25HS_NLP

Lecture Notes

目录 / Contents

1 Semirings

1.1 Motive: one algo 解决多个问题

核心洞察：计算 normalizer Z 和寻找 highest-scoring path 本质上都是 **shortest path problems**。与其为每个任务设计单独 algo，不如用 semiring 参数化：

$$\sum \rightsquigarrow \oplus, \quad \prod \rightsquigarrow \otimes$$

若原 algo 计算 $Z = \sum_y \prod_n \exp \mathrm{score}(y_n)$，则 semiringified 版本计算：

$$\bigoplus_y \bigotimes_n \exp \mathrm{score}(y_n)$$

这之所以可行，是因为我们只需 associativity、commutativity (for $\oplus$) 和 distributivity。

1.2 常用 Semirings 速查

Name	$\mathbb{K}$	$\oplus$	$\otimes$	$\mathbf{0}$	$\mathbf{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Real[1]	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1
Tropical[2]	$\mathbb{R}\cup\{\infty\}$	min	$+$	∞	0
Viterbi	$\mathbb{R}\cup\{-\infty\}$	max	$+$	$-\infty$	0
Log	$\mathbb{R}\cup\{\pm\infty\}$	lse[3]	$+$	$-\infty$	0

Table 1: Semiring 对照表

1.3 代数结构

Monoid

三元组 $\langle \mathbb{K}, \otimes, \boldsymbol{e}\rangle$ 满足：

1. **Associativity:** $(x\otimes y)\otimes z = x\otimes(y\otimes z)$
2. **Identity:** $x\otimes \boldsymbol{e} = \boldsymbol{e}\otimes x = x$

直觉：比 group 少一个 inverse 公理，所以更简单。

Semiring

五元组 $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ 满足：

1. $\langle \mathbb{K}, \oplus, \mathbf{0}\rangle$ 是 **commutative monoid**
2. $\langle \mathbb{K}, \otimes, \mathbf{1}\rangle$ 是 **monoid**
3. **Distributivity:** $(x\oplus y)\otimes z = (x\otimes z)\oplus(y\otimes z)$（左右皆需）
4. **Annihilation:** $\mathbf{0}\otimes x = x\otimes \mathbf{0} = \mathbf{0}$

命名由来：比 ring 少公理（ring 要求 $\oplus$ 构成 group，有逆元）。

💡 我们在逆向工程 dynamic programming 所需的最小公理集。Distributivity 是关键——它让指数 sum 变成多项式形式。

Idempotent Semiring

若 $\forall a : a\oplus a = a$，则称 semiring 是 idempotent 的。典型例子：$\max(a,a)=a$。注意 $\oplus$ 只是 binary operation 的记号，不必然是加法！

⚠️ **考试高频题型**：判断给定结构是否是 monoid/semiring。TA 强调这类题“fast, easy to check, shows understanding”。

1.3.1 Monoid 判定

Monoid 判定练习如 Table 2, Monoid 必须满足：

1. **Closure**：$a\otimes b\in\mathbb{K}$（operation 封闭）；
2. **Associativity**：$(a\otimes b)\otimes c = a\otimes(b\otimes c)$;
3. **Identity**：$\exists \boldsymbol{e} : a\otimes \boldsymbol{e} = \boldsymbol{e}\otimes a = a$

常见陷阱：

- 减法不 associative
- 要检查 identity 是否在集合内
- Monoid 不要求 *commutativity*——string concatenation 是典型例子

结构	Monoid?	原因
$\langle\mathbb{N},+,0\rangle$	✅	标准例子
$\langle\mathbb{N},-,0\rangle$	×	不封闭：$0-1=-1\notin\mathbb{N}$
$\langle\mathbb{Z},-,0\rangle$	×	不 associative：$(a-b)-c\neq a-(b-c)$
$\langle\mathbb{N},\times,1\rangle$	✅	乘法封闭、associative
$\langle\mathbb{R}_{\geq0},\max,0\rangle$	✅	max associative，$\max(a,0)=a$
$\langle\Sigma^*,\mathrm{concat},\varepsilon\rangle$	✅	**非交换** monoid 的例子！

Table 2: Monoid 判定练习

1.3.2 Semiring 判定清单

除 monoid 条件外，还需：

1. $\langle\mathbb{K},\oplus,\mathbf{0}\rangle$ 是 **commutative** monoid
2. $\langle\mathbb{K},\otimes,\mathbf{1}\rangle$ 是 monoid
3. **Distributivity**：$a\otimes(b\oplus c) = (a\otimes b)\oplus(a\otimes c)$
4. **Annihilation**：$\mathbf{0}\otimes a = a\otimes\mathbf{0} = \mathbf{0}$

结构	Semiring?	关键点
$\langle\mathbb{N},+,\times,0,1\rangle$	✅	counting paths
$\langle\mathbb{R}_{\geq0},\max,\times,0,1\rangle$	✅	unnormalized probabilities
$\langle\mathbb{R}_{\geq0},\max,+,0,0\rangle$	×	$\mathbf{0}=\mathbf{1}=0$ 矛盾！
$\langle\mathbb{R}\cup\{-\infty\},\min,+,+\infty,0\rangle$	✅	shortest path (tropical)
$\langle\mathbb{R}\cup\{+\infty\},\min,+,-\infty,0\rangle$	×	不 distributive!
$\langle\mathcal{P}(\Sigma^*),\cup,\mathrm{concat},\emptyset,\{\varepsilon\}\rangle$	✅	语言的集合，**非交换**乘法

Table 3: Semiring 判定练习

💡

关键陷阱：$\mathbf{0}=\mathbf{1}$ 时必然失败。因为：

- $a\otimes\mathbf{0}=\mathbf{0}$（annihilation）
- $a\oplus\mathbf{0}=a$（identity）
- 若 $\mathbf{0}=\mathbf{1}$，则 $a\otimes\mathbf{1}=\mathbf{0}$，但应有 $a\otimes\mathbf{1}=a$

Distributivity 检验：$\min(1,2)+3=4$，但 $\min(1+3,2+3)=4$? ✅
反例：$\min(1,2)+3\neq\min(1,2+3)$（错误方向的 distributivity）

1.4 Closed Semiring 与 inftysum

Closed Semiring

增设 **Kleene star** 运算：$a^* = \bigoplus_{n=0}^{\infty} a^{\otimes n}$，满足：

$$a^* = \mathbf{1}\oplus a\otimes a^* = \mathbf{1}\oplus a^*\otimes a$$

对 real semiring 在 $(-1,1)$ 上：$a^*=\sum_{n\geq0}a^n=\frac{1}{1-a}$（geometric series）。这是 globally normalized language model 的理论基础。

1.5 动态规划的代数推导

⚠️ **考点**：Ryan 明确说这是“fundamental slide”，历年必考。

目标：计算 $Z(\boldsymbol{w}) = \sum_{\boldsymbol{t}\in\mathcal{T}^N}\exp\mathrm{score}(\boldsymbol{t},\boldsymbol{w})$

[1] 严格来说，$\mathbb{R}_{\geq0}$ 上的 $(+,\times)$ 严格来说不是 semiring（$0.6+0.6=1.2\notin[0,1]$，不封闭），但文献中常如此称呼。

[2] 因曲线形态得名。Tropical geometry 与 ReLU 网络的 decision boundary 几何相关——并非冷门领域。

[3] 其中 $\mathrm{lse}(x,y)=\log(e^x+e^y)$. **Log-Sum-Exp Trick**: 若 $x\geq y$，则 $\log(e^x+e^y)=x+\log(1+e^{y-x})$ 因 $y-x\leq0$，故 $e^{y-x}\leq1$，数值稳定。Motiv 是计算 $\log(e^x+e^y)$ 时，直接 exp 会 overflow。所有神经网络库都实现了 `logsumexp`,如 `torch.logsumexp(log_probs, dim=...)`。

Step 1：假设 score 可加分解

$$\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, \boldsymbol{w}, n)$$

Step 2：代数变换

$$\begin{aligned} Z &= \sum_{\boldsymbol{t}} \exp \sum_{n} \text{score}_n = \sum_{\boldsymbol{t}} \prod_{n} \exp \text{score}_n \quad (\exp \text{ 法则}) \\ &= \sum_{t_1} \cdots \sum_{t_N} \prod_{n} \exp \text{score}_n \quad (\text{展开}) \\ &= \sum_{t_1} \exp \text{score}_1 \times \left(\cdots \sum_{t_N} \exp \text{score}_N \right) \quad (\text{distributivity}) \end{aligned}$$

关键：最后一步用 distributivity 把内层 sum"推进去"。complexity 从 $O(|\mathcal{T}|^N)$ 降到 $O(N\ |\mathcal{T}|^2)$。

💡 **高频考题**："若 score 依赖连续 3 个 tags 而非 2 个？"答：complexity 变为 $O(N\ |\mathcal{T}|^3)$——多一层 for 循环，推导同理。

2 Part-of-Speech Tagging

Figure: POS Graph
$\mathcal{T} = \{\text{N}, \text{V}, \text{Det}\}$ 时的示意图。Inference 找最优 N-path；training 对所有 N-paths sum。虚线为 backpointers，粗线为最优路径。

2.1 问题定义

给定 sentence $\boldsymbol{w} \in \Sigma^N$，输出 tag sequence $\boldsymbol{t} \in \mathcal{T}^N$。Output space 大小 $|\mathcal{T}|^N$ 指数增长，需高效 algo。
与 language modeling 的区别：这里是**有限** set 上的 sum，无收敛问题，只有计算 complexity 问题。

💡 **linguistic 备注**：POS 范畴因语言而异。欧洲语言中 adjective/verb 分明；汉语中形容词常可直接作谓语（"我高兴"而非"我是高兴的"）。

2.2 Conditional Random Fields

CRF 是 structured labeling 的 conditional model，考虑 neighboring labels 的 context（不像独立分类器）。

First-Order Linear-Chain CRF
假设 tag 仅依赖相邻 tag：

$$\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \sum_{n=1}^{N} [\text{transition}(t_{n-1}, t_n) + \text{emission}(w_n, t_n)]$$

💡 Bigram 假设针对 **tags**，不限制 word representation。用 BiRNN 时，每个位置仍"看到"全句。
局限：无法处理 **garden-path sentences**（如 "The horse raced past the barn fell"），因为无法回溯修改早期 tagging。

2.3 Forward/Backward Algorithms

Backward Algorithm

从右向左计算 semiring-sum。
1. $\forall t_N : \beta[N, t_N] \leftarrow \mathbf{1}$
2. **For** $n = N-1, ..., 0$:
3. **For** $t_n \in \mathcal{T}$:
4. $\beta[n, t_n] \leftarrow \bigoplus_{t_{n+1}} \exp\left(\text{score}_{\{n+1\}}\right) \otimes \beta[n+1, t_{n+1}]$
5. **Return** $\beta[0, \text{BOS}]$

complexity $O(N\ |\mathcal{T}|^2)$。Forward algorithm 方向相反，形式对称。

结构上等价于 backpropagation（都是 DAG 上的路径 sum）。

2.3.1 Forward vs Backward 实现细节

💡 Forward 和 backward 有微妙的不对称性，源于 BOS (beginning of sequence) 存在但 EOS 不显式处理。

Pseudo Code 对比
Backward Algorithm:
```
beta[N, t] = 1  # 直接初始化为 semiring 1
for n = N-1, ..., 0:
    for t_n in T:
        beta[n, t_n] = ⊕_{t_{n+1}} exp(score) ⊗ beta[n+1, t_{n+1}]
return beta[0, BOS]
```
Forward Algorithm:
```
alpha[0, t] = exp(score(BOS -> t))  # 初始化包含 BOS 转移
for n = 1, ..., N-1:
    for t_n in T:
        alpha[n, t_n] = ⊕_{t_{n-1}} alpha[n-1, t_{n-1}] ⊗ exp(score)
return ⊕_t alpha[N-1, t]  # 需要遍历最后一列!
```

关键差异：
1. **初始化**：Backward 直接 $\mathbf{1}$；Forward 需计算 BOS 转移
2. **终止**：Backward 返回单值 $\beta[0, \text{BOS}]$；Forward 需 $\oplus$ 整个最后一列
3. **循环次数**：Forward 少一次迭代，但初始化更复杂

2.3.2 Dijkstra 的局限性

Dijkstra 在哪些 semiring 下失效？Dijkstra 依赖 **greedy property**：一旦 node 被 finalized，其值不再更新。

Dijkstra 失效示例

在 tropical semiring $\langle \mathbb{R}, \min, +, +\infty, 0 \rangle$ 中，若允许 **负权边**：

```
A --3--> B --(-9)--> C
A --7--> C
```

Dijkstra 先 finalize $A \to C$ (cost 7)，但实际最短路 $A \to B \to C$ (cost $3 + (-9) = -6$) 更优。
问题：Dijkstra 从未考虑经过 B 的路径！

Dijkstra 有效的条件：

1. 所有 edge weights 非负（tropical semiring 的标准假设）
2. 或更一般地：semiring 满足某种 **monotonicity**（加入更多 edges 不会使 path 更优）

对于 `sum` semiring（计算 Z）：Dijkstra 正确但无加速——必须考虑所有非零路径。

2.4 Viterbi Algorithm

将 backward algorithm 中的 $\sum$ 换成 max，并记录 backpointers：

Viterbi

1. $\forall t_N : \beta[N, t_N] \leftarrow 1, \mathrm{bp}[N, t_N] \leftarrow \perp$
2. **For** $n = N-1, ..., 0$; **For** t_n:
3. $\quad \beta[n, t_n] \leftarrow \max_{t_{\{n+1\}}} \exp\left(\mathrm{score}_{\{n+1\}}\right) \times \beta\left[n+1, t_{\{n+1\}}\right]$
4. $\quad \mathrm{bp}[n, t_n] \leftarrow \arg\max(\cdots)$
5. Backtrack 得 $\boldsymbol{t}^*$

💡 **为何 search-and-replace 有效？** 因为 $(\max, \times)$ 与 $(+, \times)$ 满足相同代数性质。
- min：可以（tropical semiring）
- sin 等非线性函数：不行（violates distributivity）

历史：Viterbi (1967) 因此 algo 成名，USC 工程学院以其命名。

2.5 Dijkstra 与 Semiring

Dijkstra 通过剪枝只探索可能最短的路径。对 max / min 有效，但对 $\sum$ 无加速——sum 时每条非零路径都必须计入。

2.6 Training

最大化 log-likelihood：

$$\mathcal{L} = \sum_{(\boldsymbol{w}, \boldsymbol{t}) \in \mathcal{D}} [\mathrm{score}(\boldsymbol{t}, \boldsymbol{w}) - \log Z(\boldsymbol{w})]$$

对 forward algorithm 做 backprop 即可求梯度。

💡 HMM 的 forward-backward 本质上就是在算这个梯度，EM 因此类似 gradient descent。

2.7 Structured Perceptron

对 CRF 引入 temperature T：

$$p_{T(\boldsymbol{t}|\boldsymbol{w})} = \frac{\exp(\mathrm{score}/T)}{Z_T}$$

令 $T \to 0$，softmax 变 hard max，梯度简化为：

$$\nabla \mathcal{L} = \varphi(\boldsymbol{t}^{\mathrm{gold}}) - \varphi(\hat{\boldsymbol{t}}), \quad \hat{\boldsymbol{t}} = \arg\max \mathrm{score}$$

其中 arg max 由 Viterbi 计算。

💡 Collins (2002) EMNLP best paper。在我们的框架下只需一行推导。

3 Weighted Finite-State Automata

3.1 动机：Transliteration

将英文名转写为日语片假名“California” -> “カリフォルニア”,日语只有一个 coda 辅音 (N)，需插入额外元音。Source-target 对齐 **not one-to-one**——这正是 CRF 无法直接处理的原因。

3.2 Formal Definitions

基本概念

Alphabet Σ：非空有限集，元素称 letters

String：letters 的有限序列；ε 为 empty string

Unambiguous：每个 string 至多一条 accepting path（$\neq$ deterministic！）

💡 $\cup_{n=0}^{\infty}$ 中 n 遍历自然数，$\infty \notin \mathbb{N}$。set 无穷大，但每个元素有限长。

FSA

Tuple $\langle \Sigma, Q, I, F, \delta \rangle$：alphabet、states、initial states、final states、transitions。

String w 被 **accept** 当且仅当存在从 I 到 F 的 path 拼出 w。

WFSA

在 semiring $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ 上的 weighted 版本，增设：

- $\lambda : Q \to \mathbb{K}$ (initial weights)
- $\rho : Q \to \mathbb{K}$ (final weights)
- Transitions 带权重

Path Sum

Path weight：沿途权重的 $\otimes$-积。

$$Z(\mathcal{A}) = \bigoplus_{\pi \in \Pi(\mathcal{A})} w(\pi)$$

有 cycle 时 path 数无穷，可能发散——类似 language model 的 tightness 问题。

3.3 收敛条件

考虑 self-loop 权重 x：$Z = 1 + x + x^2 + \cdots = \frac{1}{1-x} \quad (|x| < 1)$

能做 globally normalized inference 的 language model class 本质上是 **generalized geometric distributions**。

3.4 Automata 性质

Accessible/Co-accessible：从 initial 可达 / 可达 final

Trim：所有 states 都 useful

Unambiguous：每个 string 至多一条 accepting path（$\neq$ deterministic！）

💡 Ambiguity 在 idempotent semiring 中无影响（$1 \vee 1 = 1$），但在 real semiring 中需对多条 path sum。

3.5 Finite-State Transducers

FST
Transition 形如 (q, a, b, w, q')，$a \in \Sigma, b \in \Omega$。给定 input x，定义 output y 上的条件分布。

Composition
$T_1 : \Sigma \to \Omega$，$T_2 : \Omega \to \Gamma$，则：

$$(T_1 \circ T_2)(x, z) = \bigoplus_{y \in \Omega^*} T_1(x, y) \otimes T_2(y, z)$$

形如 matrix multiplication（带 marginalization）。可层叠构建复杂 model。

3.6 Transliteration model

Composition 自动处理 alignment 的 sum。

1. 定义 permissive FST：任意 symbol 可映射到任意 symbol
2. 学习 transition weights
3. Training：最大化 likelihood，**marginalize over latent alignments**

3.7 Acyclic WFSA 的 Backward Algorithm

⚙ Backward Algorithm (Acyclic WFSA)

按 reverse topological order 遍历 nodes，应用 distributivity。

1. 对 nodes 按 topological order 排序：$q_1, ..., q_M$
2. **For** $m = M, ..., 1$:
3. $\quad \beta[q_m] \leftarrow \rho(q_m) \oplus \bigoplus_{(q_m, a, w, q') \in \delta\}} w \otimes \beta[q']$
4. **Return** $\bigoplus_{\{q \in I\}} \lambda(q) \otimes \beta[q]$

Complexity: $O(|Q| + |\delta|)$（linear time）

对于 **acyclic** WFSA，可直接应用 backward algorithm。关键性质：directed acyclic graph 可做 **topological sort**。

💡 Topological sort 非唯一。对 CRF 中长度 N、$|\mathcal{T}|$ 个 tags 的 lattice，topological orderings 数量为 $N \times (|\mathcal{T}|!)$——同一 time step 内的 nodes 可任意顺序更新。

与 CRF 版本对比：这里只需 topological order，不再显式遍历 time steps 和 tags。抽象使一切更简单, which is worth it.

3.8 Closed Semiring 与 Kleene Star

处理 **cyclic** WFSA 需要 infinite sums。关键工具是 **Kleene star**。

Closed Semiring (revisited)
Semiring 称为 **closed** 若存在 Kleene star 运算 a^* 满足：

$$a^* = \mathbf{1} \oplus a \otimes a^*$$
$$a^* = \mathbf{1} \oplus a^* \otimes a$$

这两条公理看似抽象，实则 geometric series 天然满足：$\sum_{n \geq 0} x^n = 1 + x \cdot \sum_{n \geq 0} x^n = 1 + \left(\sum_{n \geq 0} x^n\right) \cdot x$

对 $|x| < 1$，closed form 为 $x^* = 1/(1 - x)$。

💡 **Real semiring 本身不 closed**：若 $x = 2$，则 $\sum x^n$ 发散。需扩展到 **extended reals** $\mathbb{R} \cup \{\infty\}$。

Matrix Version: 若 $\boldsymbol{M}$ 是 semiring 值矩阵，则：

$$\boldsymbol{M}^* = \sum_{n \geq 0} \boldsymbol{M}^n$$

在 real semiring 中，这收敛当且仅当 $\boldsymbol{M}$ 的 **largest eigenvalue** $< \mathbf{1}$。此时：

$$\boldsymbol{M}^* = (\boldsymbol{I} - \boldsymbol{M})^{-1}$$

这给出 cubic time algorithm（matrix inversion）。但问题是：semiring 没有 minus 和 inverse！

3.9 Lehmann's Algorithm

Lehmann's algorithm 是处理 cyclic WFSA 的通用 dynamic program，无需 inverse 操作。

⚙ Lehmann's Algorithm

Input: Weight matrix $\boldsymbol{W} \in \mathbb{K}^{|Q| \times |Q|}$, closed semiring

1. $\boldsymbol{R}^{(0)} \leftarrow \boldsymbol{W}$
2. **For** $j = 1, ..., |Q|$:
3. $\quad$ **For** $i, k = 1, ..., |Q|$:
4. $\qquad \boldsymbol{R}_{ik}^{(j)} \leftarrow \boldsymbol{R}_{ik}^{(j-1)} \oplus \boldsymbol{R}_{ij}^{(j-1)} \otimes \left(\boldsymbol{R}_{jj}^{(j-1)}\right)^* \otimes \boldsymbol{R}_{jk}^{(j-1)}$
5. **Return** $\boldsymbol{R}^{(|Q|)}$

Complexity: $O(|Q|^3)$

Lehmann's Recursion
令 $\boldsymbol{R}_{ik}^{(j)}$ 为从 q_i 到 q_k、仅经过 $\{q_1, ..., q_j\}$ 的所有 paths 的 semiring-sum。
Base case: $\boldsymbol{R}^{(0)} = \boldsymbol{W}$（direct edges）
Recursion:

$$\boldsymbol{R}_{ik}^{(j)} = \boldsymbol{R}_{ik}^{(j-1)} \oplus \boldsymbol{R}_{ij}^{(j-1)} \otimes \left(\boldsymbol{R}_{jj}^{(j-1)}\right)^* \otimes \boldsymbol{R}_{jk}^{(j-1)}$$

直观理解：经过 $\{1, ..., j\}$ 的 paths = 不经过 j 的 paths $\oplus$ 经过 j 的 paths。后者分解为：$i \to j$（不经过 j），j 上的 cycles，$j \to k$（不经过 j）。

3.9.1 Floyd-Warshall 作为特例

Floyd-Warshall 是 Lehmann's algorithm 在 tropical semiring $\langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0 \rangle$ 下的特例。

💡 **关键观察**：Floyd-Warshall 中没有 Kleene star！因为在 shortest path 问题中，**走 cycle 永远使路径更长**，所以 $a^* = 0$（identity of min）。这也解释了为何 Floyd-Warshall 存在：它比 naive $|\mathcal{V}|^2$ 次 Dijkstra 快 $O(|\mathcal{V}|)$ 倍。

3.9.2 Gauss-Jordan 作为特例

在 real semiring 中，Lehmann's algorithm 等价于 Gauss-Jordan elimination（matrix inversion）：

$$\boldsymbol{M}^* = (\boldsymbol{I} - \boldsymbol{M})^{-1}$$

推导（注意公理 2 的使用）：

$$\boldsymbol{M}^* = \boldsymbol{I} + \boldsymbol{M}^* \otimes \boldsymbol{M} \quad \text{(Axiom 2)}$$
$$\boldsymbol{M}^* - \boldsymbol{M}^* \boldsymbol{M} = \boldsymbol{I}, \quad \boldsymbol{M}^*(\boldsymbol{I} - \boldsymbol{M}) = \boldsymbol{I}. \blacksquare$$

3.9.3 Kleene's Algorithm 作为特例

将 Lehmann 应用于 **regular expression semiring**（$\oplus$ = union, $\otimes$ = concatenation），得到 FSA → regex 转换 algorithm。这是 Kleene's theorem 的构造性证明。

3.10 Path Sum 计算总结

给定 WFSA $\mathcal{A}$，计算 $Z(\mathcal{A})$：

1. 构造 symbol-specific transition matrices $\boldsymbol{W}_a$
2. Sum 得 $\boldsymbol{W} = \bigoplus_a \boldsymbol{W}_a$
3. 应用 Lehmann's algorithm 得 $\boldsymbol{R}^{(|Q|)}$
4. $Z = \bigoplus_{q_i \in I, q_f \in F} \lambda(q_i) \otimes \boldsymbol{R}_{if}^{(|Q|)} \otimes \rho(q_f)$

对 transliteration：compose transducers，然后用 Lehmann 计算 Z。可 backprop 训练，用 Viterbi semiring 做 inference。

4 Constituency Parsing

4.1 Syntax 与 Hierarchical Structure

Syntax 是 sentence structure 的数学研究，或曰 word order 的研究。核心事实：

💡 **Language is structured hierarchically.** 这是 overwhelming evidence 支持的事实，非假设。

4.1.1 Constituency

Constituent 是在 hierarchical structure 中作为 single unit 运作的 word group。

Example: Constituency

- John speaks **Spanish** fluently. → John speaks **Chinese** fluently. ✅
- Mary programs the homework **in the lab**. → Mary programs the homework **for eternity**. ✅

可替换的部分即为 constituent。

4.1.2 Syntactic Ambiguity

同一 string 可对应多个 parse trees，产生不同 meanings。

Classic Ambiguity Examples

"Fruit flies like..."
- [Fruit flies] [like]... — 果蝇喜欢
- [Fruit] [flies like]... — 水果像...飞

"...elephant in my pajamas"
- PP attach high — 我穿睡衣
- PP attach low — 大象穿睡衣

Modifier **scope**
- "plastic cup holder"
- "plastic-cup holder"

这些 ambiguities 是**事实**（data）。我们通过 introspection 收集, which is linguistics 独特之处。

4.1.3 Constituency Tests

判定 constituent 的 linguistic 测试：

1. **Pronoun replacement**: "Eleanor ate [the pad thai]" → "Eleanor ate **it**"
2. **Clefting**: "John loves [the red car]" → "**It is [the red car]** that John loves"
3. **Pro-form substitution**: "Papa eats caviar [with a spoon]" → "**How** does Papa eat caviar?"

💡 Clefting 可消解歧义：
- "It is **the fruit** that flies like a green banana" — 只保留解读 2
- 这说明 syntactic operations 作用于 tree structure，而非 string

4.1.4 与 Programming Languages 对比

| | Programming Lang | Natural Lang | |---|---|---| | Constituents | 明确标记（brackets） | 隐式 | | Parsing | Linear time | Cubic time | | Ambiguity | 设计上避免 | 无处不在 | | Grammar | 已知 | 需 reverse engineer |

4.2 Context-Free Grammars

Context-Free Grammar (CFG)

四元组 $\langle \mathcal{N}, S, \Sigma, \mathcal{R} \rangle$：
- $\mathcal{N}$: non-terminal symbols（大写字母）
- $S \in \mathcal{N}$: start symbol
- Σ: terminal symbols（小写字母）
- $\mathcal{R}$: production rules，形如 $N \to \boldsymbol{\alpha}$，$\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$

String w 属于 language 当且仅当存在从 S 开始 yield w 的 derivation。

称"context-free"是因为 rule 的应用不依赖左右 context——N 无论出现在哪都可被替换。

💡 **CFG 是 model，非 ground truth。**
- 它是解释 linguistic data 的工具，非大脑中真实存在的结构
- Tree annotations 是某人的 modelling choice
- 切勿将 treebank 视为"ground truth"

4.2.1 Ambiguous Grammars

若同一 string 有多于一个 derivation tree，则 grammar 是 ambiguous 的。

Parse Tree Factorization

给定 CFG，tree 的 probability/score 分解到 rules：

$$\mathrm{score}(\boldsymbol{t}, \boldsymbol{w}) = \sum_{r \in \boldsymbol{t}} \mathrm{score}(r)$$

即 tree 只是 rules 的 **multiset**——任何两棵用相同 rules（含重数）的 tree 有相同 score。

4.3 Probabilistic & Weighted CFGs

PCFG

五元组 $\langle \mathcal{N}, S, \Sigma, \mathcal{R}, p \rangle$，其中 $p : \mathcal{R} \to [0, 1]$ 是 locally normalized distribution：

$$\forall N \in \mathcal{N} : \sum_{N \to \alpha \in \mathcal{R}} p(N \to \alpha) = 1$$

问题：PCFG 可能 non-tight（类似 LM 的问题）；可能有 infinite trees for one string。

Weighted CFG (WCFG)

用任意 non-negative weights 替代 probabilities。Globally normalized：

$$p(\boldsymbol{t} \mid \boldsymbol{w}) = \frac{\exp \mathrm{score}(\boldsymbol{t}, \boldsymbol{w})}{\sum_{\boldsymbol{t}' : \, \mathrm{yield}(\boldsymbol{t}') = \boldsymbol{w}} \exp \mathrm{score}(\boldsymbol{t}', \boldsymbol{w})}$$

问题：$Z(\boldsymbol{w})$ 可能发散！

Divergence Example

Rules: $S \to S$ (weight 1), $S \to a$ (weight 1)

String "a" 有无穷多 trees（$S \to S \to \ldots \to S \to a$），每棵 weight 1。$Z$ 发散。

4.4 Chomsky Normal Form

Chomsky Normal Form (CNF)

所有 rules 形如：

- $N_1 \to N_2 N_3$（binary branching）
- $N \to a$（terminal emission）
- $S \to \varepsilon$（仅对 start symbol，仅当 $\varepsilon \in L(G)$）

CNF Theorem
任何 CFG G 可转换为 CNF grammar G'，使得 $L(G') = L(G)$（或 $L(G') = L(G) - \{\varepsilon\}$）。prob 也可保持。

CNF 的关键后果：
1. **Decidability**：长度 N 的 string 的 trees 数量有限（历史上证明 CFL membership decidable 的关键步骤）
2. **Tree size fixed**：长度 N 的 string 对应的 binary tree 有 $2N-1$ 个 nodes
3. **No cycles**：不存在 $N \to \dots \to N$ 的 chain

Trees 数量虽有限但仍指数级：**Catalan number** $C_N \approx O\left(\frac{4^N}{N^{\frac{3}{2}}}\right)$。

4.5 Parsing Problem
给定 sentence $\boldsymbol{w}$，求 distribution over trees with yield $\boldsymbol{w}$：

$$p(\boldsymbol{t} \mid \boldsymbol{w}) = \frac{\exp \operatorname{score}(\boldsymbol{t}, \boldsymbol{w})}{Z(\boldsymbol{w})}, \quad Z(\boldsymbol{w}) = \sum_{\boldsymbol{t}:\ \operatorname{yield}(\boldsymbol{t}) = \boldsymbol{w}} \exp \operatorname{score}(\boldsymbol{t}, \boldsymbol{w})$$

与 CRF 的类比：| CRF | Parsing | |—|—| | 给定 $\boldsymbol{w}$，distribution over $\boldsymbol{t} \in \mathcal{T}^N$ | 给定 $\boldsymbol{w}$，distribution over trees yielding $\boldsymbol{w}$ | | Score 分解到 bigrams | Score 分解到 rules |

关键：我们只需对**特定 string** 的 trees 求和，不需整个 WCFG 的 Z。

> 💡 与 WFSA 不同，WCFG 的 general Z 需解 **quadratic equations**（iterative methods like Newton's method），无 closed-form algo。但 per-string $Z(w)$ 可用 CKY 高效计算。

4.6 CKY Algorithm
此 algorithm 的意义：证明了 **CFL membership 可在 polynomial time 决定**。这在当时是开放问题。

4.6.1 为何需要 CKY?
Programming languages 设计上保证 linear-time parsing（unambiguous, deterministic CFG）。但 natural language 天然 ambiguous，需要更通用的 algorithm。

> 💡 **与 matrix multiplication 的关系**：存在 tight reduction——更快的 matrix multiplication → 更快的 parsing。Sub-cubic parsing 由 Leslie Valiant（PAC learning 发明者，Turing Award）给出。

Jay Earley 进一步证明：对**任意** CFG（非 CNF），可达到 $O(N^3\ |G|)$ 而非 $O(N^3\ |G'|)$（G' 是 CNF 转换后可能变大的 grammar）。

4.6.2 Algorithm 核心思想
Span：sentence 的 contiguous substring，如 “like a green” 在 “fruit flies like a green banana” 中。
Chart：dynamic programming table，Chart$[i,k,X]$ 存储 non-terminal X 覆盖 span $[i,k)$ 的所有 derivations 的 semiring-sum。
三层 for loops：
1. 初始化 length-1 spans（terminal rules）
2. 按 span length 递增枚举
3. 对每个 span，枚举所有 split points

⚙ CKY Algorithm

Input: Sentence $\boldsymbol{w} = w_1 \dots w_N$, CNF grammar $\langle \mathcal{N}, S, \Sigma, \mathcal{R} \rangle$, scoring function

$\boldsymbol{C} \leftarrow \boldsymbol{0}$
For $i = 1, \dots, N$:
  For $X \to w_i \in \mathcal{R}$:
    $\boldsymbol{C}[i, i+1, X] \leftarrow \boldsymbol{C}[i, i+1, X] \oplus \exp(\operatorname{score}(X \to w_i))$
For $\ell = 2, \dots, N$:
  For $i = 1, \dots, N - \ell + 1$:
    $k \leftarrow i + \ell$
    For $j = i+1, \dots, k-1$:
      For $X \to Y Z \in \mathcal{R}$:
        $\boldsymbol{C}[i,k,X] \leftarrow \boldsymbol{C}[i,k,X] \oplus \exp(\operatorname{score}(X \to YZ)) \otimes \boldsymbol{C}[i,j,Y] \otimes \boldsymbol{C}[j,k,Z]$
Return $\boldsymbol{C}[1, N+1, S]$

Complexity: $O(N^3\ |\mathcal{R}|)$

> 💡 **CNF 在哪里体现？** 每个 span 由恰好 2 个 sub-spans 组成（binary branching）。若允许 k 个 children，complexity 变为 $O(N^{k+1})$——这就是为何需要 CNF。

4.6.3 CKY 详细示例
⚠ **考试常见**：手工填写 CKY chart。TA 强调“做 3-4 遍就记住了”。

CKY Chart 索引

Chart 索引 $\boldsymbol{C}[i,k,X]$：
- i: span 起点（word 之前的 position）
- k: span 终点（word 之后的 position）
- X: non-terminal

Position 在 **words 之间**：0 | fruit | 1 | flies | 2 | like | 3 | ...
Span $[i,k)$ 覆盖 words $w_i, \dots, w_{k-1}$，其长度 = $k - i$

⚙ CKY 填表步骤

1. 初始化 (length-1)

2. 递推 ($\ell = 2..N$)

3. 结果

对每个 w_i，查找 $X \to w_i$：
$C[i, i+1, X] = \text{score}(X \to w_i)$

枚举 i, k, j，对 $X \to YZ$：
$C[i, k, X] \leftarrow C[i, k, X] \oplus$
$\text{score} \otimes C[i, j, Y] \otimes C[j, k, Z]$

$C[0, N, S]$
完整 sentence 被 start symbol 覆盖

💡 **填表顺序**：按 span 长度递增 ($\ell = 1, 2, ..., N$); 遍历顺序：**同一 length 内任意**（topological order 的自由度）。

- **对角线** ($\ell = 1$)：词性标注
- $C[0, 1] = \{N\} \leftarrow$ "fruit"
- $C[1, 2] = \{N, V\} \leftarrow$ "flies"

- **上三角** ($\ell \geq 2$)：组合规则
- $C[0, 2] = \{NP\} \leftarrow N + N$
- $C[0, 5] = \{S\} \leftarrow$ 最终结果 ✓

i \ j	**1** fruit	**2** flies	**3** like	**4** a	**5** banana
0	N	NP (N N)	S (NP VP)		S (NP VP)
1		N, V	VP (V NP)	VP (V NP)	VP
2			V, P	PP (P NP)	PP, VP
3				Det	NP (Det N)
4					N

Table 4: Syntax Parsing 填表例子

4.6.4 Catalan Number 与 Parse Trees 数量

Catalan Number
长度 N 的 string 的 binary parse trees 数量：

$$C_N = (1, N+1)\binom{2N}{N} \approx \left(4^N, N^{\frac{3}{2}}\sqrt{\pi}\right)$$

, $C_1 = 1, C_2 = 2, C_3 = 5, C_4 = 14, C_5 = 42, C_{10} = 16796$

这解释了为何 CKY 必要：即使只有 5 个 words，也有 42 种可能的 binary tree structures。

4.6.5 CNF 转换

💡 CKY 要求 CNF。转换步骤繁琐 & 机械：
1. 消除 ε-productions（除 $S \to \varepsilon$）
2. 消除 unit productions（$A \to B$）
3. 将长 RHS 拆成 binary（引入新 non-terminals）
4. 将 terminals 与 non-terminals 混合的 RHS 分离

Example
VP → V NP PP 不是 CNF（3 个 children）
转换：
- 引入 VP′ → NP PP
- 改为 VP → V VP′

现在两条 rules 都是 binary。

4.6.6 CRF 与 CFG 的对应

⚠ **Exercise 考点**：将 CRF 写成 CFG 形式，理解两者结构对应。

CRF 是一种 **right-recursive CFG**：
结果：$O(|\mathcal{T}|^2)$ 条 transition rules，与 CRF 的 transition matrix 对应。

CRF as CFG
给定 tag set $\mathcal{T}$，构造 CFG：
- Non-terminals: B_t for each $t \in \mathcal{T}$, plus S
- Rules:
  - $S \to B_t$ for each $t \in \mathcal{T}$（起始）
  - $B_t \to A_t B_{\{t'\}}$ for each $t, t' \in \mathcal{T}$（transition）
  - $A_t \to w$ for each word w, tag t（emission）

这强制 *linear* structure——parse tree 必须是 right-branching chain。

4.6.7 Topological Order 视角

CKY 的 for loops 实际上是在遍历一个 **generalized topological order**：
- 枚举所有 triples (i, j, k) 满足 $0 < i < j < k \leq N$
- 按 span length $k - i$ 递增
- 同一 length 内，任意顺序皆可

这与 CRF 中同一 time step 内 tags 可任意顺序更新是同一 insight。

4.6.8 Semiring 化与 Viterbi

CKY 可用任意 semiring：
- **Real semiring**：计算 $Z(w)$（normalizer）
- **Viterbi semiring**：找 best parse（配合 backpointers）
- **Entropy semiring**：计算 parse distribution 的 entropy

4.6.9 Training

Scoring function 可以是任意 neural network。Training 方式与 CRF 相同：

$$\mathcal{L} = \sum_{(\boldsymbol{w}, \boldsymbol{t}) \in \mathcal{D}} [\text{score}(\boldsymbol{t}, \boldsymbol{w}) - \log Z(\boldsymbol{w})]$$

对 CKY forward pass 做 backprop 即可求 gradient。

✅ **与 Assignment 2 的联系**：Given 已经用 semiring 算过 entropy。同样的 semiring 直接 plug into CKY 即可算 parse trees 的 entropy。这就是 abstraction 的 power。

4.6.10 Weighted CKY 与 Semirings

与 CRF 相同，CKY 可用不同 semirings：

Semiring	操作 $(\oplus, \otimes)$[4]	计算内容 / 应用场景[5]
Real/Probability	$(+, \times)$	$Z(\boldsymbol{w})$ (partition function/normalizer)；前向-后向算法
Tropical/Viterbi	$(\max, \times)$	最优路径/解析树；Viterbi 算法；最大似然解码
Log	$(\text{logsumexp}, +)$	$\log Z(\boldsymbol{w})$；数值稳定的 prob 计算；避免下溢
Boolean	$(\vee, \wedge)$	是否存在有效路径；可达性判断；语法解析存在性
Counting	$(+, \times)$	路径/推导数量；歧义度计算；派生树计数
k-best Tropical	$(\max_k, \times)$	Top-k 最优路径；k-best Viterbi；束 search (beam search)
Expectation	$(+, \times)$ over $\mathbb{R} \times \mathbb{R}$	特征期望；梯度计算；EM 算法 E-step
MinPlus/Tropical	$(\min, +)$	最短路径；编辑距离；CKY 最小代价解析
Inside	$(+, \times)$	Inside prob；PCFG 内向算法；子树 prob
Outside	$(+, \times)$	Outside prob；PCFG 外向算法；上下文 prob
Entropy	特殊组合	Shannon 熵计算；不确定性度量；模型置信度
Risk/Loss	$(+, \times)$ with loss	期望风险；最小贝叶斯风险解码；损失感知训练

Table 5: Semiring 及其应用

5 Dependency Parsing

5.1 Dependency Grammar 简介

Dependency grammar 是 constituency grammar 的替代传统——两者都是 **models**（某人对 language structure 的 opinion），解释不同 phenomena，可互补。
核心思想：sentence 中每个 word 与其 **syntactic head** 连接，形成 directed tree。

> Dependency Tree
> - 每个 word 有唯一 parent（head）
> - 有唯一 root（通常是 main verb）
> - Edges 带 grammatical relation labels（subject, object, etc.）

"The boy eats Rösti"
Relations: boy $\xrightarrow{\text{subj}}$ eats, Rösti $\xrightarrow{\text{obj}}$ eats, The $\xrightarrow{\text{det}}$ boy

5.1.1 为何 Dependency 与 Function Application 相关?

Verb 可视为 function，arguments 是其 dependents：

$$\text{eats} = \lambda y.\lambda x.\ \text{Eats}(x, y)$$

应用后：$\text{eats}(\text{Rösti})(\text{boy}) = \text{Eats}(\text{boy}, \text{Rösti})$
这是 **argument structure** 的浅层建模——verb 接受哪些 arguments、如何标记（preposition, case marking 等）。

5.1.2 Dependency vs Constituency

Aspect	Constituency	Dependency
基本单位	Phrases (constituents)	Head-dependent pairs
标注内容	Phrase structure	Grammatical relations
信息	Hierarchy, scope	Head selection, valency
相互关系	可互相转换	但 tree 结构不同

从 constituency tree 提取 dependency tree：为每个 rule 指定 head child（Collins' rules），然后向上传播 head。

[4]关键性质 **幺元 (Identity)**：加法幺元 $\mathbf{0}$，乘法幺元 $\mathbf{1}$；**零元 (Annihilator)**：加法零元使 $a \otimes \mathbf{0} = \mathbf{0}$；**分配律**：$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$；**结合律**：两个操作都满足结合律；**交换律**：加法操作通常满足交换律（乘法不一定）

[5]NLP 应用示例：CRF/HMM：使用 Log 半环进行训练，Viterbi 半环进行解码；PCFG：Inside-Outside 算法使用 Real 半环；神经网络：前向传播使用 Real 半环，反向传播涉及 Expectation 半环；机器翻译：束 search 使用 k-best 半环；依存句法分析：最大生成树使用 MaxPlus 半环

5.1.3 Projectivity

Projective Dependency Tree
若将 arcs 画在 words 上方，**没有 crossing arcs**，则称 tree 是 **projective** 的。

- **Projective**：与 constituency structure 紧密相关，可用 CKY 变体 parse
- **Non-projective**：有 crossing arcs，需要不同 algorithms

Non-projectivity Example
"Ryan ate Rösti for lunch, which was delicious."
"which" 修饰 "Rösti"，但 "for lunch" 介于两者之间。Arc (Rösti, which) 跨过 arc (ate, for)。
这是英语中的 **non-contiguous constituent** 现象——CFG 假设开始 break。

大多数语言 **nearly projective**——non-projective structures 存在但相对稀少。

5.1.4 Universal Dependencies

UD (Universal Dependencies) 是跨语言的 dependency annotation 标准，覆盖数百种语言。

> 💡 **为何 dependency parsing 如此流行?** 主要是 **convenience**。UD 提供免费、统一格式的 data，便于 benchmark。相比之下，constituency treebanks 多在 paywall 后且格式各异。
> 这是 NLP 的社会学现象：data 的 accessibility 极大影响 research 方向。

5.2 Probability Model for Non-Projective Trees

目标：给定 sentence $\boldsymbol{w}$，定义 spanning trees 上的 distribution。
问题规模：N 个 nodes 的 directed spanning trees 数量是 $(N-1)^{N-2}$（Cayley's formula 的 directed 版本）——比 parse trees 的 Catalan number 还大。

5.2.1 Edge Factorization

Edge-Factored Model
假设 scoring function 分解到 edges：

$$\text{score}(\boldsymbol{t}, \boldsymbol{w}) = \text{score}(r, \boldsymbol{w}) + \sum_{(i \to j) \in \boldsymbol{t}} \text{score}(i, j, \boldsymbol{w})$$

其中 r 是 root choice。

将 scores 组织成 matrices：
- $\boldsymbol{A}_{ij} = \exp \text{score}(i, j, \boldsymbol{w})$：weighted adjacency matrix
- $\boldsymbol{\rho}_j = \exp \text{score}(r = j, \boldsymbol{w})$：root scores

> 💡 **为何不能更强?** Edge factorization 是能保持 tractability 的 **strongest** assumption。若允许 second-order（同时看两条 edges），可 encode Hamiltonian path problem（NP-hard）。

5.3 Matrix-Tree Theorem

5.3.1 Root Convention

> 💡 Dependency tree 引入 external root node（不在 sentence 内），有一条 arc 指向 sentence 的 syntactic head（通常是 main verb）。这让 root choice 也变成普通的 edge choice。

两种等价写法：
- Root as special node 0：edges $0 \to j$ 表示 w_j 被选为 root
- Root scores vector $\boldsymbol{\rho}$：$\boldsymbol{\rho}_j = \exp \text{score}(r = j, \boldsymbol{w})$

两者最终都落到对 Laplacian $\boldsymbol{L}$ 第一行的修改（Koo et al. trick）。

5.3.2 Arc Scoring: First-Order vs Higher-Order

First-Order (Arc-Factored)
Score 仅依赖单条 arc：
$\text{score}(\boldsymbol{t}, \boldsymbol{w}) =$
$\text{score}(r, \boldsymbol{w}) + \sum_{(i \to j) \in \boldsymbol{t}} \text{score}(i, j, \boldsymbol{w})$
其中 i = head, j = dependent。Complexity: $O(N^2)$ arcs。

Second-Order: Grandparent
Score 还依赖 grandparent g（i 的 parent）：

$$\text{score}(\boldsymbol{t}) = \sum_{(g \to i \to j) \in \boldsymbol{t}} \text{score}(g, i, j, \boldsymbol{w})$$

Example: "eat a red apple" 则 Arc: apple → red; Grandparent: eat（因为 eat → apple）

Second-Order: Sibling
Score 还依赖 sibling s（同一 head 下的其他 dependents）：

$$\text{score}(\boldsymbol{t}) = \sum_{(i \to j, i \to s) \in \boldsymbol{t}} \text{score}(i, j, s, \boldsymbol{w})$$

Example: "eat an apple and an orange" 则 Head: eat; Siblings: apple, orange

5.3.3 Complexity 分析与 Extreme Structures

Arc Scoring Templates
Extreme Trees for Complexity 直觉:
一般来说 First-order: $\boldsymbol{O(N^2)}$; Second-order: $\boldsymbol{O(N^3)}$

Flat tree：one head with $N-1$ dependents
- First-order: $O(N)$ arcs
- Sibling: $O(N^2)$（每条 arc 有 $N-2$ siblings）

Chain tree：each node has exactly one child (a path)
- First-order: $O(N)$ arcs
- Grandparent: $O(N)$（每条 arc 只有 1 grandparent）

> 💡 为何 first-order 够用？Neural scoring function（BiLSTM/Transformer）已在 input representation 中编码丰富 context。Arc-factored assumption 作用于 scoring，不限制 representation。Edge-factored 是"还能做 exact inference"的最强可用假设。

5.3.4 MTT 证明结构与 Sanity Checks

✅ **Assignment 5 的证明路线图**

Step 1: Undirected Case (Kirchhoff)
- 构造 adjacency matrix $\boldsymbol{A}$
- 构造 Laplacian $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$
- 证明 $\det(\hat{\boldsymbol{L}}_i)$ = spanning trees 数量

Step 2: Directed Case (Tutte)
- 修改 Laplacian 为 directed version
- 加入 root constraint (Koo et al.)

Laplacian Matrix 构造
Undirected：

$$L_{ij} = \begin{cases} \sum_k \boldsymbol{A}_{ik} & \text{if } i = j \text{ (degree)} \\ -\boldsymbol{A}_{ij} & \text{otherwise} \end{cases}$$

Directed with root (课程写法，incoming sum on diagonal)：

$$\boldsymbol{L}_{ij} = \begin{cases} \sum_{k \neq j} \boldsymbol{A}_{kj} & \text{if } i = j \\ -\boldsymbol{A}_{ij} & \text{if } i \neq j \end{cases}$$

注入 root scores：$\boldsymbol{L}_{1,j} \leftarrow \boldsymbol{\rho}_j$

Sanity Checks:

1. Undirected：列和为 0
$\det(\boldsymbol{L}) = 0$，需 cofactor

2. Directed + root：
$\det(\boldsymbol{L}) \neq 0$ 即 $Z(\boldsymbol{w})$

3. 若 undirected det≠0
→ 构造错误

MTT Workflow (Mechanical)

1. Build Laplacian：
diag = col sum
off-diag = $-\boldsymbol{A}_{ij}$

2. Inject root：
$\boldsymbol{L}_{1,j} \leftarrow \boldsymbol{\rho}_j$

3. Partition function：
$Z(\boldsymbol{w}) = \det(\boldsymbol{L})$

5.3.5 Chu-Liu-Edmonds 详解

CLE Hand-Run Checklist

Repeat until no cycle:
1. Greedy step：对每个 non-root node，选 highest incoming arc
2. Cycle detection：检查 greedy graph 是否有 cycle
3. If no cycle：done
4. If cycle exists：
   - Contract cycle into super-node c
   - Reweight entering edges
5. Recurse on contracted graph
6. Expand cycles using recorded choices

Reweighting 公式

设 cycle C 内节点 v 的 best incoming arc 权重为 w_v。
对外部节点 u 到 $v \in C$ 的 arc：

$$\text{new\_weight}(u, v) = \text{weight}(u, v) - w_v$$

直觉：选择 (u, v) 意味着放弃 v 在 cycle 内的 arc。Reweight 确保 total cost 正确。

Root Constraint 处理

CLE base version 允许 root 有多条 outgoing arcs，但 dependency parsing 要求 root 只有 1 outgoing。
Naive：对每条 root arc 分别运行 CLE，取最优。Complexity $O(N \cdot \text{CLE})$。
Clever (Gabow et al.)：计算 swap score = next-best incoming - current incoming，删除 swap score 最小的多余 root edge。

5.3.6 Arc Scoring Functions (Implementation)

Arc Scoring Templates

Let $\boldsymbol{h}_i$ be representation of word i (from BiLSTM/Transformer encoder).
Then set $\boldsymbol{A}_{ij} = \exp \text{score}(i, j, \boldsymbol{w})$

Common choices:
- Bilinear: $\text{score}(i, j) = \boldsymbol{h}_i^\top \boldsymbol{W} \boldsymbol{h}_j$
- MLP: $\text{score}(i, j) = \boldsymbol{v}^\top \tanh(\boldsymbol{W}_h \boldsymbol{h}_i + \boldsymbol{W}_d \boldsymbol{h}_j)$

💡 If you add label types ℓ (subj/obj/etc):
$\text{score}(i, j) = \max_{\ell} \text{score}(i, j, \ell)$
或在 decoding 时 explicitly 保留 labels。

5.3.7 与 CRF 的结构对比

Aspect	CRF (Sequence)	Dependency Parsing
Structure	Linear chain	Tree
Z computation	Forward algorithm	Determinant (MTT)
Complexity	$O(N\ \|\mathcal{T}\|^2)$	$O(N^3)$
Inference	Viterbi	CLE
Factorization	Edge-factored (transitions)	Arc-factored

5.3.8 Kirchhoff's Theorem (Undirected Case)

Kirchhoff's Matrix-Tree Theorem

给定 undirected graph $\mathcal{G}$，令 Laplacian matrix：

$$\boldsymbol{L}_{ij} = \begin{cases} -\boldsymbol{A}_{ij} & \text{if } i \neq j \\ \sum_{k \neq i} \boldsymbol{A}_{kj} & \text{if } i = j \end{cases}$$

则 spanning trees 数量 $= \det(\hat{\boldsymbol{L}}_i)$，其中 $\hat{\boldsymbol{L}}_i$ 是删去第 i 行列后的 matrix。

5.3.9 Tutte's Extension (Directed & Weighted)

Tutte 推广到 **directed weighted** graphs：

$$Z(\boldsymbol{w}) = \det(\boldsymbol{L})$$

其中 $\boldsymbol{L}$ 用 directed adjacency matrix 构造。

💡 Undirected case 中 $\boldsymbol{A}$ 是 symmetric；directed case 不对称。

5.3.10 Adding Root Constraint (Koo et al., 2007)

为满足 single-root constraint，修改 Laplacian：

$$\boldsymbol{L}_{ij} = \begin{cases} \boldsymbol{\rho}_j & \text{if } i = 1 \\ -\boldsymbol{A}_{ij} & \text{if } i \neq j \\ \sum_{k \neq i} \boldsymbol{A}_{kj} & \text{otherwise} \end{cases}$$

结论：$Z(\boldsymbol{w}) = \det(\boldsymbol{L})$，complexity $O(N^3)$（determinant computation）。

💡 **魔法公式**：整个 normalizer 就是一个 matrix determinant。这不是 dynamic program，而是 linear algebra。
Semiring 问题：Determinant 需要 **subtraction**（Laplacian 定义中有负号）。若没有 subtraction，需 exponential time。这就是为何无法 semiringify。

5.4 Inference: Chu-Liu-Edmonds Algorithm

Matrix-tree theorem 算 Z，但不给 argmax。需要另一个 algorithm 找 best tree。

5.4.1 为何 Kruskal 不 work?

Kruskal's algorithm（greedy add edges，不成 cycle）对 **undirected** minimum spanning tree 有效。
但 directed case 失败：

Greedy 失败示例

Greedy 选 highest incoming edge to each node，得到 score 7 的 tree。但 optimal tree score 是 10。
原因：directed edges 有 constraints（每个 node 最多一条 incoming edge），locally optimal choices 可能 block globally optimal solutions。

5.4.2 Algorithm Overview

Chu-Liu-Edmonds (1965) / Edmonds (1967)：

⚙ **Chu-Liu-Edmonds**

1. **Greedy graph**：每个 non-root node 选 highest incoming edge
2. **If** no cycle: done（greedy graph 即 optimal）
3. **If** cycle exists:
   - **Contract** cycle 成单个 node
   - **Reweight** entering edges：新 weight = 原 weight + (被替换的 cycle edge 的 weight)
   - **Recurse** on contracted graph
4. **Expand** contracted cycles，得到 final tree

5.4.3 Edge Cataloging

对 contracted node c：

- **Dead edges**：cycle 内部的 edges（已处理）
- **External edges**：不涉及 cycle 的 edges
- **Enter edges**：进入 cycle 的 edges（需 reweight）
- **Exit edges**：离开 cycle 的 edges

5.4.5 Complexity

- Edmonds' original: $O(N^3)$ 或 $O(MN)$
- Tarjan's improvement: $O(N^2)$ 或 $O(M \log N)$

5.4.4 Root Constraint Handling

Naive：对每个可能的 root edge 分别运行 algorithm，比较结果。Complexity 增加 factor of N。

Clever（Gabow et al.）：在 greedy graph 中若 root 有多条 outgoing edges，删除 **swap score** 最小的（swap score = next-best incoming edge - current incoming edge）。

✅ **非 Dynamic Program**：这是 assignment 中唯一一个非 DP 的 algorithm。无法 semiringify——想要不同的 computation（如 entropy）需要用 Matrix-Tree Theorem 的 gradient tricks。

6 Semantic Parsing

6.1 什么是 Meaning?

Syntax 研究 sentence structure；semantics 研究 meaning, which is a philosophical question.

Truth-Conditional Semantics

理解一个 expression 的 meaning，即知道它在何种条件下为 true。类比数学：理解 $F = F(x)$ 的 meaning，即知道哪些 F 使之为 true/false。

为何必须成立？我们能理解 unheard, novel sentences 从;这只可能因为其由可重用的 parts 组成; Plagiarism detection 的基础：language 太 expressive，独立产生相同句子的 prob 极低

Example: Quantifier Scope Ambiguity

"Everybody loves somebody else" 有两个 readings：

1. $\forall p[\text{Person}(p) \to \exists q[\text{Person}(q) \land p \neq q \land \text{Loves}(p, q)]]$
   - 每个人都有（可能不同的）某个他们爱的人
2. $\exists q[\text{Person}(q) \land \forall p[\text{Person}(p) \land p \neq q \to \text{Loves}(p, q)]]$
   - 存在某个特定的人，被所有人爱

这是 **semantic ambiguity**——非 lexical（词义歧义）、非 syntactic（结构歧义），而是 quantifier scope 的歧义。

6.1.1 Logical Form

Logical form 是 meaning 的形式化表示[6]

关键：logical form 是 **可执行/可求值** 的——可以判断 true/false 或产生 action。

💡 如果学过 programming language theory：这与 PL 中的 denotational semantics 是同一思想——通过 evaluation 定义 meaning。

6.2 Principle of Compositionality

Frege's Principle of Compositionality

The meaning of a complex expression is a **function** of the meanings of its constituent parts.

为何必须成立？我们能理解 **novel sentences**（从未听过的句子）；这只可能因为 sentence 由可重用的 parts 组成; Plagiarism detection 的基础：language 太 expressive，独立产生相同句子的 prob 极低

💡 **Idioms** 是例外（如 "kick the bucket" = die），但：

- 可以修改 idiom："He kicked the bucket **yesterday**"
- 打破 idiom："He kicked the **red** bucket" 失去 idiomatic reading
- Idioms 是更大的 lexical units，compositionality 仍在更高层面成立

6.3 Lambda Calculus

Lambda calculus (Church, 1932) 是 computation 的形式化 model，与 Turing machine 等价（Church-Turing thesis）。
对我们而言，它主要是 **notation for anonymous functions**——Python 中的 `lambda` 即来源于此。

6.3.1 语法定义

Lambda Calculus Terms

Terms 归纳定义：

Base case: Variables $x, y, z, \ldots$ 是 terms

Recursive rules:

- **Abstraction:** 若 M 是 term，x 是 variable，则 $\lambda x.M$ 是 term
- **Application:** 若 M, N 是 terms，则 (MN) 是 term（M applied to N）

Scope Example

$(\lambda x.x(\lambda x.x)x)$

第 1、3 个 $x \to$ 外层 λx；第 2 个 $x \to$ 内层 λx

类似 Python 嵌套 function 的 variable scoping——内层 scope 遮蔽外层

6.3.2 Free vs Bound Variables

Free and Bound Variables

Bound：在某个 λ 的 scope 内

Free：不在任何 abstraction 的 scope 内

归纳定义:

- $\text{FV}(x) = \{x\}$
- $\text{FV}(\lambda x.M) = \text{FV}(M) - \{x\}$
- $\text{FV}(MN) = \text{FV}(M) \cup \text{FV}(N)$

💡 **为何需要 free variables?**

[6] 还可以是 First-order logic formulas, Lambda calculus expressions, SQL queries, Python code, Robot commands

虽然最终我们关心 **closed terms**（无 free variables），但定义 semantics 时必须处理 sub-expressions，而 sub-expressions 自然包含 free variables。这是 compositionality 在 formalism 中的体现。

6.3.3 Alpha Conversion

α-conversion
重命名 bound variable 及其所有 bound occurrences：

$$\lambda x.M \underset{\alpha}{\to} \lambda y.M[x := y]$$

条件：y 不能是 M 中的 free variable（否则会改变 meaning）。

需要 α-conversion 的情况
$(\lambda x.\lambda y.xy)y$ 直接 β-reduce 会得到 $\lambda y.yy$——但原本外层的 y 是 free 的，现在变 bound 了！
正确做法：先 α-convert 内层 $\lambda y.xy \underset{\alpha}{\to} \lambda z.xz$，再 reduce。另外 $\lambda x.x \underset{\alpha}{\to} \lambda y.y$ 这种是无所谓的✅。

6.3.4 Beta Reduction

β-Reduction
Function application——将 argument 代入 function body：

$$(\lambda x.M)N \underset{\beta}{\to} M[x := N]$$

即：找到 M 中所有被该 λx 绑定的 x，替换为 N。

Example
$(\lambda x.\lambda y.xy)z \quad \underset{\beta}{\to} \lambda y.zy$ （x 被替换为 z）

⚠️ Ryan 明确说会考 beta reduction。给定 Lambda expression，simplify it（反复 apply α-conversion 和 β-reduction 直到无法继续）。TA 也在 tutorial 中强调了至少是 10 分题。

6.3.5 Lambda Calculus 实战

💡 TA 建议的应试技巧：

1. 写 Lambda calculus 时**使用不同变量名**，避免 α-conversion
2. 每步标注**哪个 λ 正在 apply**
3. 检查 free variables 是否会被 capture
4. 遇到 identity function $(\lambda x.x)$ 直接替换

例题 A：无需 α-conversion
化简 $(\lambda z.z)((\lambda y.yy)(\lambda x.xa))$
Step 1: 识别最外层 application：$(\lambda z.z)$ applied to $(\ldots)$
Step 2: $(\lambda z.z)$ 是 identity function，直接返回 argument：

$$\underset{\beta}{\to} (\lambda y.yy)(\lambda x.xa)$$

Step 3: 继续 reduce：$y := (\lambda x.xa)$

$$\underset{\beta}{\to} (\lambda x.xa)(\lambda x.xa)$$

Step 4: 再次 reduce：外层 $x := (\lambda x.xa)$

$$\underset{\beta}{\to} (\lambda x.xa)a$$

Step 5: 最终：$x := a$

$$\underset{\beta}{\to} aa$$

结果：aa（两个 free variables，无法再 reduce）

例题 C：需要 α-conversion
化简 $(\lambda x.(\lambda y.y)x)((\lambda y.y)y)$
Trap：若直接把 outer y（free）代入 inner λy，会被错误 bind！
Step 1: 先 α-convert 内层 λy 为 λa：

$$(\lambda x.(\lambda a.a)x)((\lambda y.y)y)$$

Step 2: 化简 argument：$(\lambda y.y)y \underset{\beta}{\to} y$

$$(\lambda x.(\lambda a.a)x)y$$

Step 3: 代入 $x := y$：

$$(\lambda a.a)y$$

Step 4: 最终：

$$\underset{\beta}{\to} y$$

6.3.6 First-Order Logic 翻译

English Pattern	FOL Formula	Example
条件与蕴含 (Conditionals & Implications)		
"If A, then B" "A implies B"	$A \to B$	"If it rains, then the ground is wet" $\text{Rain}(x) \to \text{Wet}(x)$
"A if and only if B" "A iff B"	$A \leftrightarrow B$	"You pass iff you score ≥60" $\text{Pass}(x) \leftrightarrow \text{Score}(x) \geq 60$
"Unless A, B" "B unless A"	$\neg A \to B$ or $A \lor B$	"You fail unless you study" $\neg\,\text{Study}(x) \to \text{Fail}(x)$
逻辑连接词 (Logical Connectives)		
"A and B" "Both A and B"	$A \land B$	"It's cold and raining" $\text{Cold}() \land \text{Raining}()$
"A or B" "Either A or B"	$A \lor B$	"Tea or coffee" $\text{Tea}(x) \lor \text{Coffee}(x)$
"Not A" "It is not the case that A"	$\neg A$	"Not happy" $\neg\,\text{Happy}(x)$
"Neither A nor B"	$\neg A \land \neg B$ or $\neg(A \lor B)$	"Neither rich nor famous" $\neg\,\text{Rich}(x) \land \neg\,\text{Famous}(x)$
全称量词 (Universal Quantifiers)		
"All/Every/Each A is B" "Everyone/Everything"	$\forall x[A(x) \to B(x)]$	"All dogs bark" $\forall x[\text{Dog}(x) \to \text{Bark}(x)]$
"No A is B" "No one/Nothing"	$\neg\exists x[A(x) \land B(x)]$ or $\forall x[A(x) \to \neg B(x)]$	"No student is lazy" $\neg\exists x[\text{Student}(x) \land \text{Lazy}(x)]$
"Only A is B"	$\forall x[B(x) \to A(x)]$	"Only students can register" $\forall x[\text{Register}(x) \to \text{Student}(x)]$
存在量词 (Existential Quantifiers)		

English Pattern	FOL Formula	Example
"Some/A/An A is B" "Someone/Something"	$\exists x[A(x) \wedge B(x)]$	"Some student is smart" $\exists x[\text{Student}(x) \wedge \text{Smart}(x)]$
"There exists/is an A"	$\exists x[A(x)]$	"There exists a solution" $\exists x[\text{Solution}(x)]$
"At least n A's"	$\exists x_1 ... x_n \left[\bigwedge_{i=1}^{n} A(x_i) \wedge \bigwedge_{i<j} x_i \neq x_j\right]$	"At least 2 witnesses" $\exists x_1 x_2[\text{Witness}(x_1) \wedge \text{Witness}(x_2) \wedge x_1 \neq x_2]$
Complex Patterns		
"Most A's are B"	需要二阶逻辑或计数	"Most birds fly" (超出 FOL 表达能力)
"The A" (唯一性)	$\exists x[A(x) \wedge \forall y[A(y) \to y = x]]$	"The king of France" $\exists x[\text{King}(x, \text{France}) \wedge \forall y[\text{King}(y, \text{France}) \to y = x]]$
"Every A has a B"	$\forall x[A(x) \to \exists y[B(y) \wedge \text{Has}(x, y)]]$	"Every person has a mother" $\forall x[\text{Person}(x) \to \exists y[\text{Mother}(y, x)]]$
"A's B" (所有格)	$B(x) \wedge \text{Possess}(A, x)$ or iota $x[B(x) \wedge \text{Of}(x, A)]$	"John's car" $\text{Car}(x) \wedge \text{Possess}(\text{John}, x)$

💡 多种正确答案可能存在——只要逻辑等价即可。关键是**结构正确**。

Example
例题：嵌套量词 "If one of Abigail's brothers makes noise, Abigail cannot sleep."
分析：
- 结构：If-then → 使用 "→"
- "One of Abigail's brothers" → $\exists x.\ \text{Brother}(x, \text{Abigail})$
- "makes noise" → $\text{MakeNoise}(x)$
- "cannot sleep" → $\neg\ \text{Sleep}(\text{Abigail})$

FOL：
$$(\exists x.\ \text{Brother}(x, \text{Abigail}) \wedge \text{MakeNoise}(x)) \to \neg\ \text{Sleep}(\text{Abigail})$$

6.3.7 Linear Indexed Grammar 构造策略

⚠ **TA 强调**："In the final exam last year, there was a very similar question... the idea is the same."

LIG 构造思路
核心问题：CFG 无法"计数"——无法保证 $a^n b^n c^n$ 中三个 n 相等。
LIG 解决方案：用 **stack** 记录计数信息。
策略选择：
1. 两端向中间：先生成首尾（如 a 和 d），再生成中间（如 b 和 c）
2. 左向右：先生成前半部分，stack 记录信息，再生成后半部分

例题：$a^n b^n c^n d^n$
策略：两端向中间。先生成 $a...d$，再生成 $b...c$。
Rules：
$$S[\sigma] \to aS[f\sigma]d \quad \text{(push f, 生成 a 和 d)}$$
$$S[\sigma] \to T[\sigma] \quad \text{(转换，开始生成 b 和 c)}$$
$$T[f\sigma] \to bT[\sigma]c \quad \text{(pop f, 生成 b 和 c)}$$
$$T[] \to \varepsilon \quad \text{(stack 空，结束)}$$
验证：生成 $aabbccdd$（$n = 2$）：
- $S[] \to aS[f]d \to aaS[ff]dd \to aaT[ff]dd$
- $\to aabT[f]cdd \to aabbT[]ccdd \to aabbccdd$ ✓

例题：$w h(w)$ where $w in Sigma^*$
策略：左向右。生成 w 时记录每个 symbol，再用 stack 生成 $h(w)$。
Rules：
$$S[\sigma] \to aS[a\sigma] \mid bS[b\sigma] \mid ... \quad \text{(生成 w，push symbols)}$$
$$S[\sigma] \to T[\sigma] \quad \text{(转换)}$$
$$T[a\sigma] \to h(a)T[\sigma] \quad \text{(pop a, 输出 h(a))}$$
$$T[b\sigma] \to h(b)T[\sigma] \quad \text{(pop b, 输出 h(b))}$$
$$T[] \to \varepsilon \quad \text{(stack 空)}$$

6.3.8 CCG 推导练习

⚠ **TA 建议**："Start from basic intuition... 'every dog' is a phrase, 'likes every dog' is a phrase." FOC 要注意¬的优先级.

CCG 推导步骤
1. Lexicon assignment：为每个 word 分配 categories
2. 从语义直觉出发：哪些 words 应该先组合？
3. 应用 combinatory rules：Forward (>) 或 Backward (<)
4. 同步计算 semantics：每步 apply Lambda terms

💡 **Type raising 的作用**：将 NP 提升为 $T/(T \setminus \text{NP})$，使得 proper noun 可以"主动"与 verb phrase 组合。在 "Alex" 的语义中用 $\lambda P.P(\text{Alex})$ 体现。

例题：'Alex likes every dog'
Lexicon：
- Alex : NP : Alex
- likes : $(S \setminus \text{NP}) / \text{NP} : \lambda P.\lambda Q.Q(\lambda x.P(\lambda y.\ \text{Likes}(x, y)))$
- every : NP / N : $\lambda P.\lambda Q.\forall x.P(x) \to Q(x)$
- dog : N : Dog

Derivation：
```
every                dog
NP/N:λP.λQ.∀x.P(x)→Q(x)     N:Dog
──────────────────────────────────────── >
        NP:λQ.∀x.Dog(x)→Q(x)

likes                       [every dog]
(S\NP)/NP:...               NP:λQ.∀x.Dog(x)→Q(x)
──────────────────────────────────────────────────── >
       S\NP:λQ.Q(λx.∀y.Dog(y)→Likes(x,y))

Alex        [likes every dog]
NP:Alex     S\NP:...
───────────────────────────────── <
    S:∀x.Dog(x)→Likes(Alex,x)
```
最终语义：$\forall x.\ \text{Dog}(x) \to \text{Likes}(\text{Alex}, x)$

"对所有 x，如果 x 是狗，则 Alex 喜欢 x"

6.3.9 Termination 与 Turing Completeness

Lambda calculus 的 Turing completeness 来源于：β-reduction 可能**不终止**。

Non-terminating Example
令 $\Omega = (\lambda x.xx)(\lambda x.xx) \underset{\beta}{\to} (\lambda x.xx)(\lambda x.xx) = \Omega$
无限循环！这是 Russell's paradox 在 Lambda calculus 中的体现。

💡 **Undecidable problem:** 判断两个 Lambda terms 是否 equivalent（即能否通过 α/β 互相到达）是不可判定的。

6.3.10 Extended Lambda Calculus

NL semantics
Constants: 表示 entities（Alex, Bob, Texas, ...） **Predicates:** 表示 relations（$\mathrm{Likes}(\cdot,\cdot)$, $\mathrm{Person}(\cdot)$, ...） **Quantifiers:** $\forall, \exists$ **Logical connectives:** $\wedge, \vee, \neg, \to$

α-conversion 和 β-reduction 规则不变。

Semantic Composition Example
Lexicon: $\mathrm{Alex} : \mathrm{Alex}$; $\mathrm{Brit} : \mathrm{Brit}$; $\mathrm{likes} : \lambda y.\lambda x.\ \mathrm{Likes}(x, y)$
Derivation of "Alex likes Brit":

1. $\mathrm{likes}(\mathrm{Brit}) = (\lambda y.\lambda x.\ \mathrm{Likes}(x, y))(\mathrm{Brit}) \underset{\beta}{\to} \lambda x.\ \mathrm{Likes}(x, \mathrm{Brit})$
2. $(\lambda x.\ \mathrm{Likes}(x, \mathrm{Brit}))(\mathrm{Alex}) \underset{\beta}{\to} \mathrm{Likes}(\mathrm{Alex}, \mathrm{Brit})$

💡 **为何 likes 是 $\lambda y.\lambda x$ 而非 $\lambda x.\lambda y$?**
因为英语语序是 Subject-Verb-Object。Verb 先接 object（右边），再接 subject（左边）。Lambda 的参数顺序反映了 syntactic composition 的顺序。

6.4 Combinatory Logic

Combinatory logic (Curry, 1958) 是 Lambda calculus 的替代——不使用 abstraction，只用 **primitive combinators** 构建 functions。

💡 *S 和 K 构成 **complete basis**——任何 Lambda term 都可用 S, K 表示。例如 $I = SKK$。*

Combinators
Identity: $Ix = x$; **Constant:** $Kxy = x$; **Substitution:** $Sxyz = xz(yz)$
Convention: left-associative，即 $Kxy = (Kx)y$

其他常用 combinators： **Composition:** $Bxyz = x(yz)$; **Flip:** $Cxyz = xzy$; **Type-raising:** $Txy = yx$

6.5 Combinatory Categorial Grammar (CCG)

6.5.1 为何需要 CCG?

Context-free grammars 无法优雅处理某些 phenomena：

1. **Coordination with gapping:** "I like to play bridge and Sarah handball"
2. **Cross-serial dependencies:** Dutch/Swiss German 的 verb-object 交叉依赖（recall Lecture 1）

CCG 是 **mildly context-sensitive**——比 CFG 更 expressive，但仍 polynomial-time parsable。
更重要的是：CCG 提供了 **syntax-semantics interface**——将 Lambda calculus 优雅集成到 grammar 中。

6.5.2 Linear Indexed Grammars（热身）

Linear Indexed Grammar
类似 CFG，但 non-terminals 可带 **stack**，且 stack 只能传给 **one** child：

$$N[\sigma] \to \alpha M[\sigma]\beta$$
$$N[\sigma] \to \alpha M[f\sigma]\beta \quad \text{(Push)}$$
$$N[f\sigma] \to \alpha M[\sigma]\beta \quad \text{(Pop)}$$

LIG 可生成 $\{a^n b^n c^n \mid n \in \mathbb{N}\}$——CFG 无法做到。
直觉：CFG 等价于 pushdown automata（无限 states via stack）。LIG 进一步扩展了这种"controlled infinity"。

6.5.3 CCG 形式定义

Combinatory Categorial Grammar
五元组 $\langle V_T, V_N, S, f, R\rangle$：

- V_T: terminals（词汇）
- V_N: atomic categories（基本范畴，如 S, NP, N）
- $S \in V_N$: start category
- $f : V_T \cup \{\varepsilon\} \to \mathcal{P}(C(V_N))$: lexicon（词 $\to$ categories 集合）
- R: combinatory rules

$C(V_N)$ 是 **categories** 的无限集：

- $V_N \subset C(V_N)$
- 若 $c_1, c_2 \in C(V_N)$，则 $c_1/c_2, c_1 \setminus c_2 \in C(V_N)$

6.5.4 Categories 与 Type-Theoretic View

Category 编码 **argument structure**：

- $S \setminus \mathrm{NP}$: 左边要 NP $\longrightarrow$ S (intransitive)
- $(S \setminus \mathrm{NP})/\ \mathrm{NP}$: 右边要 NP $\longrightarrow S \setminus \mathrm{NP}$ (transitive)

Example Lexicon
Mary, John : NP; walks : $S \setminus \mathrm{NP}$;
likes : $(S \setminus \mathrm{NP})/\ \mathrm{NP}$

6.5.5 Combinatory Rules

Application

$$X/Y \quad Y \Rightarrow X \ (>)$$
$$Y \quad X \setminus Y \Rightarrow X \ (<)$$

Composition

$$X/Y \quad Y/Z \Rightarrow X/Z \ (\boldsymbol{B}_>)$$
$$Y \setminus Z \quad X \setminus Y \Rightarrow X \setminus Z \ (\boldsymbol{B}_<)$$

Type-raising

$$X \Rightarrow T/(T \setminus X) \ (\boldsymbol{T}_>)$$
$$X \Rightarrow T \setminus (T/X) \ (\boldsymbol{T}_<)$$

💡 Rules 是 **schematic**——适用于所有 matching categories。这是 CCG 的设计哲学：rules 是 universal，language-specific 信息全在 lexicon。

6.5.6 Syntax-Semantics Integration

CCG 的优雅之处：category 的 argument structure 直接对应 Lambda term 的 type！

Derivation with Semantics
Lexicon:

- $\mathrm{Mary} : \mathrm{NP} : \mathrm{Mary}$
- $\mathrm{likes} : (S \setminus \mathrm{NP})/\ \mathrm{NP} : \lambda y.\lambda x.\ \mathrm{Likes}(x, y)$

6.5.7 Practical Application: Semantic Parsing to SQL

给定 question "What states border Texas?"，CCG parse 可得：

$$\lambda x.\ \mathrm{State}(x) \wedge \mathrm{Borders}(x, \mathrm{Texas})$$

```
SELECT x FROM states WHERE borders(x, 'Texas') #一步之遥即为 SQL
#同样适用于 robot commands、database queries、code generation 等。
```

- John : NP : John

Parse "Mary likes John":

```
Mary        likes                            John
NP:Mary    (S\NP)/NP:λy.λx.Likes(x,y)        NP:John
           ──────────────────────────────────────── >
                 S\NP:λx.Likes(x,John)
────────────────────────────────────────────────── <
              S:Likes(Mary,John)
```

💡 **与 LLM 的关系**：现代 LLM 可以直接生成 SQL/code、但无法 **guarantee** syntactic validity。CCG 等 grammar-based methods 提供 formal guarantees——在 safety-critical 应用中仍有价值。

[Bonus]:CCG parsing 可在 $O(N^6)$ 完成（类似 CKY，但因 composition/type-raising 导致更高 complexity）。

✅ **不考**：CCG parsing algorithm 细节不在考试范围。但理解 CCG 如何集成 syntax 和 semantics 是重要的 conceptual point。

7 Transformer

7.1 Machine Translation: 问题定义

Task：给定 source language 句子 $\boldsymbol{x} = (x_1, \ldots, x_M)$，生成 target language 句子 $\boldsymbol{y} = (y_1, \ldots, y_N)$。

MT as Optimization

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \operatorname{score}(\boldsymbol{y} \mid \boldsymbol{x})$$

目标：学习 $p(\boldsymbol{y} \mid \boldsymbol{x})$，从 input space 映射到 output space。特殊 token：BOS 和 EOS 标记序列边界。

💡 MT 不是 one-to-one mapping！同一句子可有多种合法翻译。例如 "The cat is black" ⟶ Spanish 需决定 grammatical gender（el gato / la gata），而英语不携带此信息——model 需 hallucinate 额外信息。

历史演进：
- **Rule-based**（1950s）：手工词典 + 语法规则
- **Statistical MT**（1990s）：多阶段 pipeline（morphology ⟶ alignment ⟶ generation），每步独立建模
- **Neural MT**（2014+）：**end-to-end** 单 model，直接输出 $p(\boldsymbol{y}|\boldsymbol{x}) \longleftarrow$ 今日主题

7.2 Sequence-to-Sequence Models

Encoder-Decoder Architecture
1. **Encoder**：$\boldsymbol{z} = \operatorname{Encode}(\boldsymbol{x})$，将输入压缩为 representation
2. **Decoder**：$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{n=1}^{N} p(y_n \mid y_1, \ldots, y_{n-1}, \boldsymbol{z})$

训练：MLE + backpropagation。

💡 **Information Bottleneck**：无论输入长度如何，encoder 输出 **fixed-length** vector $\boldsymbol{z}$。如 3 词句子与 100 词句子被压缩到同维向量——信息损失不可避免。

7.3 Attention Mechanism

7.3.1 动机与直觉

Attention 解决 fixed-length bottleneck：允许 decoder 在每一步 **动态关注** 输入的不同部分，不仅从单一压缩向量 $\boldsymbol{z}$ 解码，而是每步动态查询 encoder 全部 hidden states. 核心隐喻：**Soft Hash Table**。

7.3.2 从 Hard 到 Soft

Hash Table ⟶ Attention 的演进

Step 1: Hard Hash Table"

$$V = \operatorname{lookup}(K, \text{query}) = \begin{cases} v_i & \text{if } k_i = \text{query} \\ \text{null} & \text{otherwise} \end{cases}$$

问题：discrete lookup，不可微。仍是 hard retrieval。

Step 2: Algebraic View 用 one-hot vector $\boldsymbol{\alpha} \in \{0,1\}^n$（仅一个位置为 1）检索：

$$\boldsymbol{c} = \boldsymbol{\alpha}^\top \boldsymbol{V} = \sum_i \alpha_i \boldsymbol{v}_i$$

Step 3: Soft Attention 用 prob 分布$\boldsymbol{\alpha} \in \Delta^{|K|}$替代 one-hot:

$$\alpha_i = \frac{\exp \operatorname{score}(\boldsymbol{q}, \boldsymbol{k}_i)}{\sum_j \exp \operatorname{score}(\boldsymbol{q}, \boldsymbol{k}_j)} = \operatorname{softmax}(\operatorname{score}(\boldsymbol{q}, \boldsymbol{K}))_i$$

$$\boldsymbol{c} = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i = \boldsymbol{\alpha}^\top \boldsymbol{V} \quad \text{(context vector, weighted average)}$$

现在 $\boldsymbol{\alpha}$ 表示"关注度分布"，continuous，differentiable；$\alpha_i \geq 0, \sum_i \alpha_i = 1, \boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v}_i$.

7.3.3 Math Formulation

Attention Mechanism

给定：
- **Query** $\boldsymbol{q} \in \mathbb{R}^{d_q}$：当前 decoder 状态（"我在找什么"）
- **Keys** $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$：候选位置的表示（"有什么可选"）
- **Values** $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$：实际要检索的信息

$$\alpha_i = \operatorname{softmax}_i(\operatorname{score}(\boldsymbol{q}, \boldsymbol{k}_i)) = \frac{\exp \operatorname{score}(\boldsymbol{q}, \boldsymbol{k}_i)}{\sum_j \exp \operatorname{score}(\boldsymbol{q}, \boldsymbol{k}_j)}$$

$$\boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v}_i = \boldsymbol{\alpha}^\top \boldsymbol{V} \quad \text{(context vector)}$$

Scoring Functions（实践中常用）：
- Dot-product: $\boldsymbol{q}^\top \boldsymbol{k}$，最常用，高效
- Scaled dot-product: $\boldsymbol{q}^\top \boldsymbol{k} / \sqrt{d_k}$，Transformer 默认[7]
- Additive: $\boldsymbol{w}^\top \tanh(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k})$，Bahdanau 2015

⚙ **Encoder-Decoder Attention Flow**

7.3.4 Self-Attention Complexity

Self-Attention Complexity & Comparison with N-gram:

Operation / Aspect	Self-Attention	N-gram
Complexity Analysis		
Sequence length	N	N
Hidden dimension	H	–
Compute $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	$O(NH^2)$	–
$\boldsymbol{Q}\boldsymbol{K}^\top$	$O(N^2 H)$	–
Softmax	$O(N^2)$	–
Multiply $\boldsymbol{V}$	$O(N^2 H)$	–
Total	$\boldsymbol{O(N^2 H + N H^2)}$[8]	–
Bottleneck ($N \gg H$)	$O(N^2)$	–

[7] Scaled dot-product 中除以 $\sqrt{d_k}$：防止 d_k 很大时 dot product 值过大，导致 softmax 梯度消失（进入饱和区）。Dot product 的 variance 与 dimension 成正比。若不 normalize，当 d_k 很大时，softmax 输入值过大，梯度 saturation 趋近于 0

[8] 不难发现当 $N \gg H$ 时，$O(N^2)$ 是主要瓶颈。

```
# Encoder 阶段
K, V = Encoder(x)  # shape: (m, d_model)

# Decoder 逐步生成
for t in 1..n:
    q = decoder_hidden[t]      # query: (d_model,)
    scores = score(q, K)       # (m,)
    α = softmax(scores)        # attention weights
    c = weighted_sum(α, V)     # context: (d_model,)

    p_t = softmax(FFN([c; q])) # 融合context生成prob
    y_t ~ p_t
```

Qualitative 对比	Self-Attention	N-gram
Context	Full sequence	Fixed window k
Parameters	Fixed	Depends on vocab size
Small dataset	Prone to overfit	Works with smoothing
Large dataset	Stronger	Limited by sparsity
Runtime	$O(N^2)$	$O(N)$

7.3.5 Multi-Head Attention

Multi-Head Attention

并行运行 h 个 attention（各有独立参数），拼接后：

$$\mathrm{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)\boldsymbol{W}^O$$

$$\mathrm{head}_i = \mathrm{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right)$$

💡 不同 head 可能关注句法/语义/位置等不同模式; 增加 model 容量，类似 CNN 多通道; 经验上 $h = 6$ 或 8 效果好。

✅ Assignment 6: 证明 multi-head self-attention 可以表示任意 conv 层。[9]

7.3.6 Encoder-Decoder Attention

MT 中的 Attention

- $\boldsymbol{K} = \boldsymbol{V}$： encoder 各位置的 hidden states $\boldsymbol{h}_1^{\mathrm{enc}}, ..., \boldsymbol{h}_M^{\mathrm{enc}}$
- $\boldsymbol{Q}$： decoder 当前 hidden state $\boldsymbol{h}_n^{\mathrm{dec}}$

语义：decoder 在生成第 n 个词时，询问"源句中哪些词与当前生成最相关？"

7.3.7 Self-Attention

Self-Attention

$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ 均来自 **同一序列** 的不同 linear projections：

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

每个位置的表示由 **整个序列** 加权得到，捕获 long-range dependencies。

7.4 Trsf Architecture

Self-attention 的输出为 contextual embeddings —— 每个 token 的表示包含了序列中其他 tokens 的信息。 Self-attention 的意义：

- **Encoder-only**（如 BERT）：双向 self-attention
- **Decoder-only**（如 GPT）：causal self-attention（只看过去）
- 不再需要 encoder-decoder 架构！

💡 Transformer 的核心贡献，关键不在精度提升，而是：**parallelization**。RNN 必须顺序处理，Transformer 可并行处理整个序列——这是 scaling 的基础。
训练并行 vs. 推理串行：训练时 teacher forcing 可并行计算全序列，但生成时仍需逐词 sampling（autoregressive bottleneck）

Component	Complexity
Encoder (L layers)	$O(LND^2)$
Decoder (1 layer, no attention)	$O(MD^2)$
Cross-attention	$O(MND)$
Total with attention	$O(LND^2 + MND)$

Table 8: 设 encoder 长度 N, decoder 长度 M, hidden dim D， encoder 层数 L

7.4.1 整体结构

Transformer Encoder Block 图略

7.4.2 关键组件

1. Positional Encoding

Self-attention 是 permutation-invariant[10] ——丢失位置信息 hence 需 positional embedding。

Sinusoidal Positional Encoding

$$\mathrm{PE}_{\mathrm{pos},2i} = \sin\left(\mathrm{pos} \ /10000^{2i/d}\right)$$

$$\mathrm{PE}_{\mathrm{pos},2i+1} = \cos\left(\mathrm{pos} \ /10000^{2i/d}\right)$$

性质： bounded in $[-1, 1]$，远距离衰减，相对位置可通过线性变换表示。

这是 engineering hack——没有理论证明为何 sine/cosine 最优。

2. Residual Connection:

每个 sub-layer 输出：$\boldsymbol{x}_{l+1} = \boldsymbol{x}_l + \mathrm{SubLayer}(\boldsymbol{x}_l)$ 作用：缓解 vanishing gradient，允许更深网络。信息可 bypass 中间层直接传递(允许梯度直接回传)。

3. Layer Normalization:

$$\mathrm{LayerNorm}(\boldsymbol{x}) = \gamma \odot \frac{\boldsymbol{x} - \mu}{\sigma} + \beta$$

其中 μ, σ 是单个样本内[11] hidden states 的均值/标准差。作用：稳定训练(缓解梯度消失)，加速收敛。

7.4.3 Encoder vs Decoder vs Encoder-Decoder

⚙️ **Transformer Encoder Layer**

完整 Transformer 架构

Encoder:

1. Input Embedding + Positional Encoding

---

[9]Theorem: 若 multi-head self-attention 有 K^2 个 heads（K 是 kernel size），且使用特定 Gaussian positional encoding，则可精确表示任意 $K \times K$ 卷积。这解释了 ViT 的成功——Transformer 至少和 CNN 一样 expressive，且更 general。

[10]Permutation Equivariance: 若 f 是 permutation equivariant，则对任意 permutation π,

$$f(\pi(X)) = \pi(f(X))$$

, 即打乱输入顺序，输出以相同方式打乱。
设 $\boldsymbol{P}$ 是 permutation matrix, 具体证明利用了 softmax 对 row-wise 操作的性质和 $\boldsymbol{P}^\top\boldsymbol{P} = \boldsymbol{I}$。 则:

$$\begin{aligned}\mathrm{Attention}(\boldsymbol{P}\boldsymbol{X}) &= \mathrm{softmax}\left(\frac{1}{\sqrt{d}}(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_Q)(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_K)^\top\right)(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_V) \\ &= \mathrm{softmax}\left(\boldsymbol{P}\boldsymbol{Q}\boldsymbol{K}^\top\frac{\boldsymbol{P}^\top}{\sqrt{d}}\right)\boldsymbol{P}\boldsymbol{V} = \boldsymbol{P}\ \mathrm{softmax}\left(\boldsymbol{Q}\frac{\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V} = \boldsymbol{P}\ \mathrm{Attention}(\boldsymbol{X})\end{aligned}$$

若$\boldsymbol{Q}$ fixed（如常数矩阵），则 attention 变成 permutation invariant——输出完全不依赖输入顺序。

[11]注意各种 norm 的区别： Layer Norm: 在特征维度上归一化（更适合序列 data）; Batch Norm: 在 batch 维度上归一化

```
#
def encoder_layer(x):
# 1. Multi-head self-attention
attn_out = MultiHeadAttention(Q=x, K=x, V=x)
x = LayerNorm(x + attn_out)  # residual + norm

# 2. Feed-forward network
ffn_out = FFN(x)  # 2-layer MLP: ReLU(xW₁)W₂
x = LayerNorm(x + ffn_out)

return x
```

堆叠 N=6 层（原论文）。Decoder 类似，但增加 encoder-decoder attention。

2. $\times N$ layers:
   - Multi-Head Self-Attention
   - Add & Norm
   - Feed-Forward Network (FFN)
   - Add & Norm

Decoder:
1. Output Embedding + Positional Encoding
2. $\times N$ layers:
   - Masked Multi-Head Self-Attention（causal）
   - Add & Norm
   - Encoder-Decoder Attention（$\boldsymbol{Q}$ from decoder，$\boldsymbol{K}, \boldsymbol{V}$ from encoder）
   - Add & Norm
   - FFN + Add & Norm
3. Linear + Softmax $\longrightarrow P(y_t \mid y_{<t}, \boldsymbol{x})$

7.5 Decoding: 如何生成文本

7.5.1 问题：指数爆炸

设 vocabulary size $|\mathcal{V}| = 30000$，最大长度 $N = 20$：$|\mathcal{Y}| = |\mathcal{V}|^N = 30000^{20} >$ 宇宙粒子数无法穷举！Dynamic programming 也不适用（无 Markovian structure, Viterbi $O(n|\mathcal{V}|^k)$不适用）。
Transformer 的 scoring function 考虑 **entire context** 有 global 依赖性，不满足 local 分解假设 ⟶ search 空间/search 图是 **tree** 而非 DAG，状态不合并 ⟶ 指数复杂度 ⟶ 考虑 heuristic 剪枝

7.5.2 Decoding 策略

Decoding Strategies 对照

类型	方法	特点
Deterministic	Greedy search	每步取 arg max，快但 suboptimal
Deterministic	Beam search	保留 top-K candidates，trade-off
Stochastic	Top-k sampling	从前 k 高 prob 词中随机 sampling
Stochastic	Nucleus (top-p)	从累积 prob 达 p 的词中 sampling

常见问题：greedy 导致重复，高温 sampling 产生乱码。
随机 sampling 族（stochastic decoding）:
e.g. Nucleus Sampling (Top-p) 从累积 prob $\geq p$ 的最小词集 sampling:

$$V^{(p)} = \operatorname{argmin} \left\{ V' \subset V : \sum_{w \in V'} P(w \mid \boldsymbol{y}_{<t}, \boldsymbol{x}) \geq p \right\}$$

动态调整候选集大小。常用 $p = 0.9$。
Beam Search：Pruned BFS，每步保留 K 个最高分 partial sequences。
- $K = 1$ 退化为 greedt; K 越大越接近 exact search，但计算量 $O(K \cdot |\mathcal{V}|)$
- 实践中 $K = 4 \sim 10$ 常用。无 formal guarantee 但 works well。

Beam Search

维护 top-K 候选路径（beam width = K）:

```
beams = [(score=0, seq=[BOS])]

for t in 1..max_len:
    all_candidates = []
    for (s, seq) in beams:
        probs = model.predict(seq, x)  # P(y_t | seq, x)
        for w in top_K(probs):
            new_score = s + log(probs[w])
            all_candidates.append((new_score, seq + [w]))

    # 剪枝：只保留global top-K
    beams = sorted(all_candidates)[:K]

    # 终止条件
    if all EOS in beams: break

return max(beams, key=score)
```

复杂度：$O(nK\,|\mathcal{V}|)$。权衡：$K = 1$ 退化为 greedy，$K = \infty$ 穷举。

Parallelization 的限制：training 时可并行（所有 tokens 已知），但 **inference 仍是 sequential**（autoregressive generation）——当前 LLM 推理速度瓶颈。

7.6 Evaluation: BLEU Score

人工评估不可扩展 ⟶ 需自动化指标。核心困难：一句多译皆合理。

BLEU Score
基于 n-gram 精确率 + 长度惩罚:

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$p_n = \frac{\sum_{\text{ngram}} \text{count}_{\text{clip(ngram)}}}{\sum_{\text{ngram}} \text{count(ngram)}} \quad \text{(modified precision)}$$

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{otherwise} \end{cases} \quad \text{(brevity penalty)}$$

其中：c = 候选长度，r = 参考长度，$w_n = 1/N$（通常 $N = 4$）；
p_n：n-gram precision（预测中出现在 reference 的比例）；BP：brevity penalty，惩罚过短翻译

Clipped Count：防止重复词刷分

$$\text{count}_{\text{clip(the)}} = \min\left(\text{count}_{\text{pred(the)}}, \max_{\text{ref}} \text{count}_{\text{ref(the)}}\right)$$

💡 **BLEU 家族**：ROUGE（摘要）：Recall-oriented; METEOR：考虑同义词、词干; chrF：字符级 n-gram. 现代趋势：NN 评估（BERTScore）+ 人类评估（仍是金标准）

局限性：BLEU 只是 proxy metric。MT evaluation 仍是 open problem: 只看词重叠，忽略语义（“not good” vs. “bad”）；对改写、同义替换不友好; 需要高质量参考译文

8 Axes of Modelling

8.1 问题定义：从 data 到任务

核心问题：给定 data $\boldsymbol{X}$，学习映射 $f : \boldsymbol{X} \to \boldsymbol{Y}$。**Task Characterization**（决定后续所有选择）：

- Classification：$\boldsymbol{Y}$ 是离散 label 集合
- Structured Prediction：$\boldsymbol{Y}$ 是 exponentially large 的结构化输出（如序列、树）
- Representation Learning：学习 $\boldsymbol{X}$ 的 dense embedding
- Density Estimation：建模 $p(\boldsymbol{X})$

💡 data 质量 >> model 复杂度。High-quality data 是 NLP 的 “magic element”

8.2 model 分类体系

Model Taxonomy

```
Models
├── Probabilistic（建模 p(Y|X) 或 p(X,Y)）
│   ├── Discriminative: 直接建模 p(Y|X)
│   │   └── Logistic Regression, CRF, Neural Classifiers
│   └── Generative: 建模 joint p(X,Y) = p(Y)p(X|Y)
│       └── N-gram LM, Naive Bayes, HMM
└── Non-Probabilistic
    ├── Learned: SVM, MLP, Perceptron
    └── Handcrafted: CFG, Rule-based systems
```

8.2.1 Probabilistic vs Non-Probabilistic

Aspect	Probabilistic	Non-Probabilistic
输出	prob 分布 $p(y\|x)$	决策边界 / 得分
优势	Uncertainty quantification, 理论框架成熟	Interpretable, geometric intuition
劣势	Independence 假设可能不成立	难以估计 uncertainty
典型	Naive Bayes, LM	SVM, rule-based

8.2.2 Generative vs Discriminative

Trade-off：Generative 需更多假设但可处理 missing data；Discriminative 通常分类性能更优。

Generative：建模 $p(\boldsymbol{x}, y) = p(y) \cdot p(\boldsymbol{x}|y)$，可生成新样本

Discriminative：直接建模 $p(y|\boldsymbol{x})$，专注分类边界

8.3 Structured Prediction

当 output space $|\mathcal{Y}|$ 指数级大（如所有可能句子）时，无法为每个 y 单独建模。

8.3.1 Local vs Global Normalization

Normalization 对比

Type	Locally Normalized	Globally Normalized
定义	$p(\boldsymbol{y}\|\boldsymbol{x}) = \prod_n p(y_n \mid y_{<n}, \boldsymbol{x})$	$p(\boldsymbol{y}\|\boldsymbol{x}) = \exp(\text{score}(\boldsymbol{y}))/Z(\boldsymbol{x})$
归一化	每步独立归一化	整个序列归一化
代表	N-gram, RNN LM, GPT	CRF, EBM
优点	训练高效，teacher forcing	无 label bias
缺点	**Label bias**：早期错误无法恢复	计算 Z 代价高

Label Bias Problem

Locally normalized model 中，若某状态转移 prob 集中于少数 successor，则该路径被“锁定”——即使后续 observation 强烈反对，也难以修正。

8.3.2 Independence Assumptions 的 Trade-off

有假设（如 Markov）
- 可用 DP 进行 **exact** decoding
- model 参数少，不易 overfit
- 可能 underfit（假设过强）

无假设（如 Transformer）
- 建模能力强，捕获 long-range dependency
- 只能 **approximate** decoding（beam search 等）
- 易 overfit，需大量 data
- 可解释性差

8.4 Loss Functions

8.4.1 定义与性质

Loss/Objective/Cost Function
将 model 参数 $\theta \in \Theta$ 映射到实数，量化 model 在训练 data 上的拟合程度。

$$\mathcal{L} : \Theta \to \mathbb{R}$$

Desirable Properties：**Convexity**：保证收敛到 global minimum; **Differentiability**：可用 gradient-based optimization; **Computational efficiency**：快速计算; **Robustness to noise**：对异常值不敏感; **Statistical guarantees**：如 consistency, efficiency

8.4.2 Maximum Likelihood → Cross-Entropy Loss

MLE：$\hat{\theta} = \arg\max_\theta \prod_i p(y_i \mid x_i; \theta)$ 取负对数，转化为 loss：

$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\sum_i \log p(y_i \mid x_i; \theta)$$

💡 **MLE 优点**：Consistent estimator（data → ∞ 时收敛到真实参数）；Asymptotically efficient（达到 Cramér-Rao lower bound）；低 KL divergence from true distribution **MLE 局限**：仅适用于 probabilistic models; 易 overfit; 要求 $p(y|x) > 0$（否则 $\log 0$ 爆炸）

8.4.3 其他 Loss Functions

常用 Loss 对比:

Loss	Formula	Convex?
0-1 Loss	$\mathbb{1}[\hat{y} \neq y]$	No
Hinge (Max-margin)[12]	$\max(0, 1 - y \cdot f(x))$	Yes
Logistic	$\log\left(1 + e^{-y \cdot f(x)}\right)$	Yes
Exponential	$e^{-y \cdot f(x)}$	Yes
Cross-Entropy	$-\sum_c y_c \log \hat{y}_c$	Yes

8.5 Regularization

目标：防止 overfitting，提升 generalization（在 unseen data 上的表现）。

Regularized Loss

$$\mathcal{L}_{\mathrm{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \cdot R(\theta)$$

其中 $R(\theta)$ 是 penalty term，$\lambda > 0$ 控制正则化强度。

其他正则化技术：
- **Dropout**：训练时随机置零部分神经元
- **Early stopping**：validation loss 不再下降时停止
- **Weight decay**：等价于 L2 in certain optimizers

8.5.1 L1 vs L2 Regularization

Aspect	L1 (Lasso)	L2 (Ridge)
Penalty	$\lambda \sum_j \lvert\theta_j\rvert$	$\lambda \sum_j \theta_j^2$
效果	Sparse：多数 $\theta_j = 0$	Shrinkage：θ_j 趋近 0 但非零
Bayesian 解释	Laplace prior	Gaussian prior
适用	Feature selection	防止 collinearity

8.6 Evaluation Metrics

✅ **Loss ≠ Evaluation Metric**. Loss 用于训练优化；Evaluation metric 用于评估 trained model 在 held-out set 上的表现。两者可以相同（如 perplexity），但通常不同。

8.6.1 Classification Metrics

Confusion Matrix 衍生指标

	Predicted +	Predicted −	
Actual +	TP	FN	$P = \mathrm{TP} + \mathrm{FN}$
Actual −	FP	TN	$N = \mathrm{FP} + \mathrm{TN}$

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad \mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \mathrm{F}_1 = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \quad \text{(harmonic mean)13}$$

✅ **为何不直接用 Accuracy?**
- Class imbalance：99% negative 时，“always predict negative” 得 99% accuracy
- 不同错误代价不同：cancer 漏诊 (FN) 比误诊 (FP) 严重得多

8.6.2 Intrinsic vs Extrinsic Evaluation

Type	Intrinsic	Extrinsic
定义	Task-neutral，评估 model 本身	评估 model 在下游任务的表现
LM 例子	Perplexity, likelihood	MT BLEU, QA accuracy
Embedding 例子	Word analogy, similarity	下游分类性能

NLP 评估的挑战
主观性：语言风格因人而异（表情符号、句末标点的“攻击性”）；多正确答案：同一问题可有多个合理回答; Human evaluation 昂贵：且 annotator diversity 难保证; Alignment 问题：model 价值观与人类期望的对齐; Data contamination：benchmark 泄露到 training data

[12]**Hinge Loss**: 不仅要分类正确，还要 margin ≥ 1。Convex 但在 0 点不可微（可用 subgradient）。
[13]F1 是 NLP 标准：平衡 precision 和 recall。

优点	快速、可复现	更贴近实际应用
缺点	可能与实际性能脱节	昂贵、任务依赖

8.7 Model Selection

8.7.1 为何需要 Model Selection

Hyperparameters（如 learning rate, hidden size, regularization λ）显著影响 model 性能，但不能通过 training 优化——需单独选择。
Two 目标常冲突：

1. **Inference**：选择最能 explain data 的 model（interpretability）[14]
2. **Prediction**：选择 predict 性能最佳的 model

8.7.2 Cross-Validation

K-Fold Cross-Validation

1. 将数据随机分为 K 份（folds）。
2. **Iteration**: 循环 K 次，每次取第 k 份作为 Test/Validation Set，其余 $K-1$ 份作为 Training Set。
3. **Aggregation**: 报告 K 次结果的 mean $\pm$ std。

✅ exm21b 中：test set size：$\frac{N}{K}$；training set size: $N \times \frac{K-1}{K}$；total model created: K(必须完整跑完每一折)

用途区分：

- Model Assessment: 用于评估模型泛化能力（此时留出份为 Test Set）。
- Model Selection: 用于超参数调优（此时留出份为 Validation Set）。

Nested CV (嵌套交叉验证)：用于同时进行调参和无偏评估。

- Inner Loop: Model Selection。在训练集中再次做 CV 来寻找最佳超参数。
- Outer Loop: Model Assessment。使用 Inner Loop 选出的最佳参数，在外层 Test fold 上评估泛化误差。
- 稳定性检测: 若外层不同 fold 选出的最佳参数差异巨大，说明模型不稳定或数据过少。

8.7.3 Statistical Significance Testing

问题：Model A 比 B 好 2%，是真实差异还是随机噪声？

- Multiple Testing Problem：若比较 20 个 model，$\alpha = 0.05$ 时期望有 1 个 false positive。
- Bonferroni Correction：比较 m 个 model 时，使用 $\alpha' = \frac{\alpha}{m}$。

常用**检验**：

- Parametric：paired t-test（假设正态分布）
- Non-parametric：McNemar test, permutation test（无分布假设）

8.7.4 McNemar's Test

标准 t-test 假设数据服从 normal/t 分布——textual data 通常不满足。因此 NLP 常用 **non-parametric tests**：无需指定 parametric family。

McNemar's Test

用于比较 **两个 classifiers** 在同一数据集上的表现。
构造 contingency table：

	Model B Correct	Model B Wrong
Model A Correct	n_{00}	n_{01}
Model A Wrong	n_{10}	n_{11}

Insight：只关注 **disagreement cells** n_{01}, n_{10}（两模型都对/都错的样本不提供区分信息）。
Test statistic：

$$\chi^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

H_0：两 classifier 性能相同。要求 $n_{01} + n_{10} \geq 25$。

8.7.5 Permutation Test

Permutation Test

检验 classifier 是否学到了有意义的 pattern（vs random chance）。
Algorithm：

1. 在原始数据上训练模型，记录 performance P_0
2. Repeat B 次（$B \geq 1000$）：
   - 随机 permute labels（打乱 y 与 x 的对应）
   - 重新训练，记录 performance P_b
3. p-value ≈ tion of $P_b \geq P_0$

直觉：若 labels 包含信息，原始模型应显著优于 permuted versions。

5×2 CV t-Test

解决问题：标准 CV 中各 fold 的样本 **相互依赖**，违反 t-test 独立性假设。
Procedure：

1. 将数据 50-50 split 为 train/test
2. 训练两个 classifiers，计算 performance difference d
3. 交换 train/test，再次计算 d'
4. 重复 5 次（共 10 个 difference values）
5. 计算特殊 t-statistic（考虑 variance）

优点：保持样本独立性，结果 **conservative**（不易 false positive）。

💡 **实践建议**：小样本时务必检查 test 的 statistical power; 多重比较必须做 **Bonferroni correction**：$\alpha' = \frac{\alpha}{m}$; 不要过度依赖 p-value，关注 **effect size**

8.7.6 Statistical Power 与 Type I Error

常用检验的 Type I Error 对比

Test	**Type I Error**	**Note**
Resampled t-test	High	不推荐用于 NLP

[14] 例：银行信贷必须使用 interpretable model（如 logistic regression），即使 neural network 准确率更高——因法规要求解释拒贷原因。

McNemar's test	Low	适合比较两个 classifiers
Permutation test	Low	最常用，无分布假设
5×2 CV t-test	Low	适合 CV 场景

8.8 Occam's Razor in NLP

模型选择原则

Prefer simpler models：
- 两个 model 的 Loss 相近 → 选系数/参数更小的
- 简单模型更 stable，泛化更好
- L1/L2 regularization ⇒ enforce simplicity
- Parameter sharing[15] ⇒ *implicit* regularization

💡 即使 NN 看似 over-parameterized，good practice：实验多种复杂度，选满足性能的最简模型

8.9 Domain Adaptation (Unsupv: density ratio; Supv: feature augmentation)

Train/test 分布不同（**covariate shift**）→ 性能下降

Setup：
- $P_{\text{old}}(x, y)$：training 分布 (有 labels)
- $P_{\text{new}}(x, y)$：test 分布 (可能无 labels)

Goal：学习在 P_{new} 上表现好的 classifier

Domain Shift

"Very small" 对 USB drive 是 positive，对 hotel room 是 negative——同一 feature 在不同 domain 语义相反。

Importance Sampling

当 P_{new} 无 label 时，用 density ratio 重新加权训练样本：

$$\mathcal{L}_{\text{new}} = \mathbb{E}_{P_{\text{new}}}[\ell(x, y)] = \mathbb{E}_{P_{\text{old}}}\left[\frac{P_{\text{new}}(x)}{P_{\text{old}}(x)} \cdot \ell(x, y)\right]$$

直觉：给与 P_{new} 更相似的训练样本更高权重。

Unsupervised: Importance Sampling

当 P_{new} 无 label 时，用 density ratio 重新加权训练样本，用 density ratio 重新加权：

$$\mathcal{L}_{\text{new}} = \mathbb{E}_{P_{\text{old}}}\left[\frac{P_{\text{new}}(x)}{P_{\text{old}}(x)} \cdot \ell(x, y)\right]$$

实现：训练 binary classifier 区分 old/new, use prob ratio as weight:

$$w(x) = \frac{P(s = 0|x)}{P(s = 1|x)}$$

其中 $s = 1$ 表示来自 old distribution; 若 *classifier* 准确率高 → *shift* 严重 → 更难 *adapt*

Supervised: Feature Augmentation

若 P_{new} 有少量 labels，可共享 feature：

$$\varphi_{\text{aug}}(x) = [\varphi_{\text{shared}}; \varphi_{\text{old}}; \varphi_{\text{new}}]$$

- Shared features: 两 domains 都激活（通用 sentiment words）
- Domain-specific features: 仅对应 domain 激活

8.10 Bias and Fairness in NLP

✅ **Non-exam content**，但对 responsible AI practice 至关重要。

8.10.1 Bias 来源

Example

Bias 进入 NLP 系统的途径:

1. Labeling bias annotators 偏见
2. Sample selection 数据偏向 Western, male
3. Task design 如 binary gender 假设
4. Feature omission 缺信息 → 模型"猜测"
5. Majority pattern 优化 avg loss，忽视 minority
6. Feedback loops 部署后放大偏差

8.10.2 Train-Test Mismatch 视角

训练数据主要来自 group A　测试时遇到 group B (*distribution shift*)　→ 性能在 group B 显著下降

8.10.3 伦理框架

Framework	**Core Idea**	**NLP 应用**
Consequentialism	行为的道德性由后果决定	评估 model deployment 的社会影响
Utilitarianism	最大化总体 utility	权衡不同群体的 performance
Deontology	遵循规则，无论后果	Hard constraints（如禁止生成特定内容）
Social Contract	相互尊重的隐含契约	Fairness 作为社会规范

8.11 Debiasing Word Embeddings

8.11.1 Bolukbasi et al. (2016)：线性 Bias Subspace

Gender Bias as Linear Subspace

核心假设：Gender bias 在 embedding space 中是 **linear subspace**。

Vocabulary Partition：
- Neutral words：无固有 gender（programmer, homemaker）
- Gendered word pairs：(w_m, w_f)（he/she, king/queen）

Identify Bias Subspace：
1. 构造 difference matrix：$\boldsymbol{D} = [\boldsymbol{w}_{\text{she}} - \boldsymbol{w}_{\text{he}}; \boldsymbol{w}_{\text{her}} - \boldsymbol{w}_{\text{him}}; \ldots]$

[15] 如 word embeddings 跨位置共享

2. 对 $\boldsymbol{D}$ 做 PCA，前 k 个 principal components 定义 bias subspace $\boldsymbol{B}$

Debiasing Steps：

1. Neutralize：对 neutral words，投影去除 bias 分量
$$\boldsymbol{w}_n' = \boldsymbol{w}_n - \mathrm{proj}_{\boldsymbol{B}}(\boldsymbol{w}_n)$$
2. Equalize：确保 gendered pairs 与 neutralized words 等距
   - 将 neutral word 置于 gendered pair 的"中点"
   - 调整 gendered words s.t.到 neutral words 距离相等

💡 **Equalization 的几何**：设 neutral word 为 $\boldsymbol{w}_n$，gendered pair 为 $(\boldsymbol{w}_m, \boldsymbol{w}_f)$。目标：$\|\boldsymbol{w}_n' - \boldsymbol{w}_m'\| = \|\boldsymbol{w}_n' - \boldsymbol{w}_f'\|$。通过 Pythagorean theorem 解析求解 λ 参数。

8.11.2 Kernel PCA Debiasing (Cotterell et al.)

Motivation：为何 bias 必须是 linear？

Idea：用 kernel trick[16]将 embeddings 映射到高维 feature space $\varphi : \mathbb{R}^d \to \mathcal{H}$，在 $\mathcal{H}$ 中做 linear debiasing（等价于原空间的 non-linear debiasing）。

Kernel Trick

无需显式计算 $\varphi(\boldsymbol{w})$，只需定义 kernel function：
$$K(\boldsymbol{w}, \boldsymbol{w}') = \langle \varphi(\boldsymbol{w}), \varphi(\boldsymbol{w}') \rangle$$
常用 kernels：polynomial, RBF/Gaussian。

8.11.3 Word Embedding Association Test (WEAT)

WEAT

量化 embedding 中的 implicit association。

给定 target sets X, Y（如 male/female names）和 attribute sets A, B（如 career/family words）：
$$s(w, A, B) = \frac{1}{|A|} \sum_{a \in A} \cos(\boldsymbol{w}, \boldsymbol{a}) - \frac{1}{|B|} \sum_{b \in B} \cos(\boldsymbol{w}, \boldsymbol{b})$$
$$\mathrm{WEAT} = \sum_{x \in X} s(x, A, B) - \sum_{y \in Y} s(y, A, B)$$
Effect size 类似 Cohen's d。

8.11.4 Debiasing 的局限

💡 *Residual bias*：即使移除 gender subspace，gender prediction 仍可达 70-75% accuracy（vs 原始 97%）——信息仍 encoded elsewhere。
Quality preservation：验证 debiased embeddings 在 SimLex-999 等 benchmark 上与 human similarity judgments 的相关性未下降。

[16]幽默的是，结果显示 Non-linear debiasing 没有显著优于 linear 方法——linearity assumption 对 gender bias 足够。