图）。
- 无模型 (Model-free)：未知 P 和 R（两眼一抹黑，只能亲自去试）。

6.1.1 基于 Monte Carlo 采样的价值函数更新

基本思想 Monte Carlo（MC）方法通过**完整轨迹采样**来估计价值函数，不依赖环境模型，仅使用与环境交互得到的样本序列：

$$(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T)$$

其核心思想是：

> 一次完整回合（*episode*）的真实回报，可以作为当前状态或状态–动作对真实价值的无偏估计。

—

回报（Return）的定义 从时间步 t 开始的累计折扣回报定义为：

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$$

其中 T 表示该回合终止时刻。

—

Monte Carlo 状态价值函数更新 对于给定策略 π，Monte Carlo 方法使用回报 G_t 直接更新状态价值函数：

$$V(s_t) \leftarrow V(s_t) + \alpha\big[G_t - V(s_t)\big]$$

其中：

- G_t 是从状态 s_t 出发的真实累计回报；
- 该更新等价于对 $V(s)$ 进行样本均值估计；
- 不存在 bootstrap（自举），仅依赖真实回报。

—

Monte Carlo 动作价值函数更新 类似地，动作价值函数可按以下方式更新：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[G_t - Q(s_t, a_t)\big]$$

该方法直接利用一次完整轨迹中状态–动作对的真实回报进行更新。

—

Monte Carlo 方法的特点

- **Model-free**：不需要转移概率或奖励模型；
- **无偏估计**：G_t 是真实期望回报的无偏估计；
- **高方差**：单条轨迹波动大，收敛较慢；
- **必须终止**：仅适用于 episodic 任务；
- **延迟更新**：需等回合结束才能更新价值。

6.1.2 动态规划 (DP): 策略迭代与价值迭代

动态规划是解决有模型 (Model-based) MDP 的核心方法。前提是我们已知环境的动力学 $P(s'|s,a)$ 和奖励函数 $R(s,a)$。

策略评估 (Policy Evaluation)

目标：给定一个策略 π，计算该策略下的状态价值函数 $V_\pi(s)$。这是一个解线性方程组的过程，但在计算机中通常使用迭代法：

$$V_{k+1}(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right]$$

随着迭代次数 $k \to \infty$，价值函数收敛到 V_π。

策略提升 (Policy Improvement)

目标：有了 $V_\pi(s)$ 后，如何找到一个更好的策略 π'？方法是贪婪化 (Greedy)。在某个状态 s，直接选择那个能让 $Q_\pi(s,a)$ 最大的动作：

$$\pi'(s) = \arg\max_a Q_\pi(s,a) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_\pi(s') \right]$$

根据策略提升定理，π' 至少不比 π 差。

策略迭代 (Policy Iteration)

将上述两步循环交替进行：

1. 评估：计算当前策略 π 的 V_π（通常需要多次迭代直到收敛）。
2. 提升：根据 V_π 生成新的贪婪策略 π'。

一直重复，直到策略不再改变，此时得到的即为最优策略 π^*。

价值迭代 (Value Iteration)

策略迭代在“评估”步骤需要完全收敛，太慢了。价值迭代将其简化：每评估一步，立刻进行提升（即把 Bellman 最优方程转为更新规则）。

重要提示

价值迭代更新公式

$$V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right]$$

迭代直到 V 值收敛，最后提取贪婪策略即可。

6.1.3 TD 与 SARSA

时序差分学习 TD(0)

提出背景 在基于 Monte Carlo 的价值评估方法中，状态价值函数 $V(s)$ 的更新依赖于一次完整回合的累计回报 G_t。这意味着：

- 必须等到 episode 终止后才能进行更新；
- 不适用于持续性任务（continuing tasks）；
- 回报 G_t 方差较大，学习过程不稳定。

为克服上述问题，时序差分（Temporal Difference, TD）学习方法被提出。

—

核心思想（Temporal Difference） TD 方法的核心思想是：

利用下一时刻的价值估计，来修正当前时刻的价值估计。

即：

用“估计的未来”来更新“当前的估计”，这一过程称为 bootstrap（自举）。

TD 方法结合了：

- Monte Carlo 的 **model-free** 特性；
- Dynamic Programming 的 **bootstrapping** 思想。

—

TD(0) 更新公式 最简单的 TD 方法为 TD(0)，其更新公式为：

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD Target}} - \underbrace{V(S_t)}_{\text{当前估计}}\Big]$$

其中：

- $\alpha \in (0, 1]$ 为学习率；
- TD Target 是基于一步真实交互得到的回报估计；
- 更新方向由 TD Error 决定。

—

TD Error（时序差分误差） 定义 TD Error 为：

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

其物理意义是：

- $\delta_t > 0$：当前状态价值被低估，应上调；
- $\delta_t < 0$：当前状态价值被高估，应下调；
- $\delta_t = 0$：当前估计与一步观测一致。

TD(0) 的更新可简写为：

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t$$

—

TD(0) 的期望形式 对 TD Target 在条件 $S_t = s$ 下取期望，有：

$$\mathbb{E}\big[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s\big] = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)\big[R(s,a,s') + \gamma V(s')\big]$$

该期望形式正是 Bellman 期望方程中的备份操作，说明 TD(0) 可以视为 **对 Bellman 更新的一步随机近似**。

—

TD(0) 的算法流程

1. 初始化价值函数 $V(s)$（任意值）；
2. 按策略 π 与环境交互，得到 (S_t, R_{t+1}, S_{t+1})；
3. 计算 TD Error：
$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
4. 更新价值函数：
$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$
5. 转移到下一状态，重复上述过程。

—

TD(0) 的特点总结

- **Model-free**：不需要环境模型；
- **在线学习**：每一步都可更新；
- **支持持续任务**：无需终止状态；
- **低方差**：相较 MC，更新目标更稳定；
- **有偏估计**：由于 bootstrap，引入偏差。

—

Monte Carlo、TD 与 DP 的统一视角

方法	是否使用模型	更新目标	是否 Bootstrap
DP	是	Bellman 期望	是
MC	否	G_t	否
TD(0)	否	$R_{t+1} + \gamma V(S_{t+1})$	是

TD 方法位于 Monte Carlo 与 Dynamic Programming 之间，在实际强化学习问题中被广泛采用。

SARSA (On-policy)

SARSA 是 TD 思想在控制（Control）问题上的应用。数据序列为：$S_t \to A_t \to R_{t+1} \to S_{t+1} \to A_{t+1}$。

重要提示

SARSA 更新公式

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$$

关键区别：

- Q-Learning (Off-policy)：计算 Target 时使用 $\max_{a'} Q(S',a')$。即：虽然我下一步可能乱走（探索），但我更新时假设我下一步会走得最好（贪婪）。
- SARSA (On-policy)：计算 Target 时使用 $Q(S',A')$。即：我下一步实际走了 A'，我就用 A' 的价值来更新。言行一致。

* 直观理解：SARSA 比较胆小（保守），遇到悬崖会绕着走；Q-Learning 比较大胆，倾向于走最优路径即便离悬崖很近（因为它假设自己不会失误）。*

6.1.4 Q-Learning 算法

核心思想 Q-Learning 是一种 **model-free、off-policy** 的强化学习算法。算法不需要事先知道环境的状态转移概率 $P(s'|s,a)$，而是通过与环境交互得到的样本数据 (s,a,r,s')，直接对动作价值函数 $Q(s,a)$ 进行迭代更新，最终逼近最优动作价值函数 $Q^*(s,a)$。

其核心目标是：

$$Q(s,a) \xrightarrow[\text{迭代}]{} Q^*(s,a)$$

—

重要提示

Q-Learning 更新公式

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{TD Target（现实回报）}} - \underbrace{Q(s,a)}_{\text{Estimate（当前估计）}}\Big]$$

等价写法：

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a')\Big)$$

- $\alpha \in (0,1]$（学习率）：决定新信息对旧估计的覆盖程度。
  - α 大：学得快，但震荡大；
  - α 小：学得稳，但收敛慢。
- TD Target（时序差分目标）：基于**一次真实交互**得到的对当前状态–动作对真实价值的估计，包括即时奖励 r 和下一状态的最优预期回报。
- TD Error（时序差分误差）：

  $$\delta = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

  表示当前估计与现实观测之间的偏差，是驱动学习更新的直接信号。

—

探索与利用（Exploration vs. Exploitation）

在 Q-Learning 训练过程中，智能体需要在每一步决策时回答一个关键问题：

是选择当前看来最优的动作，还是尝试尚不确定的动作？

这正是强化学习中的 **探索–利用困境**。

利用（Exploitation） 利用指在状态 s 下，选择当前 Q 值最大的动作：

$$a = \arg\max_{a} Q(s,a)$$

优点：

- 充分利用已有知识；
- 短期回报最大化。

缺点：

- 可能过早锁定次优策略；
- 无法发现潜在更优动作（局部最优问题）。

—

探索（Exploration） 探索指随机选择动作，以获取新的环境信息。

优点：

- 有机会发现更优策略；
- 提升状态–动作空间的覆盖度。

缺点：

- 短期回报较低；
- 过度探索会降低学习效率。

—

ϵ-Greedy 策略（最常用） ϵ-Greedy 是一种在实践中最常用的探索–利用折中策略，其定义如下：

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a), & \text{以概率} 1-\epsilon \quad \text{(利用)} \\ \text{随机选取} a \in \mathcal{A}, & \text{以概率} \epsilon \quad \text{(探索)} \end{cases}$$

其中 $\epsilon \in [0,1]$ 控制探索强度。

- ϵ 大：探索多，适合训练初期；
- ϵ 小：利用多，适合训练后期。

在实际应用中，常采用 **递减探索策略**：

$$\epsilon_t \downarrow 0 \quad \text{随训练轮数逐渐减小}$$

以保证算法在早期充分探索，在后期稳定收敛到最优策略。

6.2 策略梯度：从目标函数到 REINFORCE

6.2.1 为什么需要策略梯度 (PG)?

Q-Learning 等方法是基于价值的 (Value-based)，最终策略是确定的（看谁 Q 值大）。但 PG 是基于策略的 (Policy-based)，直接参数化策略 $\pi_\theta(a|s)$。

- 可以处理连续动作空间。
- 可以学习随机策略（如石头剪刀布）。

6.2.2 策略梯度定理与 REINFORCE

目标是最大化期望回报 $J(\theta) = \mathbb{E}_{\pi_\theta}[G]$。我们要对 θ 求梯度上升。根据策略梯度定理：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \cdot G_t]$$

这导出了 REINFORCE 算法：

1. 玩一整局游戏，收集轨迹。
2. 对每一步，如果结果 G_t 是好的（正值），就增加这一步动作的概率（即 $\nabla \log \pi$ 方向）；如果是坏的，就减少概率。

6.2.3 基线 (Baseline) 降方差

REINFORCE 采样方差极大。为了让训练更稳定，我们引入一个基线 $b(s)$（通常是状态价值 $V(s)$）：

$$\nabla_\theta J(\theta) \approx \sum (\nabla_\theta \log \pi_\theta(a_t|s_t))(G_t - b(s_t))$$

项 $(G_t - V(s_t))$ 被称为优势函数 (Advantage) A_t。* 直觉：如果拿到的分 G_t 比平均水平 $V(s_t)$ 高，才去鼓励这个动作；否则即便 G_t 是正的但低于平均，也要抑制它。*

6.3 PPO: 剪切目标函数 (重点公式)

PPO (Proximal Policy Optimization) 是目前最主流的 RL 算法（包括 ChatGPT 都在用）。它是 TRPO 的简化版。

6.3.1 重要性采样比率 (Importance Sampling Ratio)

为了重复利用旧策略 π_{old} 产生的数据来更新新策略 π_θ，我们定义比率：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$$

当 $\theta = \theta_{old}$ 时，$r_t = 1$。

6.3.2 PPO-Clip 目标函数

为了防止策略更新步子迈得太大导致崩盘，PPO 强制限制 $r_t(\theta)$ 在 $[1-\epsilon, 1+\epsilon]$ 之间。

重要提示

PPO Clip 目标 (背诵)

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

- $\hat{A}_t > 0$ (动作好)：限制上限。如果 r_t 太大（新策略在这个动作上的概率提升太多），就被 clip 住，防止过拟合。
- $\hat{A}_t < 0$ (动作差)：限制下限。如果 r_t 太小（概率下降太多），也 clip 住。

6.3.3 GAE (常用优势估计)

Generalized Advantage Estimation (GAE) 是一种计算优势 A_t 的方法，用于平衡偏差和方差。它利用了 λ 参数在 TD 误差和 MC 误差之间做折中。

6.4 RLHF (对齐训练) 常用数学形式

RLHF (Reinforcement Learning from Human Feedback) 是将 LLM 与人类价值观对齐的关键技术。

6.4.1 Step 1: 监督微调 (SFT)

在一个高质量的数据集（Prompt, Response）上进行标准的监督学习 (MLE)。

$$\max_{\phi} \sum_{(x,y)\in D} \log \pi_\phi(y|x)$$

得到的模型作为后续的起点。

6.4.2 Step 2: 奖励模型 (RM) 从偏好学习

训练一个模型 $r_\psi(x, y)$ 来模仿人类打分。人类标注员会对两个回答 y_w (win) 和 y_l (loss) 进行比较。RM 的目标是让 y_w 的得分比 y_l 高：

$$\mathcal{L}_{RM} = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma(r_\psi(x, y_w) - r_\psi(x, y_l))\right]$$

6.4.3 Step 3: 用 PPO 优化并加 KL 约束

我们希望语言模型 π_θ 生成的内容能获得 RM 的高分，但又不能通过“钻空子”（乱输出乱码骗分）偏离原来的语言能力。因此，总奖励函数设计为：

$$R(x, y) = r_\psi(x, y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{SFT}(y|x)}$$

其中第二项是 KL 散度 (KL Divergence) 惩罚项，用于约束当前模型 π_θ 不要偏离 SFT 模型 π_{SFT} 太远。

Chapter 7

语言模型与 NLP: 从 N-gram 到 Transformer

语言模型 (Language Model, LM) 是 NLP 的核心。简单来说，语言模型的任务是计算一个句子（单词序列）出现的概率 $P(w_1, w_2, \ldots, w_T)$，或者预测下一个词出现的概率 $P(w_{t+1}|w_1, \ldots, w_t)$。

7.1 链式法则与语言模型

7.1.1 联合概率与链式法则 (Chain Rule)

给定一个由 T 个单词组成的句子 $S = (w_1, w_2, \ldots, w_T)$，我们希望计算这个句子作为一个整体存在的概率。根据概率论中的链式法则，联合概率可以分解为条件概率的乘积：

$$P(S) = P(w_1, w_2, \ldots, w_T) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_T|w_1, \ldots, w_{T-1}) \tag{7.1}$$

或者写成通用的乘积形式：

重要提示

语言模型通用公式

$$P(S) = \prod_{t=1}^{T} P(w_t|w_1, \ldots, w_{t-1})$$

- 物理含义：一个句子出现的概率，等于“第一个词出现的概率”乘以“在第一个词已知的情况下第二个词出现的概率”……以此类推。

- 参数空间爆炸：随着句子变长，历史上下文 $(w_1, \ldots, w_{t-1})$ 的组合数量呈指数级增长。如果词表大小为 $|V|$，上下文长度为 L，则参数量为 $|V|^L$。这使得直接统计变得不可能（数据稀疏问题）。

7.1.2 N-gram 模型：马尔可夫假设

为了解决参数爆炸问题，我们引入马尔可夫假设 (Markov Assumption)：假设当前词 w_t 的出现只与它前面 $n-1$ 个词有关，而与更早的历史无关。

这就是 N-gram (N 元语法) 模型：

$$P(w_t|w_1, \ldots, w_{t-1}) \approx P(w_t|w_{t-n+1}, \ldots, w_{t-1}) \tag{7.2}$$

常见的 N-gram 类型

1. Unigram (一元，N=1)：

$$P(S) \approx \prod_{t=1}^{T} P(w_t)$$

完全忽略上下文，词与词之间相互独立（“词袋模型”）。

2. Bigram (二元，N=2)：

$$P(S) \approx \prod_{t=1}^{T} P(w_t|w_{t-1})$$

当前词只取决于前一个词。

3. Trigram (三元，N=3)：

$$P(S) \approx \prod_{t=1}^{T} P(w_t|w_{t-2}, w_{t-1})$$

参数估计：最大似然估计 (MLE)

N-gram 的概率通常通过统计语料库中的词频来计算。对于 Bigram 模型，概率计算公式为：

$$P_{MLE}(w_t|w_{t-1}) = \frac{\text{Count}(w_{t-1}, w_t)}{\text{Count}(w_{t-1})} \tag{7.3}$$

即：“单词对 (w_{t-1}, w_t) 一起出现的次数”除以“单词 w_{t-1} 单独出现的总次数”。

通俗讲解：N-gram 的局限性

- 长距离依赖 (Long-term Dependency)：N-gram 无法捕捉超过 N 个词距离的语义关系（例如：“The **boy** ..... **is** running”）。
- 稀疏性 (Sparsity)：如果一个 N 元组在训练语料中从未出现，其概率为 0，会导致整个句子概率为 0。通常需要引入平滑 (Smoothing) 技术（如 Laplace 平滑）来解决。

7.2 评估指标：交叉熵与困惑度

如何评价一个语言模型好不好？好的模型应该给真实的句子分配高概率，给不通顺的句子分配低概率。

7.2.1 负对数似然 (Negative Log-Likelihood, NLL)

在深度学习中,我们通常通过最小化损失函数来训练模型。对于数据集 $D = \{w_1, \ldots, w_N\}$，最大化似然概率 $P(D)$ 等价于最小化负对数似然：

$$\mathcal{L}_{NLL} = -\sum_{t=1}^{T} \log P(w_t | w_{<t}) \tag{7.4}$$

注：由于概率是 $[0, 1]$ 之间的小数，连乘会导致数值下溢，取对数可以将乘法转换为加法。

7.2.2 交叉熵 (Cross Entropy)

从信息论角度，语言模型的训练目标是让模型预测的概率分布 Q 尽可能接近真实数据的分布 P。对于分类问题（预测下一个词就是在词表中做分类），交叉熵定义为：

$$H(P, Q) = -\sum_{x} P(x) \log Q(x)$$

在语言模型中，真实分布 P 是 One-hot 向量（下一个词是确定的），因此交叉熵损失实际上就是目标词的 NLL。

7.2.3 困惑度 (Perplexity, PPL)

困惑度是 NLP 中最常用的评价指标，它是交叉熵的指数形式。

重要提示

困惑度公式 (背诵)

$$\text{PPL}(S) = P(w_1, \ldots, w_T)^{-\frac{1}{T}} = \sqrt[T]{\frac{1}{P(w_1, \ldots, w_T)}}$$

或者利用交叉熵 H 表示：

$$\text{PPL} = 2^{H(P,Q)}$$

困惑度的直观解释

- 物理含义：PPL 表示模型在预测下一个词时，感到“困惑”的程度。也可以理解为平均分支系数 (Average Branching Factor)。
- 例子：
  - 如果 PPL $= 1$，说明模型极其确信，完全能猜对下一个词（完美模型）。
  - 如果 PPL $= 1000$，说明模型觉得下一个词有 1000 种可能，相当于在一个 1000 面的骰子上盲猜。
- 结论：PPL 越小，模型越好。

7.3 语言的表示：从符号到向量

计算机无法直接理解“苹果”这个中文符号，必须将其转化为数字。

7.3.1 独热编码 (One-hot Encoding)

这是最原始的方法。假设词表有 V 个词，每个词就是一个长为 V 的向量。

- 苹果：$[1, 0, 0, 0, \ldots]$
- 香蕉：$[0, 1, 0, 0, \ldots]$

致命缺陷：

1. 维度爆炸：词表如果有 10 万个词，向量长度就是 10 万，极度稀疏。
2. 语义鸿沟：任意两个词向量是正交的（点积为 0）。计算机无法计算“苹果”和“香蕉”的相似度，也无法知道它们都属于水果。

7.3.2 词嵌入 (Word Embedding) 与 Word2Vec

核心假设：分布式假设 (Distributional Hypothesis) “*A word is known by the company it keeps.*” ——一个词的含义由它周围的词决定。

Word2Vec 将每个词映射到一个低维（如 512 维）、稠密的实数向量空间中。

通俗讲解：Word2Vec 的两种训练架构

假设句子为：“The cat sat on the mat”。

- CBOW (Continuous Bag of Words)：
  - 任务：根据周围词预测中心词。
  - 输入：‘”The”, ”cat”, ”on”, ”the”‘ → 输出：‘”sat”‘
  - 特点：像做完形填空，适合小数据。
- Skip-gram：
  - 任务：根据中心词预测周围词。
  - 输入：‘”sat”‘ → 输出：‘”The”, ”cat”, ”on”, ”the”‘
  - 特点：数据利用率高，对生僻词效果好（必考对比）。

数学特性：向量空间能够捕捉语义关系。

$$\vec{v}(\text{King}) - \vec{v}(\text{Man}) + \vec{v}(\text{Woman}) \approx \vec{v}(\text{Queen})$$

7.4 序列建模：从 RNN 到 Transformer

在 Transformer 出现之前，RNN 是处理序列的霸主，但也存在严重问题。

7.4.1 循环神经网络 (RNN) 及其缺陷

RNN 处理序列是“串行”的：读完第一个词，生成隐状态 h_1，再读第二个词，结合 h_1 生成 h_2。

- 优点：能够处理变长序列。
- 缺点 (梯度消失/长距离依赖)：当句子很长时（如 100 个词），读到最后时，第 1 个词的信息早已在反复的矩阵乘法中丢失了。

注：*LSTM (长短时记忆网络)* 通过引入“门控机制”缓解了这个问题，但依然无法并行计算，速度慢。

7.4.2 Transformer 模型

2017 年 Google 提出 Transformer，彻底抛弃了循环结构，完全基于 Attention（注意力）机制。

- 并行计算：所有词同时输入，不再是一个个读。
- 全局视野：无论句子多长，任意两个词之间的距离都是 1（直接计算相关性）。

7.5 详解：自注意力机制 (Self-Attention)

这是大模型的灵魂。请务必理解每一个符号的含义。

7.5.1 Q, K, V 的物理含义

对于输入序列中的每一个词向量 x，我们通过三个可学习的矩阵 W^Q, W^K, W^V 将其线性变换为三个向量：

- Query (Q)：查询向量。代表“当前这个词想找什么样的相关信息”。
- Key (K)：键向量。代表“当前这个词包含什么样的特征标签”。
- Value (V)：值向量。代表“当前这个词的实际内容”。

7.5.2 核心公式推导

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

1. QK^T (相似度计算)：计算当前词的 Q 和句中所有词的 K 的点积。点积越大，说明两个词关系越紧密（比如“吃”和“苹果”）。得到的是一个分数矩阵 (Attention Scores)。
2. $\frac{1}{\sqrt{d_k}}$ (缩放 Scaling)：考点！为什么除以根号 d_k？
   - 当向量维度 d_k 很大时，点积结果会非常大。
   - 大数值经过 Softmax 后，梯度会趋近于 0（梯度消失）。缩放是为了把数值拉回梯度敏感区。
3. Softmax (归一化)：将分数转化为概率分布（权重和为 1）。
4. $\times V$ (加权求和)：根据权重，把所有词的 V 加起来，得到当前词的新表示。

7.5.3 多头注意力 (Multi-Head Attention)

为什么要“多头”?

- 就像人看文章，既关注语法结构，又关注指代关系，还关注情感色彩。
- 多头意味着把向量切分成多组，每一组（Head）去关注不同的语义空间，最后拼接起来。

7.5.4 位置编码 (Positional Encoding)

因为 Transformer 是并行输入的，它不知道“我爱你”和“你爱我”中词序的区别。**解决**：人为给每个词加上一个位置向量（基于正弦/余弦函数），告诉模型词在句中的位置。

7.6 大语言模型训练三部曲 (ChatGPT 原理)

现在的 LLM（如 GPT-4）训练过程可以概括为：“博览群书 $\rightarrow$ 拜师学艺 $\rightarrow$ 查漏补缺”。

7.6.1 阶段一：预训练 (Pre-training) ——获得知识

任务：Next Token Prediction (文字接龙)。**数据**：互联网上的海量文本（TB 级别）。**目标**：最大化似然函数 $P(w_t|w_1,\ldots,w_{t-1})$。**结果**：模型学会了语法、世界知识、推理能力。但它是个“话痨”，不懂人类指令（你问它“怎么做蛋糕”，它可能接着写“的 10 种方法”而不是直接回答）。

7.6.2 阶段二：有监督微调 (SFT) ——学习指令

Supervised Fine-Tuning。**数据**：人工编写的高质量“问题-答案”对（几万条）。**过程**：在预训练模型基础上，用这些数据继续微调。**结果**：模型学会了“对话模式”，知道当用户提问时应该回答，而不是续写。缺点：依靠人工写答案太贵了，而且人类很难写出完美的答案，只能写出“还不错”的。

7.6.3 阶段三：RLHF (基于人类反馈的强化学习) ——对齐价值观

这是让 ChatGPT 变得“像人”的关键。分为三个子步骤：

1. SFT (已完成)：得到一个会说话的模型 π^{SFT}。
2. 训练奖励模型 (Reward Model, RM)：

- 让 SFT 模型对同一个问题生成 4 个回答 (A, B, C, D)。
- 人类标注员进行排序（比如 $A > C > B > D$），而不是直接打分（排序比打分容易且客观）。
- 训练一个神经网络 RM，输入是“问题 + 回答”，输出是一个标量分数。目标是让 RM 的打分符合人类的排序。

3. PPO 强化学习 (Proximal Policy Optimization)：

- 环境：用户的 Prompt。
- 智能体：语言模型。
- 动作：生成的单词。
- 奖励：RM 给出的分数。
- 目标：更新语言模型的参数，让它生成的回答能从 RM 那里拿到高分。
- 约束 (KL Divergence)：为了防止模型为了高分胡言乱语（Hack 奖励模型），加一个约束，让新模型的分布不要偏离 SFT 模型太远。

重要提示

BERT vs. GPT (常见考点)

- BERT (Encoder-only)：双向理解。像做完形填空。擅长理解任务（情感分析、分类）。
- GPT (Decoder-only)：单向生成。像做文字接龙（只能看左边）。擅长生成任务（对话、写作）。

Chapter 8

模拟视觉：计算机视觉

8.1 视觉的本质：从矩阵到特征

8.1.1 计算机眼中的图像

人类看到的是“一只猫”，计算机看到的是一个三维矩阵 (Tensor)。对于一张 224×224 的彩色图片，计算机存储为：

$$\text{Image} \in \mathbb{R}^{H \times W \times C}$$

其中 $H = 224$ (高), $W = 224$ (宽), $C = 3$ (RGB 通道)。**挑战：**如果直接把这 $150,528$ 个像素点作为输入（Flatten）扔进全连接神经网络，参数量会爆炸，且会丢失像素之间的空间邻域关系。

8.2 卷积神经网络 (CNN) ——视觉的基石

CNN 受生物学启发（Hubel 和 Wiesel 发现猫脑皮层中有对特定方向线条反应的细胞），专门用于提取空间特征。

8.2.1 核心组件详解

1. 卷积层 (Convolution Layer) ——提取特征

通俗讲解：通俗比喻：手电筒扫描

想象你在看一幅巨大的壁画，手里只有一个小的手电筒（卷积核/Filter/Kernel）。

- 你从左上角开始，照亮一小块区域。
- 你的手电筒不仅是照明，还是一个“匹配器”。如果手电筒里的图案（比如一条竖线）和墙上的图案吻合，由于点积运算，数值就会很大（激活）。
- 你按固定的步子（Stride）滑动移动手电筒，直到扫描完整个墙壁。

关键超参数 (Hyperparameters)：

- Kernel Size (K)：手电筒的大小（通常 3×3 或 5×5）。
- Stride (S)：步长。每次滑动的距离。$S = 1$ 扫得细，$S = 2$ 会导致输出变小（降维）。
- Padding (P)：填充。为了不让边缘信息丢失，或者为了保持输出尺寸不变，在原图周围补一圈 0。

重要提示

必考计算：输出尺寸公式 输入尺寸为 W_{in}，卷积核 K，步长 S，填充 P。输出尺寸 W_{out} 为：

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

注：如果计算结果不是整数，通常向下取整。

2. 激活函数 (Activation - ReLU)

卷积只是线性运算（乘加）。为了拟合复杂的非线性世界，必须引入非线性激活。

$$\text{ReLU}(x) = \max(0, x)$$

作用：把负值置零。生物学解释是“抑制”不重要的信号，只保留激活强烈的特征。

3. 池化层 (Pooling) ——降维打击

目的：减少参数量，扩大后续层的感受野 (Receptive Field)，防止过拟合。

- Max Pooling：在 2×2 的窗口内只取最大的那个值。

- 物理含义：只要这个区域内出现了“眼睛”这个特征，我就保留它，具体它在区域的左上还是右下不重要（平移不变性）。

8.3 图像生成模型：让 AI 拥有想象力

8.3.1 生成模型概览

- 判别式模型 (Discriminative)：输入 x (图)，输出 y (标签)。(是猫还是狗？)
- 生成式模型 (Generative)：学习数据的分布 $P(x)$，然后从中采样生成新的 $\hat{x}$。(画一只猫)

8.3.2 GAN (生成对抗网络)

原理：博弈论。

- 生成器 (Generator, G)：造假钞的。输入随机噪声 z，输出假图 $G(z)$。目标是骗过 D。
- 判别器 (Discriminator, D)：验钞机。输入图片，判断是真图还是假图。目标是揪出 G。
- 结局：纳什均衡。G 生成的图极其逼真，D 瞎猜（判断概率为 0.5）。

缺点：训练极不稳定（两个网络很难同步进步），容易产生模式坍塌（Mode Collapse，即生成的图长得都一样）。

8.4 扩散模型 (Diffusion Models) 详解

这是目前最先进的生成模型（*Stable Diffusion, Sora, DALL-E 3* 的基础）。

8.4.1 核心直觉：拆楼与盖楼

扩散模型包含两个互逆的过程。

- 前向过程 (Forward / Diffusion Process)：有序 $\rightarrow$ 无序。就像往一杯清水里滴墨水，或者把一座精致的沙堡推倒。随着时间推移，图像逐渐变成纯粹的高斯噪声。
- 反向过程 (Reverse / Denoising Process)：无序 $\rightarrow$ 有序。这是 AI 要学的神技。给它一团杂乱无章的噪声，让它像倒放电影一样，一步步“去噪”，还原出清晰的图像。

8.4.2 数学原理解析

1. 前向过程 (加噪)

这是一个固定的马尔可夫链，不需要训练。对于时刻 t，我们在 $t-1$ 时刻的图像上加入少量高斯噪声 ϵ：

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

只要 t 足够大（比如 1000 步），x_T 就变成了纯高斯噪声。

2. 反向过程 (去噪与训练)

我们要训练一个神经网络（通常是 UNet 架构），来模拟反向过程 $p_\theta(x_{t-1}|x_t)$。

重要提示

关键考点：网络到底在预测什么？初学者常误以为神经网络直接输入噪声，输出一张高清大图。错！

- 输入：当前时刻的噪声图 x_t + 时间步 t (+ 文本提示词 embedding)。
- 输出：预测这张图里包含的噪声 ϵ_θ 是什么。

去噪公式：

$$\text{上一刻图像}x_{t-1} = \text{当前图像}x_t - \text{预测的噪声}\epsilon_\theta(x_t, t)$$

（注：实际公式还要乘系数和加方差，但核心逻辑是**减去预测的噪声**）。

3. 文本引导 (Text Conditioning)

为什么我们可以通过输入“一只骑车的狗”来控制生成？

- 使用 CLIP 等模型将文本转化为向量 Embedding。
- 将这个向量通过 Cross-Attention (交叉注意力) 机制注入到 UNet 的每一层中。
- UNet 在预测噪声时，会被文本向量“带偏”，只保留符合文本描述的图像特征，去除不符合的特征。

8.4.3 GAN vs. Diffusion 对比 (复习总结)

特性	GAN (生成对抗网络)	Diffusion (扩散模型)
生成质量	高，但容易失真	**极高**，细节丰富
多样性	低 (模式坍塌)	**高** (覆盖完整分布)
训练难度	极难 (对抗不稳定)	较易 (只有简单的损失函数)
推理速度	**快** (一次生成)	慢 (需迭代几十/上百步)

8.5 GAN: Min-Max 目标与最优判别器推导

8.5.1 基本目标函数 (Min-Max Objective)

GAN 的核心是一个零和博弈 (Zero-Sum Game)。

- 判别器 D：希望最大化区分真假的能力。即 $D(x) \to 1$ (真图)，$D(G(z)) \to 0$ (假图)。
- 生成器 G：希望最小化判别器发现假图的概率。即让 $D(G(z)) \to 1$。

这构成了一个极小-极大值问题：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1-D(G(z)))] \tag{8.1}$$

8.5.2 最优判别器 $D^*(x)$ 的推导

问题：固定生成器 G，什么样的判别器 D 才是最好的？我们将期望写成积分形式。令生成的分布为 $P_g(x)$，则第二项可以写成关于 x 的积分：

$$V(D,G) = \int_x \left[P_{data}(x)\log D(x) + P_g(x)\log(1-D(x))\right] dx$$

为了求极大值，对积分内部关于 $D(x)$ 求导并令其为 0。设 $y = D(x)$，函数为 $f(y) = a\log y + b\log(1-y)$。

$$f'(y) = \frac{a}{y} - \frac{b}{1-y} = 0 \implies y = \frac{a}{a+b}$$

代入 $a = P_{data}(x), b = P_g(x)$，得到最优判别器公式：

重要提示

最优判别器

$$D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$$

直觉：如果某处的真实数据密度 P_{data} 远大于生成密度 P_g，判别器就该输出 *1*。如果两者相等，输出 *0.5*。

8.5.3 与 JS 散度的关系

当判别器达到最优 D^* 时，把它代回目标函数 $V(D^*, G)$，经过数学变换可以证明：

$$\max_D V(D, G) = 2 \cdot JS(P_{data} \| P_g) - \log 4$$

结论：原始 GAN 的训练本质上是在最小化真实分布与生成分布之间的 JS 散度 (Jensen-Shannon Divergence)。

8.5.4 非饱和生成器损失 (Non-saturating Loss)

在训练初期，生成器很烂，判别器很容易区分，$D(G(z)) \approx 0$。此时原始损失 $\min \log(1 - D(G(z)))$ 的梯度非常平缓（Saturating，梯度消失），导致学不动。工程实战做法：不最小化 $\log(1 - D)$，改为最大化 $\log D$。

$$\mathcal{L}_G = -\mathbb{E}_z[\log D(G(z))]$$

这也是目录中提到的“非饱和损失”，能提供更强的梯度。

8.6 VAE: 变分自编码器 (Variational Auto-Encoder)

8.6.1 目标与变分后验

我们希望通过隐变量 z 生成数据 x，即建模 $P(x) = \int P(x|z)P(z)dz$。但这个积分很难求。我们引入一个推断网络（编码器）$Q_\phi(z|x)$ 来近似真实的后验 $P(z|x)$。

- Encoder $Q_\phi(z|x)$：输入图 x，输出 z 的均值 μ 和方差 σ^2。
- Decoder $P_\theta(x|z)$：输入 z，还原图 x。

8.6.2 ELBO 推导 (最常用推导之一)

类似于 EM 算法，我们希望最大化对数似然 $\log P(x)$。

$$\log P(x) = \log \int P(x, z)dz = \log \mathbb{E}_{z\sim Q}\left[\frac{P(x, z)}{Q(z|x)}\right]$$

利用 Jensen 不等式：

$$\log P(x) \geq \mathbb{E}_{z\sim Q}\left[\log \frac{P(x, z)}{Q(z|x)}\right] = \text{ELBO}$$

将 ELBO 拆解为两部分（VAE 的损失函数）：

重要提示

VAE 损失函数 = 重构损失 + KL 散度

$$\mathcal{L}_{VAE} = \underbrace{\mathbb{E}_{z\sim Q}[\log P_\theta(x|z)]}_{\text{重构项 (Reconstruction)}} - \underbrace{KL(Q_\phi(z|x)\|P(z))}_{\text{正则项 (Regularization)}}$$

- 重构项：希望 z 能还原出原图（通常用 MSE 衡量）。
- KL 项：希望编码出的 z 的分布接近标准正态分布 $\mathcal{N}(0, I)$，防止编码器“作弊”死记硬背。

8.6.3 重参数技巧 (Reparameterization Trick)

痛点：在网络中间进行“随机采样” $z \sim \mathcal{N}(\mu, \sigma^2)$ 是无法反向传播梯度的（因为采样是离散的随机行为）。解决方案：将随机性转移到一个独立的噪声 ϵ 上。

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

这样，z 就变成了关于 μ 和 σ 的平滑函数，梯度可以顺利流过。

8.7 扩散模型：前向加噪、反向去噪与训练目标

8.7.1 前向过程 $q(x_t|x_{t-1})$

前向过程是一个固定的马尔可夫链，每一步加入高斯噪声。

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

任意时刻 t 的直接采样性质：不需要一步步算，可以直接从 x_0 算出 x_t。令 $\alpha_t = 1-\beta_t, \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$：

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

8.7.2 反向过程 $p_\theta(x_{t-1}|x_t)$

我们要训练神经网络逼近真实的后验 $q(x_{t-1}|x_t)$。虽然 $q(x_{t-1}|x_t)$ 很难求，但如果已知 x_0，它是可解的。我们让神经网络去预测高斯分布的均值 μ_θ 和方差 Σ_θ：

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

8.7.3 训练目标：预测噪声的均方误差

经过复杂的数学简化（DDPM 论文），最大化 ELBO 最终等价于让神经网络预测加入的噪声 ϵ。

重要提示

简单有效的损失函数 我们在 x_0 上叠加噪声 ϵ 得到 x_t，然后把 x_t 扔给网络，问网络："刚才加的噪声是什么？"

$$\mathcal{L}_{simple} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon}_{x_t}, t)\|^2\right]$$

8.7.4 条件生成与 CFG (概念写法)

如何让 AI 听懂人话？比如"画一只戴墨镜的猫"。Classifier-Free Guidance (CFG) 是现在的标准做法。训练时，有时给网络看文本 c (Cond)，有时不给 (Uncond, 空字符)。生成采样时，我们结合两者的预测：

$$\tilde{\epsilon}_\theta = \epsilon_\theta(x_t) + w \cdot (\epsilon_\theta(x_t, c) - \epsilon_\theta(x_t))$$

- w (Guidance Scale)：引导系数。
- $w > 1$ 时，会增强文本对图像的控制力，让图像更符合描述，但过大会导致图像失真（炸得五颜六色）。

Chapter 9

模拟听觉：语音信号处理

9.1 声音的物理与数字化

9.1.1 声音的三要素

- 音调 (Pitch)：由频率 (Frequency) 决定。频率高声音尖（女声），频率低声音沉（男声）。
- 响度 (Loudness)：由振幅 (Amplitude) 决定。振幅大声音大。
- 音色 (Timbre)：由波形 (Waveform) 决定。区分钢琴和小提琴的关键。

9.1.2 模拟信号数字化 (ADC)

计算机存不了连续的波，只能存点。

- 采样 (Sampling)：时间轴上的离散化。每秒钟切多少刀（如 44.1kHz）。
- 量化 (Quantization)：纵轴（振幅）的离散化。每一刀切下来的值用多少位二进制表示（如 16-bit）。

9.2 特征提取：从时域到频域的魔法

这是 AI 处理语音的前置核心步骤。直接把波形扔进神经网络效果很差，我们需要提取特征。

9.2.1 傅里叶变换 (Fourier Transform)

通俗讲解：通俗比喻：奶昔与食谱

- 时域信号 (Time Domain)：你手里的一杯混合果汁奶昔（随时间变化的波形）。你只能看到它现在的颜色和味道，看不出成分。
- 傅里叶变换 (FT)：一台神奇的机器，它能把奶昔还原成食谱。
- 频域信号 (Frequency Domain)：食谱清单。它告诉你：这里面有 30% 的草莓（低频波）、20% 的香蕉（中频波）和 50% 的牛奶（高频波）。

结论：任何复杂的波形，都可以拆解为无数个不同频率、不同振幅的正弦波的叠加。

9.2.2 声谱图 (Spectrogram)

将声音转化为图像，这是 CNN/Transformer 处理语音的标准输入。

- 横轴：时间 (Time)。
- 纵轴：频率 (Frequency)。
- 颜色深浅：能量/响度 (Magnitude)。颜色越亮，说明这个时间点、这个频率的声音越响。

9.2.3 梅尔刻度 (Mel Scale) ——必考概念

问题：物理上的 100Hz → 200Hz 的差距，和 10000Hz → 10100Hz 的差距是一样的（都是 100Hz）。但人耳听起来，前者区别很大，后者几乎听不出来。**原理**：人耳对频率的感知是对数的。低频敏感，高频迟钝。**公式**：

$$\text{Mel}(f) = 2595 \cdot \log_{10}(1 + \frac{f}{700})$$

Mel Spectrogram：在普通声谱图的基础上，把纵轴（频率）按照 Mel 刻度拉伸或压缩，使其符合人类听觉特性。

9.3 语音合成 (TTS) 的进化史

任务：输入文本 (Text) → 输出语音 (Audio)。

9.3.1 端到端合成的两个流派

1. 自回归模型 (Autoregressive) ——代表：Tacotron

原理：像 RNN 一样，生成完这一秒的声音，再生成下一秒。依赖上一步的输出。**核心机制：**基于 Attention（注意力）来对齐文本和语音。**致命缺陷：**

- 推理慢：必须串行生成，无法并行。
- 鲁棒性差 (Robustness)：容易出现漏读 (Skipping) 或复读 (Repeating)。*原因：Attention 矩阵有时候没对齐好，模型不知道“我爱你”读完“爱”字没有，可能卡在那里一直读“爱爱爱...”。*

2. 非自回归模型 (Non-Autoregressive) ——代表：FastSpeech

这是考试的重点算法，解决了 Tacotron 的痛点。

核心创新：显式时长预测 (Explicit Duration Prediction) FastSpeech 认为：不要让模型自己在 Attention 里瞎猜每个字读多久，而是专门训练一个模型来预测时长。

通俗讲解：FastSpeech 的工作流程

1. 音素编码：输入文本序列（如“你好”），过 Encoder 得到向量。
2. 时长预测器 (Duration Predictor)：
   - 这是一个小的回归网络。
   - **输入：**“你”的向量。
   - **输出：**数字 10（代表“你”这个字要占 10 个时间帧）。
3. 长度调节器 (Length Regulator) ——关键步骤：
   - 根据预测的时长，把“你”的向量复制 10 份。
   - 把“好”的向量复制 15 份（假设预测为 15）。
   - 拼接起来，形成一个与最终音频长度一致的长序列。
4. 解码生成：Decoder 并行地把这些向量转化为声谱图。

重要提示

FastSpeech 的三大优势

1. 速度极快：并行生成，比 Tacotron 快 38 倍甚至更多。
2. 极度稳定：因为时长是显式预测并强制复制的，绝不会出现漏字或重复。
3. 可控性强：想调节语速？只需要把预测出来的时长统一 $\times 1.5$ 或 $\times 0.5$ 即可。

9.4 第七部分：总结与展望

9.4.1 多模态融合 (Multimodal Fusion)

未来的 AI 不再是瞎子（只懂 NLP）或聋子（只懂 CV）。

- Think-Then-React：看到动作（视觉）$\rightarrow$ 思考意图（语言/逻辑）$\rightarrow$ 做出反应（行为）。
- AV-ASR (Audio-Visual ASR)：利用唇语（视觉信息）来辅助在嘈杂环境下的语音识别。