

Transformer架构与训练笔记

第三版 · 2026-10-17（北京时间）

这一版从原分享问答重组：保留原来讲得清楚的解释、例子、公式和推导，先补前置问题，再把相关追问接在一起。新增内容只为补足跳步和说明条件。

从第1章顺着读即可。第4章保留完整方差推导，第11章保留原来的单参数训练例子；第13章提供额外的完整数值核验，第15章供矩阵细节回查。它们不代替主线解释。

文中教学分词、注意力比例和参数数值均为示意，不是真实模型测量，也不是临床判断依据。

正文目录

第 1 章：先弄清输入、预测、训练和输出	3
第 2 章：Q、K、V 从哪里来，WQ 又是什么	6
第 3 章：点积怎样一步步变成注意力比例	10
第 4 章：dk 是什么，为什么要除以它的平方根	16
第 5 章：为什么还要乘 V，得到的 z 有什么用	28
第 6 章：把逐个向量的计算写成矩阵，再变成多头	38
第 7 章：注意力以外还有什么，为什么要一层层处理	43
第 8 章：没有掩码会怎样，怎样防止偷看未来	50
第 9 章：向量怎样变成下一 Token 的预测概率	55
第 10 章：用同一句话看训练和生成两种情况	62
第 11 章：损失、导数与反向传播，沿原问答逐步推导	71
第 12 章：输入、内部和输出层怎样一起训练	82
第 13 章：补算一个小模型，看到梯度真正回到 WQ	89
第 14 章：每生成一个 Token 后，接下来发生什么	96
第 15 章：需要时回查的矩阵公式与补充推导	98
第 16 章：按原来的问题回查，以及来源与检查说明	105

第 1 章：先弄清输入、预测、训练和输出

[返回目录](#)

你原来的追问里，有两句应该先放到最前面：“我们输入的是一句话吧？”“这一步到底是在训练还是在输出？”

先明确这两件事，后面看到Q、K、V时，才知道它们处在整个流程的哪里。这里先看全貌，不要求现在理解注意力内部公式。

1.1 先固定同一个例子

我们沿用原问答中的句子：

患者服用华法林后出现黑便。

本文主要用同一套教学分词：

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (\text{患者}, \text{服用}, \text{华法林}, \text{后}, \text{出现}, \text{黑便}, \text{。})$$

t_i 表示第*i*个位置的Token；下标是位置，不是向量坐标。真实Tokenizer不一定这样切，这里只为清楚地对应每个位置。

如果输入到“出现”为止，要预测的是“黑便”这个候选；如果输入已经包含“黑便。”，那么要预测的就是句号之后的内容。不要把“读取已经给出的黑便”和“在它还没给出时预测黑便”当成同一件事。

1.2 一句话如何成为整个输入矩阵

首先，每个Token有一个编号，编号用来查嵌入表，取出一行数值向量。例如“患者”的向量可以写成 x_1 ，“出现”的向量写成 x_5 。

输入前五个Token时，模型接收的是：

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

本文统一把 x_i 视为行向量；每行有d个数，X就是5行d列。这叫 $X \in \mathbb{R}^{5 \times d}$ ，只表示这张实数表的行列数。

模型处理的是整个X。 解释时只盯着某一行，是为了看清这个位置怎样计算，不是说模型只收到一个孤立词语。位置的信息也要进入计算，第7章说明具体做法。

1.3 这一步究竟属于训练，还是使用模型

训练和生成都会先根据输入算一次预测，这一段叫前向传播。

训练时，数据里已经有完整句子。模型根据到“出现”为止的前文，算“黑便”等候选的概率，再拿真实目标“黑便”评分。评分结果用来计算各参数怎样调整，最后执行更新。

生成时，后续还没提供。模型同样先计算概率，再选一个Token追加到前文，接着计算下一项。通常不会因为这次生成而改变参数。

所以我们之后看到的注意力运算，不是“只在训练时算”或“只在输出时算”，而是两者共用的前向计算。完整同句示例在第10章；这里先不引入内部字母。

1.4 把训练与推理放在一起对照

做的事情	训练	生成
把整段前文变成向量	要做	要做
用注意力与后续网络计算预测	要做	要做
已知数据给出的真实下一Token	是	未提供后续目标
按真实目标计算损失、梯度	要做	普通生成通常不做
更新可训练参数	要做	普通生成通常不做
把选出的Token接回去继续生成	标准训练使用真实前文	要做

“输出”还可能指模块输出的一串数，而不是最终文字。后面会分别叫它们中间向量、预测概率和生成文字。

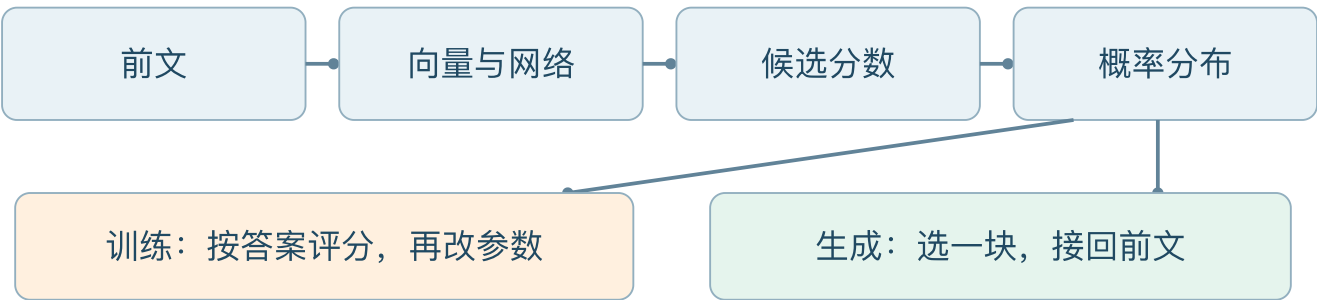


图1：前向预测相同，概率之后的用途不同。

1.5 先只记住这一条流程

共同的前向计算是：

输入文字 → Token向量 → 多层Transformer → 隐藏表示 → 输出打分 → 下一Token概率。

训练再接“读取真实目标 → 算损失 → 求梯度 → 更新参数”；生成再接“选一个Token → 放回前文 → 继续预测”。

后面说“输出”，可能指某个模块输出的向量，也可能指最终生成的文字。本文会区分：z是注意力的中间结果，h是整个网络处理后的表示，候选概率和文字还在它们之后。

第 2 章：Q、K、V 从哪里来，WQ 又是什么

[返回目录](#)

你问“这一步是在算WQ还是算q”，说明必须先分清规则与本次结果。下面保留原回答中三个阶段的拆分，但先补上Q/K/V为什么分成三个角色。

2.1 同一个输入，经过三个不同的变换

Transformer 不会直接使用原始 x_i 计算所有关系，而是先进行三个线性变换：

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \end{aligned}$$

其中 W_Q, W_K, W_V 都是训练得到的参数矩阵。

你可以把它理解为：同一个向量经过三种不同的线性变换，得到三种不同用途的表示。

向量	英文	可以怎么理解
Q	Query（查询）	我需要寻找什么信息？
K	Key（键）	我具有什么信息可供匹配？
V	Value（值）	我实际可以传递什么信息？

举个医学语言的例子。

当模型处理「黑便」时，可能需要关注「华法林」这个词，因为两者之间存在与出血相关的联系。

那么：

- 「黑便」的 Query 可能与「华法林」的 Key 高度匹配。
- 这种匹配会产生较大的注意力权重。
- 「华法林」的 Value 就会更多地参与「黑便」的新向量的构造。

这是机制上的示意，并不意味着模型内部真的有一个人为设定的「出血维度」。

这里用“黑便”读取“华法林”，说的是黑便已经作为输入出现之后的表示更新；若要在它尚未出现时预测它，应使用前一个“出现”位置的最终表示。这两个时刻不要混淆。

2.2 第一阶段：计算本次的 q 、 k 、 v

假设某个 Token 的原始向量是 x_i 。

首先通过矩阵变换：

$$\begin{aligned} q_i &= x_i W_Q \\ k_i &= x_i W_K \\ v_i &= x_i W_V \end{aligned}$$

这里有两类不同的东西：

- W_Q, W_K, W_V ：模型通过训练学到的参数矩阵，在一次普通推理过程中保持不变。
- q_i, k_i, v_i ：根据当前输入计算出来的向量，会随着输入的 Token 及其表示改变。

因此， W_Q 是用于计算 q_i 的矩阵，二者不是同一个东西。

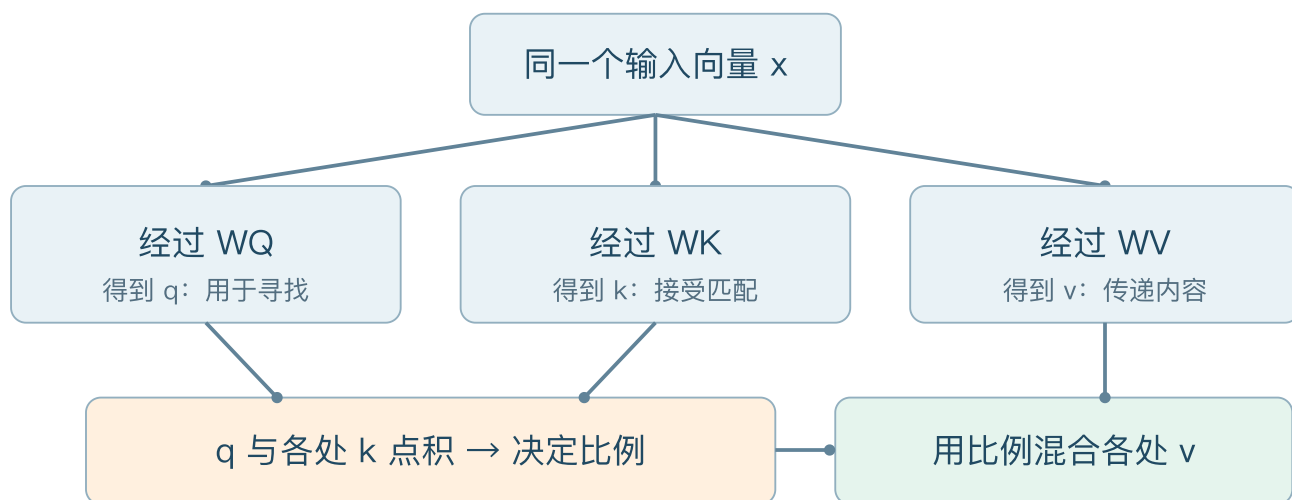


图2：参数矩阵是规则，q/k/v是本次输入经过规则算出的结果。

2.3 一个小计算，看清 WQ 与 q 的区别

假设某行输入为：

$$\mathbf{x}_i = (2, 1), \quad \mathbf{W}_Q = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}.$$

则 $\mathbf{q}_i = \mathbf{x}_i \mathbf{W}_Q = (2, 3)$ 。如果输入变为 $(1, 4)$ ，WQ不变，q就变成 $(1, 12)$ 。

所以WQ是一套可训练的数值规则；q是这次输入按规则算出的结果。训练更新的是WQ；下一次前向传播再用更新后的WQ算新的q。不是先单独“训练出一份固定Q”给所有输入使用。

2.4 第二阶段：使用 q、k 计算分数

这才是你选中的「Q 与 K 如何确定权重」那一步。

我们已经有了：

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}_Q, \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}_K$$

现在计算：

$$s_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

它表示第 i 个 Token 的 Query 与第 j 个 Token 的 Key 的匹配分数。

这里没有计算新的 W_Q ，也没有计算新的 q_i ，只是使用第一阶段得到的向量进行点积。

2.5 第三阶段：把分数变成比例

将分数 s_{ij} 转换为：

$$a_{ij} = \frac{e^{s_{ij}}}{\sum_l e^{s_{il}}}$$

得到真正的注意力权重 a_{ij} ，然后计算：

$$z_i = \sum_j a_{ij} v_j$$

2.6 遇到下标时，按谁在读、从哪里读来理解

s_{ij} 中， i 是当前需要读取信息的位置， j 是候选提供信息的位置。 q_i 属于 i ， k_j 属于 j ， a_{ij} 表示 i 分给 j 的比例。

例如 $a_{5,3}$ 是“出现”位置从“华法林”位置读回信息时所用的系数。它不是 WQ 里的第 5 行第 3 列，也不是“黑便”的输出概率。

下面先只看一个位置面对几个候选的情况，暂时不让整张矩阵掩盖每一步。

第 3 章：点积怎样一步步变成注意力比例

[返回目录](#)

这里对应你说的“还是太快，不太好理解”。原回答后来把点积、取指数、归一化和加权分开讲，这个展开顺序保留下来。缩放因子的推导紧接在下一章。

3.1 先理解：为什么需要计算点积？

假设一句话里有三个 Token，我们希望知道当前 Token 应该更加关注哪一个。

首先，模型已经通过线性变换得到：

- 当前 Token 的查询向量 q (Query)。
- 三个候选 Token 的键向量 k_1, k_2, k_3 (Key)。

为了方便，我们假设所有向量都是二维的：

$$q = (1, 0)$$

$$k_1 = (1, 0)$$

$$k_2 = (0, 1)$$

$$k_3 = (-1, 0)$$

你已经学过点积公式：

$$q \cdot k = \|q\| \|k\| \cos \theta$$

这里所有向量的模长都是 1，于是点积仅由夹角决定。

逐个计算：

$$q \cdot k_1 = 1 \times 1 + 0 \times 0 = 1$$

$$q \cdot k_2 = 1 \times 0 + 0 \times 1 = 0$$

$$q \cdot k_3 = 1 \times (-1) + 0 \times 0 = -1$$

于是得到三个分数：

$$\mathbf{s} = (1, 0, -1)$$

它们可以表示三种匹配程度：

Key	点积分数	在这个例子中的直观含义
k_1	1	匹配程度较高
k_2	0	匹配程度居中
k_3	-1	匹配程度较低

到这里，我们还没有涉及概率，也没有进行信息组合。

点积只是给出了一个分数，用于比较不同 Key 与当前 Query 的匹配程度。

需要注意，实际模型的 Q、K 不一定是单位向量，点积不仅由夹角决定。所谓「高分意味着更值得关注」，是模型通过训练学出的规则，而不是任何两个语义相近的向量天然都有高点积。

3.2 为什么不能直接拿 $(1, 0, -1)$ 作为权重？

现在假设三个 Token 各自携带信息：

$$\mathbf{v}_1, \quad \mathbf{v}_2, \quad \mathbf{v}_3$$

我们想构造：

$$\mathbf{z} = a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + a_3 \mathbf{v}_3$$

你最近学过凸组合，知道如果把三个系数解释为信息分配比例，应当满足：

$$a_1, a_2, a_3 \geq 0$$

$$a_1 + a_2 + a_3 = 1$$

这样我们才能说：

「从第一个 Token 获取 60% 的权重，从第二个获取 30%，从第三个获取 10%。」

但刚才点积得到的：

$$(1, 0, -1)$$

有两个问题：存在负数，而且总和是 0。

不能直接将它理解成三个非负的信息分配比例。

因此需要一个额外的数学函数：

把任意实数分数转换成总和为 1 的非负权重，同时尽可能保留原来的高低顺序。

这就是 Softmax 的作用。

3.3 Softmax 分成两个步骤，其实很简单

我们仍然使用：

$$s = (1, 0, -1)$$

第一步：每个分数取指数

也就是计算 e^s 。

你可以先把 $e \approx 2.718$ 当作一个固定常数。

$$\begin{aligned}e^1 &\approx 2.718 \\e^0 &= 1 \\e^{-1} &\approx 0.368\end{aligned}$$

得到：

$$(2.718, 1, 0.368)$$

为什么这样做？

因为指数函数有几个非常适合这里的性质：

- 对任何实数，指数结果始终大于 0。
- 原来分数越大，指数结果也越大。
- 较高的分数经过指数变换后，会获得相对更大的权重。

因此，它既消除了负数，又保留了匹配分数的排序。

第二步：让三个数字的和等于 1

目前三个数字之和为：

$$2.718 + 1 + 0.368 = 4.086$$

那就把每个数字都除以 4.086：

$$\begin{aligned}a_1 &= \frac{2.718}{4.086} \approx 0.665 \\a_2 &= \frac{1}{4.086} \approx 0.245 \\a_3 &= \frac{0.368}{4.086} \approx 0.090\end{aligned}$$

现在：

$$0.665 + 0.245 + 0.090 = 1$$

我们成功得到了三个凸组合系数。

如果把三个分数全部设为0。

这时 Softmax 会得到：

$$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

因为三个 Token 的匹配分数完全相同，所以它们获得同样的权重。

这说明一件很重要的事情：

Softmax 不是独立判断某个 Token 的分数够不够高，而是在多个候选 Token 之间进行相对分配。

3.4 现在才把它和凸组合联系起来

经过上面的计算，我们得到了：

$$a = (0.665, 0.245, 0.090)$$

假设三个 Value 向量为：

$$v_1 = (2, 0), \quad v_2 = (0, 2), \quad v_3 = (0, 0)$$

我们进行加权组合：

$$z = 0.665v_1 + 0.245v_2 + 0.090v_3$$

代入：

$$\begin{aligned} z &= 0.665(2, 0) + 0.245(0, 2) + 0.090(0, 0) \\ &= \boxed{(1.33, 0.49)} \end{aligned}$$

这时候就可以用你熟悉的凸包知识理解了。

因为三个权重全部非负，而且总和为 1，所以得到的新点 z ，一定在 v_1, v_2, v_3 的凸包内部。

这就是注意力机制中的信息整合过程。

注意：Q、K 负责计算权重，但最终被加权的是 V，而不是 Q 或 K。

3.5 为什么加同一个数不改变结果

如果把所有分数都加c，则分子分母都多了同一个因子 e^c ，它会约去：

$$\frac{e^{s_j+c}}{\sum_r e^{s_r+c}} = \frac{e^c e^{s_j}}{e^c \sum_r e^{s_r}} = \frac{e^{s_j}}{\sum_r e^{s_r}}.$$

这说明Softmax使用候选之间的相对分数，而不是拿一个固定分数线判断是否重要。数值实现常先减去最大分数，利用同一性质避免指数太大。

注意，上面把比例说成“取多少信息”，是解释计算分工，不是已测量的语义贡献百分比。V的方向、大小和后续网络，也会影响最终结果。

原来两个候选的比例还满足：

$$\frac{a_j}{a_k} = \frac{e^{s_j}}{e^{s_k}} = e^{s_j-s_k}.$$

因此它们的分数相差1，权重之比就是e；所有分数一起平移，则这个比不变。

第 4 章：dk 是什么，为什么要除以它的平方根

[返回目录](#)

这一章把你连续追问的dk、方差定义、方差相加和缩放放在一起。先说明符号，再用原来的正负1例子；证明用到均值为0时，立即解释依据，不再让你跳到后面寻找。

4.1 dk先表示什么

d_k 指的是 Key 向量（键向量）的维度，也等于 Query 向量的维度。

回顾 Transformer 中的注意力公式：

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

假设输入的每个 Token 是一个 512 维向量：

$$x_i \in \mathbb{R}^{512}$$

经过线性变换：

$$q_i = x_i W_Q, \quad k_i = x_i W_K$$

我们可以把 Query 和 Key 都映射到 64 维：

$$q_i, k_i \in \mathbb{R}^{64}$$

此时：

$$\boxed{d_k = 64}$$

它们的点积为：

$$q_i \cdot k_j = \sum_{r=1}^{64} q_{ir} k_{jr}$$

也就是 64 个乘积项相加。

因此需要除以：

$$\sqrt{d_k} = \sqrt{64} = 8$$

注意区分三个不同的维度

符号	含义	假设值
d_{model}	Token 在模型中的表示维度	512
d_k	每个注意力头的 Query、Key 维度	64
d_v	每个注意力头的 Value 维度	64

如果有 8 个注意力头，经典配置中通常是：

$$d_k = d_v = \frac{d_{\text{model}}}{8} = 64$$

但这只是常见的设计选择，并不是数学上必须如此。

最重要的一点： d_k 不是 Token 的数量，而是用于计算一次 Query-Key 点积的向量维度。

例如一句话有 20 个 Token，每个注意力头的 Key 是 64 维，那么：

$$n = 20, \quad d_k = 64$$

最终 QK^T 的大小是 20×20 ，而缩放因子仍然是 $\sqrt{64} = 8$ 。

4.2 先看缩放的理由，再把方差细节补全

前面为了说明 Softmax，故意省略了这一步。

实际 Transformer 并不是直接计算：

$$s_j = q \cdot k_j$$

而是：

$$s_j = \frac{q \cdot k_j}{\sqrt{d_k}}$$

原因与高维向量的点积有关。

你知道二维点积是：

$$q \cdot k = q_1 k_1 + q_2 k_2$$

如果是 64 维：

$$q \cdot k = \sum_{r=1}^{64} q_r k_r$$

现在是 64 项相加。

假设这些乘积项大致独立，每项均值为 0、方差为 1，那么 64 项之和的方差就是 64，标准差是：

$$\sqrt{64} = 8$$

而如果只有 4 维，其标准差则是：

$$\sqrt{4} = 2$$

所以在这样的统计假设下，维度越高，点积结果的典型波动幅度越大，规模大约正比于 $\sqrt{d_k}$ 。

如果不缩放，Softmax 很容易得到特别极端的结果，比如：

$$(0.999, 0.001, \text{接近 } 0)$$

过于极端的 Softmax 可能导致梯度很小，不利于训练。

因此，把点积除以 $\sqrt{d_k}$ ，就能使其数值尺度相对稳定。

这是一个基于典型统计假设的解释，并不是说所有高维向量的点积都必然很大。

4.3 先不考虑 Transformer，假设有 4 个随机数

假设每个随机数 X_i 都有以下分布：

$$X_i = \begin{cases} +1 & \text{概率为 } 1/2 \\ -1 & \text{概率为 } 1/2 \end{cases}$$

并且它们相互独立。

每一个随机数的期望为：

$$E(X_i) = 0.5(1) + 0.5(-1) = 0$$

方差为：

$$\begin{aligned} \text{Var}(X_i) &= E[(X_i - E(X_i))^2] \\ &= 0.5(1)^2 + 0.5(-1)^2 \\ &= 1 \end{aligned}$$

因此，每个随机数都是：

$$\text{均值} = 0, \quad \text{方差} = 1, \quad \text{标准差} = 1$$

现在把 4 个随机数相加：

$$S = X_1 + X_2 + X_3 + X_4$$

你可能直觉上认为：四个随机数的标准差都是 1，加起来之后标准差应该是 4。

但我们实际上可以把所有可能结果列出来。

总和 S	16 种组合中出现次数	概率
-4	1	1/16
-2	4	4/16
0	6	6/16
2	4	4/16
4	1	1/16

你会发现，虽然理论上可以取到 4 或 -4，但最常见的结果反而是 0。

原因很简单：随机的正负数会相互抵消。

我们计算这 4 个数之和的方差。

因为 $E(S) = 0$ ，所以：

$$\begin{aligned}\text{Var}(S) &= E(S^2) \\ &= \frac{1}{16}(16 + 4 \times 4 + 6 \times 0 + 4 \times 4 + 16) \\ &= \boxed{4}\end{aligned}$$

因此标准差为：

$$\text{SD}(S) = \sqrt{4} = 2$$

注意：虽然 4 个随机数相加可以得到最大值 4，但总和的标准差只有 2。

4.4 先补上你追问的方差定义

因为我们前面假设 X 和 Y 的期望都等于 0，所以方差的定义可以简化。

我们从方差（Variance）的定义开始。

方差的定义

对于任意随机变量 Z ，方差定义为：

$$\text{Var}(Z) = E[(Z - E[Z])^2]$$

这里：

- $E[Z]$ ：随机变量 Z 的期望，也就是均值。
- $Z - E[Z]$ ：随机变量与均值的偏差。
- $(Z - E[Z])^2$ ：偏差的平方。
- $E[(Z - E[Z])^2]$ ：平方偏差的平均值。

所以，方差就是随机变量相对于自身均值的平方偏差的期望。

为什么这里可以省略均值？

我们令：

$$Z = X + Y$$

代入方差定义：

$$\text{Var}(X + Y) = E[((X + Y) - E[X + Y])^2]$$

因为期望满足线性性质：

$$E[X + Y] = E[X] + E[Y]$$

而我们之前假设：

$$E[X] = E[Y] = 0$$

因此：

$$E[X + Y] = 0 + 0 = 0$$

代回方差公式：

$$\begin{aligned}\text{Var}(X + Y) &= E[((X + Y) - 0)^2] \\ &= \boxed{E[(X + Y)^2]}\end{aligned}$$

这就是那个等式成立的原因。

注意：这里根本不需要假设 X 和 Y 独立。只要它们的和的期望为 0，就能这样简化。

如果均值不是 0 呢？

例如：

$$E[X] = 2, \quad E[Y] = 3$$

那么：

$$E[X + Y] = 5$$

此时必须写成：

$$\text{Var}(X + Y) = E[(X + Y - 5)^2]$$

而不能直接写成 $E[(X + Y)^2]$ 。

实际上，把方差定义展开，还可以得到一个常用的恒等式：

$$\boxed{\text{Var}(Z) = E[Z^2] - (E[Z])^2}$$

它对任何具有有限方差的随机变量都成立。

因此，当 $E[Z] = 0$ 时：

$$\text{Var}(Z) = E[Z^2]$$

总结：方差并不总是平方的期望，只有均值为 0 时，二者才相等。

另外，你可以注意到两种性质所需的条件不同：

- $E[X + Y] = E[X] + E[Y]$ ：不需要独立。
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$ ：需要 $\text{Cov}(X, Y) = 0$ ，独立是保证这一点的充分条件，但不是必要条件。

这一区别正是理解 Transformer 中点积方差推导的关键。

4.5 为什么方差是直接相加的？

我们现在从数学上证明这个性质。

假设有两个独立随机变量 X, Y ，其期望都为 0。

那么：

$$\text{Var}(X + Y) = E[(X + Y)^2]$$

展开平方：

$$= E[X^2 + 2XY + Y^2]$$

利用期望的线性性质：

$$= E[X^2] + 2E[XY] + E[Y^2]$$

这里最关键的是：

如果 X, Y 相互独立，那么：

$$E[XY] = E[X]E[Y] = 0$$

所以中间的交叉项消失了！

得到：

$$\boxed{\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)}$$

因此，对于 n 个相互独立、方差都是 1 的随机变量：

$$\text{Var} \left(\sum_{i=1}^n X_i \right) = n$$

而标准差是方差的平方根：

$$\boxed{\text{SD} \left(\sum_{i=1}^n X_i \right) = \sqrt{n}}$$

这里就出现了我们需要的平方根。

4.6 回到 Transformer 的点积

假设 Query 和 Key 都是 64 维：

$$\mathbf{q} = (q_1, q_2, \dots, q_{64})$$

$$\mathbf{k} = (k_1, k_2, \dots, k_{64})$$

点积为：

$$\mathbf{q} \cdot \mathbf{k} = \sum_{i=1}^{64} q_i k_i$$

我们将每一项记为：

$$X_i = q_i k_i$$

如果假设每个 q_i, k_i 都独立、均值为 0、方差为 1，那么：

$$E(X_i) = E(q_i)E(k_i) = 0$$

而且：

$$\begin{aligned} \text{Var}(X_i) &= E(q_i^2 k_i^2) \\ &= E(q_i^2)E(k_i^2) \\ &= 1 \end{aligned}$$

也就是说，**每一个维度的乘积项**，也具有均值 0、方差 1 的性质。

如果不同维度之间也相互独立，那么根据上一节的结论：

$$\text{Var}(\mathbf{q} \cdot \mathbf{k}) = 64$$

因此：

$$\boxed{\text{SD}(q \cdot k) = 8}$$

我们看维度不断增加时会发生什么。

由上述推导可见：维度越高，原始点积的标准差越大；但除以 $\sqrt{d_k}$ 之后，标准差在这一假设下就保持为 1。

要特别区分：

- 标准差为 8，不是说点积必然等于 8。
- 而是说点积围绕均值 0 的波动尺度为 8。
- 如果分布近似正态，那么约 68% 的点积结果会落在 -8 到 8 之间。

4.7 为什么要除以 $\sqrt{64}$ ，而不是除以 64？

这时就很好理解了。

64 维时：

$$\text{SD}(q \cdot k) = 8$$

如果直接除以 64：

$$\text{SD}\left(\frac{q \cdot k}{64}\right) = \frac{8}{64} = 0.125$$

这样标准差反而会随着维度增加而不断减小。

但是如果除以 8：

$$\text{SD}\left(\frac{q \cdot k}{8}\right) = \frac{8}{8} = 1$$

就回到了单位标准差。

因此 Transformer 使用：

$$\frac{q \cdot k}{\sqrt{d_k}}$$

目的不是让点积的值落在某个固定范围内，而是在上述统计假设下，使不同维度的点积分数具有大致相同的波动尺度，避免 Softmax 因输入分数过大而容易饱和。

4.8 还有一个可以联系你已有知识的地方：随机游走

你可以把每一个 $+1$ 或 -1 看成数轴上随机向左或向右走一步。

假设走 64 步：

- 如果始终往右走，位移可以达到 64。
- 但如果方向完全随机，很多步会彼此抵消。
- 最终位移的均方根是 $\sqrt{64} = 8$ 。

这和点积中 64 个随机乘积项相加，具有相同的数学结构。

所以 $\sqrt{d_k}$ 的来源，本质上就是独立随机项求和时，标准差按平方根规律增长。

最后还有一个细节：这个推导依赖于各维度独立、均值及方差满足特定条件的假设。真正训练好的 Transformer 中，Q、K 的各维度并不一定严格独立或服从这种分布，因此除以 $\sqrt{d_k}$ 是合理的尺度调整方法，而不是能保证所有情况下方差都恰好为 1 的数学定理。

4.9 尺度为什么影响Softmax

把分数差放大，Softmax就更容易把比例集中到一个候选。例如(1,0)还会给第二项明显份额，而(100,0)几乎全部给第一项。

对比例 a_j 和分数 s_k 求导，有：

$$\frac{\partial a_j}{\partial s_k} = a_j(\delta_{jk} - a_k).$$

δ_{jk} 在 $j=k$ 时为1，否则为0。若一项接近1、其余接近0，这些导数就接近0，经过注意力比例传回的梯度可能变小。除以根号 d_k ，是帮助控制输入分数尺度，而不是保证所有训练问题都被解决。

这个导数怎样从Softmax求出，第15章保留了商法则的逐步推导。这里不要求先学会矩阵求导。

第 5 章：为什么还要乘 V ，得到的 z 有什么用

[返回目录](#)

现在你已经知道比例从哪里来。但你原来的追问还没有解决：“为什么又乘上 V ？”以及“ x 变成 z ，到底有什么意义？”下面沿用原问答的两层解释：先区分比例与内容，再说明上下文表示的用途。

5.1 先回想你熟悉的凸组合

先不考虑人工智能。

假设三维空间中有三个向量：

$$v_1, v_2, v_3 \in \mathbb{R}^3$$

我们构造一个新向量：

$$z = 0.2v_1 + 0.5v_2 + 0.3v_3$$

你已经知道，因为：

$$0.2 + 0.5 + 0.3 = 1$$

而且每个系数非负，所以 z 是三个向量的凸组合，必然落在它们构成的凸包内。

Transformer 的注意力机制，在很大程度上就是做这件事情：

$$z_i = \sum_{j=1}^n a_{ij} v_j$$

其中：

- v_j ：第 j 个词所提供的信息向量。
- a_{ij} ：第 i 个词分配给第 j 个词的注意力权重。
- z_i ：第 i 个词综合其他词的信息后，得到的新向量。

并且：

$$a_{ij} > 0, \quad \sum_j a_{ij} = 1$$

因此，单个注意力头的每一个输出向量，都是对应 Value 向量的凸组合。

这就是整个架构最值得首先理解的地方。

前两章已经解释这些比例怎样算出，现在重点看被组合的 Value 和结果 z 。

5.2 假设我们只有 Q 和 K，没有 V

例如一句话：

患者服用华法林后出现黑便。

假设现在模型正在处理「黑便」这个 Token，需要从前面的 Token 中获取信息。

第一步，利用「黑便」的 Query 和其他 Token 的 Key 计算匹配分数：

$$s_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

第二步，经过 Softmax，假设得到以下注意力权重：

Token	注意力权重
患者	0.1
服用	0.1
华法林	0.6
黑便	0.2

这意味着对于当前这个注意力头，「华法林」获得最高的注意力权重。

但是你注意到没有？

我们现在只有 0.1、0.1、0.6、0.2 这四个数字。

它们只能告诉我们应该给哪个 Token 分配更多权重，并不能直接告诉我们要从这些 Token 获取什么信息。

因此需要第三种向量： V 。

5.3 V 的作用：提供实际被组合的信息

假设每个 Token 都有一个 Value 向量：

$$v_1, \quad v_2, \quad v_3, \quad v_4$$

那么我们就可以按照刚才计算出的权重进行加权求和：

$$z = 0.1v_1 + 0.1v_2 + 0.6v_3 + 0.2v_4$$

它的意思是：

当前「黑便」这个 Token 的新表示，整合了其他 Token 的信息，其中「华法林」的 Value 所占的权重最大。

这正好对应你之前学过的凸组合。

注意：这里所谓的信息并不是直接把「华法林」这个词复制过去，而是组合它经过线性投影得到的数值向量。

5.4 为什么不直接拿 K 来加权，而要另外计算 V ？

这是更深层的问题。

实际上，从纯数学上讲，完全可以设计一种注意力机制，直接对 K 加权：

$$z = \sum_j a_j k_j$$

但是标准 Transformer 并不这么设计，而是另外引入：

$$k_j = x_j W_K$$

$$v_j = x_j W_V$$

原因是我们希望把两个任务分开：

- **K 负责匹配：** 这个 Token 是否与当前 Query 相关？
- **V 负责传递：** 如果相关，这个 Token 应该向当前 Token 提供什么信息？

这样，模型可以分别训练 W_K 和 W_V ，不要求用于判断相关性的特征与用于传递的特征完全一致。

你可以用一个数据库检索的类比来理解：

假设你想查找某种药物。

- Query：你输入的检索条件。
- Key：数据库用来匹配检索条件的索引。
- Value：匹配之后返回的药物信息。

你不是为了拿到索引本身，而是为了获取索引对应的内容。

5.5 把点积、缩放、Softmax和V连起来再手算一次

假设某个 Token 的 Query 为：

$$q = (\sqrt{2}, 0)$$

三个 Key 为：

$$k_1 = (\ln 2, 0), \quad k_2 = (0, 1), \quad k_3 = (0, -1)$$

取 $d_k = 2$ ，那么缩放后的点积分数为：

$$s = (\ln 2, 0, 0)$$

Softmax 之后：

$$a = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{4} \right)$$

现在假设三个 Value 是平面上的三个点：

$$v_1 = (0,0), \quad v_2 = (4,0), \quad v_3 = (0,4)$$

那么注意力输出：

$$\begin{aligned} z &= \frac{1}{2}v_1 + \frac{1}{4}v_2 + \frac{1}{4}v_3 \\ &= \boxed{(1,1)} \end{aligned}$$

可以看到，我们只做了三件事：

1. 使用点积确定匹配程度。
2. 使用 Softmax 把分数转换成凸组合系数。
3. 使用这些系数对 Value 进行加权平均。

这三个操作就是 Scaled Dot-Product Attention 的核心。

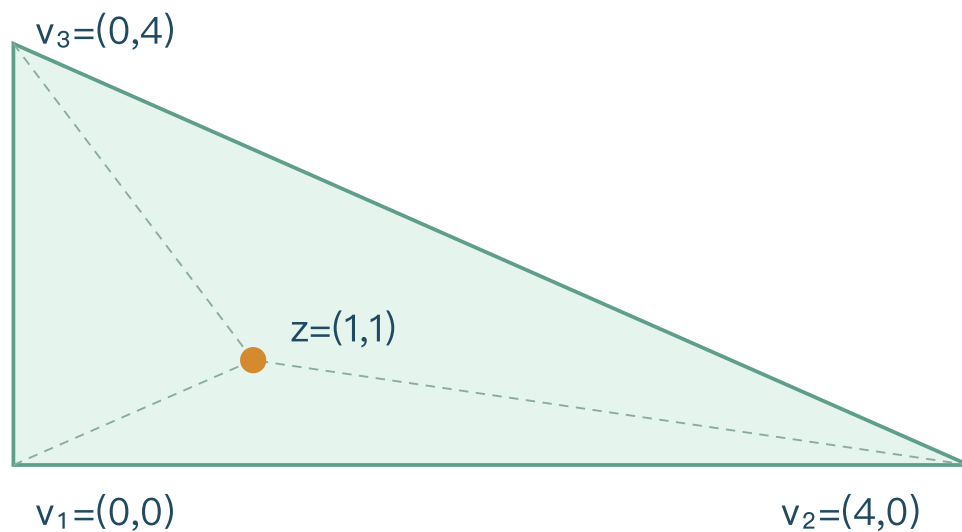


图3：单头输出是当前Value的凸组合。这里 $z=(1,1)$ ，在三个Value围成的三角形内。

5.6 假设没有 Attention，会怎样？

例如：

我刚吃了苹果。

我投资了苹果。

假设我们单独考虑「苹果」这个 Token。

最初通过 Embedding 得到：

$$\mathbf{x}_{\text{苹果}} \in \mathbb{R}^d$$

在这两个句子里，「苹果」这个 Token 的初始词嵌入可以是完全相同的。

于是出现问题：

同样的向量，怎么区分第一个是水果，第二个是公司？

如果我们只对这个向量进行固定的线性变换：

$$\mathbf{y} = \mathbf{x}_{\text{苹果}} \mathbf{W}$$

那么只要 $\mathbf{x}_{\text{苹果}}$ 相同，输出就相同。

也就是说，它无法根据其他词语区分语境。

5.7 Attention 产生的 z 就是为了解决这个问题

对于「苹果」这个 Token，我们计算：

$$z_{\text{苹果}} = \sum_j a_{\text{苹果},j} v_j$$

在第一句话中，模型可能更多地从「吃了」等位置提取信息。

在第二句话中，模型可能更多地从「投资了」等位置提取信息。

因此，虽然初始的 $x_{\text{苹果}}$ 相同，但由于其他 Token 不同，得到的 $z_{\text{苹果}}$ 也可以不同。

这就是 Attention 的价值：让一个 Token 的表示受到上下文的影响。

5.8 但 z 到底是什么？它还是一个普通的向量吗？

是的。

例如：

$$z_{\text{苹果}} = (0.2, -0.7, 1.3, \dots)$$

它仍然只是一个高维向量。

你可能会进一步问：

这些数字有什么实际意义？0.2 代表水果吗？-0.7 代表公司吗？

通常不是。

Transformer 并没有预先规定每一维必须对应一个具体概念。语义信息可以分布在许多维度中，而且一个维度也可能参与表示许多不同的特征。

z 的价值不在于人类能够直接读懂它，而在于后续网络能够利用它。

例如，模型可以通过后面的层，把新获得的上下文信息与当前 Token 原有的信息结合起来。

5.9 实际上 z 还不是最终输出

这里需要纠正一个容易产生的误解。

Attention 计算：

$$z_i = \sum_j a_{ij} v_j$$

得到的 z_i 是一个注意力头的信息聚合结果。

经过多头拼接、输出投影之后，在常见的残差结构中，还会与原来的输入相加。

用单头简化表示：

$$x_i^{\text{new}} = x_i + z_i W_O$$

假设采用行向量约定，并令矩阵维度匹配。

你可以理解成：

$$\underbrace{x_i}_{\text{原来具有的信息}} + \underbrace{z_i W_O}_{\text{从上下文整合的新信息}} = \underbrace{x_i^{\text{new}}}_{\text{更新后的表示}}$$

这个新向量再进入后续的神经网络和 Transformer Block。

经过多层处理后，GPT 最终利用最后位置的隐藏向量 h_T 计算下一个 Token 的概率：

$$P(t_{T+1} \mid t_1, \dots, t_T) = \text{softmax}(h_T W_{\text{out}} + b)$$

这里的输出才真正用于生成语言。

5.10 所以，从数学上看， z 的意义究竟是什么？

你可以把它理解成一种依赖于上下文的信息更新量。

普通线性变换：

$$x_i \longrightarrow x_i W$$

只依赖于 x_i 。

而 Attention:

$$x_i \longrightarrow z_i = \sum_j a_{ij}(X) x_j W_V$$

不仅依赖于 x_i ，还依赖于其他 x_j 。

因此，原本彼此独立表示的 Token，现在可以相互传递信息。

这使模型能够建立词语之间的联系，而不只是逐个处理词语。

但也必须强调：数学上仅仅计算出 z ，不保证它就包含有用的语义。只有经过训练，使整个网络能够利用这些向量降低预测误差时， z 才真正具有功能上的意义。

5.11 这里的“有用”需要训练来建立

即使参数随机，也能照公式算出 z ，但它未必对预测有用。我们不是看见 z 里出现某个神奇坐标，就宣布模型理解了语言；而是后面会看到，它怎样被输出层利用，以及训练怎样改变产生它的参数。

只对归一化后的注意力组合而言，系数非负、和为1，所以属于当前Value的凸包。多头投影、残差与FFN不受同一个凸包约束；训练时若对系数做未重新归一化的dropout，当次组合也不能直接套用和为1的结论。

5.12 再看一遍标明分工的完整Attention公式

$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}_{\text{计算注意力权重 } A} \underbrace{V}_{\text{提供待整合的信息}}$$

其中：

$$A \in \mathbb{R}^{n \times n}, \quad V \in \mathbb{R}^{n \times d_v}$$

矩阵乘法：

$$Z = AV$$

得到：

$$Z \in \mathbb{R}^{n \times d_v}$$

这意味着，每个 Token 都得到一个新的向量表示。

最终记住一句话：Q 用来提问，K 用来匹配，V 用来提供答案所需的信息；Softmax 决定每份信息占多大权重。

而你已经理解的凸组合，恰好就是 AV 这一步的数学本质。

第 6 章：把逐个向量的计算写成矩阵，再变成多头

[返回目录](#)

前面我们逐个q与k计算，再逐个v加权。现在才把这些运算装进矩阵，说明一次处理整句时每一行仍在做同样的事。大写字母只是把已经理解的计算合在一起。

6.1 先看前五个位置怎样同时得到五个新向量

这里用前五块“患者、服用、华法林、后、出现”作输入。下面的5x5只表示这个输入长度；完整句子包含七块，长度变化后矩阵的行数与列数相应变化。

我们已经知道：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

假设输入 5 个 Token，每个注意力头的 Q、K 都是 64 维。

那么：

$$Q, K \in \mathbb{R}^{5 \times 64}$$

接下来：

$$QK^T \in \mathbb{R}^{5 \times 5}$$

也就是说，模型计算了一个 5×5 的匹配分数矩阵。

经过缩放、因果掩码和 Softmax，得到注意力矩阵 A 。

下面用一个假设的权重矩阵展示它的结构。

最终：

$$\boxed{Z = AV}$$

这里的 Z 不只是一个向量，而是一个矩阵：

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

也就是说，输入整句话的五个向量，经过注意力计算，得到五个新的向量。

它们随后还会经过输出投影、残差连接、前馈网络等处理，进入下一个 Transformer Block。

6.2 一次矩阵乘法怎样完成全部点积与组合

你之前也讨论过矩阵乘法能够把大量向量运算合并进行。

假设输入有 n 个 Token：

$$X \in \mathbb{R}^{n \times d}$$

投影后：

$$Q, K \in \mathbb{R}^{n \times d_k}, \quad V \in \mathbb{R}^{n \times d_v}$$

那么：

$$QK^T \in \mathbb{R}^{n \times n}$$

这个矩阵中的第 i 行、第 j 列，就是：

$$(QK^T)_{ij} = q_i \cdot k_j$$

也就是说，一次矩阵乘法就计算了每一个 Token 对其他所有 Token 的匹配分数。

接下来，对每一行使用 Softmax：

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right)$$

再做一次矩阵乘法：

$$Z = AV$$

把整个过程合并：

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

这是 Transformer 最经典的一条公式。

注意它的几何含义：

- QK^T ：计算点积，决定权重。
- Softmax：把每一行变成非负且和为 1 的系数。
- AV ：逐行构造 Value 的凸组合。

你可以把注意力矩阵 A 理解为一个由当前输入动态生成的加权组合矩阵。

这点很重要：

W_Q, W_K, W_V 是训练后相对固定的参数，而 A 会随着每次输入的文本发生改变。

6.3 把矩阵乘法展开一项，就能核对

QK^T 第 i 行第 j 列为：

$$(QK^T)_{ij} = \sum_{r=1}^{d_k} Q_{ir} K_{jr} = q_i \cdot k_j.$$

r 在这里遍历向量坐标， i/j 遍历输入位置。接着：

$$(AV)_{iu} = \sum_{j=1}^n a_{ij} V_{ju}.$$

u 是输出向量中的一个坐标。让 u 遍历全部坐标，得到的正是 $z_i = \sum_j a_{ij} v_j$ 。因此矩阵写法与逐向量计算严格相同。

6.4 为什么还需要多头注意力

假设输入：

患者服用华法林后出现黑便，血红蛋白明显下降。

模型可能同时需要识别不同种类的联系：

- 药物与不良反应的联系。
- 症状与实验室检查的联系。
- 句法结构中不同词语之间的联系。

如果仅有一个注意力头，那么每个 Token 就只有一套 Q、K、V 变换。

Transformer 采用多个不同的投影矩阵：

$$W_Q^{(1)}, W_K^{(1)}, W_V^{(1)} \dots$$

更准确地写，各头有各自的 $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}$ 。

每个头独立计算：

$$Z^{(h)} = \text{Attention}(Q^{(h)}, K^{(h)}, V^{(h)})$$

最后拼接并进行线性变换：

$$Z = \text{Concat}(Z^{(1)}, \dots, Z^{(H)})W_O$$

这就是 **Multi-Head Attention（多头注意力）**。

可以把它类比为多个不同的坐标投影方式，各自提取信息后再重新整合。但各头并不保证严格对应某种可解释的语法或医学关系。

6.5 拼接后为什么还要乘 W_O

设有 H 个头，每头分别计算 $Z^{(r)} \in \mathbb{R}^{n \times d_v}$ 。拼接与投影是：

$$O = \text{Concat}(Z^{(1)}, \dots, Z^{(H)})W_O, \quad W_O \in \mathbb{R}^{Hd_v \times d}.$$

拼接后是 $n \times Hd_v$ ，乘 W_O 回到 $n \times d$ ，才能与原输入相加。多头允许多个混合分布同时存在，不要求“药物头”“语法头”这样人为固定的语义分工。

第 7 章：注意力以外还有什么，为什么要一层层处理

[返回目录](#)

注意力负责不同位置之间的信息交换，但完整Transformer不止这一步。这里保留原文中FFN、残差、位置和多层复合的解释，只补清它们怎样接起来。较细的归一化导数放在第15章，不挡住这条主线。

7.1 注意力只是一个Block的一部分

到目前为止，我们只解释了注意力。

真正的 Transformer Block 还有其他结构。

这里有两个你尚未重点接触的概念，但它们并不复杂。

前馈网络 (Feed-Forward Network)

注意力负责不同 Token 之间的信息整合。

但整合之后还要对每个 Token 自身的特征进行加工。

一个经典的 FFN 可以写成：

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2$$

其中 σ 是非线性激活函数，例如 ReLU 或 GELU；现代模型也经常使用门控结构。

从你熟悉的线性代数角度看，它相当于：

线性变换 $\rightarrow$ 非线性变换 $\rightarrow$ 线性变换

为什么要加入非线性？

因为如果始终只有固定的线性变换：

$$(xW_1)W_2 = x(W_1W_2)$$

多个线性变换仍然可以合并成一个线性变换。

加入非线性函数后，整体一般就不能简化成一个矩阵乘法，能够表达更复杂的关系。

残差连接 (Residual Connection)

其基本形式是：

$$x_{\text{new}} = x + F(x)$$

也就是把模块计算的新信息加到原信息上。

这样提供一条原输入的直连路径，通常有助于深层训练与梯度传播；并不保证任何情况下信息都不丢失。

这也是为什么我前面特别强调：

只有注意力头中的 AV 是凸组合，整个 Transformer Block 的输出一般不再局限于原始 Value 的凸包。

因为后面还有线性投影、残差相加和非线性变换。

7.2 顺序信息怎样进入模型

你可能已经发现一个问题。

单纯从：

$$QK^T$$

来看，在没有位置编码、也没有位置相关掩码时，改变Token排列会相应重排注意力输出，结构本身不提供顺序标签。因果掩码已引入方向约束，不能把无掩码结论直接照搬。

比如：

「医生治疗患者」与「患者治疗医生」

词语相同，但意义不同。

Transformer 因此需要位置相关的信息（Positional Information）。

经典的一种做法是：

$$\tilde{x}_i = x_i + p_i$$

其中 p_i 是表示位置的向量。

现代语言模型也常使用 RoPE（Rotary Position Embedding），将位置信息引入 Q、K 的计算。

你之前学过旋转矩阵，可以把 RoPE 看作一个很好的进阶理解对象：它通过与位置有关的旋转，让注意力分数反映相对位置信息。

7.3 普通的加权平均，与 Attention 有什么本质区别？

假设有三个向量：

$$x_1, x_2, x_3$$

最普通的平均：

$$z = \frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3$$

稍微复杂一点，可以设置不同的固定权重：

$$z = 0.2x_1 + 0.5x_2 + 0.3x_3$$

但是不管输入是什么，这三个系数始终不变。

而 Attention 不一样：

$$z_i = \sum_j a_{ij}(X)v_j$$

注意这里的：

$$a_{ij}(X)$$

权重是输入 X 的函数。

也就是说，模型不是简单地执行一个事先规定的加权平均，而是根据这一次的输入，现场计算应该采用怎样的加权系数。

这就是核心差别：**模型能根据当前任务和上下文，改变信息整合的方式，而不需要人提前指定每个词应该关注谁。**

7.4 从线性代数来看，它已经不是普通的线性变换了

你之前学过：

$$y = xW$$

给定矩阵 W ，这是一个固定的线性变换。

我们将 Transformer 的注意力公式完全展开：

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

代入：

$$Z = \text{softmax} \left(\frac{XW_Q W_K^T X^T}{\sqrt{d_k}} \right) XW_V$$

你仔细观察它。

其中 W_Q, W_K, W_V 是训练得到的固定矩阵。

但是中间出现了：

$$XW_Q W_K^T X^T$$

这意味着注意力分数取决于输入向量之间的两两相互作用。

再经过非线性的 Softmax，得到：

$$A(X)$$

最终：

$$Z = A(X)XW_V$$

与普通的 $Y = XW$ 不同，这里的 $A(X)$ 并不是事先固定的矩阵。

所以，虽然一次注意力计算的最后一步是加权平均，但整个从 X 到 Z 的映射一般是非线性的。

这就突破了固定线性变换只能按照固定规则处理数据的限制。

7.5 为什么堆叠几十层、上百层会产生更复杂的能力？

假设最开始每个 Token 只有比较基础的信息。

第一层 Attention：

$$X^{(1)} = F_1(X^{(0)})$$

让它们整合其他 Token 的信息。

第二层：

$$X^{(2)} = F_2(X^{(1)})$$

注意，这次输入已经不是最初的词向量了，而是经过第一层上下文整合的新向量。

这意味着第二层不仅能关注原始词语，还能进一步处理第一层已经提取出的关系。

第三层再继续：

$$X^{(3)} = F_3(X^{(2)})$$

最终形成：

$$X^{(L)} = F_L \circ F_{L-1} \circ \dots \circ F_1(X^{(0)})$$

这就和你之前学习过的复合映射联系起来了。

可以想象一个医学场景：

患者使用抗凝药物后出现黑便，血红蛋白由 120 g/L 降至 75 g/L。下一步应该考虑什么？

某些注意力头或中间表示可能整合出药物与症状的关系，其他部分可能捕捉血红蛋白变化，再通过后续层将这些信息进一步结合。

需要强调，这只是概念示意，并不是说真实模型严格按照「第一层识别药物、第二层判断出血、第三层推理」这样的人工步骤运作。

而且，Transformer 每一层还有前馈神经网络、残差连接等结构，因此整个网络远不止反复进行凸组合。

7.6 最值得惊讶的其实是：这些规则并不是人写进去的

这一点可能比注意力公式本身更重要。

我们并没有手动告诉模型：

- 遇到「华法林」，要联系「出血」。
- 遇到「血红蛋白下降」，要考虑失血。
- 遇到「它」，需要识别它指代前文的什么。

我们只是设定了 Transformer 的计算结构，然后通过大量训练数据，优化 W_Q, W_K, W_V 及其他参数。

模型在不断降低预测误差的过程中，逐渐学会了如何组织中间表示、匹配相关信息、组合不同位置的信息。

不过也不能据此直接认定模型具有与人类相同的理解或推理机制。注意力本身只是计算工具，复杂能力来自注意力、非线性网络、训练目标、数据以及规模等因素的共同作用。

7.7 归一化、残差与完整一层的关系

前面讲了模块用途，再明确一种常见排列。令 X 是当前层收到的整句表示，MHA表示多头注意力，LN表示逐位置的LayerNorm：

$$T = X + \text{MHA}(\text{LN}_1(X)),$$

$$Y = T + \text{FFN}(\text{LN}_2(T)).$$

先归一化，算注意力，加回X，得到T；再归一化T，过FFN，加回T，得到Y，送进下一层。这叫Pre-LN。原始2017年论文用的是子层相加后再归一化的Post-LN，不要把这两种顺序当成同一精确配置。

LayerNorm只在同一位置的坐标里取均值和波动尺度。例如(1,2,3)先减均值2，得到(-1,0,1)，再缩放。它不需要未来位置的数。完整公式和残差梯度依据仍保留在第15章。

第 8 章：没有掩码会怎样，怎样防止偷看未来

[返回目录](#)

你说“没有M，每个词都会联系其他词吧”，这个判断正是理解掩码的入口。先看没有限制的情况，再看为什么下一个Token训练必须限制。

8.1 没有 M：双向注意力（Bidirectional Attention）

计算：

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right)$$

得到一个 5×5 的注意力矩阵。

可以看到：

- 没有 M ：全部 25 个位置都可以参与注意力计算。
- 有因果掩码 M ：只有主对角线及其下方的 15 个位置可以参与。

8.2 既然如此，为什么 GPT 要加 M？

关键在于 GPT 的训练目标是预测下一个 Token。

例如：

患者服用华法林后出现_____

我们希望模型根据已有内容预测后面的词。

假设训练数据的完整句子是：

患者服用华法林后出现黑便。

如果不使用因果掩码，在训练时，当模型使用「出现」位置的隐藏向量预测「黑便」时，它可以直接关注后面的「黑便」。

这就相当于**预测答案之前，已经提前看到了答案。**

训练过程会发生信息泄漏（Information Leakage）。

所以 GPT 加入：

$$M_{ij} = \begin{cases} 0, & j \leq i \\ -\infty, & j > i \end{cases}$$

使未来位置的注意力权重严格等于 0。

这样，第 i 个位置的隐藏向量只能依赖前 i 个 Token：

$$h_i = F_{\theta}(x_1, \dots, x_i)$$

就可以用它预测第 $i + 1$ 个 Token，而不会提前看到正确答案。

8.3 那没有 M 的 Transformer 就没有用吗？

当然不是。

例如 BERT 就主要采用双向注意力。

如果任务是理解一整句话，而不是逐个预测后面的词，那么同时利用前后文通常很有帮助。

比如：

他在银行办理了贷款。

如果我们希望理解「银行」的意义，后面的「贷款」也能提供有用的信息。

BERT 可以让「银行」同时关注前后的 Token，而 GPT 的因果注意力不允许这个位置关注后面的「贷款」（但 GPT 在处理最后一个位置时，仍然可以整合整句话的信息）。

所以两者并不是谁绝对更高级，而是对应不同的计算约束和训练任务。

最后要补充一个精确的数学细节：无掩码时，各个位置的 Softmax 权重通常都严格大于 0，因此所有 Token 在数学上都会参与加权；但这不意味着它们具有同样的重要性，有些权重可以非常接近 0。

8.4 用三个位置把它算成一张表

例如：

患者 服用 华法林

为了简化，假设这句话恰好分成三个 Token：

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

其中：

- x_1 : 患者
- x_2 : 服用
- x_3 : 华法林

现在我们对整个矩阵 X 进行 Self-Attention:

$$Z = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V$$

这里多了一个 M ，它就是因果掩码 (Causal Mask)。

它的作用是：每个 Token 只能获取自己以及自己之前的 Token 的信息，不能看未来。

8.5 看最后一行怎样读取完整前缀

假设经过 Softmax，得到：

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0.4 & 0.6 & 0 \\ 0.2 & 0.5 & 0.3 \end{bmatrix}$$

这是一个假设的注意力矩阵，不是真实模型的计算结果。

你注意最后一行：

$$z_3 = 0.2v_1 + 0.5v_2 + 0.3v_3$$

这说明，虽然 z_3 对应的是「华法林」这个位置，但它不再只依赖于「华法林」，而是同时整合了「患者」「服用」「华法林」的信息。

这就是为什么最后一个位置如此重要。

	患者	服用	华法林	后	出现	黑便	。
患者	可见	屏蔽	屏蔽	屏蔽	屏蔽	屏蔽	屏蔽
服用	可见	可见	屏蔽	屏蔽	屏蔽	屏蔽	屏蔽
华法林	可见	可见	可见	屏蔽	屏蔽	屏蔽	屏蔽
后	可见	可见	可见	可见	屏蔽	屏蔽	屏蔽
出现	可见	可见	可见	可见	可见	屏蔽	屏蔽
黑便	可见	可见	可见	可见	可见	可见	屏蔽
。	可见	可见	可见	可见	可见	可见	可见

图4：每行表示当前位置允许读取哪里。橙框“出现”行不能读取答案“黑便”。这里没有展示注意力大小。

8.6 为什么叠很多层也不能绕路偷看

先看第一层：位置*i*只读取*j*≤*i*的输入，因此只可能依赖前*i*个Token。假设上一层每个位置*j*也只依赖前*j*个Token，那么位置*i*下一层读取的那些*j*≤*i*，仍然只携带前*i*个Token的信息。

FFN、残差和逐位置归一化不引入别的位置，所以逐层归纳后，最终*h_i*仍只依赖前*i*个Token。这个论证需要所有跨位置操作都遵守限制；若另外偷偷做全句汇总，就不能照用。

为写得精确，可以不把负无穷当普通实数，而直接规定：只在允许位置集合 $J_i = \{1, \dots, i\}$ 中做Softmax，其他系数设0：

$$a_{ij} = \begin{cases} \frac{e^{s_{ij}}}{\sum_{r \in J_i} e^{s_{ir}}}, & j \in J_i, \\ 0, & j \notin J_i. \end{cases}$$

允许读取自己，保证每行至少有一项。数学上无掩码且分数有限时各权重严格正，但可能极小；实际浮点计算还可能下溢为0。

第 9 章：向量怎样变成下一 Token 的预测概率

[返回目录](#)

你后来问“这里还得有一个什么预测函数吧”，这一问不能等到讲损失时才补。我们先完整说明h怎样变成候选分数和概率，再进入训练例子。

下面用 θ 统称网络参数，用 $F\theta$ 表示整套网络； h_i 是它在第i位置的最终行向量。下标i在这一章始终表示位置。

9.1 最后位置的向量，为什么可以用于整个前缀的预测

这里必须区分两个概念：

最后一个 Token 的原始向量 x_3 和 经过多层 Transformer 计算后的最终隐藏向量 h_3 。

它们不是同一个东西。

最开始：

$$x_3 = \text{Embedding}(\text{华法林})$$

这时它主要是「华法林」这个 Token 的初始表示，外加可能的位置相关信息。

经过第一层 Attention：

$$x_3 \longrightarrow z_3^{(1)}$$

它开始获得前面 Token 的信息。

再经过输出投影、残差连接、前馈网络，得到第一层更新后的隐藏表示。

第二层又重新计算 Q、K、V，进一步整合信息。

反复进行 L 层后：

$$h_3 = F_{\theta}(x_1, x_2, x_3)_3$$

也就是说，最后得到的 h_3 已经是一个依赖整段前缀的向量表示。

但这里有一个细节：它并不需要精确地保存句子中的全部原始信息。

它只需要在训练目标下，保留和组织足够有用的信息，以便模型预测后续 Token。

例如：

患者服用华法林后出现.....

最后一个位置的隐藏向量可能编码了与患者、用药及当前句子结构有关的信息，使模型能够为下一个 Token 计算概率。

9.2 先看完整的数学关系

假设输入：

患者服用华法林后出现

首先经过 Transformer：

$$X \xrightarrow{\text{Transformer}} H$$

其中 H 是各个 Token 经过多层处理后的隐藏向量矩阵。

我们取最后一个位置的隐藏向量：

$$h_5 \in \mathbb{R}^{d_{\text{model}}}$$

然后进行一次线性变换：

$$\ell = h_5 W_{\text{out}} + b$$

其中：

- W_{out} ：输出层的权重矩阵，也需要通过训练学习。
- b ：可选的偏置向量。
- ℓ ：输出分数向量（Logits），每个候选 Token 对应一个分数。

如果模型的隐藏向量维度是 512，词表有 50,000 个 Token：

$$h_5 \in \mathbb{R}^{512}$$

$$W_{\text{out}} \in \mathbb{R}^{512 \times 50000}$$

因此：

$$\ell \in \mathbb{R}^{50000}$$

也就是模型一次性算出了 50,000 个候选 Token 的分数。

9.3 但是分数还不是概率

这就又需要我们之前学过的 Softmax。

假设词表只有三个候选 Token，输出层计算得到：

$$\ell = [2 \quad 1 \quad 0]$$

它们分别对应：

Token	输出分数 (Logit)
黑便	2
呕血	1
呼吸困难	0

经过 Softmax：

$$P_j = \frac{e^{\ell_j}}{\sum_k e^{\ell_k}}$$

得到：

$$\begin{aligned} P(\text{黑便}) &= \frac{e^2}{e^2 + e^1 + e^0} \approx 0.665 \\ P(\text{呕血}) &\approx 0.245 \\ P(\text{呼吸困难}) &\approx 0.090 \end{aligned}$$

现在我才真正得到了预测概率。

注意，这是第二次遇到 Softmax：

- Attention 的 Softmax：计算不同输入 Token 的注意力权重。
- 输出层的 Softmax：计算下一个 Token 的预测概率。

两者作用不同。

9.4 所以完整的预测函数是什么？

我们将整个 Transformer 记为一个函数：

$$H = F_{\theta}(X)$$

其中 θ 表示 Transformer 内部的所有可训练参数，包括各层的 W_Q, W_K, W_V 等。

取最后一个位置的向量：

$$h_5 = F_{\theta}(X)_5$$

那么预测函数就是：

$$P(t_6 | X) = \text{softmax}(F_{\theta}(X)_5 W_{\text{out}} + b)$$

注意，这个函数的输出不是一个概率，而是整个词表上的概率分布。

我们取其中「黑便」对应的概率，再计算损失：

$$L = -\ln P(t_6 = \text{黑便} | X)$$

因此整个过程实际上是：

9.5 把预测函数中的每个符号读出来

F_{θ} 表示前面整套Transformer计算； θ 是其中参数的统称，不是输入文本。 $F_{\theta}(X)_5$ 表示结果矩阵的第5行，不是第五个坐标。

W_{out} 是输出层可训练矩阵。有些资料记作U，意思相同；本版主线沿用原对话的Wout。b是可选偏置， ℓ 是输出分数向量，不是损失L。

式子 $P(t_6 | X)$ 在这里简写整个下一Token分布。要读“黑便概率”这一个数，应写成 $P(t_6 = \text{黑便} | X)$ 。竖线读作“在X已知的条件下”。

Attention的Softmax沿输入位置分比例；输出Softmax沿词表分概率。前面没出现过的Token，也有自己的输出列，可以被预测出来。

9.6 为什么训练还需要前面各位置的向量

你可能已经想到：

既然最终只用 h_n ，那么前面的 $h_1, h_2, \dots, h_{n-1}$ 是不是多余的？

并不是。

首先，前面位置在各层产生的表示，会参与后续层对其他Token的注意力计算。

其次，训练阶段可以利用所有位置预测各自的下一个Token。

例如：

当前位置的隐藏向量	对应的预测目标
h_1	t_2
h_2	t_3
h_3	t_4
h_4	t_5

因果掩码保证了每个位置都不能提前看到自己的预测目标。

所以训练时可以并行计算多个位置的损失；推理时，前面的位置已有对应的Token，我们主要需要预测尚未生成的下一个Token，因此只使用最后位置的输出分布。

9.7 从五个位置的例子推广到任意长度

假设输入总共有 n 个 Token，经过所有 Transformer 层以后得到：

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}$$

在推理时，我们取最后一行：

$$h_n \in \mathbb{R}^{d_{\text{model}}}$$

然后再做一次线性变换：

$$\ell = h_n W_{\text{out}} + b$$

假设：

$$d_{\text{model}} = 512$$

词表中有 50,000 个可能的 Token。

那么：

$$W_{\text{out}} \in \mathbb{R}^{512 \times 50000}$$

矩阵乘法后：

$$\ell \in \mathbb{R}^{50000}$$

这意味着：从一个 512 维的上下文向量，计算出对词表中 50,000 个候选 Token 的预测分数。

再经过 Softmax：

$$P(t_{n+1} = j) = \frac{e^{\ell_j}}{\sum_k e^{\ell_k}}$$

就能得到下一个 Token 的概率分布。

这里有一个你之前可能没有意识到的区别：

Attention 里面的 Softmax，是决定从已有 Token 获取多少信息；输出层的 Softmax，是决定下一个 Token 应该是什么。

它们虽然使用同一个数学函数，但作用完全不同。

第 10 章：用同一句话看训练和生成两种情况

[返回目录](#)

现在输入、注意力、h与输出层都已经有了意义，可以把原问答中整句“患者服用华法林后出现黑便。”的训练和生成示例完整接上。这里的概率仍然是示意值，不是真实模型运行结果。

10.1 首先，把一句话转换成 Token

为了方便理解，假设模型将这句话切分成 7 个 Token（实际 Tokenizer 不一定这样切分）。

每个 Token 转换成向量，再组成矩阵：

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

我们现在分别看两种情形。

10.2 训练阶段：模型已经有整句话作为标准答案

训练数据中已经存在：

患者服用华法林后出现黑便。

模型的任务是学习：给定前面的 Token，后面应该是什么 Token？

一次训练可以同时预测多个位置

当前已知前缀	要预测的正确 Token
患者	服用
患者 服用	华法林
患者 服用 华法林	后
患者 服用 华法林 后	出现
患者 服用 华法林 后 出现	黑便
患者 服用 华法林 后 出现 黑便	。
患者 服用 华法林 后 出现 黑便 。	结束标记（示意）

重点是：训练时可以把完整的句子一次性输入 Transformer，而不是必须分别运行七次。

但既然输入里已经有「黑便」，不就泄漏答案了吗？

这就是我们刚刚讨论的因果掩码 M 的作用。

当模型计算第 5 个位置「出现」的注意力时，只允许访问：

$$x_1, x_2, x_3, x_4, x_5$$

不允许访问：

$$x_6, x_7$$

因此虽然整个矩阵 X 包含「黑便」，但第 5 个位置的计算不能利用它。

这里要记住：第 5 个位置经过多层 Transformer 后得到的 h_5 ，就是用来预测「黑便」的。

假设模型预测错误了

假设输出层给出的概率为：

候选 Token	模型预测概率（示意）
呼吸困难	45%
黑便	30%
呕血	15%
其他	10%

正确答案是「黑便」，但模型给它的概率只有 30%。

于是计算交叉熵损失：

$$L = -\ln P(\text{黑便} \mid \text{前文})$$

对于这个位置：

$$L = -\ln(0.3)$$

同样，训练还会对其他位置计算损失，再进行汇总。

然后才是真正的训练

通过反向传播，计算损失对参数的梯度：

$$\frac{\partial L}{\partial W_Q}, \quad \frac{\partial L}{\partial W_K}, \quad \frac{\partial L}{\partial W_V}$$

优化器根据这些梯度更新权重，例如以最简单的梯度下降表示：

$$W_Q^{\text{new}} = W_Q^{\text{old}} - \eta \frac{\partial L}{\partial W_Q}$$

其他参数也类似。

通过大量训练，模型逐渐调整这些矩阵，使正确的下一个 Token 获得更高的预测概率。

还有一个容易忽略的地方：训练时即使模型错误地预测了「呼吸困难」，用于其他位置预测的输入仍然是训练数据里的正确 Token 「黑便」。这通常称为 Teacher Forcing（教师强制）。

10.3 推理阶段：模型不知道后面的正确答案

现在换成已经训练完成的 GPT。

你输入：

患者服用华法林后出现

注意，此时输入里没有「黑便」。

模型收到的 Token 是：

$$(t_1, t_2, t_3, t_4, t_5)$$

组成：

$$X \in \mathbb{R}^{5 \times d}$$

第一步：进行 Attention 运算

仍然计算：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

然后：

$$Z = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V$$

经过多个 Transformer Block，得到：

$$H = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix}$$

取最后一个位置：

$$h_5$$

第二步：预测下一个 Token

模型利用 h_5 计算词表中所有候选 Token 的概率。

例如，这次预测结果可能是：

候选 Token	预测概率（示意）
黑便	60%
呕血	20%
头晕	10%
其他	10%

模型根据解码策略选择一个 Token。

假设选择了「黑便」。

第三步：将生成的 Token 放回输入序列

新的序列是：

患者服用华法林后出现黑便

现在有 6 个 Token。

模型再次根据最后一个位置的隐藏向量预测后续 Token。

假设又生成「。」。

于是得到：

患者服用华法林后出现黑便。

但请注意，模型不一定会生成这句话。这里只是展示一种可能的生成过程。

整个过程中， W_Q, W_K, W_V 一直是训练完成后已经确定的参数，不会因为本次生成而更新。

10.4 把两种情况直接对比

现在我们专门研究「出现」后面生成「黑便」的这个位置。

	训练阶段	推理阶段
模型能看到的前文	患者服用华法林后出现	患者服用华法林后出现
数据中是否有正确答案「黑便」	有，但被因果掩码挡住	尚未提供
是否计算 Q, K, V	是	是
是否计算 Z 和 h_5	是	是
是否预测第 6 个 Token	是	是
预测错误后	计算损失，更新参数	按解码策略生成 Token
是否知道标准答案	是	否
W_Q, W_K, W_V 是否更新	是	否

你应该注意到一件非常重要的事情：

对于完全相同的前缀，如果训练和推理使用的是完全相同的模型参数，那么在相同计算条件下，两者得到的下一 Token 概率分布也是相同的。

区别主要在于后续如何使用这个概率分布，而不是 Attention 采用了不同的数学公式。

10.5 如果推理时输入的已经是完整句子呢？

这也正是你最开始提出的疑问。

假设你直接输入：

患者服用华法林后出现黑便。

那么 GPT 不会再把「黑便」当成下一步需要预测的内容。

因为「黑便」已经属于输入。

它会计算整个输入序列最后位置「。」的隐藏向量 h_7 ，用来预测第 8 个 Token。

如果你输入的是：

患者服用华法林后出现黑便。请分析可能的原因：

那么模型就会从冒号之后开始生成内容，例如讨论消化道出血及抗凝相关出血风险。

所以实际上，ChatGPT 接收的不只是孤立的一句话，还可以接收系统指令、前面的对话历史、当前问题等共同组成的上下文序列。随后它从整个前缀的末尾继续生成回答。

最后再把 X、Z 和 W 的关系串起来

这是整个过程最容易混淆的地方。

你可以这样记忆：

- **W** 是模型长期学到的「规则」。
- **X** 是这一次输入的内容。
- **Z** 是按照规则，从输入中动态整合出来的信息。

- h_n 是经过整个模型处理、用于预测下一个 Token 的最终表示。

而所谓训练与推理的区别，本质上就是：

训练：用已知的正确答案，调整规则 W 。

推理：使用已经学好的规则 W ，预测未知的后续内容。

如果你接下来希望继续深入数学原理，我认为最重要的问题已经从 Attention 转移到了**反向传播**：模型究竟怎样根据「黑便」预测错误所产生的损失，反过来知道应该修改 W_Q 中的哪些数字？这一步才真正解释了神经网络的「学习」是怎样发生的。

10.6 最后再把 X 、 Z 和 W 的关系串起来

这是整个过程最容易混淆的地方。

你可以这样记忆：

- W 是模型长期学到的「规则」。
- X 是这一次输入的内容。
- Z 是按照规则，从输入中动态整合出来的信息。
- h_n 是经过整个模型处理、用于预测下一个 Token 的最终表示。

而所谓训练与推理的区别，本质上就是：

训练：用已知的正确答案，调整规则 W 。

推理：使用已经学好的规则 W ，预测未知的后续内容。

如果你接下来希望继续深入数学原理，我认为最重要的问题已经从 Attention 转移到了**反向传播**：模型究竟怎样根据「黑便」预测错误所产生的损失，反过来知道应该修改 W_Q 中的哪些数字？这一步才真正解释了神经网络的「学习」是怎样发生的。

10.7 几个容易混淆的条件，放在这里说明

- 第一，句号本身不是EOS。训练表中“句号预测结束标记”，要以数据确实追加了EOS为前提；若没有，就忽略最后没有下一目标的位置。
- 第二，标准下一个Token交叉熵训练并不需要真的先抽样错误答案。直接从完整分布取真实目标的概率即可。其他位置仍使用数据中的真实前文。
- 第三，训练可把输入设为t1到t6，标签设为t2到t7；也可输入七项再忽略最后目标。目标错开一位，才是这张训练表的要点。

第 11 章：损失、导数与反向传播，沿原问答逐步推导

[返回目录](#)

这一章保留原对话中从负对数、单参数导数、一次具体更新，再到矩阵和链式法则的整段展开。它本来就是帮助理解训练的关键内容，不再压缩成“用梯度下降”一句话。

11.1 为什么损失函数是 $-\ln p$?

假设模型预测下一 Token 的概率分布为：

候选 Token	预测概率
呼吸困难	0.45
黑便	0.30
呕血	0.15
其他	0.10

这里「黑便」是正确答案。

我们希望用一个数字衡量模型的预测有多差。

最直观的要求是：

- 如果正确答案的概率是 100%，损失应该是 0。
- 如果正确答案的概率很小，损失应该较大。
- 如果正确答案的概率接近 0，损失应该非常大。

函数：

$$L = -\ln p$$

恰好符合这些要求。

例如：

正确答案的概率 p	损失 $-\ln p$
0.01	4.605
0.10	2.303
0.30	1.204
0.50	0.693
0.90	0.105
1.00	0

所以刚才模型的损失是：

$$L = -\ln(0.3) \approx 1.204$$

但这个 1.204 并不会直接告诉模型「应该增加 W_Q 中哪个元素」。

它只是告诉模型：按照当前参数，这一次预测产生了多大的损失。

为什么特别选择对数？其中一个重要原因是概率模型的最大似然原理。假设多个训练目标的条件概率相乘，取负对数就能把乘积转换为求和：

$$-\ln(p_1 p_2 \cdots p_n) = -\sum_{i=1}^n \ln p_i$$

这样便于把多个 Token 的预测误差汇总为训练目标。

11.2 模型怎么知道修改参数的方向？

现在涉及你学过的**导数**。

先不要考虑复杂的矩阵 W_Q 。

假设模型只有一个可调整的参数 w ，而损失是它的函数：

$$L = f(w)$$

我们想找到让 L 尽可能小的 w 。

你知道函数在某一点的导数：

$$f'(w) = \frac{dL}{dw}$$

反映函数在该点附近的变化率。

如果：

$$\frac{dL}{dw} > 0$$

说明在当前位置附近，增加 w 会使损失增加，所以我们应该减小 w 。

反之，如果：

$$\frac{dL}{dw} < 0$$

说明增加 w 会使损失减小，所以我们应该增加 w 。

因此可以统一写成：

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{dL}{dw}$$

其中 η 称为学习率（Learning Rate），控制每次调整的幅度。

这就是梯度下降（Gradient Descent）最基本的原理。

11.3 我们实际计算一次参数更新

为了看清楚导数的作用，假设现在模型只需要在两个选项中预测：

- 正确答案：「黑便」
- 其他答案

我们用一个简单的函数表示模型给「黑便」的概率：

$$p = \frac{e^w}{e^w + 1}$$

这里 w 是一个可训练的标量，代表正确答案相对其他答案的输出分数。它不是 W_Q ，只是为了展示训练机制而构造的简化模型。

损失函数变成：

$$L(w) = -\ln \left(\frac{e^w}{e^w + 1} \right)$$

利用对数性质：

$$\begin{aligned}L(w) &= -\ln e^w + \ln(e^w + 1) \\ &= -w + \ln(e^w + 1)\end{aligned}$$

现在对 w 求导：

$$\begin{aligned}\frac{dL}{dw} &= -1 + \frac{e^w}{e^w + 1} \\ &= -1 + p\end{aligned}$$

因此：

$$\boxed{\frac{dL}{dw} = p - 1}$$

这是一个很漂亮的结果。

当正确答案的预测概率为 0.3 时：

$$\frac{dL}{dw} = 0.3 - 1 = -0.7$$

导数为负，说明增加 w 有助于减小损失。

假设学习率为：

$$\eta = 0.1$$

那么梯度下降：

$$\begin{aligned}w_{\text{new}} &= w_{\text{old}} - 0.1(-0.7) \\ &= \boxed{w_{\text{old}} + 0.07}\end{aligned}$$

也就是说，模型根据导数，自动知道应该把这个参数增加 0.07。

最初 $p = 0.3$, 对应:

$$w_{\text{old}} = \ln \frac{0.3}{0.7} \approx -0.8473$$

更新以后:

$$w_{\text{new}} \approx -0.7773$$

重新计算预测概率:

$$p_{\text{new}} = \frac{e^{-0.7773}}{1 + e^{-0.7773}} \approx 0.315$$

你可以看到:

模型并不需要有人告诉它「把参数增加 0.07」。只需要计算损失对参数的导数, 就能自动找到局部降低损失的方向。

11.4 但 Transformer 的参数是矩阵, 而不是一个数字

现在才回到真正的 Transformer。

我们有:

$$Q = XW_Q$$

假设 W_Q 是一个 512×64 的矩阵。

那么它包含:

$$512 \times 64 = 32768$$

个元素。

我们把这些元素记为：

$$W_Q = \begin{bmatrix} w_{11} & w_{12} & \cdots \\ w_{21} & w_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

模型需要决定这 32768 个数字分别应该增加还是减少多少。

于是我们对每个元素都计算一个偏导数：

$$\frac{\partial L}{\partial w_{ij}}$$

把所有偏导数组成一个与 W_Q 同样大小的矩阵：

$$\frac{\partial L}{\partial W_Q} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \cdots \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

这就是梯度矩阵。

假设其中一小部分为：

$$\frac{\partial L}{\partial W_Q} = \begin{bmatrix} 0.3 & -0.2 \\ -0.5 & 0.1 \end{bmatrix}$$

而学习率为 **0.1**。

那么参数更新量是：

$$-\eta \frac{\partial L}{\partial W_Q} = \begin{bmatrix} -0.03 & +0.02 \\ +0.05 & -0.01 \end{bmatrix}$$

于是：

- 左上角参数减小 0.03。
- 右上角参数增加 0.02。
- 左下角参数增加 0.05。
- 右下角参数减小 0.01。

真实模型可以包含数十亿甚至更多参数，但基本原理仍然是这样。当然，实际训练中通常使用 AdamW 等优化器，而不仅是最简单的梯度下降。

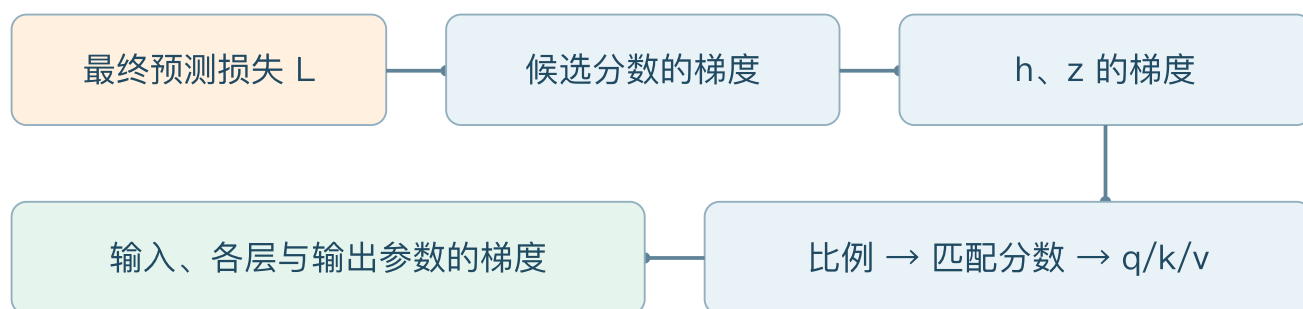


图5：沿计算依赖反向求导，再更新参数；不是把前向过程逆运算还原。

11.5 最关键的问题：损失 L 怎么对 WQ 求导？

这是反向传播真正厉害的地方。

回顾我们学过的前向计算：

$$X \xrightarrow{W_Q} Q$$

然后：

$$Q, K \longrightarrow S \longrightarrow A$$

再与 V 运算：

$$A, V \longrightarrow Z$$

经过其他 Transformer 模块，得到最终隐藏向量 H ，然后通过输出层计算概率 p ，最后得到损失 L 。

注意：这里从 W_Q 到 L ，并不是一个简单函数，而是许多函数的复合。

你学过复合函数求导法则：

如果：

$$y = f(u), \quad u = g(x)$$

那么：

$$\boxed{\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}}$$

这就是链式法则（Chain Rule）。

对于神经网络，原理完全一样。

如果把中间运算简化成一条依赖链，那么：

$$\frac{\partial L}{\partial W_Q} \quad \text{需要沿着} \quad L \rightarrow p \rightarrow Z \rightarrow A \rightarrow S \rightarrow Q \rightarrow W_Q$$

逐步反向计算导数。

真实网络有分支、矩阵和残差连接，因此反向传播还需要汇总不同路径传回来的梯度，但基本数学原理仍然是多元函数的链式法则。

甚至我们可以对最初的：

$$Q = XW_Q$$

直接写出矩阵形式的梯度：

$$\frac{\partial L}{\partial W_Q} = X^T \frac{\partial L}{\partial Q}$$

这里把来自下游所有路径的梯度都汇总到了 $\partial L / \partial Q$ ，并把 X 看成已知输入。

所以只要我们算出了损失相对于 Q 的梯度，就可以进一步算出损失相对于 W_Q 的梯度。

这就是反向传播不是在倒着重新猜参数，而是在沿计算图系统地应用链式法则。

最后把训练过程与推理过程对应起来

阶段	进行的计算
前向传播	用 W_Q, W_K, W_V 等参数处理输入，计算预测概率
损失计算	根据正确 Token，计算 $-\ln p$
反向传播	使用链式法则计算所有参数的梯度
参数更新	使用梯度和优化器调整参数
下一轮训练	使用更新后的参数重新预测

而你实际与 ChatGPT 对话的时候，通常只进行前向计算和 Token 生成，不执行这种针对本次对话的梯度更新。

最重要的结论是：

Transformer 的「学习」不是给某个词手动赋予意义，也不是直接修改注意力权重 A 。

它是通过计算最终预测误差对各个参数的偏导数，不断修改 W_Q, W_K, W_V 等矩阵。随后这些矩阵在新的前向传播中产生新的 Q, K, V ，进而改变注意力分配和最后的预测结果。

而这整个训练机制，从数学上讲，核心就是你已经接触过的多元函数求导、链式法则和梯度下降。

11.6 最后把训练过程与推理过程对应起来

阶段	进行的计算
前向传播	用 W_Q, W_K, W_V 等参数处理输入，计算预测概率
损失计算	根据正确 Token，计算 $-\ln p$
反向传播	使用链式法则计算所有参数的梯度
参数更新	使用梯度和优化器调整参数
下一轮训练	使用更新后的参数重新预测

而你实际与 ChatGPT 对话的时候，通常只进行前向计算和 Token 生成，不执行这种针对本次对话的梯度更新。

最重要的结论是：

Transformer 的「学习」不是给某个词手动赋予意义，也不是直接修改注意力权重 A 。它是通过计算最终预测误差对各个参数的偏导数，不断修改 W_Q, W_K, W_V 等矩阵。随后这些矩阵在新的前向传播中产生新的 Q、K、V，进而改变注意力分配和最后的预测结果。而这整个训练机制，从数学上讲，核心就是你已经接触过的多元函数求导、链式法则和梯度下降。

11.7 为什么选负对数：把原来的依据补成公式链

下面把目标概率、交叉熵与整段训练目标连成完整公式。目标分布 r 就是：真实Token那一项为1，其余候选为0；这叫one-hot。 r_w 表示候选w上的这个数。

若目标编号是 y ，用 one-hot 目标分布 $r_w = \mathbf{1}_{w=y}$ ，交叉熵为：

$$L_i = - \sum_w r_w \ln p_{i,w} = - \ln p_{i,y}.$$

正确目标概率提高，损失严格下降。但仅凭“概率低时罚得多”并不能唯一确定对数形式。

更系统的依据是条件最大似然。对一段已知文本，条件概率链式分解给出：

$$p_{\theta}(t_2, \dots, t_m \mid t_1) = \prod_{i=1}^{m-1} p_{\theta}(t_{i+1} \mid t_{\leq i}).$$

这来自概率链式法则，不要求 Token 独立。最大化这个乘积等价于最小化其负对数，即各位置负对数之和。多个训练样本则按给定训练采样与目标求和或平均。[来源 R2 提供神经网络语言模型背景]

定义有效目标集合 I ，平均损失写为：

$$L = \frac{1}{|I|} \sum_{i \in I} - \ln p_{i,t_{i+1}}.$$

padding 与不计分位置应排除。某些对话微调仅在助手回答位置计损失；这是损失掩码，作用不同于因果注意力掩码。后者控制信息流，前者控制哪些位置计入目标。

第 12 章：输入、内部和输出层怎样一起训练

[返回目录](#)

现在回到你最后的问题：“输出层也得有一个训练过程吗，还是和输入层一起？”下面保留原回答的联合训练说明，并把输入、输出共享参数的含义说清。

嵌入表在原问答记作 W_E ，在补充查表式中记作 E ；二者是同一个角色。词表大小统一记 N ，以免与 Value 矩阵 V 重名。

12.1 整个模型实际上是一个联合训练的系统

还是用：

患者服用华法林后出现_____

正确答案是「黑便」。

模型不会先把输入层训练好，再把 Transformer 训练好，最后单独训练输出层。

在通常的预训练中，它们作为一个整体进行端到端训练（End-to-End Training）。

12.2 从数学上看，所有参数都属于同一个优化问题

我们将模型所有参数写成：

$$\theta = \{W_E, W_Q, W_K, W_V, \dots, W_{\text{out}}\}$$

那么：

$$L = L(\theta)$$

训练目标是找到使损失尽可能小的参数：

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

反向传播时，分别得到：

$$\frac{\partial L}{\partial W_E}, \quad \frac{\partial L}{\partial W_Q}, \quad \frac{\partial L}{\partial W_K}, \quad \frac{\partial L}{\partial W_{\text{out}}}$$

随后更新这些参数。

例如，使用最简单的梯度下降：

$$W_Q^{\text{new}} = W_Q - \eta \frac{\partial L}{\partial W_Q}$$
$$W_{\text{out}}^{\text{new}} = W_{\text{out}} - \eta \frac{\partial L}{\partial W_{\text{out}}}$$

所有参数都根据同一次前向传播产生的损失接受训练信号。

这里的argmin是在写训练希望实现的优化目标，不保证实际算法一定达到全局最小值，甚至不预先保证该最小值在所用参数空间取得。实际训练是在给定资源和算法下尝试降低平均损失。

12.3 但输出层和输入层还有一种非常巧妙的关系

你可能会觉得：

输入层需要把 Token 变成向量：

$$\text{Token} \longrightarrow x_i$$

输出层则要把向量变回 Token 的概率分布：

$$h_i \longrightarrow P(\text{下一个 Token})$$

这两个过程在用途上一个读入Token、一个给Token打分，但不是互为逆函数。

因此，许多语言模型采用一种叫做 **Weight Tying**（权重共享）的设计。

假设词表大小为 N ，模型隐藏维度为 d 。

输入 Embedding 矩阵：

$$W_E \in \mathbb{R}^{N \times d}$$

输出层可以使用：

$$W_{\text{out}} = W_E^T$$

也就是说，**输出层和输入 Embedding 层直接共享同一组参数**。

这样做既能减少模型参数量，也能让输入和输出的 Token 表示使用同一个权重矩阵。

不过需要注意：不是所有 Transformer 语言模型都采用权重共享，有些模型会分别训练这两个矩阵。

12.4 为什么联合训练特别重要？

假设模型对「黑便」的预测概率只有 0.3。

出现这个问题，可能有许多原因：

- 输入层没有形成足够有用的 Token 表示。
- Q、K 的匹配没有很好地整合相关上下文。
- V 没有提供足够有用的信息。
- 后续 Transformer 层没有形成合适的隐藏表示。
- 输出层没有将隐藏表示正确映射到「黑便」的分数。

我们事先并不知道问题主要出现在哪一层。

联合训练的优势，就是不需要人为确定应该修改哪一层。

反向传播根据损失对各参数的梯度，自动计算不同参数的局部调整方向。

当然，这不意味着每次更新都一定能提高所有样本的预测准确率。优化器是在大量训练数据上不断尝试降低总体损失。

所以最准确的理解是：

一次前向传播得到多个位置的预测，同一个汇总损失提供训练信号；反向传播计算输入层、Transformer各层和输出层的梯度，再由优化器更新。它们通常共同学习。

这也解释了为什么单独讨论 W_Q 的语义比较困难：一个参数是否有用，要看它与整个网络中其他参数组合起来，最终能否改善预测结果。

12.5 输入查表与输入层梯度，也用式子接起来

下面的查表公式只是把“按编号取一行”写清楚。 $E_{t_i,:}$ 里冒号表示该行的全部列；one-hot 行向量 u_i 只在编号 t_i 处为1，其他项为0，所以 $u_i E$ 会选出E的那一行。

Tokenizer 把文本变成离散编号。假设“患者”编号为 17，“服用”编号为 92，这两个数之差 75 不表示语义距离。编号用于索引，不直接拿来做语言意义上的加减。

输入嵌入矩阵定义为：

$$E \in \mathbb{R}^{N \times d}.$$

第 w 行是词表中第 w 个 Token 的可训练向量。第 i 个位置先查表得到行向量：

$$e_i = E_{t_i,:} \in \mathbb{R}^{1 \times d}.$$

若用 one-hot 行向量 $u_i \in \mathbb{R}^{1 \times N}$ 表示编号，则 $e_i = u_i E$ 。查表和这个乘法数学上等价；实现无需真的构造巨大 one-hot。

若 $x_i = E_{t_i,:} + p_i$ ，则：

$$\bar{E}_{w,:} = \sum_{i:t_i=w} \bar{X}_{i,:}.$$

可训练的位置向量同样收到对应梯度；固定位置编码不更新。若 $U = E^T$ ，还需加来自输出路径的 $\bar{U}^T$ 。训练更新可以同时影响“输入如何表示”和“输出如何打分”，无需分开进行两个训练过程。

12.6 预测函数、损失函数和优化目标的区别

对第*i*个位置，若加上输出偏置*b*：

$$\ell_i = h_i U + b, \quad p_i = \text{softmax}(\ell_i).$$

下标*i*表示这是第*i*位置作出的下一Token预测。 p_i 是一整组概率，不是单个数；要问“黑便”的概率，需要取这一组中属于“黑便”的那一项。

有时会写：

$$p_{\theta}(\text{黑便} \mid t_1, \dots, t_5).$$

这里*t*₁到*t*₅是已有五块前文，竖线读作“在这些前文已知的条件下”。 θ 代表模型当前全部参数；把它写在*p*的下标，是提醒你换一套参数，预测概率也可能改变。整个式子表示一个数：这套模型给“黑便”分了多少概率。

公式并没有额外执行一步操作，只是把前面讲过的整个预测过程压缩记下来。

对可微标量损失 $L(\theta)$ ，把所有参数展开成向量。由一阶展开：

$$L(\theta - \eta g) = L(\theta) - \eta \|g\|^2 + o(\eta), \quad g = \nabla L(\theta).$$

若 $g \neq 0$ ，充分小的正 η 使损失下降。有限步长不自动满足条件；批次损失下降也不保证所有样本或测试集表现改善。

反向传播是多元链式法则在计算图上的高效应用，求的是当前参数处的局部导数。梯度下降是更新规则；Adam 类方法再使用梯度的历史统计量调整更新。本文数值例子只用最简单的梯度下降。

12.7 这和反向传播有什么关系？

现在逻辑才真正完整。

因为损失函数可以写成：

$$L = L(W_Q, W_K, W_V, \dots, W_{\text{out}})$$

所以我们才能对这些矩阵求偏导。

例如：

$$\frac{\partial L}{\partial W_Q}$$

它的意思是：

如果稍微改变 W_Q 中的某个元素，最终预测「黑便」的损失会如何变化？

虽然 W_Q 并不直接产生最终概率，但它会影响：

$$W_Q \rightarrow Q \rightarrow A \rightarrow Z \rightarrow h_5 \rightarrow \ell \rightarrow P \rightarrow L$$

因此可以通过链式法则，一直把梯度传递回 W_Q 。

这就是反向传播。

而且，输出层的 W_{out} 也需要训练，并不是一开始就知道「哪个隐藏向量应该对应黑便」。

12.8 现在我们可以把整个训练写成一个优化问题

最终，模型希望找到一组参数：

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

这里把 Transformer 和输出层的所有可训练参数统称为 θ 。

更完整地说，是希望在大量训练样本上，使平均的下一个 Token 预测损失尽可能小。

所以三个概念需要明确区分：

概念	数学对象	作用
Transformer	$F_{\theta}(X)$	将输入转化为上下文相关的隐藏表示
预测函数	$\text{softmax}(hW_{\text{out}} + b)$	计算下一个 Token 的概率
损失函数	$-\ln P(\text{正确Token})$	衡量预测结果与正确答案的差异

这里最重要的理解是：Transformer 并不是直接输出文字，而是先计算一个高维向量，再通过同样需要训练的输出矩阵，把它映射到词表上的概率分布。

第 13 章：补算一个小模型，看到梯度真正回到 WQ

[返回目录](#)

原问答已经用一个标量算过更新。但“最终误差怎样真的传到WQ”仍然值得多算一小段。这里保留第二版的可核验小模型作为补充练习：不替换前面的一般公式，也不要求先看完这一章才能继续读生成过程。

13.1 先缩小教具，但不把数字与文字脱开

完整模型参数太多，不适合手算。我们做一个极小的教学模型，只留下“注意力 → 残差 → 输出 → 损失”这条主线。FFN、归一化和位置编码暂时不放进去；因此它不是完整GPT。

为了仍然对应那句填空，这一章专门使用一个只有三项的玩具词表：

简写	这一项代表的文本
A	患者服用华法林后
B	出现
C	黑便

把一整段前文当成一个Token，只是为了缩成两个输入位置。这是本章特设的玩具分词，和前文七块分词不同，也不是说合并以后与原模型计算等价。本章不处理句号或结束标记。

输入是A、B，即“患者服用华法林后出现”；目标是C，即“黑便”。注意力要让“出现”这个位置读取前文A，再由它的最终向量预测C。

13.2 先列出本次所有参数

给三项Token各指定一个二维嵌入。把这张表叫E：

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

三行依次对应A、B、C。于是输入A、B查出 $(1, 0)$ 、 $(0, 1)$ ，按行排列：

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

指定匹配变换和内容变换：

$$W_Q = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

WQ把二维输入变成一维q，WK把二维输入变成一维k。WV在这个初值下不改变向量。q/k只有一个坐标，所以本例dk=1，缩放因子也是1。

再设注意力输出投影WO也为二维单位矩阵，并指定词表输出矩阵：

$$W_O = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$

U三列依次给A、B、C打分。这里没有偏置，E与U也不共享。以上参数都是为了方便核验而指定的起点，不是已经学会医学知识的一套数。

13.3 逐行算 Q、K、V

对第一位置A， $x_1 = (1, 0)$ ：

$$q_1 = 0, \quad k_1 = 1, \quad v_1 = (1, 0).$$

对第二位置B， $x_2 = (0, 1)$ ：

$$q_2 = 1, \quad k_2 = 0, \quad v_2 = (0, 1).$$

为什么第二位置q等于1？就是 $(0, 1)$ 乘WQ那一列，得到 $0 \times 0 + 1 \times 1 = 1$ 。每一个数都来自普通乘加。

现在考察第二位置“出现”。它对前文A的分数是 $q_2 k_1 = 1$ ，对自己B的分数是 $q_2 k_2 = 0$ 。

所以它需要分配的是两个分数 $(1, 0)$ 。Softmax给出：

$$\left(\frac{e}{e+1}, \frac{1}{e+1} \right) \approx (0.731059, 0.268941).$$

为了后面算式短一点，把这两个数暂叫a和c。a不是Token A，c也不是Token C；它们只是两个比例：a约0.731059，c约0.268941，而且a+c=1。

13.4 取内容，再加回原表示

第二位置读到的结果是：

$$z_2 = av_1 + cv_2 = (a, c).$$

再经过WO投影。因为本例WO恰好是单位矩阵，投影不变。与第二位置原输入 $(0, 1)$ 相加：

$$h = x_2 + z_2 W_O = (a, 1 + c) \approx (0.731059, 1.268941).$$

这里h就是本章小模型的最终向量。请注意：第一个坐标约0.731059不是“黑便概率”，第二个坐标也不是另一个Token概率。h仍是给输出层读取的两个数。

第一位置只能读自己，权重恒为1。若同时把两行写出，注意力比例表为：

$$A_{\text{attn}} = \begin{bmatrix} 1 & 0 \\ a & c \end{bmatrix}.$$

加上下标attn是为了把这张表与词表中的Token A区分开。

13.5 终于得到三个候选的概率

把h乘U：

$$\ell = hU = (a, 1 + c, -a) \approx (0.731059, 1.268941, -0.731059).$$

第三项是C、“黑便”的分数。当前参数居然给它负分，但负分没有问题：logits只是相对分数，接下来才变概率。

对三项取指数再归一化，得到：

$$p \approx (0.339660, 0.581626, 0.078714).$$

即A约33.97%，B约58.16%，C“黑便”只有约7.87%。输入中没有C，也不妨碍输出层给它打分。

训练目标是C，所以损失为：

$$L = -\ln p_C \approx 2.541928.$$

这一次概率不是凭空指定的30%。它从输入、投影、比例、内容、残差与输出矩阵逐步算出来。

13.6 先看损失怎样要求输出分数改变

对“Softmax加负对数损失”，可以推导出一个非常实用的结果：

$$\frac{\partial L}{\partial \ell_j} = p_j - \mathbf{1}_{j=C}.$$

符号 $\mathbf{1}_{j=C}$ 的意思是：如果当前候选j正是目标C，取1，否则取0。推导在第15章，不需要凭记忆接受。

代入本例，三个分数的梯度约为：

$$(0.339660, 0.581626, -0.921286).$$

这说明：若能独立改这三个分数，降低A、B分数，提高C分数，会在当前位置附近降低损失。但它们并非独立参数，而是由h和U算出，所以还要继续求到h与U。

13.7 再看误差怎样传到“读多少”

U第一行是 $(1, 0, -1)$ 。h第一个坐标增加一点，会让A分数同量增加、C分数同量减少。因此L对h第一坐标的导数是：

$$0.339660 - (-0.921286) = 1.260945.$$

U第二行是 $(0, 1, 0)$ ，所以h第二坐标只直接影响B分数，对应导数是0.581626。

于是h的梯度约为 **(1.260945, 0.581626)**。由于本例的残差直连和WO初值很简单，这份梯度也沿z2向前传。

而z2是 $av_1 + cv_2$ 。若先把a、c当成两个独立变量，增加a就沿 **(1, 0)** 方向改变z2，对损失的影响率是1.260945；增加c则沿 **(0, 1)** 方向改变，影响率是0.581626。

所以损失对两项混合比例的梯度，是这两个数。到这里，我们已经从“C预测得太低”，追到了“怎样的内容混合更能降低损失”。

13.8 比例互相牵制，不能各自随便改

a与c经过Softmax产生，必须相加为1。如果提高第一个匹配分数，a会增加，c会相应减少。因此要把这两种影响同时算进去。

对分数 (s_1, s_2) ，两项Softmax中有：

$$\frac{\partial a}{\partial s_1} = ac, \quad \frac{\partial c}{\partial s_1} = -ac.$$

把两条影响加起来：

$$\frac{\partial L}{\partial s_1} = ac(1.260945 - 0.581626) \approx 0.133562.$$

同理，第二个分数的梯度约为-0.133562。这里终于出现了“匹配分数该往哪个方向变”的数值。

注意，它不是简单地说“让看起来有关的前文权重越来越高”。这个方向取决于当前V内容、输出U和整个损失，不是人为指定的语义关系。

13.9 现在才到 WQ

本例q2来自第二行输入乘WQ，而两个分数是：

$$s_1 = q_2 k_1 = q_2, \quad s_2 = q_2 k_2 = 0.$$

求 q_2 的局部影响时，把当前 k 视为给定，得到：

$$\frac{\partial L}{\partial q_2} = 0.133562 \times 1 + (-0.133562) \times 0 = 0.133562.$$

再看WQ第二个元素，记为 w 。因为输入第二行是 $(0, 1)$ ，有

$q_2 = 0 \times W_{Q,1} + 1 \times w = w$ ，故损失对 w 的导数也是0.133562。

取学习率0.1，它从1更新为：

$$w_{\text{new}} = 1 - 0.1 \times 0.133562 \approx 0.986644.$$

至此，“预测错了怎样影响WQ中的一个数”已经走通：输出误差 $\rightarrow h \rightarrow$ 内容比例 $\rightarrow$ 匹配分数 $\rightarrow q \rightarrow WQ$ 。

13.10 不是只改这一个数，而是共同更新

同样的链式法则也能算出WK、WV、WO、U和输入E的梯度。第15章列出矩阵规则，配套程序保存全部24个参数元素的结果。

例如，U第二行、C列最初为0。它的梯度为：

$$\frac{\partial L}{\partial U_{2,C}} = h_2(p_C - 1) \approx 1.268941 \times (-0.921286) \approx -1.169057.$$

注意这里 h_2 是本章二维 h 的第二个坐标，不是“第2位置的向量”。因为本节已经固定最后位置，只对向量内部坐标编号。写明这一点，是为了不把位置下标与坐标下标混用。

学习率0.1时，这个输出参数从0增加到约0.116906。输入表示的数也会更新，Q/K/V的各变换同样参与。

所有梯度先基于同一套旧参数算出，再一起执行更新。不能先改U，再拿新U去拼接旧前向的梯度，那是在混用不同计算点。

13.11 全部更新后，再跑一次看看

使用学习率0.1联合更新所有可训练矩阵，再从查表开始重新计算：

观察的量	更新前	更新后
“出现”读取前文A的比例	0.731059	0.713748
输出“黑便”的概率	0.078714	0.171807
正确目标损失	2.541928	1.761382

目标概率从约7.87%提高到17.18%，损失降低。但“黑便”还不是概率最大的候选，一次训练不必立刻答对。

读取前文A的比例反而下降了。说明训练可以通过同时改变输入表示、传递内容和输出打分来进步，不是“把每条看起来相关的注意力都加大”。

这个小模型共有24个参数元素。解析梯度与中心差分逐项核对，最大绝对误差小于 3×10^{-11} 。这支持本例计算与程序实现一致，不等于证明真实模型学到了医学知识。

想核查数值，可以打开[全部参数和梯度](#)或[运行程序](#)。正文显示值经四舍五入，程序始终使用完整精度。

第 14 章：每生成一个 Token 后，接下来发生什么

[返回目录](#)

现在参数怎样训练已经解释完，再回到使用模型。这里只发生新的前向计算与选择，通常不针对这次对话更新参数。

14.1 选出一个Token后，把它放回前缀

比如模型预测出了一个新的 Token t_4 。

那么输入序列就由：

$$(t_1, t_2, t_3)$$

变成：

$$(t_1, t_2, t_3, t_4)$$

模型再计算第四个位置的最终隐藏向量 h_4 ，预测第五个 Token。

实际推理通常使用 KV Cache（键值缓存），保存前面 Token 在各层已经算出的 K、V，避免每次生成新 Token 时都重复计算整个前缀。

14.2 为什么使用最后一行

每一行 h_i 受训练目标约束，用于预测当前位置之后的 Token。前面各行预测的是已经出现在输入里的后续位置；最后一行对应尚未知的下一位置。因此标准续写取最后一行词表分布。

它能依赖整个允许前缀，不意味着它无损保存了全部原文，也不是把句子压缩成一个独立、永久不变的句子向量。下次生成时，新位置仍可读取历史各层 K/V。

14.3 概率怎样变成选择

贪心解码： 取概率最大编号。它最大化当前一步的概率，不保证整个输出序列的联合概率最大。

采样： 按概率随机选。即使概率较小也可能被选中。Softmax 本身只给分布，不执行抽样。

温度： 对 $T > 0$ 使用 $\text{softmax}(\ell/T)$ 。较小 T 通常使分布更尖锐，较大 T 更平缓； $T = 0$ 不直接代入除法，而可讨论趋零极限。Top-k 或 top-p 还会截断候选并重新归一化。这些是解码规则，不是本次把模型训练了一遍。

14.4 KV Cache为什么能复用旧结果

在因果网络中，追加新 Token 不改变旧位置允许看到的前缀。参数固定、位置定义与计算配置不变时，旧位置各层 K/V 无需重新计算。保存它们，就能让新位置的 Q 读取旧 K/V 加上新位置自己的 K/V。

这依赖第8章的因果归纳性质。缓存要逐层保存，不是只保存最终向量。旧位置的中间表示在多层网络中早已参与构造后续所需的 K/V，并没有“只用最后一行所以前面都没用”。

窗口截断、位置重映射、参数改变等会影响缓存可复用条件。缓存通常以额外内存换取减少重复计算，也不消除新 Token 读取历史内容的代价。

14.5 怎样停止并变回文字

可因选中 EOS、达到长度上限或触发应用停止规则而结束。句号不自动等于结束。输出 Token 序列还要由 tokenizer 的解码规则变回可见文本，Token 与完整汉字、完整词语并非一一对应。

普通生成过程保持参数不变；上下文和缓存改变。把用户信息写进本次上下文，可以改变后续输出，并不等于本次进行了梯度训练。

14.6 训练、上下文适应与应用记忆

参数训练：损失反传后更新参数。上下文适应：参数固定，用本次上下文改变激活与预测。工程记忆：应用存储信息并在未来检索或插入上下文。三者可配合，但不能互相替代为同一个概念。

第 15 章：需要时回查的矩阵公式与补充推导

[返回目录](#)

下面保留第一版与第二版里有用的补充公式，供你在主文某一步想核查时使用。原对话直接用到的 Softmax、方差、负对数与单参数更新，已经在对应章节展开；这里不把那些解释抽走。

本章沿用补充推导里的U记输出矩阵，与主文Wout同义。例子里的WQ、WK等仍是同一组参数，不新增一套模型。

15.1 Softmax导数，从商法则逐项推出

设 $D = \sum_r e^{s_r}$, $a_j = e^{s_j}/D$ 。若对自己的分数 s_j 求导，商法则给出：

$$\frac{\partial a_j}{\partial s_j} = \frac{e^{s_j} D - e^{2s_j}}{D^2} = a_j(1 - a_j).$$

若对别的分数 s_k 求导，只有分母受影响：

$$\frac{\partial a_j}{\partial s_k} = -\frac{e^{s_j} e^{s_k}}{D^2} = -a_j a_k, \quad k \neq j.$$

这解释了“提高一项，会挤掉其他项的比例”。若一项已接近1、其他项接近0，上述导数都可能很小，所以过尖的注意力分配可能不易调整。缩放分数是帮助控制这种情况的方法，不是所有训练问题的保证解法。

为了数值稳定，所有分数可先减去最大值。Softmax不变，因为分子分母中的公共指数因子会约去。这能避免直接对很大的分数取指数。

15.2 输出Softmax与交叉熵合起来怎样求导

对目标编号 y ，输出分数是 $\ell_1, \dots, \ell_N$ ，Softmax概率为 p 。将目标概率代入负对数：

$$L = -\ln \frac{e^{\ell_y}}{\sum_r e^{\ell_r}} = -\ell_y + \ln \sum_r e^{\ell_r}.$$

对第 j 个分数求导：第一项在 $j=y$ 时给 -1 ，否则给0；第二项给 $e^{\ell_j} / \sum_r e^{\ell_r} = p_j$ 。所以：

$$\frac{\partial L}{\partial \ell_j} = p_j - \mathbf{1}_{j=y}.$$

因此即使输出对错误候选很自信，目标项梯度仍可能接近-1。不能仅根据Softmax本身的导数很小，就推断“输出错误且自信时一定无法学习”。合并损失后要看完整导数。

15.3 内容混合、Q/K/V与矩阵反传的完整连接

用上横线表示“损失对这个量的梯度”，比如 $\bar{Z} = \partial L / \partial Z$ 。梯度与原量同形状。对矩阵X，它的含义由微分关系规定：

$$dL = \sum_{ir} \bar{X}_{ir} dX_{ir}.$$

对输出 $\ell = hU + b$ ：

$$\bar{U} = h^\top \bar{\ell}, \quad \bar{h} = \bar{\ell} U^\top, \quad \bar{b} = \bar{\ell}.$$

第一式为什么是外积？因为第j分数是 $\ell_j = \sum_r h_r U_{rj} + b_j$ ，故 $\partial L / \partial U_{rj} = h_r \bar{\ell}_j$ 。多位置时将各贡献累加；若损失取平均，梯度也带同样的平均系数。

对于本版第13章定义的简化模块 $R = X + ZW_O, Z = AV$ ，给定 $\bar{R}$ ：

$$\bar{W}_O = Z^\top \bar{R}, \quad \bar{Z} = \bar{R} W_O^\top, \quad \bar{A} = \bar{Z} V^\top, \quad \bar{V} = A^\top \bar{Z}.$$

对允许的注意力分数，用Softmax各导数汇总：

$$\bar{S}_{ij} = a_{ij} \left(\bar{A}_{ij} - \sum_{r \in J_i} a_{ir} \bar{A}_{ir} \right).$$

被禁止位置梯度为0。然后：

$$\bar{Q} = \bar{S} K / \sqrt{d_k}, \quad \bar{K} = \bar{S}^\top Q / \sqrt{d_k},$$

$$\bar{W}_Q = X^\top \bar{Q}, \quad \bar{W}_K = X^\top \bar{K}, \quad \bar{W}_V = X^\top \bar{V}.$$

输入X通过残差与三条投影路径同时影响损失，故收到的梯度要相加：

$$\bar{X} = \bar{R} + \bar{Q}W_Q^T + \bar{K}W_K^T + \bar{V}W_V^T.$$

这些式子针对本节明确的小模块，不冒充整个完整GPT的反向公式。完整网络还要沿FFN、归一化和更多层继续用链式法则。

如果X来自嵌入查表，某个词表行收到的梯度，就是本次所有使用该行的位置梯度之和。没有用到的行不从输入路径收到梯度；重复Token的贡献相加。

15.4 小模型其余参数的梯度

同一次前向得到的梯度约为：

$$\bar{W}_Q = \begin{bmatrix} 0 \\ 0.133562 \end{bmatrix}, \quad \bar{W}_K = \begin{bmatrix} 0.133562 \\ -0.133562 \end{bmatrix},$$

$$\bar{W}_V = \bar{W}_O = \begin{bmatrix} 0.921825 & 0.425203 \\ 0.339120 & 0.156423 \end{bmatrix},$$

$$\bar{U} = \begin{bmatrix} 0.248311 & 0.425203 & -0.673514 \\ 0.431008 & 0.738049 & -1.169057 \end{bmatrix},$$

$$\bar{E} = \begin{bmatrix} 1.055387 & 0.425203 \\ 1.466503 & 0.871611 \\ 0 & 0 \end{bmatrix}.$$

C“黑便”的嵌入行没作为输入出现，所以这里E第三行梯度为0；但U的C输出列有非零梯度。输入角色和输出角色是不同路径。WV与WO梯度相等，则是本例单位矩阵初值造成的特殊结果。

配套程序用 $[L(\theta + \epsilon e_r) - L(\theta - \epsilon e_r)]/(2\epsilon)$ 对24个参数逐项做中心差分， $\epsilon = 10^{-5}$ ， e_r 表示只在第r个参数位置改动。差分有数值误差，是对解析推导的核验，不代替推导本身。

15.5 FFN与LayerNorm的公式及反向计算

主线只解释各模块用途，这里明确一种可计算的排列。令 X 是本层输入，LN1/LN2为两次归一化，MHA是多头注意力：

$$T = X + \text{MHA}(\text{LN}_1(X)),$$

$$Y = T + \text{FFN}(\text{LN}_2(T)).$$

这叫Pre-LN，因为归一化放在各子层前面。 Y 进入下一层。最后还可以做最终归一化，得到用于输出的 H 。原始2017年Transformer使用子层与残差相加之后再归一化的Post-LN排列，不要混成同一精确配置。[R1]

对一行 x ，LayerNorm先定义均值 μ 和平方偏差平均 v ：

$$\mu = \frac{1}{d} \sum_r x_r, \quad v = \frac{1}{d} \sum_r (x_r - \mu)^2.$$

再逐坐标计算：

$$\text{LN}(x)_r = \gamma_r \frac{x_r - \mu}{\sqrt{v + \varepsilon}} + \beta_r.$$

ε 为小正数，避免分母为0； γ/β 是可训练的逐坐标缩放和偏移。这里 v 是归一化定义的平方偏差平均，不是除以 $d-1$ 的统计样本方差。因为有 ε 及 γ/β ，不能声称最后输出严格均值0、方差1。[R3]

FFN可设 $B = XW_1 + b_1, C = \sigma(B), Y = CW_2 + b_2$ 。给定 $\bar{Y}$ ，按乘积与链式法则：

$$\bar{W}_2 = C^\top \bar{Y}, \quad \bar{C} = \bar{Y} W_2^\top, \quad \bar{B} = \bar{C} \odot \sigma'(B),$$

$$\bar{W}_1 = X^\top \bar{B}, \quad \bar{X} = \bar{B} W_1^\top.$$

符号 $\odot$ 表示逐坐标相乘；偏置梯度对位置相加。ReLU在0不可微，自动微分实现会使用约定值；平滑激活按自己的导数计算。

LayerNorm也可逐步求导。令 $s = \sqrt{v + \varepsilon}$ 、 $\hat{x} = (x - \mu)/s$ ，已知输出梯度 g ，令 $u = g \odot \gamma$ ，可整理为：

$$\bar{x} = \frac{1}{s} [u - \text{mean}(u)\mathbf{1} - \hat{x} \text{mean}(u \odot \hat{x})].$$

mean沿这一个位置的坐标取平均。用 $d\mu = \text{mean}(dx)$ 、 $dv = 2 \text{mean}((x - \mu) \odot dx)$ 与 $ds = dv/(2s)$ 代入 $d\hat{x}$ ，即可推出。 γ/β 梯度分别为 $g \odot \hat{x}$ 与 g ，多位置时相加。完整网络就是把这些局部规则按计算图连接起来。

15.6 位置旋转与输入输出共享

没有位置编码、也没有位置相关掩码的全注意力，对输入位置的重排会相应重排输出，这叫置换等变。因为Q/K/V都随行重排，分数表随行列重排，每行Softmax与内容组合也随之重排。

因果掩码本身已经引入方向约束，不能把这个“没有掩码”的论证原封不动套给因果网络。显式位置信息还可提供位置或距离结构。

RoPE的一种理解是：先给不同位置的Q/K施加不同角度的二维旋转，再点积。本段专门采用列向量，写成 $\tilde{q}_i = R_i q_i, \tilde{k}_j = R_j k_j$ 。那么：

$$\tilde{q}_i^T \tilde{k}_j = q_i^T R_i^T R_j k_j.$$

若一个二维块的旋转角是位置乘固定频率 ω ，则 $R_i^T R_j = R((j - i)\omega)$ 。相对位置差 $j-i$ 便进入点积分数。换回主文行向量约定时，相应转置即可。[R4]

最后，输入嵌入 E 为 $N \times d$ ，输出 U 为 $d \times N$ ，可以有意设置 $U = E^T$ ，这叫权重共享。此时输出是拿 h 与各Token嵌入点积，而不是对 E 求逆。共享参数收到输入、输出两条路径的梯度之和；不共享时分别训练。[R1、R6]

15.7 残差的直接梯度路径

残差形式是 $y = x + F(x)$ 。相加要求形状相同。它提供一条不必经过 F 的直接路径，而不是保证原信息永远不丢失，例如 $F(x) = -x$ 会得到零。

在列向量的 Jacobian 记号下：

$$J_y = I + J_F.$$

所以梯度有一条恒等路径。这有助于解释深层优化，但不保证 $I + J_F$ 永远良好，也不构成“绝不会梯度消失”的证明。

15.8 学习两套Q/K投影，比直接点积多了什么

用行向量写：

$$q_i k_j^T = x_i W_Q W_K^T x_j^T.$$

中间矩阵 $B = W_Q W_K^T$ 可训练，秩至多 d_k ，通常不要求对称。若强制 $W_Q = W_K$ ，那么 B 变成 $W_Q W_Q^T$ ，额外受到对称半正定约束。

一个小例子： $x_a = (1, 0), x_b = (0, 1)$ ，取 $W_Q = (1, 0)^T, W_K = (0, 1)^T$ ，则a读b时点积为1，b读a时为0。原来两个输入向量的直接点积却是0。可见投影后的匹配不是原始向量相似度的简单重复。

15.9 原始架构和GPT不要混为一谈

架构	主要信息流	常见任务形式
Encoder-only	输入位置双向互读	表示、分类、掩码预测等
Decoder-only	因果自注意力	自回归续写
Encoder-Decoder	编码器双向；解码器因果并读取编码结果	条件序列生成，如翻译

编码器—解码器的交叉注意力中，Q 来自解码端，K/V 来自编码端，长度可以不同，注意力矩阵是目标长度乘源长度。它不是把同一个 X 投影三次的 self-attention。原始 Transformer 属于第三类；本笔记训练与生成主线研究第二类。[来源 R1、R5]

15.10 关于能力，哪些是机制解释，哪些还需证据

训练目标奖励能预测文本的数据处理方式。有些预测需要句法、指代、事实关联或多步依赖；模型若学到可复用特征，可以在多个样本上减少损失。输入嵌入、内容混合和输出层共同适应，使隐藏向量成为后续计算可利用的表示。

这是训练机制与经验结果之间的解释，不能由矩阵公式单独推出“必然具有可靠推理能力”。表达能力、优化是否找到好参数、数据是否提供信号、泛化是否可靠，是不同问题。

较大注意力系数表示该头对某个 Value 使用了较大系数；它不直接量化最终答案受该词影响多少。Value 的方向与大小、其他头、投影、残差和后续非线性都会改变最终效果。

“华法林与黑便有关”可以作为教学情境，但不能在未测真实模型时声称某个头已经学到这个关系。第13章甚至展示了损失降低时某条注意力比例下降的情况。

对数损失以数据实际提供的后续为目标，不直接判断命题真伪。一段错误文本也能成为训练样本；降低该样本损失不等于提高事实可靠性。模型有能力拟合复杂关系，也可能依赖表面共现、偏差或记忆。

所以本文能严格建立的是“定义后的模型怎样计算、梯度怎样产生、给定例子怎样更新”；对现实模型理解、推理、可靠性的判断，需要额外实验与证据。

第 16 章：按原来的问题回查，以及来源与检查说明

[返回目录](#)

这里按原来的提问顺序回查本版对应位置。正文按理解所需的先后重新排列，回查表仍保留你的提问链。

16.1 原始16轮问题的去向

原来的问题	本版位置
1 用我会的知识解释架构	输入与任务、整个Block
2 点积转权重这里太快	保留原来的两步Softmax展开
3 为什么64项标准差是8	正负1例子与一般推导
4 为什么方差等于平方的期望	方差定义紧接在推导前
5 dk到底是什么	先定义坐标数再解释缩放
6 这一步是在算WQ还是q	规则、本次结果与三个阶段
7 为什么又乘V	匹配比例与传递内容
8 这有什么高级的	动态混合和多层复合
9 x变成z有什么意义	上下文表示与后续用途
10 一句话输入，训练还是输出	整句矩阵和阶段区别、同句对照
11 继续：为何用最后一行预测	最后位置与输出层、生成循环
12 没有M就可以读所有位置吧	双向与因果约束
13 用这句话展示训练和输出	原完整示例
14 解释损失、偏导与参数更新	原单参数推导和矩阵解释、补算WQ
15 还得有一个预测函数吧	h到分数到概率
16 输出层单独训练还是一起	联合目标、共享参数与输入梯度

16.2 这一版怎样保留原文，怎样作修正

本版以原分享问答的可见正文为底稿，保留其中逐步解释、例子和推导，主要调整顺序、统一符号、补必要衔接。相同公式的重复出现可以合并，但不以“容易理解”为理由删除推理环节。

修正的地方主要是：行向量与列向量约定、固定教学分词、苹果例子对未来词的依赖、句号与EOS的区别、权重共享不是求逆，以及示意分数不能当作真实模型测量。注意力比例也不直接等于最终答案的因果贡献。

原对话有几处交互图。本文把它们改为嵌入的静态图或明确表格，正文公式与解释保留；原始页面和可见正文仍一并存档。

第一版增加的矩阵反传与归一化公式留在补充章，不要求主线先建立一整套新理论。第二版的小模型也作为补充核验，而不代替原来易懂的单参数例子。

16.3 与你已有数学知识的对应，保留原来的整理

你已掌握的数学概念	Transformer 中对应的结构
向量与坐标	Token Embedding
矩阵乘法、线性变换	Q、K、V 投影
向量点积	Query–Key 匹配分数
凸组合、凸包	Attention 对 Value 的加权整合
矩阵按行运算	Softmax 权重归一化
多种坐标投影	Multi–Head Attention 的部分直觉
非线性函数	FFN 中的激活函数
复合映射	多个 Transformer Block 堆叠

你已掌握的数学概念	Transformer 中对应的结构
函数极值、偏导数	损失函数和梯度优化

最后再给你一个更抽象、但很重要的理解：

你以前研究的几何变换，大多是在研究一个已经给定的映射 T ，例如 $T(x) = Qx$ ，并分析它的性质。

而在 Transformer 中，我们考虑的是一个参数化映射：

$$F_{\theta} : \mathbb{R}^{n \times d} \longrightarrow \mathbb{R}^{n \times d}$$

它的参数 θ 包含众多矩阵，而这些矩阵不是由人事先指定的，而是通过训练数据优化得到的。

其中注意力机制还有一个非常特别的性质：它的加权矩阵 $A(X)$ 也是由输入 X 动态决定的。

所以注意力部分可以抽象成：

$$Z = A(X)XW_V$$

这里 $A(X)$ 的每一行都是一组凸组合系数。

这和普通固定矩阵变换 $Y = XW$ 最大的区别之一，就是注意力的混合系数本身依赖于输入。

如果只记住一个数学本质，我希望是这个：

Transformer 利用可学习的线性投影和点积，为每个 Token 动态构造一组凸组合系数，让不同 Token 交换信息；然后通过非线性变换、残差连接和多层复合，形成复杂的语言表示。

你已经理解了其中最重要的线性代数部分。接下来最值得深入的其实是：**为什么 $A(X)XW_V$ 这样一种看似简单的向量运算，经过多层复合之后，竟然能够表现出推理和语言理解能力？** 这就涉及非线性、表示学习和模型训练的本质了。

16.4 原始来源与参考资料

分享来源：[解释Transformer架构](#)。本地可查看原对话正文与读取核对报告。共16次提问，会话链连续；另有一条被源页面隐去的插件输出，不能声称恢复其内容。

数学基础使用矩阵乘法、指数与对数求导、多元链式法则、概率链式法则、期望与有限二阶矩下的方差性质。本文展开与问题直接有关的依据，不从实数构造或测度论重建这些基础。

- **R1** Vaswani等，[Attention Is All You Need](#)：缩放点积、多头、原始编码器—解码器与FFN结构。
- **R2** Bengio等，[A Neural Probabilistic Language Model](#)：概率语言模型与分布式表示背景。
- **R3** Ba等，[Layer Normalization](#)：归一化定义。
- **R4** Su等，[RoFormer: Enhanced Transformer with Rotary Position Embedding](#)：旋转位置编码。
- **R5** Devlin等，[BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)：双向编码器与不同训练目标。
- **R6** Press与Wolf，[Using the Output Embedding to Improve Language Models](#)：输入输出共享参数。

正文与公式可单文件离线阅读，以上网络来源和本地资料链接只是补充。阅读索引见[全部笔记](#)；历史版本入口也在那里保留。