

第 0 周 · Day 3：词的表示

时长 6-7 小时：§1-§2 表示问题与计数派（1h）· §3 Word2Vec 与负采样（2.5h）· §4 两派统一（1h）· §5-§7 几何、天花板、分词（1.5h）· §A-§C 1h

配套代码 `day3_checks.py`（纯 numpy）：全 softmax 梯度的结构、SGNS 梯度、0.75 指数、算力对比、Levy-Goldberg 的数值求解验证、BPE 玩具实现

§B 参考实现（code/ 目录）：`d3_sgns_numpy.py`、`d3_gensim_explore.py`、`d3_bpe.py`、`d3_tokenizer_demo.py`。先自己写，再对照；一键自检 `python code/run_selftests.py`，说明见 `code/README.md`。

§0 今天的主线

0.1 一个问题

前两天讲的是"怎么训练"，今天第一次讲"训练什么"：

怎么把一个词变成一个向量，使得这个向量携带语义？

0.2 论证链

节	回答什么	撞到的墙
§1	one-hot 为什么不行	三宗罪，第三宗（无泛化）是致命的
§2	计数派：共现 → PMI → SVD	思路对，但无法扩展到大语料
§3	预测派：Word2Vec	全 softmax 的分母要遍历词表 → 负采样
§4	两派其实是一回事	表面之争，底下是同一个矩阵分解
§5	词向量的几何	类比成立的原因与它的缺陷
§6	静态词向量的天花板	一词一向量无法表达多义 → 通往注意力的门
§7	分词	为什么 LLM 既不用词也不用字符 → BPE

今天真正要带走的三样东西：

- §3.2-§3.3 的负采样推导思路："候选太多、分母算不起时，把'从所有候选里选一个'换成'判断真假'"。这个套路在对比学习、检索模型里会反复出现，第 4 周做 RAG 时你会再见到它。
- §4.2 的 Levy-Goldberg 推导：一个看起来是"预测"的方法，最优解却是一个显式的统计量。它是"很多看似思想之争的东西，拆开看只是工程之争"的典范。
- §6 的天花板论证：它直接解释了注意力为什么会被发明出来。

0.3 和正课的关系

Lecture 1 前 30 分钟会飞快扫过 NLP 背景、分词、词嵌入和 Word2Vec。今天学完，那 30 分钟可以 2 倍速。§6 是 Lecture 1 整堂课的动机，§7 是 Lecture 1 会正式讲的分词。

诚实提示：这是第 0 周对 Transformer 最不"刚需"的一天。时间紧时 §2、§4.1、§5 可以快读，但 §3、§4.2、§6 不要跳。

§1 表示的问题

1.1 one-hot 的三宗罪

把词表 V 里第 i 个词表示成 $e_i \in \{0, 1\}^{|V|}$ 。

罪一：维度灾难。 $|V|$ 通常 10^5 量级，后面每一层的参数量都要乘这个数。

罪二：不携带相似性。 任意两个不同词的 one-hot 点积都是 0，欧氏距离都是 $\sqrt{2}$ 。"猫"和"狗"的距离等于"猫"和"量子力学"的距离。表示里没有任何语义信息。

罪三（致命）：不能泛化。 训练数据里出现过"我养了一只猫"，模型学到了规律。现在来了"我养了一只狗"。因为 $e_{\text{猫}} \perp e_{\text{狗}}$ ，从 Day 1 §5.3 的外积公式 $\partial L / \partial W = g x^\top$ 能直接看出来：输入是 $e_{\text{猫}}$ 时，只有 W 的"猫"那一列被更新，"狗"那一列一动不动。模型学到的关于猫的一切，对狗完全用不上。

罪一是工程问题，罪二罪三是根本问题。

1.2 分布式表示

直觉。 罪三的根源是"不同词的表示没有共享的部分"。那就让它们共享：用一个稠密低维向量 $v_w \in \mathbb{R}^d$ (d 通常 100–1000)，让语义相近的词在空间中靠近。这样更新"猫"的时候，"狗"附近的区域也会被影响，泛化就发生了。

"分布式"的意思是：语义分散在所有维度上，没有哪一维专门代表"动物性"，语义是整个向量的方向。

但问题来了：这些向量从哪来？ 总不能人工标注 10 万个词的 300 维坐标。

1.3 分布式假设

Firth (1957) :

You shall know a word by the company it keeps.

一个词的意义由它的上下文决定。两个词经常出现在相似的上下文里，它们的意义就相近。

这句话把一个哲学问题（什么是词义）变成了一个统计问题（上下文分布的相似度）。今天剩下的所有内容，都是在实现这一句话。

§2 计数派：直接统计共现

2.1 共现矩阵

开一个窗口（比如前后 5 个词），数"词 w 和词 c 在同一窗口中共现了多少次"，得到矩阵 $C \in \mathbb{R}^{|V| \times |V|}$ ，取第 w 行当 w 的表示。

这直接实现了分布式假设，而且完全不需要训练。但有三个问题。

2.2 问题一：原始计数是坏的度量 $\rightarrow$ PMI

动机。"the"、"of" 和谁的共现次数都高。原始计数主要反映词频，而不是关联。

直觉。真正有信息的不是" c 出现了多少次"，而是"**知道了 w 之后， c 出现的可能性比平时高了多少**"。

推导。把这句话直接写成公式：

$$\text{PMI}(w, c) = \log \frac{P(c | w)}{P(c)} = \log \frac{P(w, c)}{P(w)P(c)}$$

分子是实际共现概率，分母是"假设两者独立"时的期望共现概率。比值大于 1（取对数后为正）说明关联超出偶然。

实践中用 **PPMI**（取正部 $\max(\text{PMI}, 0)$ ）。负 PMI 表示"比偶然更少共现"，在稀疏数据下极不可靠：一对词没一起出现过，可能只是语料不够大。

2.3 问题二：维度太高且稀疏 → SVD

$|V| \times |V|$ 的矩阵存不下也用不了。做**截断 SVD** 降到 d 维：

$$M \approx U_d \Sigma_d V_d^\top, \quad W = U_d \Sigma_d^{1/2}$$

取 W 的行作为词向量。这是 LSA（潜在语义分析）的做法。

降维不只是为了省空间，它本身就在做泛化。强迫 $|V|$ 个词挤进 d 维，模型就不得不让出现在相似上下文里的词共享坐标。"猫"和"狗"的共现行不完全一样，但在最重要的那几个奇异方向上很接近，压缩之后就被放到了一起。

2.4 问题三：工程上不可扩展

SVD 在 $|V| = 10^5$ 时就很吃力；更要命的是，**加一批新语料就得整个重算**，无法增量更新。

于是有了**预测派**：**不构造矩阵，用 SGD 一对一对地学**。这就是 Word2Vec。

§3 Word2Vec：把表示学习变成预测任务

3.1 Skip-gram 的目标

直觉。§2.4 的墙是"构造矩阵"。能不能不看整个矩阵，每次只看一对（中心词，上下文词），用 SGD 慢慢把向量调好？可以，只要把"共现"改写成一个**预测任务**：给定中心词，预测它的上下文。

两种架构：CBOW（用上下文预测中心词）和 **Skip-gram**（用中心词预测上下文）。Skip-gram 在低频词上更好，是主流，下面只讲它。

给定语料 $w_1, \dots, w_T$ 和窗口半径 m ，最大化

$$\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t), \quad P(c | w) = \frac{\exp(u_c^\top v_w)}{\sum_{c' \in V} \exp(u_{c'}^\top v_w)}$$

每个词有两套向量：作中心词时的 v_w ，作上下文时的 u_c （§3.6 解释为什么）。这其实就是 Day 1 §2 的交叉熵，只是类别数是整个词表。

3.2 全 softmax 的梯度：看清瓶颈在哪

动机。在跳到负采样之前，先把全 softmax 的梯度推出来。不看清原方案的梯度长什么样，就不会明白负采样替换掉的到底是什么。

推导。用 Day 1 §5.2.4 的捷径：

$$\log P(c | w) = u_c^\top v_w - \log \sum_{c'} \exp(u_{c'}^\top v_w)$$

对 v_w 求导，第二项的导数是 softmax 加权平均：

$$\frac{\partial \log P(c | w)}{\partial v_w} = \underbrace{u_c}_{\text{真实上下文}} - \underbrace{\sum_{c'} P(c' | w) u_{c'}}_{\text{模型"期望"的上下文}}$$

`day3_checks.py` 验证，误差 10^{-10} 。

直觉。这个梯度说的是：把 v_w 往真实上下文 u_c 拉，同时往模型当前认为的"平均上下文"的反方向推。模型已经预测对时，两项相等、梯度为 0。

这个"数据项 - 模型期望项"的结构是所有 softmax 型对数似然梯度的通式（Day 1 §5.2 的 $\hat{y} - y$ 是它在 logits 上的样子），在能量模型里也叫"正相位 - 负相位"。

瓶颈就在第二项： $\sum_{c'}$ 要遍历全部词表，每处理一个训练对就要算 $|V|$ 次内积。用真实数字感受一下（`day3_checks.py`, $|V| = 10^5$, $d = 300$, 50 亿训练对）：

方案	FLOPs
完整 softmax	3.0×10^{17}
负采样 ($k = 5$)	1.8×10^{13}

差 16,667 倍。这不是"优化一下更快"，这是可行与不可行的分界线。Word2Vec 真正的贡献不是架构（架构简单得像个玩具），而是怎么绕开这一项。

3.3 负采样：推导的思路

这是今天最值得学的一段，因为这个套路会反复出现。

第一步：看清要替换的是什么。§3.2 的瓶颈是一个对整个词表的期望。精确计算它需要归一化常数 $\sum_{c'} \exp(\cdot)$ ，而这个常数正是 softmax 分母。

第二步：换问题。原问题是"在 $|V|$ 个词里选出正确的上下文"，一个 $|V|$ 类分类，分母躲不掉。换成：

"这个 (词, 上下文) 对是真的从语料里来的，还是我瞎编的？"

一个二分类问题。二分类只要一个 sigmoid，没有分母。

第三步：构造数据。真实对 (w, c) 来自语料，标签 1；对每个真实对，从噪声分布 P_n 采 k 个词编造假对，标签 0。

第四步：写目标。对每个真实对：

$$J = \log \sigma(u_c^\top v_w) + \sum_{i=1}^k \mathbb{E}_{c_i \sim P_n} [\log \sigma(-u_{c_i}^\top v_w)]$$

计算量从 $|V|$ 降到 $k+1$ (典型 $k = 5 \sim 20$)。

直觉：它替换掉了什么。 对照 §3.2 和下面 §3.4 的梯度，结构完全一样：拉近真实上下文，推远"别的词"。区别只在"别的词"从"模型的精确期望（要遍历词表）"换成了" k 个随机样本"。

诚实版：负采样和全 softmax 不是同一个目标。 有一个更严格的方法叫 **NCE** (Gutmann & Hyvärinen 2010)，它在 logit 里多减一项 $\log(kP_n(c))$ ，可以证明当 $k \rightarrow \infty$ 时它恢复出**正确归一化**的 softmax 模型。负采样 (NEG) 为了简单把这一项扔掉了，所以它**不是** softmax 的一致估计，学到的也不是归一化的条件概率。它优化的是另一个目标，而这个目标的最优解恰好是一个很好的表示 (§4.2 会精确算出来是什么)。做**词向量**时这完全没问题；要做**需要校准概率的语言模型**，就不能用 **NEG**。

意义。 提炼成一句可迁移的套路：

模型需要一个"对所有候选归一化"的分母、而候选太多时，就把"从所有候选中选一个"换成"区分真样本和假样本"。归一化常数就消失了。

这条线后来发展成 **InfoNCE** (van den Oord 等, 2018)：从 k 个候选里挑出唯一的正样本，是一个小规模 softmax。CLIP、SimCLR、以及第 4 周 RAG 用的 embedding 模型，训练时全是它，通常还直接把同一个 batch 里的其他样本当负样本。

3.4 SGNS 的梯度

用 Day 1 的工具推。零件： $\frac{d}{dx} \log \sigma(x) = 1 - \sigma(x)$, $\frac{d}{dx} \log \sigma(-x) = -\sigma(x)$ 。

$$\boxed{\frac{\partial J}{\partial v_w} = \underbrace{(1 - \sigma(u_c^\top v_w)) u_c}_{\text{正样本: 拉近}} - \underbrace{\sum_{i=1}^k \sigma(u_{c_i}^\top v_w) u_{c_i}}_{\text{负样本: 推远}}}$$

$$\frac{\partial J}{\partial u_c} = (1 - \sigma(u_c^\top v_w)) v_w, \quad \frac{\partial J}{\partial u_{c_i}} = -\sigma(u_{c_i}^\top v_w) v_w$$

(三个公式均已数值验证，误差 10^{-10} 。)

读一下两项的系数，非常漂亮：

- 正样本的系数 $1 - \sigma(\cdot)$ 是"还差多少才算学会"。已经学好的对 ($\sigma \rightarrow 1$) 不再产生梯度。
- 负样本的系数 $\sigma(\cdot)$ 是"这个假对被误判得多厉害"。一眼就能看出是假的负样本 ($\sigma \rightarrow 0$) 也不产生梯度。

梯度自动集中在还没学会的样本上，一种天然的难例挖掘。这个"误差即权重"的结构和 Day 1 §5.2 的 $\hat{y} - y$ 是同一个味道。

3.5 为什么噪声分布是 $U(w)^{0.75}$

Word2Vec 用 $P_n(w) \propto U(w)^{3/4}$ (U 是 unigram 频率)，而不是 $U(w)$ 本身。原论文说是实验挑的，但它的作用能算清楚。

`day3_checks.py` 在 Zipf 分布 ($U \propto 1/\text{rank}$, 真实语料的经验规律) 上实测：

词频排名	原始 $U(w)$	$U(w)^{0.75}$ 归一化后	变化
第 1 名	1.02×10^{-1}	2.74×10^{-2}	0.27x
第 100 名	1.02×10^{-3}	8.65×10^{-4}	0.85x
第 10000 名	1.02×10^{-5}	2.74×10^{-5}	2.68x

最高频 100 个词占的采样概率从 **0.53 降到 0.25**。

为什么这样更好？ 按纯频率采样，一半的负样本预算会花在一百来个功能词（the、of、a.....）上。这些词又恰好几乎是所有词的**真实**上下文，模型在它们身上被正样本和负样本反复拉扯，却学不到多少语义区分。**真正需要"推开"的，是那些中低频的、语义上可能被混淆的词**：让"猫"和"狗"、"银行"和"河岸"分开，才是负样本该做的事。指数 0.75 介于 1（纯频率）和 0（均匀）之间，把预算从功能词挪给了这些词，是一个经验甜点。

（同理，Word2Vec 还对高频词做 **subsampling**：以概率 $1 - \sqrt{t/f(w)}$ 丢掉高频词的训练对，理由相同。）

§4.2 会看到，0.75 不只影响训练效率，它还改变了最终学到的矩阵本身。

3.6 两个遗留问题

为什么每个词有两套向量？ 若只有一套，一个词出现在自己的窗口里时，目标中会出现 $v_w^\top v_w = \|v_w\|^2$ ，模型可以靠把向量拉长来"作弊"，梯度行为很怪。两套向量让优化更干净。训练完通常只用 v_w ，或取 $v_w + u_w$ 。

层次 softmax 是另一条绕开分母的路：把词表组织成 Huffman 树，预测一个词变成沿树走 $\log_2 |V|$ 步的一串二分类， $O(\log |V|)$ 。实践中负采样更常用也更简单，知道它存在即可。

§4 两派其实是一回事

4.1 GloVe：从"比值"出发设计出一个模型

动机。 计数派和预测派各有优劣：计数派用了全局统计但不能增量，预测派能增量但每次只看局部窗口。Pennington 等人（2014）想要一个**直接建模全局共现统计**、又能用 SGD 训练的模型。问题是：该对共现统计的什么建模？

直觉：有意义的是比值，不是概率本身。 考虑 ice 和 steam，用探针词 k 去看 $P(k | \text{ice})/P(k | \text{steam})$ ：

探针 k	比值	含义
solid	很大	只和 ice 相关
gas	很小	只和 steam 相关
water	≈ 1	都相关， 比值把共有的部分约掉了
fashion	≈ 1	都不相关

比值把两者共有的部分约掉，只留下区分性的信息。 那就让模型去拟合比值。

推导（设计推导：每一步都在回答"要满足这个性质，形式必须是什么"）。 目标是找一个函数，使得 $F(w_i, w_j, \tilde{w}_k) = P_{ik}/P_{jk}$ ($P_{ik} = P(k | i)$)。

1. 向量空间是线性的，比较 i 和 j 最自然的方式是做差： $F(w_i - w_j, \tilde{w}_k)$ 。
2. 左边参数是向量，右边是标量，用点积把它们连起来： $F((w_i - w_j)^\top \tilde{w}_k) = P_{ik}/P_{jk}$ 。
3. 左边是减法，右边是除法，所以 F 必须把减法变成除法： $F(a - b) = F(a)/F(b)$ 。满足这个性质的连续正函数只有指数函数 e^{cx} （柯西函数方程），常数 c 可以被向量的尺度吸收，于是取 $F = \exp$ 。
4. 于是 $\exp(w_i^\top \tilde{w}_k) = P_{ik} = X_{ik}/X_i$ ，即

$$w_i^\top \tilde{w}_k = \log X_{ik} - \log X_i$$

5. $\log X_i$ 与 k 无关，吸收进偏置 b_i ；为了让词和上下文地位对称，再加一个 $\tilde{b}_k$ ：

$$w_i^\top \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$

6. 用加权最小二乘拟合，只对 $X_{ik} > 0$ 的项求和（避开 $\log 0$ ）：

$$J = \sum_{i,k: X_{ik} > 0} f(X_{ik}) (w_i^\top \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik})^2$$

权重 $f(x) = (x/x_{\max})^{0.75}$ ($x < x_{\max}$ 时，否则为 1) 压低罕见对（统计上太嘈杂）、封顶高频对（不让它们主导）。又是 0.75，和 §3.5 一样是经验选择。

意义。 GloVe 是显式地对对数共现矩阵做低秩分解。由第 4 步， $(w_i - w_j)^\top \tilde{w}_k = \log(P_{ik}/P_{jk})$ 加上偏置差，向量差编码的就是对数共现比值。§5.1 的类比现象就是从这里来的。

4.2 Levy-Goldberg：负采样在隐式分解什么

动机。 GloVe 是明着分解矩阵。Word2Vec 看上去是在“预测”，和矩阵分解毫无关系。真的吗？

直觉。 如果词向量维度足够大，每一对 (w, c) 的内积 $x_{wc} = v_w^\top u_c$ 几乎可以独立地取任何值。那就把 SGNS 在整个语料上的目标，按 (w, c) 对拆开，看每个 x_{wc} 的最优值是多少。

推导。 设 $\#(w, c)$ 是共现次数， $\#(w) = \sum_c \#(w, c)$ ，噪声分布 $P_n(c)$ 。对整个语料求和，SGNS 目标中和某一对 (w, c) 有关的只有两部分：

- 正样本： (w, c) 在语料里出现了 $\#(w, c)$ 次，贡献 $\#(w, c) \log \sigma(x)$
- 负样本： w 共出现 $\#(w)$ 次，每次采 k 个负样本，其中恰好采到 c 的期望次数是 $k \#(w) P_n(c)$ ，贡献 $k \#(w) P_n(c) \log \sigma(-x)$

所以

$$\ell(x) = \#(w, c) \log \sigma(x) + k \#(w) P_n(c) \log \sigma(-x)$$

对 x 求导（用 §3.4 的两个零件）并令其为 0：

$$\#(w, c) \sigma(-x) = k \#(w) P_n(c) \sigma(x)$$

利用 $\sigma(x)/\sigma(-x) = e^x$ ：

$$e^x = \frac{\#(w, c)}{k \#(w) P_n(c)} \Rightarrow x^* = \log \frac{\#(w, c)}{\#(w) P_n(c)} - \log k$$

当 $P_n(c) = \#(c)/|D|$ （纯 unigram，指数取 1）时，第一项正是 PMI：

$$v_w^\top u_c = \text{PMI}(w, c) - \log k$$

Word2Vec 表面上在做预测，实际上是在对"移位的 PMI 矩阵"做低秩分解，正是 §2 计数派在做的事。

验证方式。 `day3_checks.py` 不用上面的闭式解，而是对每一对 (w, c) 纯数值地求 $\ell'(x) = 0$ 的根 (ℓ' 关于 x 单调，用二分法)，然后和闭式解比较：误差 8.9×10^{-16} ，机器精度。噪声指数取 0.75 时同样成立，只是 PMI 里的 $P(c)$ 换成了平滑后的 $P_\alpha(c)$ 。所以 §3.5 的 0.75 不只是训练技巧，它改变了最终被分解的那个矩阵。

(前提是维度足够大、每对内积能独立取值。实际 $d = 300$ 远小于 $|V|$ ，SGNS 学到的是这个矩阵的一个低秩近似。这正是 §2.3 说的"降维本身在做泛化"。)

意义。"计数派 vs 预测派"的门派之争，数学上是假问题。真正的差别在工程：SGD 能增量、能扩展到任意大的语料，SVD 不行。

很多看起来是"思想之争"的东西，拆开看只是"工程可行性之争"。这个判断在读论文时非常有用。

§5 词向量的几何

5.1 类比：king - man + woman $\approx$ queen

为什么会成立。由 §4.1 第 4 步，向量差编码的是对数共现比值：

$$(v_{\text{king}} - v_{\text{man}})^\top \tilde{w}_k \approx \log \frac{P(k | \text{king})}{P(k | \text{man})}$$

"king 相对 man 多出来的那部分上下文" (crown、throne、reign.....) 被编码成一个方向。"queen 相对 woman"多出来的上下文几乎是同一批词，所以两个差向量近似平行：

$$v_{\text{king}} - v_{\text{man}} \approx v_{\text{queen}} - v_{\text{woman}} \Rightarrow v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}$$

语义关系被编码成了空间里的平移。

必须知道的真相：标准评测在找答案时会显式排除输入的三个词。不排除的话，结果向量的最近邻常常就是输入词本身（通常是 king，有时是 woman），因为 $b - a + c$ 本来就离 b 和 c 很近。§A.2 给了绕开这个排除的代码，你可以亲手验证。

类比现象是真的，但比宣传的脆弱。它在性别、国家-首都这类高频、规整的关系上成立得好，在别的关系上经常失效。知道一个方法的边界，比知道它的成功案例更重要。

5.2 各向异性

训练出的词向量在空间里**不是各向同性的**：它们挤在一个狭窄的锥里，任意两个词的余弦相似度普遍偏高（常 0.5 以上），相似度的区分度被压缩。缓解办法有去均值、白化、去掉前几个主成分。

这个问题在 BERT 的句向量上更严重。第 4 周挑 embedding 模型时你会直接碰上它，这也是为什么用 InfoNCE 专门训练过的句向量模型 (§3.3 末尾)，比直接拿 BERT 的 [CLS] 好用得多。

5.3 为什么用余弦而不是欧氏距离

词向量的**模长**主要反映词频（高频词训练次数多，模长大），**方向**才携带语义。余弦相似度归一化掉模长，只比较方向：

$$\cos(u, v) = \frac{u^\top v}{\|u\| \|v\|}$$

这条规则一路管到现代：RAG 的向量检索、embedding 相似度、对比学习的损失，默认全是余弦。

§6 天花板：为什么必须走向上下文相关表示

这一节是今天通往 **Lecture 1** 的门。

动机。 Word2Vec 给每个词一个**固定**向量。看这两句：

我去**银行**取钱。他坐在**河岸**边钓鱼。（英文里都是 bank）

一个固定的 v_{bank} 必须同时表示"金融机构"和"河岸"。由 §4.2，它是在拟合 bank 那一行的 PMI，而那一行是**两种语义上下文的混合**。结果 v_{bank} 落在两个语义的某种加权平均位置，哪个意思都不像。

这是静态词向量的根本天花板：一个词一个向量，无法表达一词多义。 它不是训练不够或维度不够造成的，而是**表示的形式本身**决定的。

直觉。 解法只有一个方向：让词的表示**依赖它所在的上下文**。

$$v_w \longrightarrow v_w(\text{context})$$

这个方向的历史：

1. **ELMo (2018)**：用双向 LSTM 读整句，每个词的表示由上下文动态生成。方向对了，但 LSTM 是**顺序**的，无法并行，长距离依赖仍然衰减（Day 4 §2 会看到为什么）。
2. **Transformer / BERT (2017–2018)**：用**自注意力**让每个词直接看到句子里所有其他词，**一步到位，完全并行**。

注意力机制，就是"怎么根据上下文动态生成表示"这个问题的答案。 明天 Day 4 会看到 RNN/LSTM 是怎么尝试的、为什么没做好，以及注意力是怎么被逼出来的。

§7 分词：为什么 LLM 既不用词也不用字符

Lecture 1 会正式讲分词，今天先建立框架，并把 BPE 从头推一遍。

7.1 词级分词的问题

- **OOV (未登录词)**：训练时没见过的词全变成 **<UNK>**，信息直接丢失。新词、拼写错误、专有名词是无穷无尽的。
- **词表爆炸**：形态丰富的语言（德语复合词、土耳其语黏着）词表能到百万级。
- **形态学浪费**：run / runs / running / ran 被当成四个无关的词，各学各的向量，§1.1 的罪三又回来了。
- **中日文没有空格**，"词"的边界本身就要靠另一个分词器来定义。

7.2 字符级分词的问题

- 序列长度暴增 5–10 倍。注意力是序列长度的**平方**复杂度，这个代价无法接受。
- 单个字符几乎不携带语义，模型要花很多层才能组合出词义。

7.3 BPE：把分词问题当成压缩问题

动机。 需要一个折中：常见的词保持完整，罕见的词拆成有意义的片段，最差退化到字符（所以永远没有 OOV）。问题是：**该把哪些片段放进词表？** 手工定规则不可扩展，也不跨语言。

直觉。 换个角度看：词表大小是固定的预算（比如 5 万个符号），目标是**让语料编码后的序列尽量短**（序列越短，注意力越便宜，每个 token 携带的信息越多）。**这就是一个压缩问题。** 那什么子串最值得分配一个专门的符号？当然是**出现最频繁的那个**：每给它一个符号，都能省掉最多的长度。

BPE 本来就是一个压缩算法（Gage 1994），Sennrich 等人（2016）把它拿来做分词。

算法。

1. 从字符级开始，每个词末尾加一个结束标记 `</w>`（区分"词中的 est"和"词尾的 est"）
2. 统计所有**相邻符号对**的出现次数（按词频加权）
3. 把**最高频的那一对**合并成一个新符号，加入词表
4. 重复 2-3，直到词表达达到预算

演示（`day3_checks.py`，Sennrich 原文的玩具语料：low×5、lower×2、newest×6、widest×3）：

步	合并	次数	总符号数
0	(字符级)	—	95
1	<code>e + s</code>	9	86
2	<code>es + t</code>	9	77
3	<code>est + </w></code>	9	68
4	<code>l + o</code>	7	61
5	<code>lo + w</code>	7	54
8	<code>new + est</w></code>	6	36
10	<code>w + i</code>	3	28

十次合并后序列总长从 95 压到 28。注意它自动发现了 `est</w>`（最高级后缀）和 `low`（词根）：**没有人告诉它什么是词素，频率统计自己找到了形态学结构。**

推理时怎么切？ 对新词，按训练时学到的**合并顺序**依次应用合并规则。比如 `lowest` 会先被切成字符，然后依次合并出 `low` 和 `est</w>`，得到 `low + est</w>`，即使训练语料里从没出现过 `lowest`。

意义与代价。 没有 OOV（最差退化到字符），词表可控（3 万-20 万），保留了形态信息。代价是**切分结果依赖语料统计**，不一定符合人类的语言学直觉，而且会带来下面 §7.4 那些奇怪的现象。

两个工程变体，Lecture 1 会细讲：

- **Byte-level BPE**（GPT-2 起）：在 UTF-8 字节上而不是 Unicode 字符上做 BPE，基础符号只有 256 个，任何语言、任何 emoji 都能编码，**彻底没有 OOV**。
- **WordPiece**（BERT）、**SentencePiece / Unigram**（多语言模型）：合并准则或切分方式不同，但都是"频率驱动的子词"这一个思想。

7.4 分词直接导致的三个现象

你在用 LLM 时一定遇到过：

1. **模型数不清字母**。"strawberry 里有几个 r"之所以难，是因为模型看到的是几个子词 token，而不是 10 个字母。字母 r 不是一个独立的单位。
2. **多语言不公平**。同一句话，英文可能 10 个 token，中文或泰文可能多得多。训练和推理成本、上下文窗口占用都不一样。原因就在 §7.3：BPE 的合并由训练语料的频率决定，语料以英文为主时，英文片段得到了更多合并。
3. **按 token 计费**，不是按字数。

§A.10 有验证这三条的代码。

§A gensim 与 tokenizer：API 用法与边界

A.1 `gensim.models.Word2Vec`：参数全对上今天的理论

```
from gensim.models import Word2Vec

model = Word2Vec(
    sentences,                # 可迭代的「词列表」的列表，不是字符串！
    vector_size=300,         # 词向量维度 [gensim 3.x 里叫 size]
    window=5,               # 上下文半径 m (§3.1)
    sg=1,                   # 1=Skip-gram, 0=CBOW ★ 默认是 0
    negative=5,             # 负样本个数 k (§3.3)
    ns_exponent=0.75,       # 噪声分布指数 (§3.5)
    min_count=5,            # 丢弃出现次数 < 5 的词 ★ 默认就是 5
    sample=1e-3,            # 高频词下采样阈值 (§3.5 末尾)
    epochs=5,               # 训练轮数 [gensim 3.x 里叫 iter]
    workers=8,              # 并行线程
)
```

每个参数都能在今天的文档里找到出处——这是检验你是否真的读懂了的好办法。

五个边界：

1. **sg 默认是 0 (CBOW)**，不是 Skip-gram。今天讲的全是 Skip-gram，想复现就必须显式写 **sg=1**。这是最常见的「我照着教程做了但结果不一样」的原因。
2. **sentences 必须是已分好词的列表的列表**：`[["the", "cat", "sat"], ...]`。直接传字符串列表不会报错，但 gensim 会把每个字符当成一个词，训出来全是垃圾。这个 **bug** 不报错，只产出无意义的结果。
3. **min_count=5 会静默丢词**。之后查询被丢掉的词会抛 **KeyError**。小语料上要调小，或者先 `if w in model.wv` 判断。
4. **workers>1 需要 Cython**。没装的话 gensim 会退化成单线程并给一个警告，速度差几十倍。装了 `pip install gensim` 通常自带，但用源码安装时要留意。
5. **gensim 4.0 改了参数名**：`size→vector_size`，`iter→epochs`，`model[word]→model.wv[word]`。网上大量教程还是 3.x 的写法，照抄会报 **TypeError**。

A.2 `most_similar` 与那个"排除输入词"的细节

```
model.wv.most_similar("cat", topn=10)
model.wv.most_similar(positive=["king", "woman"], negative=["man"], topn=5)
```

第二行就是类比实验。而 §5.1 讲的那个真相，就藏在这个 API 里：`most_similar` 默认会把 **positive** 和 **negative** 里出现过的词全部排除出候选。

想亲眼验证 §5.1 的说法，绕开这个排除：

```
import numpy as np
wv = model.wv
v = wv["king"] - wv["man"] + wv["woman"]
v = v / np.linalg.norm(v)
sims = wv.vectors @ v / np.linalg.norm(wv.vectors, axis=1)      # 手算余弦 (§5.3)
top = np.argsort(-sims)[:5]
print([(wv.index_to_key[i], round(float(sims[i]), 4)) for i in top])
```

排在最前面的很可能就是输入词本身（king 或 woman）。这就是 §5.1 说的「类比现象是真的，但比宣传的脆弱」——标准评测靠排除输入词把这件事掩盖了。亲手跑一次比读十遍更有说服力。

其他：`most_similar` 内部用的是归一化后的向量（余弦相似度，§5.3）；`wv.similarity(a, b)` 直接给两个词的余弦值；`wv.index_to_key` 是按词频降序的词表，`wv.key_to_index` 是反向映射。

A.3 保存与加载

```
model.save("w2v.model")          # 完整模型，可继续训练
wv = model.wv; wv.save("w2v.kv") # 只存向量，体积小得多
wv.save_word2vec_format("w2v.txt", binary=False) # 通用文本格式，跨工具
```

只要用向量就别存完整模型——完整模型包含训练用的内部矩阵（上下文向量 u_c 等，§3.6），体积通常是纯向量的两倍以上。

A.4 HuggingFace tokenizer：三个方法的区别

```
from transformers import AutoTokenizer
tok = AutoTokenizer.from_pretrained("gpt2")

tok.tokenize("Hello world")      # ['Hello', 'Ġworld']      → 子词字符串，不加特殊 token
tok.encode("Hello world")        # [15496, 995]            → id 列表，默认加特殊 token
```

```
tok("Hello world") # {'input_ids': [...], 'attention_mask': [...]} → 最常用
```

那个 **Ġ** 是什么？GPT-2 的 BPE 用 **Ġ** (U+0120) 表示前导空格。所以 **"world"** 和 **" Ġworld"** 是两个不同的 **token**。这直接导致一个实际现象：prompt 结尾多一个空格，模型行为可能明显不同——因为下一个 token 的候选集变了。

__call__ 返回的 **attention_mask** 标记哪些位置是真实 token (1)、哪些是 padding (0)。它最终会变成 Day 4 §A.4 讲的那个掩码。

A.5 padding / truncation 的参数组合

```
batch = tok(texts, padding=True, truncation=True, max_length=512,
return_tensors="pt")
```

- **padding=True** (等价 **'longest'**) 补到 batch 内最长；**padding='max_length'** 补到 **max_length**。
- **truncation=True** 不写的话，超长序列会原样返回，然后在模型那里报位置编码越界——错误信息通常指向模型内部，很难联想到是分词这一步的问题。
- **return_tensors="pt"** 才给 torch 张量，不写返回 Python list。
- 有些 **tokenizer** 没有 **pad_token** (GPT-2 就没有)，**padding=True** 会报 *"Asking to pad but the tokenizer does not have a padding token"*。标准做法是 **tok.pad_token = tok.eos_token**。

A.6 padding_side: 训练和生成要用不同的值 ★

```
tok.padding_side = "right" # 训练：右侧补齐
tok.padding_side = "left"  # 批量生成：左侧补齐
```

这是第 4 周做批量推理时最容易踩的坑，而且它不报错，只是结果乱掉。

原因：decoder-only 模型逐 token 生成时，总是取序列最后一个位置的 logits 来预测下一个。如果在右侧补了 padding，"最后一个位置"就是一个 pad token——模型基于 pad 去续写，输出自然是垃圾。左侧补齐则保证每个样本的真实内容都结束在最右端。

记法：训练时 loss 靠 **ignore_index** 跳过 padding，补哪边都行；生成时靠**"最后一个位置"**，所以必须补左边。

A.7 vocab_size vs len(tokenizer)

```
tok.add_special_tokens({"additional_special_tokens": ["<T00L>"]})
print(tok.vocab_size, len(tok)) # 50257 50258 ← 不相等!
model.resize_token_embeddings(len(tok)) # ★ 必须用 len(tok)，不是 vocab_size
```

vocab_size 是原始词表大小，不包含你后加的 token；**len(tokenizer)** 才是当前总数。

漏掉 `resize_token_embeddings` 的后果：新 token 的 id 超出 embedding 表的行数，在 CPU 上报 index out of range，在 CUDA 上则是一句几乎没法读的 **device-side assert**（Day 4 §A.5 提过这个现象）。

resize 之后新增行是随机初始化的，需要训练才有意义。第 2 周做 SFT 加特殊 token 时会用到这一整套。

A.8 fast vs slow tokenizer

```
tok = AutoTokenizer.from_pretrained("gpt2", use_fast=True) # 默认
out = tok("Hello world", return_offsets_mapping=True) # 只有 fast 版支持
print(out["offset_mapping"]) # [(0,5), (5,11)] — 每个 token 在原文中的字符区间
```

- Fast 版由 Rust 实现，批量编码快一个数量级。
- **offset_mapping** 只有 fast 版有，它把 token 映射回原文字符位置——做抽取式问答、实体识别、以及第 4 周 RAG 的引用定位时必需。
- 极少数模型只有 slow 实现，`use_fast=True` 会静默回退。需要 offset 时记得 `assert tok.is_fast`。

A.9 解码

```
tok.decode(ids, skip_special_tokens=True) # 常用：去掉
<|endoftext|> 等
tok.batch_decode(batch_ids, skip_special_tokens=True)
tok.convert_ids_to_tokens(ids) # 保留 Ġ 等标记，调试用
```

decode 不保证和原文字节级一致——BPE 的规范化（空格处理、Unicode NFKC 等）可能改变原文。需要严格还原时用 **offset_mapping** 回原文切片，而不是 **decode**。

A.10 亲手验证 §7.4 的三个现象

```
tok = AutoTokenizer.from_pretrained("gpt2")

# 现象 1: 模型数不清字母 — 它看到的是 token 不是字符
print(tok.tokenize("strawberry")) # 会被切成几个子词（具体切法看分词器），'r' 不是独立单位

# 现象 2: 多语言不公平
en, zh = "Hello, how are you today?", "你好，你今天过得怎么样？"
print(len(tok.encode(en)), len(tok.encode(zh))) # 中文 token 数明显更多

# 现象 3: 空格是 token 的一部分
print(tok.encode("world"), tok.encode(" world")) # 两个不同的 id
```

跑一遍，§7.4 那三条就从“听说”变成“见过”了。

§B 实操

1. 跑通 `day3_checks.py`，六组结论逐一看懂。第 1、2 组放在一起看，体会负采样到底替换掉了什么；第 5 组是 Levy-Goldberg 的纯数值验证。
2. 训一个 Word2Vec (`pip install gensim`，用 `text8` 或任意中等规模语料，记得 `sg=1`，见 §A.1)：
 - 看几个词的最近邻，找到成功案例
 - 重点找失败案例：一词多义的词 (`bank / apple / 苹果`) 的最近邻是不是两个意思混在一起？这直接印证 §6
 - 复现类比，再用 §A.2 的代码关掉输入词排除，看结果变成什么 (§5.1)
 - 算一批随机词对的余弦相似度，看均值是否明显大于 0 (§5.2 的各向异性)
 - 参考：先跑 `code/d3_sgns_numpy.py` (手写 SGNS，合成语料上几秒训完，能用断言检验 `bank` 的一词多义)，再跑 `code/d3_gensim_explore.py --text8` 在真实语料上看同样的现象。注意 `text8` 没有句子边界，整段传给 `gensim` 会被静默截断到前 10000 个词，脚本里的 `chunk()` 就是为此写的
3. 手写 BPE：不看 `day3_checks.py`，自己实现训练和推理两部分，在一小段英文上学 200 次合并，看它学出了哪些词根和后缀。参考：`code/d3_bpe.py`。
4. 玩分词器：跑 §A.10 的代码，对比同一句话的中英文 token 数。参考：`code/d3_tokenizer_demo.py`，把 §A.4–A.9 的边界逐个踩一遍，最后和你手写的 BPE 并排对比。

全部参考实现和一键自检见 `code/README.md`。

§C 自测

动机与直觉层

1. one-hot 的三宗罪是什么？用 Day 1 的外积公式说明为什么"无泛化"是根本的。
2. 为什么共现计数要换成 PMI？用"知道 w 之后 c 的可能性提高了多少"解释。
3. SVD 降维为什么本身就在做泛化？
4. 负采样把什么问题换成了什么问题？为什么这样分母就消失了？它替换掉的是全 softmax 梯度里的哪一项？
5. 负采样和 NCE 有什么区别？什么场景下不能用负采样？
6. GloVe 为什么要对"比值"建模而不是对概率本身？
7. 静态词向量的天花板是什么？为什么它必然导向注意力？
8. 为什么说 BPE 是一个压缩算法？它为什么能自动发现词根和后缀？

推导层

9. 推出全 softmax 下 $\partial \log P(c | w) / \partial v_w$ ，解释两项的含义。
10. 推出 SGNS 对 v_w 的梯度，解释正负两项系数的含义。
11. 从"要满足减法变除法"出发，推出 GloVe 的目标函数。
12. 完整推出 Levy-Goldberg 结论：写出每对 (w, c) 的目标 $\ell(x)$ ，求导，解出 x^* 。噪声指数取 0.75 时结论变成什么？
13. 用 §4.1 的结论解释 `king - man + woman ≈ queen`，并说出这个现象的已知缺陷。

工程与 API 层

14. $|V| = 10^5$ 、 $d = 300$ 、50 亿训练对时，完整 softmax 和负采样差多少倍？

15. 负采样分布为什么用 0.75 次方？（两个层面：训练效率，以及它对最终矩阵的影响）
 16. 为什么词向量相似度用余弦而不用欧氏距离？
 17. gensim 的 `sg` 默认值是什么？`sentences` 传字符串列表会怎样？
 18. 训练和批量生成时 `padding_side` 各应该设成什么？为什么？
 19. 给 tokenizer 加了特殊 token 之后必须做什么？不做会出什么错？
 20. 举出三个由分词直接导致、用户能感知到的 LLM 行为，并用 BPE 的原理解释第二个。
-

出关标准

- ☐ 能说清 one-hot 三宗罪，并用外积公式说明"无泛化"
 - ☐ 能解释 PMI 的含义和 PPMI 取正部的理由
 - ☐ 能推出全 softmax 梯度"数据项 - 模型期望项"的结构，并指出瓶颈
 - ☐ 能讲清负采样的推导思路，并提炼出"候选太多就改成判别真假"的套路
 - ☐ 能说出负采样与 NCE 的区别
 - ☐ 能推出 SGNS 梯度并解释两项系数
 - ☐ 能从比值出发推出 GloVe 的目标函数
 - ☐ 能完整推出 Levy-Goldberg 结论，并说明 0.75 对最终矩阵的影响
 - ☐ 能解释类比现象成立的原因和它的缺陷
 - ☐ 能完整论证静态词向量的天花板，以及它为什么导向注意力
 - ☐ 能手写 BPE 的训练和推理，并解释它为什么是压缩算法
 - ☐ SA 的 API 坑能说出现象和正确写法（尤其 `sg`、`padding_side`、`resize_token_embeddings`）
-

疑问清单

-
-
-