

第 0 周 · Day 4：序列模型与注意力的诞生

时长 8 小时： §1-§2 RNN 与 BPTT (2h) · §3 LSTM/GRU (2h) · §4-§5 Seq2Seq 与注意力 (2.5h) · §A 序列 API (1h) · §B-§C (0.5h)

配套代码 `day4_checks.py` (纯 numpy)：BPTT 连乘与 tanh 饱和、LSTM 完整雅可比 vs 直连路径 vs RNN、遗忘门 bias 的威力、 $\sqrt{d}$ 缩放与 softmax 雅可比、注意力的低温极限、softmax dim 与掩码 NaN

§B 参考实现 (`code/` 目录)：`d4_adding_problem.py`、`d4_seq2seq_attention.py`。先自己写，再对照；一键自检 `python code/run_selftests.py`，说明见 `code/README.md`。

这是第 0 周最重要的一天。 今天结束时，你应该能说出注意力是被什么困境逼出来的，Q/K/V 在类比什么，以及为什么要除以 $\sqrt{d}$ 。那正是 Lecture 1 的全部动机。

§0 今天的主线

0.1 一段发明史

今天讲的不是一堆架构，而是一段**发明史**。每个架构都是前一个撞墙之后的回应：

```

定长窗口 MLP —撞墙：变长输入、参数不共享—> RNN
RNN —撞墙：梯度沿时间连乘而消失—> LSTM / GRU (加法路径 + 门控)
LSTM+Seq2Seq—撞墙：定长向量装不下整句—> 注意力 (Bahdanau 2014)
RNN+注意力 —撞墙：循环无法并行—> Transformer (Lecture 1)
  
```

每一堵墙都能定量刻画，这就是今天要做的事。学完你会发现，Transformer 不是灵光一闪，而是这条链上第四次、也是最彻底的一次回应。

0.2 贯穿今天的三条旧结论

- Day 1 §7.2：梯度是**矩阵连乘**，增益偏离 1 就指数爆炸或消失 → §2.3 的 BPTT
- Day 1 §7.2：残差的 $I + f'$ **结构**给梯度开高速公路 → §3.4 的 LSTM
- Day 3 §6：静态词向量的天花板，需要**上下文相关表示** → §5 的注意力

另外两条会在 §5.5 用到：Day 1 §5.2 的 softmax 雅可比 $\text{diag}(p) - pp^\top$ ，以及 Day 2 §5.1 的方差传播。

§1 序列建模的问题设定

1.1 三个硬性要求

处理"一句话"这种数据，方案必须同时满足：

1. **变长输入**：句子长短不一，模型不能假定固定长度。
2. **参数共享**："猫"在第 3 位还是第 30 位，处理它的参数应该是同一套。否则参数量随长度线性增长，而且第 30 位学到的东西对第 3 位毫无帮助 (Day 3 §1.1 的"罪三"在时间维度上的翻版)。

3. **捕捉依赖**：“我昨天在北京遇到的那个人说……”，主语和谓语隔了十几个词。

1.2 在 RNN 之前人们怎么做

n-gram 语言模型（统计时代的主力）：用马尔可夫假设截断历史，

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

训练就是数数，快、简单、可解释。**致命伤**： n 增大时参数按 $|V|^n$ 爆炸，绝大多数 n-gram 出现 0 次（稀疏性灾难，靠平滑硬撑）；而且**没有泛化**，“吃苹果”的统计对“吃梨”毫无帮助，因为词是离散符号。

神经语言模型（Bengio et al. 2003）：把前 $n - 1$ 个词的词向量拼起来喂进 MLP。词向量解决了泛化（Day 3 的全部内容）。**但窗口仍是固定的**：看得更远就得加大窗口，参数随之增长，而且“第 5 个位置”和“第 1 个位置”用的是**不同的权重**，要求 2 没满足。

1.3 直觉：为什么会想到“循环”

问题变成：**怎样用固定大小的参数，处理任意长度的输入？**

答案是一个古老的工程思路，**状态机**：维护一个固定大小的状态，每读一个元素就用**同一套规则**更新它：

$$h_t = f(h_{t-1}, x_t; \theta)$$

θ 与 t 无关（参数共享 ✓），序列多长都能跑（变长 ✓）， h_t 理论上能携带任意远的历史（依赖 ✓）。**三个要求一次满足**。信号处理里的 IIR 滤波器、控制论的状态空间模型用的都是这个思路，RNN 是它的可学习版本。

§2 RNN

2.1 前向

最朴素的形式（Elman RNN，PyTorch 的 `nn.RNN`）：

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h), \quad y_t = W_{hy}h_t + b_y$$

为什么是 **tanh** 而不是 **ReLU**？ h 要被反复乘 W_{hh} 。ReLU 无上界，反复自乘极易爆炸；tanh 把状态压在 $(-1, 1)$ ，天然限幅。代价就是 §2.4 的梯度问题。

2.2 BPTT：沿时间反向传播

直觉。把 RNN 沿时间**展开**，它就是一个 T 层的深网络，只是每层**共享同一套权重**。于是 Day 1 的工具可以直接用。

由 Day 1 §6.3（参数共享 $\Rightarrow$ 梯度求和）， W_{hh} 的梯度是所有时间步贡献之和：

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial L_t}{\partial W_{hh}}, \quad \frac{\partial L_t}{\partial W_{hh}} = \sum_{s \leq t} \frac{\partial L_t}{\partial h_t} \cdot \underbrace{\frac{\partial h_t}{\partial h_s}}_{\text{关键}} \cdot \frac{\partial h_s}{\partial W_{hh}}$$

2.3 关键项： $\partial h_t / \partial h_s$

$$\frac{\partial h_t}{\partial h_s} = \prod_{k=s+1}^t \frac{\partial h_k}{\partial h_{k-1}} = \prod_{k=s+1}^t \text{diag}(1 - h_k^2) W_{hh}$$

同一个矩阵 W_{hh} 被连乘了 $t - s$ 次。这是 Day 1 §7.2 的连乘问题，而且比普通深网络更糟：普通深网络每层 W 不同，误差还有机会互相抵消；RNN 是同一个矩阵的幂，行为完全由谱半径 $\rho(W_{hh})$ 决定，没有抵消的余地。

2.4 定量：谱半径说了算

动机。 "连乘会消失或爆炸"是定性的说法。到底会发生什么，要看数字。而且单跑一个随机初始化很容易得出错误结论（这份文档的上一版就踩过这个坑），所以下面是 **40 个随机初始化的统计**。

`day4_checks.py` ($n = 32, T = 60$) :

$\rho(W_{hh})$	中位数	最小	最大	$> 10^3$ 的比例	$< 10^{-3}$ 的比例
0.5	1.0×10^{-18}	5.9×10^{-19}	3.1×10^{-18}	0	100%
0.9	6.2×10^{-4}	2.6×10^{-4}	1.8×10^{-3}	0	88%
1.0	9.2×10^{-2}	1.8×10^{-2}	4.9×10^{-1}	0	0
1.5	8.6×10^1	6.1×10^{-4}	8.0×10^3	10%	3%
2.0	7.1×10^3	4.7×10^{-3}	1.1×10^6	68%	0
3.0	2.3×10^7	1.1×10^{-1}	1.7×10^{10}	93%	0

观察一： $\rho < 1$ 时消失是确定性的。40 次里每一次都指数衰减， $\rho = 0.5$ 时 60 步后只剩 10^{-18} 。60 步之前的信息对当前损失没有梯度贡献，网络在数学上就学不到那么远的依赖。

观察二： $\rho > 1$ 时大概率爆炸，而且极不稳定。 $\rho = 2$ 时 68% 的初始化超过 10^3 ，最小和最大之间差了 8 个数量级：同样的谱半径，有的初始化几乎消失，有的爆到 10^6 。

观察三： **tanh** 的饱和确实在"刹车"，但刹不住。对照一下去掉 tanh 的线性 RNN（同样的 W ）： $\rho = 2$ 时 $\|W^{60}\|$ 的中位数是 3×10^{18} ，加上 tanh 后降到 7×10^3 ，压低了 **14 个数量级**。原因是激活变大后 tanh 进入饱和区， $\tanh' \rightarrow 0$ ，抵消了一部分增益。但剩下的仍然足够爆炸。

观察四： 即使 $\rho = 1$ 恰好平衡， $\tanh' \leq 1$ 还在按复利压缩：

$ a $	$\tanh'(a)$	连乘 60 步
0.0	1.000	1.0
0.5	0.786	5.5×10^{-7}
1.0	0.420	2.5×10^{-23}

意义： 为什么消失才是"逼着人改结构"的那个病。两种病都有，但治疗成本完全不同：

- **爆炸有便宜的解药：** 梯度裁剪（Day 1 §10.3），一行代码，把范数截到阈值以下就行。
- **消失没有：** 梯度已经是 10^{-18} 了，没有任何后处理能把丢失的信息找回来。裁剪只能往下截，不能往上抬。

所以 Pascanu 等人 (2013) 的结论是：爆炸靠裁剪，消失得靠**改结构**。§3 的 LSTM 就是这个"改结构"。

2.5 RNN 的收益与代价

收益：用固定参数处理任意长度、完全的参数共享、理论上无限的记忆，§1.1 三个要求形式上全满足。在它出现的年代，这是巨大的进步。

代价：

1. 有效记忆只有 **10–20 步** ("理论上无限"和"实际上够不着"是两回事)
2. **无法并行**： h_t 依赖 h_{t-1} ， T 步必须串行 T 次。§5.6 会看到这是压垮 RNN 的最后一根稻草
3. 训练慢且不稳

§3 LSTM：用加法路径绕开连乘

3.1 动机：面对梯度消失，有哪几条路

1990 年代中期，梯度消失的诊断已经很清楚 (Hochreiter 1991 的硕士论文、Bengio 等人 1994 的分析)。能做的事有三类：

- **治症状**：梯度裁剪。只能治爆炸，对消失无效。
- **治初值**：精心初始化 W_{hh} (比如正交初始化，让 $\rho = 1$)。有帮助，但 §2.4 观察三说明 $\tanh'$ 那一项还在，治标不治本。
- **改结构**：让梯度有一条**不经过矩阵连乘**的路径。

Hochreiter & Schmidhuber (1997) 选了第三条。

3.2 直觉：把"覆写"改成"累加"

朴素 RNN 的状态更新是**彻底覆写**： $h_t = \tanh(\dots)$ ，旧状态被揉进非线性再整个替换。每次覆写都要过一次 W_{hh} 和一次 $\tanh$ ，于是梯度每退一步就被乘一次。

如果改成累加呢？

$$c_t = c_{t-1} + (\text{本步的新信息})$$

那么沿这条边 $\partial c_t / \partial c_{t-1} = I$ ，梯度原封不动地穿过任意多步。这正是 Day 1 §7.2 残差连接里的那个 I 。

LSTM 比 ResNet 早了 18 年用上同一个思想。

纯累加有个明显问题：信息只进不出， c_t 会无限累积，模型也没办法忘记不再相关的东西（读到新句子，旧句子的主语就该丢了）。于是需要一个可学习的开关控制"保留多少"：

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$f_t \in (0, 1)$ 是遗忘门。 $f_t \approx 1$ 时退化成纯累加， $f_t \approx 0$ 时清空历史。**关键在于 f_t 是学出来的**，模型自己决定什么时候记、什么时候忘。

这就是 LSTM 的核心，其余的门都是在这个骨架上加的控制。

3.3 三个门各自在回答什么问题

门	公式	回答的问题
遗忘门 f_t	$\sigma(W_f[h_{t-1}, x_t] + b_f)$	旧记忆的每一维，保留多少？
输入门 i_t	$\sigma(W_i[h_{t-1}, x_t] + b_i)$	本步的新信息，写入多少？
输出门 o_t	$\sigma(W_o[h_{t-1}, x_t] + b_o)$	内部记忆的每一维，暴露多少给外界？

完整前向：

$$\begin{aligned}
 \tilde{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) && \text{候选记忆} \\
 c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{加法路径 (核心)} \\
 h_t &= o_t \odot \tanh(c_t) && \text{对外的隐状态}
 \end{aligned}$$

为什么门用 σ 、候选用 $\tanh$ ？门表达"保留百分之多少"，值域必须是 $(0, 1)$ 。候选是要加进 c 的内容，需要有正有负（既能增强也能抵消某个特征）。

为什么 c 和 h 要分开？ c 是内部的长期记忆通道，更新是加法的； h 是对外的的工作输出，经过输出门筛选和 $\tanh$ 压缩。把"存储"和"暴露"解耦，让记忆通道保持干净。这是 LSTM 最精妙的一点。

3.4 为什么真的能缓解梯度消失（完整版）

直连路径单独看。沿 $c_{t-1} \rightarrow c_t$ 那条边，局部导数是 $\text{diag}(f_t)$ ，于是 T 步后这条路径贡献 $\prod_t f_t$ ：

f	60 步后 $\prod f$
0.50	8.7×10^{-19}
0.90	1.8×10^{-3}
0.99	0.547
1.00	1.000

但这只是一条边。很多教材写 " $\partial c_t / \partial c_{t-1} = f_t$ " 就停了，这是不完整的。 c_{t-1} 还通过 $h_{t-1} = o_{t-1} \odot \tanh(c_{t-1})$ 影响第 t 步所有的门和候选。由 Day 1 §7.1 的多路径规则，完整的单步雅可比是

$$\frac{\partial c_t}{\partial c_{t-1}} = \underbrace{\text{diag}(f_t)}_{\text{直连}} + \underbrace{E_t}_{\text{经 } h_{t-1} \text{ 绕回各门}}$$

E_t 要穿过 W 、 σ' 、 $\tanh'$ ，行为和 RNN 的路径类似。

day4_checks.py 用有限差分算了完整的 $\partial c_T / \partial c_0$ ($T = 40$, 隐藏维 16)，并和普通 RNN 对比：

遗忘门 bias	平均 f	完整雅可比范数	直连 $\prod f$ 范数
0	0.50	1.2×10^{-7}	2.4×10^{-12}
1	0.72	5.8×10^{-3}	6.5×10^{-6}
3	0.94	1.8×10^1	4.4×10^{-1}
5	0.99	4.6×10^1	2.9×10^0

遗忘门 bias	平均 f	完整雅可比范数	直连 $\prod f$ 范数
普通 RNN	—	3.6×10^{-10}	—

三个结论，比 " f_t 连乘" 的说法更准确：

1. 间接项在范数上反而更大， $\prod f$ 只是完整梯度的一部分。
2. 真正的机制是结构： $f_t \rightarrow 1$ 时每步雅可比变成 $I + E_t$ ，与残差块 $I + f'(x)$ 同构。连乘 $\prod (I + E_t)$ 展开后含有一条"全是 I " 的路径，所以不会指数消失。RNN 的每步雅可比是 $\text{diag}(\tanh')W$ ，没有 I 。
3. 遗忘门 bias 的威力是实打实的：从 0 调到 3，40 步后的梯度差了 8 个多数量级。bias=0 时 LSTM 和 RNN 差不多一样糟。bias=5 时范数已经大于 1，LSTM 也可能爆炸，裁剪仍然需要。

一个实践技巧及其原因：遗忘门 bias 常初始化成 $+1 \sim +2$ (Jozefowicz et al. 2015 系统验证过)，让 $\sigma(b_f) \approx 0.73 \sim 0.88$ ，开局就接近"记住"。否则 $b_f = 0$ 时 $f \approx 0.5$ ，训练初期梯度就断了，模型永远学不到长依赖。

3.5 GRU：能不能更简单

Cho et al. (2014) 的思路是做减法：三个门都必要吗？

$$\begin{aligned} z_t &= \sigma(W_z[h_{t-1}, x_t]), & r_t &= \sigma(W_r[h_{t-1}, x_t]) \\ \tilde{h}_t &= \tanh(W[r_t \odot h_{t-1}, x_t]) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned} \quad \text{加法路径仍在}$$

两处简化：遗忘门和输入门合并成一个 z_t ("忘掉多少"就等于"写入多少"，两者互补而不独立)；取消 c/h 分离，只留一个状态。最后一行是旧状态与候选的凸组合， $z_t \rightarrow 0$ 时就是 $h_t = h_{t-1}$ ，同样的 I 结构。

收益：参数少约 25%，训练更快。代价：记忆和暴露耦合了，表达能力略弱。多数任务上两者打平，小数据上 GRU 常略好，大数据上 LSTM 常略好。

这是一个"简化没有明显损失"的范例。Day 2 §4.4 的 RMSNorm 砍掉 LayerNorm 的均值项，用的是完全相同的方法论：先问每个组件是否必要。

3.6 LSTM 的收益与遗留问题

收益：有效记忆从 10–20 步拉到数百步，一举成为 2014–2017 年 NLP 的绝对主力（机器翻译、语音识别、序列标注）。

遗留：

1. 仍然是串行的，无法利用 GPU 的并行能力。这一条最终要了 RNN 家族的命。
2. 长依赖只是缓解，信息从第 1 步到第 500 步仍要流过 500 次门控。
3. 四组门各一套矩阵乘，计算重。

§4 Seq2Seq：编码器-解码器

4.1 动机与直觉

机器翻译的困难在于输入输出长度不等、语序不同。统计机器翻译 (SMT) 时代要靠对齐模型、短语表、语言模型等一堆组件拼起来，工程极其复杂。

Sutskever et al. (2014) 的想法简单得惊人：

用一个 RNN (**编码器**) 把源句读成一个向量，再用另一个 RNN (**解码器**) 从这个向量把目标句写出来。

$$\text{源句} \xrightarrow{\text{Encoder}} c \xrightarrow{\text{Decoder}} \text{目标句}, \quad c = h_T^{\text{enc}}$$

c 叫 **context vector**，也叫“思想向量”。

意义：整个翻译系统变成一个**端到端可训练**的网络，不再需要任何人工设计的对齐或短语表。NLP 从“流水线工程”走向“端到端学习”的分水岭。

4.2 信息瓶颈：撞墙

这个设计有一个一眼就能看出的**结构性缺陷**：

无论源句是 5 个词还是 50 个词，都要被压进同一个固定大小的向量 c 。

信息论上就说不通。实验证据也很清楚：朴素 Seq2Seq 的翻译质量**随源句长度增长而显著下降**，长句上崩塌得厉害 (Bahdanau 等人 2014 年的论文里有 BLEU 随长度变化的曲线)。

原论文还有一个著名的 trick：**把源句反转再输入**。理由是这样源句开头和目标句开头在时间上更近。这个 **hack** 能提升效果，本身就是瓶颈存在的证据：如果那个向量真装得下，顺序不该有影响。

4.3 Teacher forcing 与 exposure bias

训练解码器时，第 t 步输入用**真实**的第 $t - 1$ 个词，还是**模型自己预测**的？

用真实的叫 **teacher forcing**。**收益**：训练稳定，而且能并行（整个目标句一次喂进去）。**代价**：训练时模型永远看到正确前缀，推理时却要基于自己的输出继续。一旦某步错了，后面就进入了训练时从未见过的分布，**错误滚雪球**。这叫 **exposure bias**。

这个概念一路活到今天：LLM 预训练就是 teacher forcing (Day 1 §5.2.5 的 $\hat{y} - y$ 就是在真实前缀上算的)，推理是自回归采样。Lecture 5 的 RLHF、Lecture 6 的 RL for reasoning，某种意义上都在修补这个训练/推理不一致，**因为 RL 是在模型自己生成的序列上算奖励的**。

§5 注意力的诞生

5.1 直觉：人是怎么翻译的

Bahdanau、Cho、Bengio (2014) 的出发点非常朴素：

人翻译长句时，不会先把整句背下来再默写，而是一边写译文，一边回头看原文的相关部分。

那为什么要强迫模型把整句压进一个向量？**让解码器在生成每个词时，都能回看编码器的全部隐状态**。

5.2 形式化：三步

编码器不再只交出最后一个 h_T ，而是**保留全部** $h_1, \dots, h_T$ 。解码器在第 t 步：

第一步，打分：当前解码状态 s_{t-1} 和每个源位置 h_j 有多相关

$$e_{tj} = \text{score}(s_{t-1}, h_j)$$

第二步，归一化：把分数变成权重

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_k \exp(e_{tk})}, \quad \sum_j \alpha_{tj} = 1$$

第三步，加权求和：得到这一步专属的上下文向量

$$c_t = \sum_{j=1}^T \alpha_{tj} h_j$$

c_t 带下标 t ：每一步都有自己的上下文向量，不再是全局唯一的那个 c 。瓶颈消失了。

5.3 为什么是这个形式：可微的软查表

直觉。这三步本质上是一次查表：拿 s_{t-1} 当查询，在 $\{h_j\}$ 里找相关的内容取回来。

硬查表（只取最相关的那一个 h_j ）要用 $\arg \max$ ，不可微，没法用梯度训练。softmax 把"选一个"松弛成"按相关度加权混合"，这是一次软化，目的就是让整个查表过程可微。这正是 Day 1 §2.4 说的"softmax 是 $\arg \max$ 的软化"。

验证。 `day4_checks.py` 把分数除以温度 τ 再 softmax，看输出和"最相关那个 key 对应的 value"差多远：

τ	最大权重	输出 - $v_{\arg \max}$
1.0	0.573	1.3×10^0
0.3	0.962	1.1×10^{-1}
0.1	1.000	8.1×10^{-5}
0.03	1.000	1.7×10^{-15}

$\tau \rightarrow 0$ 时注意力就是一次硬查表。正常温度下它是硬查表的一个光滑版本。

这个"用 softmax 把离散选择变成可微加权"的套路是深度学习最通用的工具之一。Day 3 §3.3 见过它的近亲，Lecture 3 的 MoE 路由器也用它。

白送一个好处： α_{tj} 是显式的权重矩阵，画出来就是**源词与译词的对齐图**。统计机器翻译要专门设计对齐模型才能得到的东西，在这里**作为副产品自动出现**。可解释，是注意力最早被接受的重要原因之一。

5.4 Q、K、V 从哪来：一次查表里的三种角色

动机。Lecture 1 里注意力写成 $\text{softmax}(QK^\top / \sqrt{d_k})V$ ，三个字母常让人觉得是凭空冒出来的。其实它们在 Bahdanau 的公式里早就在了，只是还没被分开。

把 §5.2 的三步和"查表"对照：

查表里的角色	在 Bahdanau 注意力里	作用
Query （我要找什么）	解码状态 s_{t-1}	拿去和每个条目比对
Key （每个条目的"标签"）	编码状态 h_j	决定这个条目有多相关

查表里的角色	在 Bahdanau 注意力里	作用
Value (每个条目的"内容")	还是 h_j	被选中后贡献出去的东西

注意 Bahdanau 里 key 和 value 是同一个向量 h_j 。这是它和 Transformer 最关键的一处区别。

为什么要把 key 和 value 拆开？因为"什么让我和你相关"与"被选中后我该给你什么"是两件不同的事。比如在"The animal didn't cross the street because it was too tired"里，"it"要找它指代谁：

- 判断"animal"是否相关，看的是句法和语义匹配（名词、有生命、单数），这是 key 该编码的
- 找到之后要取回的，是"animal"这个实体的完整信息，这是 value 该编码的

用同一个向量同时承担两种角色，就是在强迫一个表示兼顾两个不同的目标。Transformer 的做法是从同一个输入 x_j 出发，用三个不同的线性投影分别造出 $q_j = W_Q x_j$ 、 $k_j = W_K x_j$ 、 $v_j = W_V x_j$ ，让三种角色各自学习最合适的表示。这就是 Q/K/V 的全部来历。

5.5 打分函数的演化，以及为什么要除以 $\sqrt{d}$

名称	公式	说明
Bahdanau (加性)	$v^\top \tanh(W_1 s + W_2 h)$	有可学参数，表达力强，但慢
Luong (点积)	$s^\top h$	无参数，可以用矩阵乘批量算，快
Luong (general)	$s^\top W h$	折中
缩放点积	$\frac{q^\top k}{\sqrt{d}}$	Transformer 采用

演化的方向很清楚：从"有参数的小网络"走向"纯矩阵乘"，因为矩阵乘是 GPU 最擅长的操作。

为什么要除以 $\sqrt{d}$ ：一个完整的推导

动机。点积比加性打分快，但原始点积在维度大时训练很差。为什么？

推导第一步：点积的方差随维度增长。设 $q, k \in \mathbb{R}^d$ 各分量独立、均值 0、方差 1。点积 $q^\top k = \sum_{i=1}^d q_i k_i$ 是 d 个独立项之和，每项 $\mathbb{E}[q_i k_i] = 0$ ， $\text{Var}[q_i k_i] = \mathbb{E}[q_i^2] \mathbb{E}[k_i^2] = 1$ ，所以

$$\text{Var}[q^\top k] = d$$

这和 Day 2 §5.1 的方差传播是同一个计算。day4_checks.py 实测： $d = 16, 64, 256, 1024$ 时方差分别是 16.2、65.4、257.4、1019.6。

推导第二步：方差大 $\Rightarrow$ softmax 饱和 $\Rightarrow$ 梯度消失。分数的标准差是 $\sqrt{d}$ ， $d = 1024$ 时分数之间动辄相差几十，softmax 几乎变成 one-hot。而 Day 1 §5.2.1 推过 softmax 的雅可比是

$$J = \text{diag}(p) - pp^\top$$

当 p 接近 one-hot（比如 $p = e_1$ ）， $\text{diag}(e_1) - e_1 e_1^\top = 0$ ，雅可比趋于零矩阵，梯度消失。这正是 Day 1 §5.2.4 预告的那件事：注意力里的 softmax 没有交叉熵来跟它消去，这个雅可比是实打实要穿过的。

推导第三步：除以 $\sqrt{d}$ 把方差拉回 1。 $\text{Var}[q^\top k / \sqrt{d}] = d/d = 1$ ，与维度无关。

day4_checks.py 实测（32 个 key，300 次平均）：

d	最大注意力权重（不缩放）	雅可比范数（不缩放）	雅可比范数 ($\div\sqrt{d}$)
16	0.623	0.284	0.236
64	0.791	0.218	0.240
256	0.914	0.106	0.243
1024	0.948	0.067	0.241

不缩放时， d 从 16 涨到 1024，雅可比范数掉到约 1/4；缩放后纹丝不动。

意义。那个 $\sqrt{d_k}$ 不是随手加的常数，它是 Day 1（softmax 雅可比）和 Day 2（方差传播）两个结论合起来的直接推论。Lecture 1 会问这个问题，现在你能完整推出来了。

5.6 从 Bahdanau 注意力到 Transformer：三步跳跃

2014 年的注意力是 RNN 的**附加组件**，2017 年的 Transformer 把它变成了**主体**。中间是三次跳跃。

跳跃一：自注意力。Bahdanau 是**解码器看编码器**（跨序列）。如果让**序列看自己**呢？每个词根据其他所有词来更新自己的表示，这正是 **Day 3 §6 要的"上下文相关表示"**。此时 Q、K、V 都来自同一个序列，§5.4 的三个投影就成了必需品：不做投影的话，由柯西-施瓦茨，一个词和自己的点积 $\|x\|^2$ 在范数相近时往往最大，模型会倾向于只注意自己。（day4_checks.py 第 9 组：20 个随机词向量直接打分，100% 的词把最大权重给了自己；加上两个不同的投影 W_Q, W_K 后降到约 4%，接近随机。）

跳跃二：扔掉 RNN。"Attention Is All You Need"这个标题的字面意思。既然注意力能让任意两个位置**一步直达**（路径长度 $O(1)$ ，RNN 是 $O(T)$ ），RNN 还有什么用？

扔掉它的真正动机是并行：注意力对所有位置的**计算互相独立**，一次矩阵乘全部算完；RNN 必须串行 T 步。在 GPU 时代，这个差别决定了能不能把模型和数据规模放大几个数量级。§3.6 的遗留问题 1 在这里被彻底解决。代价是必须另外引入**位置编码**：扔掉循环的同时也扔掉了顺序信息。

跳跃三：多头。一次注意力只能表达一种"相关性"。并行跑多组，让不同的头关注不同类型的关系（句法、语义、指代）。

这三步，加上位置编码、残差、LayerNorm（Day 2 §4），就是 **Lecture 1** 和 **Lecture 2** 的全部内容。你现在已经知道每一件的动机了。

5.7 代价：没有免费的午餐

注意力解决了瓶颈和并行，但引入了新问题，**这些新问题构成了 Lecture 2 的主要内容：**

新问题	后果	对应的技术（Lecture 2）
复杂度 $O(T^2)$	长上下文极贵	稀疏/线性注意力、FlashAttention
推理要缓存 K/V	显存随序列线性增长	MQA / GQA
丢失顺序信息	必须外挂位置信息	正弦编码、RoPE

RNN 是 $O(T)$ 时间、 $O(1)$ 状态但串行；注意力是 $O(T^2)$ 但并行。 用"计算复杂度"换"可并行性"，在 GPU 上这笔买卖极其划算，正是它催生了今天的大模型。

§A 序列模型与注意力：API 用法与边界

全是不看文档就会踩的坑。第 1 周手写 Transformer 时会原样再来一遍。

A.1 nn.RNN / nn.LSTM / nn.GRU

```
rnn = nn.LSTM(input_size=128, hidden_size=256, num_layers=2,
              batch_first=True, dropout=0.1, bidirectional=False)
output, (h_n, c_n) = rnn(x, (h_0, c_0))  # h_0/c_0 可省，默认全零
```

坑 1: batch_first 默认是 False。 默认输入形状是 (seq_len, batch, input_size)。几乎所有人都应该显式写 batch_first=True。

坑 2: batch_first 不影响 h_n。 最阴险的一个：

张量	batch_first=True 时的形状
output	(batch, seq_len, num_directions*hidden) ← 受影响
h_n / c_n	(num_layers*num_directions, batch, hidden) ← 不受影响，层数永远在第 0 维

取最后一层最后一步的隐状态要写 `h_n[-1]`，不是 `h_n[:, -1]`。

坑 3: dropout 只作用在层与层之间，不作用在时间步之间，num_layers=1 时完全无效（PyTorch 会警告）。

坑 4: 双向输出的布局。 `bidirectional=True` 时 `output` 最后一维是 `2*hidden`，前半正向、后半反向。双向 RNN 不能用于自回归语言模型，反向那一半看到了未来。这也是 BERT（双向）不能直接生成的原因（Lecture 2）。

坑 5: h_n 里双向的排列是 (layer0_fwd, layer0_bwd, layer1_fwd, ...)，取最后一层要 `h_n[-2:]` 再拼接。

遗忘门 bias 的初始化 (§3.4)： PyTorch 的 `nn.LSTM` 把四个门的 bias 拼在一个向量里，顺序是 (i, f, g, o)。要把遗忘门 bias 设成 1:

```
for name, p in lstm.named_parameters():
    if name.startswith("bias_ih"):
        n = p.size(0) // 4
        with torch.no_grad():
            p[n:2*n].fill_(1.0)
```

只改一份，原因见下
原地改参数用 no_grad，不用 .data (Day 1 §A.7)

PyTorch 有 `bias_ih` 和 `bias_hh` 两份 bias，它们是相加的。如果写成 `if "bias" in name`，两份都会被填 1，实际效果是 2。

A.2 变长序列：pack_padded_sequence

直接把 padding 喂进 RNN，padding 位置也会参与状态更新，污染最终的 `h_n`。正确做法：

```
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
packed = pack_padded_sequence(x, lengths, batch_first=True,
                             enforce_sorted=False)
out_packed, (h_n, c_n) = rnn(packed)
out, lens = pad_packed_sequence(out_packed, batch_first=True,
                                total_length=x.size(1))
```

- `lengths` 必须在 CPU 上（即使 `x` 在 GPU 上）。
- `enforce_sorted=True`（默认）要求按长度降序排好。除非你自己排了，一律写 `False`。
- 长度不能为 0，否则报错。空样本要提前过滤。
- `pad_packed_sequence` 默认按 batch 内最大长度还原，可能比原始 padding 短。后续要对齐形状时务必传 `total_length`。
- 用了 pack，`h_n` 就是每个样本各自最后一个有效位置的状态，正是我们要的。

A.3 截断 BPTT：detach()

长序列不可能一次反传到头，显存扛不住（Day 1 §1.5）。分块，块间传递隐状态但切断梯度：

```
hidden = None
for chunk, target in chunks:
    out, hidden = rnn(chunk, hidden)
    loss = criterion(out, target)
    optimizer.zero_grad(); loss.backward(); optimizer.step()
    hidden = tuple(h.detach() for h in hidden)  # ← LSTM 是 (h, c) 元组
```

不 `detach` 会怎样？计算图一路累积，显存线性增长直到 OOM；而且第二次 `backward()` 报 *"Trying to backward through the graph a second time"*（Day 1 §A.5）。LSTM 的 `hidden` 是 `(h, c)` 元组，GRU/RNN 是单个张量，写通用代码时要区分。

A.4 注意力：三个高频 bug

Bug 1: softmax 的 dim 选错。

```
attn = F.softmax(scores, dim=-1)  # scores: (batch, T_q, T_k)
```

必须对 **key 那一维**（最后一维）归一化：语义是“每个 query 在所有 key 上分配总量 1 的注意力”。

`day4_checks.py` 显示选 `dim=0` 时每行的和变成 1.808 / 0.801 / 1.391，完全不是概率分布。这个错误不报

错、不 NaN，只会悄悄学不好。写完 softmax 就加一句 `assert torch.allclose(attn.sum(-1), torch.ones_like(attn.sum(-1)))`。

Bug 2: 掩码用 `-inf` 导致 NaN。

```
scores = scores.masked_fill(mask, float("-inf")) # 危险
```

某一行被整行掩掉（比如全是 padding 的样本）时，softmax 算出 $0/0 = \text{NaN}$ ，然后 NaN 沿反向污染整个模型。`day4_checks.py` 复现了这一点。稳妥写法：

```
NEG = -1e9 if scores.dtype == torch.float32 else -1e4 # fp16 最大约 65504
scores = scores.masked_fill(mask, NEG)
```

fp16 下绝不能用 `-1e9`，它超出范围会变成 `-inf`（Day 1 §10.2）。`masked_fill` 的 `mask` 必须是 **bool**，且 **True** 的位置被填充，很多人记反。

本质规则是“每一行至少保留一个未被掩的位置”。Day 5 §5.3 会看到一个用 `-inf` 反而安全的例子，正是因为那里能保证这一点。

Bug 3: `bmm` 与 `matmul`。

```
torch.bmm(A, B) # 严格要求两个都是 3 维: (b,n,m) @ (b,m,p)
torch.matmul(A, B) # 会广播前导维度: (b,h,n,m) @ (b,h,m,p) 也行
```

多头注意力的形状是 `(batch, heads, seq, d_head)`，4 维，`bmm` 用不了，要用 `matmul/@`，或者先 `reshape` 成 3 维。

A.5 `nn.Embedding` 的边界

```
emb = nn.Embedding(num_embeddings=50000, embedding_dim=512, padding_idx=0)
```

- `padding_idx` 那一行初始化为全零，且梯度恒为零。不设的话，padding token 也会被当成正常词去学。
- 输入必须是 **long**。
- 索引越界在 CPU 上报清晰的错误，在 CUDA 上是 **device-side assert**，错误信息极难读。调试先切回 CPU 跑一遍。

A.6 `clip_grad_norm_`

```
total_norm = nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

- 返回的是裁剪前的总范数，值得 log：梯度范数突然飙升是训练要炸的先兆。§3.4 说过 LSTM 也可能爆炸，RNN 家族尤其要盯着它。
- 必须在 `backward()` 之后、`step()` 之前调用。混合精度下还要先 `unscale_` (Day 2 §A.5)。

§B 实操

1. 跑通 `day4_checks.py`，九组结论都看懂。特别是第 1 组 $\rho > 1$ 时的巨大波动、第 4 组的"完整雅可比比直连路径大"，想清楚为什么。
2. 变长序列与长程记忆 (2h)：用 `nn.LSTM` 处理变长序列，正确使用 `pack_padded_sequence`；再故意不用 `pack` 跑一次，看 padding 污染的影响；打印 `output/h_n/c_n` 的形状，亲手确认 §A.1 坑 2。再把遗忘门 bias 分别设成 0 和 1 各训一次，对比长序列上的表现。参考：`code/d4_adding_problem.py`。它用 Hochreiter & Schmidhuber 的加法问题代替情感分类：合成数据不用下载，而且"必须把第一个标记的数带到序列末尾"，比情感分类更直接地考验长程记忆。——`compare-pack` 对比 `pack` 与否，默认模式对比 RNN / LSTM (遗忘门 bias 0 vs 1) / GRU。想做情感分类的话，把同样的 `pack` 逻辑搬到 IMDB 上即可。
3. 手写 Bahdanau 注意力 (2h, 最重要)，不许用 `nn.MultiheadAttention`：
 - 实现打分 → softmax → 加权和三步
 - 加上正确的掩码 (用 `-1e9`)
 - `assert` 注意力权重每行和为 1
 - 用 `matplotlib` 把 α_{tj} 画成热力图，看对齐是否符合直觉 (§5.3 那个白送的好处)
 - 参考：`code/d4_seq2seq_attention.py`。任务是序列反转 / 复制，正确对齐已知 (反对角线 / 对角线)，所以除了看图，脚本还直接算 argmax 对齐命中率。默认同时训一个无注意力基线，按长度分桶对比，看 §4.2 的瓶颈；`--models noattn --task copy` 与 `--task reverse` 的差距，就是 Sutskever 反转源句那个 trick 的来源
4. 换成缩放点积 (1h)：把加性打分换成 $q^T k / \sqrt{d}$ ，再把 key 和 value 用两个不同的线性层分别投影出来 (§5.4)。做到这一步，你已经在写 Transformer 的核心了。参考：`d4_seq2seq_attention.py --models additive dot scaled --separate-kv --d-attn 256`。

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. n-gram 和固定窗口 MLP 的致命伤各是什么？RNN 的"状态机"构思如何一次满足三个要求？
2. LSTM 的核心直觉是什么？为什么"把覆写改成累加"能救梯度？它和残差连接是什么关系？
3. 为什么 LSTM 要把 c 和 h 分开？
4. Seq2Seq 的信息瓶颈是什么？"源句反转"这个 trick 为什么能证明瓶颈存在？
5. 注意力的直觉来源是什么？为什么说它是"可微的软查表"？温度趋于 0 时它变成什么？
6. Bahdanau 注意力里 key 和 value 是什么？Transformer 为什么要把它们拆开？
7. 从 Bahdanau 注意力到 Transformer 的三步跳跃分别是什么？扔掉 RNN 的真正动机是什么？代价是什么？

推导层

8. 写出 $\partial h_t / \partial h_s$ ，说明为什么 RNN 比普通深网络的连乘问题更严重。

9. $\rho(W_{hh}) > 1$ 时 $\tanh$ 的饱和对爆炸有什么作用？为什么说消失比爆炸更根本，是"逼着人改结构"的那个？
10. " $\partial c_t / \partial c_{t-1} = f_t$ "这个说法哪里不完整？完整的单步雅可比是什么？LSTM 真正缓解消失的机制是什么？
11. 遗忘门 bias 为什么初始化成正数？不这样会怎样？
12. 推出 $\text{Var}[q^\top k] = d$ ，并用 softmax 雅可比说明不除以 $\sqrt{d}$ 会发生什么。
13. GRU 相对 LSTM 做了哪两处简化？它和 RMSNorm 简化 LayerNorm 是不是同一种方法论？

工程与 API 层

14. `nn.LSTM(batch_first=True)` 时 `output` 和 `h_n` 的形状各是什么？
15. 不用 `pack_padded_sequence` 会怎样？`enforce_sorted` 和 `total_length` 各解决什么？
16. 用 PyTorch 把 LSTM 的遗忘门 bias 设成 1，要改哪几个元素？为什么两份 bias 都改会变成 2？
17. 注意力的 softmax 该对哪一维做？做错了会报错吗？
18. 掩码为什么不能用 `-inf`？fp16 下该填什么？本质规则是什么？
19. 多头注意力为什么不能用 `bmm`？
20. 截断 BPTT 不 `detach` 会报什么错？为什么？

出关标准

- ☐ 能画出"定长窗口 → RNN → LSTM → 注意力 → Transformer"的撞墙链条，每一步说清墙和解法
- ☐ 能推出 BPTT 的连乘项，用多初始化的统计说清消失与爆炸，并解释为什么消失更根本
- ☐ 能说出 LSTM 的核心是加法路径，并完整说明单步雅可比 $\text{diag}(f_t) + E_t$ 与 $I + E_t$ 的残差结构
- ☐ 能用实测数据说明遗忘门 bias 的作用
- ☐ 能定量说明 Seq2Seq 的信息瓶颈
- ☐ 能把注意力讲成"可微的软查表"，并说明低温极限
- ☐ 能讲清 Q/K/V 在查表里各自的角色，以及 key 和 value 为什么要拆开
- ☐ 能完整推出为什么要除以 $\sqrt{d}$
- ☐ 能讲出扔掉 RNN 的真正动机（并行）和代价 ($O(T^2)$ + 位置编码)
- ☐ SA 的 API 坑都能说出现象和正确写法
- ☐ 手写出带掩码的 Bahdanau 注意力并画出对齐热力图

疑问清单

-
-
-