Transformer

从零开始的自学教材

面向会一点 Python 的深度学习初学者

从数学直觉与手算，到模型实现、实验设计与论文阅读



每个模块更新表示；输出头把隐藏特征变成候选 token 的分数。

阅读顺序 基础 → 注意力 → 架构 → 实现 → 实验 → 论文

配套材料 可编辑正文 • 标准库手算 • PyTorch 小型语言模型

2026年9月29日

目录

点击目录或使用 PDF 书签跳转。章内练习后附参考答案；实验按资源需求分级。

第0章 如何使用这本教材 6

0.1 学习时要同时走三条线 6

0.2 章节路线 6

0.3 本书约定 6

第1章 先理解模型究竟在学习什么 8

1.1 从一条可以写出来的规则开始 8

1.2 一次训练步骤的五个动作 8

1.3 神经网络增加了什么 8

1.4 语言模型的输入和目标 9

1.5 训练集 验证集与测试集 9

1.6 练习与答案 9

第2章 读懂向量矩阵概率和梯度 10

2.1 数字怎样组织成张量 10

2.2 点积是怎样得到一个分数的 10

2.3 矩阵乘法让许多点积一起进行 10

2.4 线性层就是学一个坐标变换 11

2.5 softmax 把分数变成一组权重 11

2.6 交叉熵为什么鼓励正确答案 11

2.7 链式法则把责任传回去 12

2.8 练习与答案 12

第3章 文字表示与序列建模 13

3.1 文本先经过分词与编号 13

3.2 嵌入是可训练的查表 13

3.3 顺序与上下文为什么重要 13

3.4 循环模型怎样处理序列 14

3.5 注意力怎样改变信息路径 14

3.6 Transformer 的总体计算路线 14

3.7 第一次实践前的准备 15

3.8 练习与答案 15

第4章 把注意力真正算一遍 16

4.1 学习目标与一个具体问题 16

4.2 从一个位置到全部位置 16
4.3 固定小矩阵 从头手算 17
4.4 用形状检查防止“看懂却写错” 18
4.5 为什么除以 √d_k 18
4.6 softmax 的数值稳定与参考实现 19
4.7 参数究竟从哪里学来 19
4.8 小练习 答案与误解纠正 20

第5章 多头注意力 批处理与遮罩 21

5.1 学习目标与引例 21
5.2 多头不是把一句话分成几段 21
5.3 四个轴的完整旅行 21
5.4 为什么必须遮住未来 22
5.5 padding 遮罩与损失遮罩解决不同问题 23
5.6 两个数值陷阱 整行遮挡与负无穷乘法 23
5.7 算力和显存为什么随长度增长很快 24
5.8 小练习 答案与误解纠正 24

第6章 把零件装成一个 Transformer 层 26

6.1 学习目标与本章路线 26
6.2 前馈网络 在每个位置内部加工特征 26
6.3 残差连接 保留一条直接通路 27
6.4 LayerNorm 与 RMSNorm 在哪个维度归一化 27
6.5 Post-LN 与 Pre-LN 只移动位置也会改变网络 28
6.6 为什么还要告诉模型顺序 28
6.7 正弦位置编码 不同速度的波形 29
6.8 RoPE 把位置放进 Q K 的旋转 29
6.9 小练习 答案与误解纠正 30

第7章 原始 encoder-decoder 完整走完一次翻译 31

7.1 学习目标与一条教学样本 31
7.2 encoder 先把源序列变成上下文表示 31
7.3 decoder 每层有三项主要计算 32
7.4 右移目标 输入与答案差一格 32
7.5 teacher forcing 为什么可以同时算很多位置 33
7.6 从隐藏向量到词表概率 33
7.7 真正翻译时 目标未来还不存在 34
7.8 一个检查训练代码的五步方法 34
7.9 小练习 答案与误解纠正 34

第8章 三条架构路线与 token 的真实面貌 36

8.1 学习目标 把结构 目标 用途分开 36
8.2 encoder-only 与 BERT 利用两侧上下文 36
8.3 decoder-only 与 GPT 在一个序列中继续写 36
8.4 encoder-decoder 与 T5 把任务写成文本到文本 37
8.5 一张架构对照表 37
8.6 token 不一定是一个词 也不一定是一个汉字 37
8.7 四种经常被混淆的“并行” 38
8.8 从任务出发选择第一个练习 39
8.9 小练习 答案与误解纠正 39

第 9 章 从零实现一个小型 Transformer 40

9.1 我们究竟要造什么 40
9.2 先把数据做对 文档 词表 窗口 40
9.3 嵌入与位置 先给每个字符一张数值名片 41
9.4 多头注意力 按形状读懂实现 41
9.5 因果遮罩 提前封住偷看答案的路 42
9.6 残差 归一化与前馈网络 42
9.7 从隐藏表示得到下一字分数 43
9.8 用性质检查 胜过只看“能运行” 43
本章练习 43

第 10 章 让模型学习 并让它逐字生成 44

10.1 交叉熵 把正确答案的概率变成学习信号 44
10.2 一次训练更新 究竟发生了什么 44
10.3 为什么训练可以一次处理全部位置 45
10.4 评估要同时处理模式 梯度和数据 45
10.5 按顺序运行 先流程 再学习 再续写 45
10.6 生成 模型给概率 程序决定怎么选 46
10.7 KV cache 为什么能加速 又为什么占内存 46
10.8 看训练曲线时 先提出可检查的解释 47
本章练习 47

第11章 从原始 Transformer 走向现代模型 48

11.1 先把改动放回正确的位置 48
11.2 RoPE 给查询和键加上旋转 48
11.3 RMSNorm 与 SwiGLU 注意力之外同样有设计空间 48
11.4 KV cache 已经读过的历史不必反复重算 49

11.5 MQA 和 GQA 让多个查询头共享键和值 49
11.6 FlashAttention 公式相同 搬运方式不同 50
11.7 LoRA RAG 蒸馏 量化分别改变什么 50
11.8 ViT 多模态与对话模型 50

第12章 十个分级实验 从看懂数字到独立研究 52

12.1 先学会做可解释的实验 52
实验1 亲手追踪一次注意力 入门 CPU 52
实验2 用扰动检测“偷看未来” 入门 CPU 52
实验3 让小模型记住一个固定批次 基础 CPU 53
实验4 训练一个字符语言模型 基础 CPU 53
实验5 采样改变了什么 基础 CPU 53
实验6 位置表示的有无与替换 进阶 需扩展 54
实验7 缓存路径和完整路径是否等价 进阶 需扩展 54
实验8 一次只替换一个模块 进阶 需扩展 54
实验9 普通注意力与优化实现 选做 GPU性能部分 54
实验10 给已训练小模型做 LoRA 适配 选做 CPU或小GPU 55
12.2 一页实验报告模板 55

第13章 推荐论文 按问题阅读 而不是按热度收藏 56

13.1 三遍阅读法 56
13.2 必读 建立完整主线 56
13.3 进阶 理解模块与计算成本 56
13.4 选读 选择一个方向继续深入 57
13.5 配套官方学习资源 58

附录A 中英术语速查 59

附录B 八周学习建议 61

附录C 综合练习与参考答案 62

附录D 一次真实训练记录与曲线解读 63

D.1 运行条件与已完成的检查 63
D.2 记录的数据 63
D.3 怎样读这两条曲线 64

第0章 如何使用这本教材

你已经会一点 Python，接下来要学的是：怎样把文字变成数字，怎样让一个模型从例子中调整参数，以及 Transformer 怎样在一段序列里交换信息。本书把这些问题串成一条能够动手验证的路线。读完核心章节后，你应能手算一次注意力、画出各层的输入输出形状、解释训练和生成的区别，并修改一个小型语言模型。

这本书以自然语言为主线。原始 Transformer 的编码器与解码器都会讲；配套程序选择结构更紧凑的 decoder-only 字符模型。现代大模型中的 RoPE、KV cache、FlashAttention、LoRA 和 RAG 放在后半部分。它们帮助你读后续论文，但不能替代前面的基本功。

0.1 学习时要同时走三条线

第一条线是直觉：每个模块解决什么问题。第二条线是计算：输入是什么形状，数值怎样变成输出。第三条线是证据：通过一个小实验判断自己的理解是否正确。例如，“因果掩码防止看见未来”只是一句话；把输入后半段改掉，验证前半段输出不变，才是能够检查的理解。

每章先看学习目标，再读例子，最后完成练习。第一次阅读不必记住每个英文缩写。遇到公式时先找输入、输出和求和的方向。遇到代码时先写下 shape，再看具体语句。读不懂一个公式，常常是某个下标的含义还没说清，而不是数学能力不够。

建议备一个学习记录，分为“我能解释的”“我能算出来的”“我能复现的”。每次学习结束，只写三五行。把“看懂了”改成“给定三个向量，我能算出第一行注意力权重”，你的进展就会更容易判断。

0.2 章节路线

阶段	章节	完成标志
预备基础	第1至3章	会读张量形状，理解损失与参数更新
核心机制	第4至6章	手算注意力，解释多头与一个完整模块
架构选择	第7至8章	区分原始翻译模型与 BERT GPT T5
代码实践	第9至10章	跑通小模型，检查标签与因果关系
实验与阅读	第11至13章	做控制变量实验，带着问题读论文

如果暂时没有合适的电脑，仍可完成前八章的纸笔计算。配套的注意力手算脚本只需要 Python 标准库；训练程序需要另行安装 PyTorch。训练小模型是学习结构和流程，不保证它生成流畅文本。你不需要先购买显卡。

0.3 本书约定

正文中“向量”一般按行排列：一句话的每个 token 占矩阵的一行。讲解单个样本时常省略 batch 维；代码默认保留 batch 维。单独写出的数学矩阵与程序中的张量必须按上下文对应，不能仅凭字母猜形状。

符号	含义	一个教学取值
B	一批样本的数量	2

符号	含义	一个教学取值
T 或 n	序列中 token 的数量	4
D 或 d_model	每个位置的隐藏向量维度	8
h 或 H	注意力头数	2
d_k 或 d_head	每个头的查询和键维度	4
d_ff	前馈网络中间层宽度	32
V_size	词表中可选 token 的数量	100
L	层数或损失	根据上下文判断

注意 V 还有另一个常见含义：注意力中的 Value 矩阵。本书把词表大小尽量写成 V_size；读论文时要检查作者自己的定义。数学中同一字母在不同章节重用非常常见。

本书中的手算值通常四舍五入到小数点后四位；程序浮点数可能略有差异。涉及论文的事实通过原论文链接核验，附录与第13章给出阅读入口。资源链接核验日期为 2026年9月29日；经典机制与现代扩展分别讲解，不把某个具体实现当成所有 Transformer 的统一标准。

第1章 先理解模型究竟在学习什么

本章目标：分清数据、模型、参数、损失与优化器；理解训练集和验证集；能够说清“预测下一个 token”是什么意思。你暂时不需要掌握微积分。

1.1 从一条可以写出来的规则开始

假设你要根据学习时长预测一道测验的得分。最简单的模型可以写成“预测得分 = w × 学习时长 + b”。w 控制增加一小时大约多预测几分，b 控制基准值。人可以手动试 w 和 b，机器学习则让程序根据一批已有例子调整它们。

输入是已知信息，目标是希望预测的结果；参数是模型内部可以调整的数。数据里的学习时长是输入，不是参数。w 和 b 是参数，不是新样本的输入。以后 Transformer 虽然拥有许多矩阵，这个区别仍然不变。

假设输入 x=2，真实目标 y=5，初始 w=1、b=0。模型预测 2，与目标差 3。我们需要一个数表示这次预测有多差，例如平方误差：

$$\hat{y} = wx + b, \qquad L = (\hat{y} - y)^2$$

代入得到 L=9。若将 w 改为2，预测变成4，损失降到1。模型是在寻找能让训练样本总体损失较低的参数。一次调整在某个例子上有效，不保证在所有例子上有效，所以训练必须使用多个例子，并在未参与参数更新的数据上检查表现。

1.2 一次训练步骤的五个动作

先取一批数据，再通过模型算预测，这叫前向传播。用预测和真实目标算损失。随后计算每个参数对损失的影响方向，这叫反向传播。优化器依据这些梯度调整参数。最后开始下一批，反复执行。

梯度可以先理解为“在当前参数附近，朝哪个方向变化会让损失变大、变化有多敏感”。为了减小损失，最简单的梯度下降朝相反方向走。学习率决定这一步走多大；学习率过大可能越走越远，过小可能学得很慢。

$$\theta_{new} = \theta_{old} - \eta \nabla_\theta L$$

这里 θ 代表全部参数，η 是学习率。倒三角符号表示梯度。模型不会直接读懂这个例子为什么错；它得到的是一个可计算的目标函数及其梯度。数据与目标设计得不好，优化器也可能非常成功地学到错误规律。

用上面的单参数例子验证一次：暂时固定 b=0，$L=(2w-5)^2$。在 w=1 时，梯度等于 $2(2w-5)\times 2=-12$。取学习率0.1，新的 $w=1-0.1\times(-12)=2.2$。预测4.4，损失0.36。这一算例说明负梯度也可能使参数增加。

1.3 神经网络增加了什么

直线表达不了所有规律。神经网络把多次线性变换和非线性函数组合起来。一个中间层先计算若干个特征，再让下一层组合这些特征。这里的“神经元”是一个数学计算单元，并不要求模仿生物神经元的完整工作方式。

若只有线性变换，连续两层可以合并成一层：先乘 W1 再乘 W2，本质上相当于乘一个 W1W2。加入非线性函数后，一般不能这样合并，模型才可以表达更复杂的输入输出关系。ReLU 把负数置零，GELU 是一种平滑的门控式非线性；第6章会把它们放进 Transformer 的前馈网络。

深度学习中的“深”主要指多层组合。参数更多、层数更多会增加表达能力，也会增加训练难度和计算开销。它们不会自动消除坏数据、标签错误或评估泄漏。

1.4 语言模型的输入和目标

考虑字符序列“我爱学习”。如果按字符分词，可构造三组预测：看到“我”预测“爱”，看到“我爱”预测“学”，看到“我爱学”预测“习”。真实任务通常使用子词或字节相关的分词方式，token 不一定等于一个汉字或一个英文单词。

模型不会直接输出一个中文字形。它先给词表中每个 token 一个分数，把分数变成概率，再按照某种规则选出下一个 token。假设候选只有“学”“吃”“看”，模型可能给出概率0.7、0.2、0.1。目标为“学”时，我们希望提高这个目标的概率，但怎样修改内部参数由梯度来决定。

一整段文本的概率可以按条件概率分解：

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

乘积符号表示把各位置的条件概率相乘。第一个 token 可以由开头标记或空上下文预测。训练时真实文本已经存在，可以同时构造许多位置的监督；生成时下一个真实 token 尚不存在，通常需要选出一个，再将它加入上下文。

这也解释了为什么低训练损失并不等于会聊天。模型可能记住很短的训练语料，生成时只是重复。对话能力、指令遵循和复杂推理还涉及数据、规模、训练目标与后续训练，不能从一个小实验直接推断。

1.5 训练集 验证集与测试集

训练集参与参数更新。验证集用于比较超参数、决定何时停止和检查过拟合。测试集应在模型和实验方案确定后用于最终评估。反复根据测试集结果修改模型，相当于把它也变成了调参依据。

过拟合的典型信号是训练损失继续下降，而验证损失不再改善甚至上升。它说明模型越来越适合训练样本，却没有同样改善未见样本上的表现。解决思路包括增加合适的数据、减少模型规模、调整正则化和训练长度，但首先要检查数据划分是否正确。

文本实验不要先把同一文档切成许多高度重叠的窗口，再随机分给训练和验证。两个集合可能共享几乎相同的句子，验证分数会过于乐观。更稳妥的入门方法是先按完整文档划分，再分别切窗口。现实任务还可能需要按用户、时间或来源划分。

1.6 练习与答案

练习1：模型输入是一段文字，输出下一词的分数。文字里的 token ID、嵌入矩阵和学习率，分别是什么？

答案：token ID 属于输入数据；嵌入矩阵属于可训练参数；学习率是人为配置的超参数。超参数一般不由这一次普通反向传播直接优化。

练习2：训练损失从2.0下降到0.1，验证损失从2.1上升到2.8，能否说模型变好了？

答案：只能说它更贴合训练数据。若目标是对未见数据泛化，验证结果提示问题。应检查重复数据、模型容量、训练时长和验证分布，不应该只挑训练曲线报告。

练习3：$L=(3w-6)^2$，在 $w=1$ 时，学习率0.05的一次梯度下降得到什么？

答案：梯度为$2(3\times1-6)\times3=-18$；新参数为1.9；新预测5.7；新损失0.09。梯度公式里的最后一个3来自内层3w的变化率，即链式法则。

第2章 读懂向量矩阵概率和梯度

本章目标：能够手算点积和小矩阵乘法，区分矩阵乘法与逐元素乘法，理解 softmax、交叉熵和链式法则。掌握这些，就足以开始阅读注意力机制。

2.1 数字怎样组织成张量

一个数是标量，例如0.5；一排数是向量，例如[1,2,3]；一个二维数表是矩阵；更多维的数数组统称为张量。这里的“维”有两种容易混淆的用法：三维向量说的是它有三个分量；三阶张量说的是它有三个轴。

假设有两句话，每句补齐到四个 token，每个 token 用八个数表示。整体张量的形状为[2,4,8]。三个轴依次是 batch、序列位置、特征。它有2×4×8=64个数。形状本身没有语义，语义来自你对各个轴的约定。

第一个样本的第三个 token 可以写成 X[0,2,:]，得到长度8的向量。冒号表示取该轴全部内容。Python 从0开始计数，因此第三个位置对应下标2。许多注意力错误来自轴搞反，而非公式本身写错。

2.2 点积是怎样得到一个分数的

两个等长向量的点积，是对应位置相乘再相加。令 q=[1,2]、k=[3,4]，则 q·k=1×3+2×4=11。结果是标量，表示在当前坐标系下的一种匹配分数。

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

求和符号的意思是让 i 从1走到 d，把每一项加起来。注意力使用点积比较查询与键，但点积并不是天然的“语义相似度”。向量的方向和长度都会影响分数；这些向量经过训练才可能编码任务所需的信息。

例如 q=[1,0] 与 k1=[1,0] 点积为1，与 k2=[10,0] 点积为10，两者方向一样，但长度不同。余弦相似度会除以两个向量的长度，标准点积注意力通常没有这一步。之后出现的除以根号 d_k 是控制分数尺度，也不是余弦归一化。

2.3 矩阵乘法让许多点积一起进行

矩阵 A 的形状为[m,k]，矩阵 B 的形状为[k,n]，则 AB 的形状为[m,n]。中间的 k 必须一致；结果的第 i 行第 j 列，是 A 的第 i 行与 B 的第 j 列的点积。

$$C_{ij} = \sum_{r=1}^{k} A_{ir} B_{rj}$$

用一个完整例子：A=[[1,2],[3,4]]，B=[[2,0],[1,5]]。结果 C=[[4,10],[10,20]]。左上角4来自1×2+2×1；右下角20来自3×0+4×5。把这个过程自己展开一次，比背口诀更有效。

转置把行和列交换。若 K 形状为[T,D]，则 K 的转置形状为[D,T]。所以 QK^T 将[T,D]和[D,T]相乘，得到[T,T]：每个查询与每个键各有一个分数。这个二维“位置对位置”分数表，就是注意力最核心的中间结果。

逐元素乘法要求相应位置配对相乘，不进行求和。在 PyTorch 中，`a @ b` 常用于矩阵乘法，`a * b` 是逐元素乘法。这两个运算符在注意力中不能互换。

2.4 线性层就是学一个坐标变换

若每个 token 有 D 个特征，一个线性层可以把它变为 D_out 个特征。按本书行向量约定，权重 W 的形状为[D,D_out]，偏置 b 长度为 D_out：

$$Y = XW + b$$

对序列中的每一行使用同一组 W 和 b，叫参数共享。它不会为第一位置和第二位置各创建一个不同的线性层。这样，无论句子有多少 token，线性层的参数量都不因序列长度改变。

若 D=8、D_out=12，权重有96个数，偏置有12个数，共108个参数。PyTorch nn.Linear(8,12) 内部存储的 weight 形状是[12,8]，前向计算会按对应规则使用转置；这是存储约定差异，并不与本书的数学公式矛盾。

2.5 softmax 把分数变成一组权重

原始分数可以为负，也不要求加起来为1。softmax 对每个分数取指数，再除以所有指数之和：

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

若分数 z=[0,1,2]，指数约为[1,2.7183,7.3891]，总和约11.1073，于是概率约为[0.0900,0.2447,0.6652]。分数越高，权重越大；三项非负且总和为1。注意：这些性质不意味着它们已经是校准良好的真实世界概率。

计算机的数值范围有限。直接对很大的数取指数可能溢出。将所有分数先减去这一行的最大值，结果不变：

$$p_i = \frac{e^{z_i - m}}{\sum_j e^{z_j - m}}, \qquad m = \max_j z_j$$

为什么不变？每个分子与分母都多乘了同一个 e 的负 m 次方，它们会约掉。原来分数[1000,1001,1002]可以变为[-2,-1,0]后再计算。不要用普通的“除以分数总和”代替 softmax：遇到负数或总和为0，性质会完全不同。

softmax 还必须指定在哪个轴上做。注意力分数[T,T]中的每一行，代表一个查询对所有键的打分。因此要沿“键的位置”轴归一化，通常就是最后一个轴。整张矩阵一起归一化会改变含义。

2.6 交叉熵为什么鼓励正确答案

对于只有一个正确类别的分类问题，目标类别为 y，交叉熵等于该类别预测概率的负对数：

$$L = -\log p_y$$

本书默认自然对数。若模型给正确答案概率0.8，损失约0.2231；给0.1，损失约2.3026。正确类别越不被模型相信，惩罚越大。概率为1时损失为0；概率接近0时损失越来越大。

一个 batch 中通常先为每个有效位置计算损失，再取平均。若有 padding，不能把补齐位置当作真实答案一起平均。还要明确平均的是 token 还是样本，因为句子长短不同会影响统计权重。

实际训练常把原始 logits 直接交给交叉熵函数，它内部用数值稳定的方式完成 log-softmax 与负对数似然。若提前做一次 softmax，再将结果当 logits 交进去，会得到另一个错误目标。第9章会展示这一点。

一个实用推导是 softmax 与交叉熵的组合梯度。若目标是 one-hot 向量 y，最终关于每个 logit 的梯度是预测概率减目标：

$$\frac{\partial L}{\partial z_i} = p_i - y_i$$

比如预测[0.2,0.5,0.3]，正确答案为第3类，梯度为[0.2,0.5,-0.7]。梯度下降会倾向于降低前两类分数、提高第3类分数。实际参数同时影响很多样本，所以真正更新是汇总多个位置的梯度之后的结果。

2.7 链式法则把责任传回去

假设中间量 a=2w，输出 b=a²，损失 L=b。w 影响 a，a 又影响 b。链式法则把沿路径的变化率相乘：

$$\frac{dL}{dw} = \frac{dL}{db}\frac{db}{da}\frac{da}{dw} = 1 \times 2a \times 2 = 8w$$

在 w=3 时，梯度为24。可以用有限差分验证：将 w 增加一个很小的 ε，观察损失变化除以 ε 是否接近24。有限差分适合检查小模型的一两个参数，但参数太多时计算代价很大；自动微分则沿计算图组织这些导数。

反向传播计算梯度，优化器使用梯度更新参数，这两步要区分。在 PyTorch 中调用 `loss.backward()` 并不会自动改变权重；通常还要调用 `optimizer.step()`。梯度默认可能累加，所以普通训练步骤还要清空上一轮梯度。

2.8 练习与答案

练习1：[2,5,8] 的输入通过8到12的线性层，输出形状与含偏置参数量各是多少？

答案：输出[2,5,12]。只变最后一个轴；参数量8×12+12=108，与 batch 和序列长度无关。

练习2：Q=[3,4]，K=[5,4]，V=[5,6]，这里方括号表示形状。QK^T和 softmax(QK^T)V 的形状是什么？

答案：QK^T为[3,5]，加权结果为[3,6]。查询数量决定输出有几行，值向量维度决定每行有多长。查询与键的数量可以不同，这是交叉注意力的重要情况。

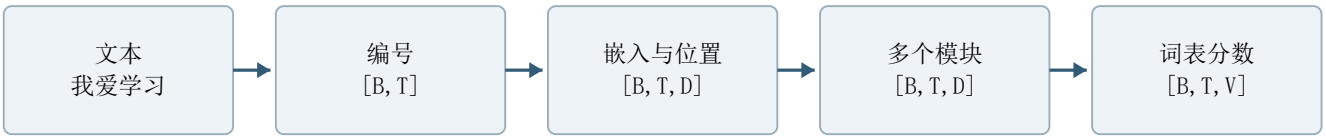
练习3：两个分数相同，softmax 权重是什么？若同时增加100又如何？

答案：都是[0.5,0.5]；同时增加100不改变结果。softmax 依赖分数差异，而不是共同偏移。

练习4：目标概率从0.25提高到0.5，损失减少多少？

答案：原损失约1.3863，新损失约0.6931，减少约0.6931。概率翻倍对应负对数减少 ln 2，但准确率可能完全没有变化，因此两种指标观察的角度不同。

第3章 文字表示与序列建模



每个模块更新表示；输出头把隐藏特征变成候选 token 的分数。

图3 1 语言模型的输入输出形状

本章目标：理解 token、嵌入和上下文表示；知道循环模型与注意力各自怎样传递信息；看懂 Transformer 从文本到输出概率的总流程。

3.1 文本先经过分词与编号

计算机需要一种把文本转换成离散符号序列的规则。最容易理解的是字符级分词：将“我爱学习”切成四个字符。随后查词表，把它们变成例如[3,7,4,9]。这些数字只是索引，9不表示这个字符比3更重要。

按完整单词切分时，生僻词和新词可能不在词表里。子词分词把常见片段作为 token，让许多新词能够由已有片段拼出来。字节级方法还会涉及编码字节，而非人眼看到的一个字。具体模型如何切分，必须使用它自己的 tokenizer 查看。

同一句中文在不同分词器下可能得到不同数量的 token。所以“上下文长度为4096”通常指最多4096个 token，不是4096个汉字。训练、推理和保存模型时，词表与分词规则都必须一致，否则同一个 ID 会被错误地解释。

开始标记 BOS、结束标记 EOS、补齐标记 PAD、未知标记 UNK 是常见的特殊 token。具体模型未必都使用它们。初学时先理解作用，不要把一套特殊标记机械套到所有模型。

3.2 嵌入是可训练的查表

词表大小为 V_size，嵌入维度为 D，可以准备一个[V_size,D]矩阵 E。token ID 为 i，就取 E 的第 i 行作为向量。若 V_size=100、D=8，嵌入层包含800个可训练数。

$$x_t = E[\mathrm{id}_t]$$

查表本身看起来很简单，但表里的数会随着任务训练不断调整。输入 ID 不需要梯度，选中的嵌入行需要梯度。若写成 one-hot 向量乘嵌入矩阵，会得到相同结果；程序一般直接查表，避免显式建立巨大稀疏向量。

设第3行是[0.2,-0.1,0.4]，一个序列中 ID=3 出现两次，刚查表得到的两个向量相同。进入 Transformer 之后，它们可能因为位置和上下文不同而变成不同表示。这就是静态 token 嵌入与上下文隐藏状态的区别。

不要给每个维度强行赋予“性别”“情绪”“词性”等固定解释。学出的表示可能把信息分散在多个维度中，不同训练还会得到不同坐标系。我们可以研究这些表示，但不能仅看一个数就声称知道模型在想什么。

3.3 顺序与上下文为什么重要

“小猫追小狗”和“小狗追小猫”包含相同的词，顺序却改变了关系。仅把所有词向量相加，无法充分保留这种差异。模型需要知道哪个 token 在哪个位置，也需要让不同位置之间交换信息。

再看“我把苹果放进背包，因为它很轻”。“它”可能指哪个实体取决于上下文与常识。一个 token 独立的词向量无法完成全部判断；模型需要结合其他位置形成表示。注意力提供了一种按当前内容选择信息来源的计算方式。

这里的语言例子用于说明任务需求，并不意味着某个注意力头必然学会了代词指代。最终学到什么要靠数据、训练和实验验证。

3.4 循环模型怎样处理序列

循环神经网络维护一个隐藏状态。每读一个 token，就根据当前输入和前一步状态更新：

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

状态像一份不断更新的记录，把前文信息传到后面。这个结构适合按时间推进，但同一层中后一步依赖前一步，使序列方向上的并行受限。长距离关系还要经过多次状态更新；是否能保留有用信息取决于结构与训练。

LSTM 和 GRU 通过门机制改善信息保留问题；它们在一些任务和资源条件下仍有价值。本书介绍循环模型，是为了比较信息路径。无需先把所有循环网络推导完，才有资格学习 Transformer。

3.5 注意力怎样改变信息路径

自注意力允许每个位置直接根据相关性读取其他允许访问位置的信息。先给每个位置生成查询、键和值；再用查询与各个键打分；最后对值向量加权求和。第4章将把每一步算出来。

一个位置可以在单层中直接组合远处位置的信息，序列中很多位置也可以组织成矩阵运算。但“路径短”不保证已经学会关系；标准稠密注意力还要计算许多位置对，长序列会带来开销。

对长度 T、宽度 D 的输入，单层稠密注意力中位置两两打分与加权通常含有 $O(T^2D)$ 量级的计算；投影与前馈层也有与 TD^2 有关的成本。不能只看到 T^2 就断言所有设置都由注意力占据最大计算量。短序列、大宽度时，投影和前馈部分可能很重要。

3.6 Transformer 的总体计算路线

先把文本变成 token ID；查嵌入并加入或施加位置信息；经过多个 Transformer 模块；最后根据任务连接输出头。语言模型的输出头把每个位置的隐藏状态映射到词表分数，分类模型则可能把整段表示映射到类别分数。

每个基本模块通常包括注意力、逐位置前馈网络、残差连接与归一化。注意力负责跨位置交换信息，前馈网络对每个位置的特征进行非线性变换；残差和归一化帮助组织深层计算与训练。位置处理方案会影响模型怎样感知顺序。

在原始翻译结构中，编码器先处理源句，解码器再根据目标前缀和源句表示预测下一个词。只保留编码器，可以发展出适合理解任务的模型；只使用带因果限制的解码器式模块，可以发展出自回归语言模型。这些分支共享许多基本部件，但训练目标并不相同。

把整条路线对照到形状：输入 ID 为[B,T]；嵌入后[B,T,D]；多个保持宽度的模块仍输出[B,T,D]；语言模型头输出[B,T,V_size]。最后一个轴从隐藏特征变为每个候选 token 的分数。

3.7 第一次实践前的准备

先确认你能运行一个 Python 文件，并理解列表、函数、类、循环和切片。本书代码采用 PyTorch，因为它能自动计算梯度，并提供张量运算和神经网络组件。无需现在熟悉它所有接口；先抓住 Tensor、Module、loss 和 optimizer 的关系。

在独立的 Python 环境中安装依赖，可以减少与已有项目冲突。CPU 就能完成手算验证和小规模结构检查。GPU 适合加速矩阵计算，但需要兼容的硬件与软件；安装命令应根据官方页面选择，不必照搬别人电脑上的 CUDA 版本。配套实验说明给出了最小入口。

补基础可阅读《动手学深度学习》的线性代数章节及 PyTorch 官方 Quickstart。本书解释与手算示例可独立阅读，外部资源用来补充练习，不是必须先读完的前置教材。

参考入口：D2L 线性代数；PyTorch Quickstart。原始结构出处：Attention Is All You Need。

3.8 练习与答案

练习1：一个词表有5000个 token，嵌入宽度128，嵌入矩阵有多少参数？句子长度从20变成40时参数量怎样变化？

答案：5000×128=640000。嵌入表参数量不变；处理时产生的激活数量增加。可学习位置表的容量是另一个配置，要与 token 嵌入区分。

练习2：两个相同 token 出现在同一句话的不同位置，它们的最终表示一定相同吗？

答案：不一定。初始 token 嵌入可以相同，但位置处理与允许读取的上下文会改变后续隐藏状态。因果模型中两个位置可见的前缀也不同。

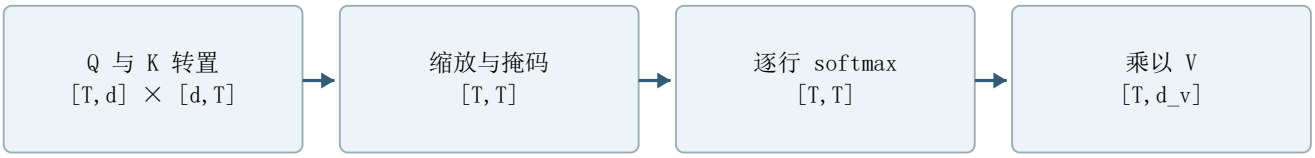
练习3：输出形状[4,16,1000]中，每个轴表示什么？若取 `logits[:, -1, :]` 得到什么？

答案：batch 为4，输入长度16，词表1000。切片得到[4,1000]，保留每个样本最后一个位置的词表分数；对自回归模型而言，它们用于预测当前上下文之后的下一个 token。

练习4：为什么把所有词向量平均一下不能代替完整 Transformer？

答案：简单平均通常丢失顺序，且各位置权重不依赖当前查询；它也缺少后续多层非线性上下文计算。平均向量仍可以作为有用的基线，用实验比较它的优缺点。

第4章 把注意力真正算一遍



一行对应一个查询；归一化沿键的位置轴进行。

图4 1 缩放点积注意力的四步计算

4.1 学习目标与一个具体问题

读完本章，你应当能独立完成三件事：给定小矩阵，算出注意力输出；解释 Q、K、V 的来源与形状；指出 softmax 沿哪个维度进行，以及为什么要做数值稳定处理。先不要追求记住所有名字，要追求能把每一步的输入、输出写出来。

考虑一句话：“小明把书递给小红，因为她要参加读书会。”如果要判断“她”更可能指谁，单看“她”的初始词向量通常不够，表示应当吸收上下文。注意力提供了一种可学习的信息混合方法：当前位置先与可访问的各位置算匹配分数，再根据这些分数混合各位置提供的向量。模型会不会学到正确的指代关系，取决于数据、训练目标和参数；仅仅装上注意力并不保证语言理解。

“查询、键、值”是一组便于讨论的名称。可以把它们暂时理解为“拿什么比较、与什么比较、比较后混合什么”，但不要把这理解成程序真的检索了一张写有语义标签的数据库。程序实际执行的是矩阵乘法、指数、归一化和加权求和。

4.2 从一个位置到全部位置

设输入有 T 个 token，每个 token 的表示是 d_model 维。把每个 token 放成一行，得到 X。单头自注意力通常用三个不同的可学习线性变换产生 Q、K、V。为突出核心计算，这里暂时省略偏置。

$$X \in \mathbb{R}^{T \times d_{model}}$$

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

$$W_Q, W_K \in \mathbb{R}^{d_{model} \times d_k}, \qquad W_V \in \mathbb{R}^{d_{model} \times d_v}$$

因此 Q、K 的形状都是 T×d_k，而 V 是 T×d_v。Q 和 K 最后一维必须相等，才能做点积；d_v 可以不同。使用相同 X 叫作自注意力，但这不意味着 Q、K、V 三个矩阵相同，因为三个投影矩阵通常不同。

把 Q 的第 i 行记作 q_i。它与 K 的第 j 行做点积，得到当前位置 i 对位置 j 的一个分数。点积同时受方向和长度影响，所以这里的“匹配”不是严格的余弦相似度，也不天然等于两个词在字典中的语义相似度。

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

固定 i，让 j 遍历全部允许访问的位置，再用 softmax 转成非负、和为 1 的权重：

$$a_{ij} = \frac{\exp(s_{ij})}{\sum_{r=1}^{T} \exp(s_{ir})}$$

最后，不是把权重加到 K 上，而是加权混合 V：

$$o_i = \sum_{j=1}^{T} a_{ij} v_j$$

这里有一个值得反复检查的下标关系：输出的第 i 行对应第 i 个查询；求和下标 j 对应被读取的位置。把所有行一起算，得到熟悉的紧凑写法：

$$S = \frac{QK^T}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}_{row}(S), \qquad O = AV$$

这个公式采用缩放点积注意力的标准形式。历史来源可查阅 Attention Is All You Need，下面的数值例子和推导为本教材自行构造。

4.3 固定小矩阵 从头手算

本节直接给出已经投影后的 Q、K、V，不再倒推它们来自哪个 X。三个位置，每个查询和键都是二维，即 d_k=2；值也是二维，即 d_v=2。为了方便算数，特意让 Q=K，这只是这个例子的选择。

$$Q = K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad V = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 1 \end{bmatrix}$$

第一步，算点积矩阵。比如第 1 行第 3 列是 [1,0] 与 [1,1] 的点积，等于 1；第 3 行第 3 列是 1×1+1×1，等于 2。

$$QK^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

第二步，所有元素除以 √2。不是除以 token 数量 3，也不是把整行的和除掉。

$$S \approx \begin{bmatrix} 0.707107 & 0 & 0.707107 \\ 0 & 0.707107 & 0.707107 \\ 0.707107 & 0.707107 & 1.414214 \end{bmatrix}$$

第三步，对每一行分别做 softmax。只看第 1 行，它对应“第 1 个查询怎样混合三个位置”。原始指数分别约为 2.028115、1、2.028115，总和约为 5.056230，所以：

$$A_{1,:} \approx [0.401112,\ 0.197776,\ 0.401112]$$

第 1 与第 3 个位置的分数相同，因此权重相同；第 2 个位置分数为 0，权重却不是 0。softmax 并不是“把负数删掉，把正数留下”的操作，分数为 0 对应指数为 1。全部三行的权重如下：

$$A \approx \begin{bmatrix} 0.401112 & 0.197776 & 0.401112 \\ 0.197776 & 0.401112 & 0.401112 \\ 0.248255 & 0.248255 & 0.503490 \end{bmatrix}$$

每一行的和约等于 1，列和则没有这种要求。虽然本例 S 对称，逐行归一化后的 A 一般并不对称，因为不同行的分母可能不同。

第四步，计算输出第 1 行。V 的三行分别是 [1,0]、[0,2]、[3,1]，所以：

$$o_1 = 0.401112[1,0] + 0.197776[0,2] + 0.401112[3,1]$$

$$o_1 \approx [1.604448,\ 0.796664]$$

例如第一维是 0.401112×1+0.197776×0+0.401112×3。第二维是 0.401112×0+0.197776×2+0.401112×1。三个输出合起来为：

$$O \approx \begin{bmatrix} 1.604448 & 0.796664 \\ 1.401112 & 1.203336 \\ 1.758725 & 1.000000 \end{bmatrix}$$

注意第 1 行输出并不等于第 1 行的 V，也不一定接近权重最大的某一个 V。它是多个向量按比例混合的结果。由于权重非负且和为 1，在没有注意力 dropout 等额外操作时，单个头的加权结果位于这些 V 向量的凸包中；但后续输出投影、残差连接和前馈网络不再受这个限制。

4.4 用形状检查防止“看懂却写错”

对象	本例形状	一般形状	每一行表示什么
Q	3×2	T_q×d_k	一个查询
K	3×2	T_k×d_k	一个可比较的键
K 的转置	2×3	d_k×T_k	把键排成列便于点积
S、A	3×3	T_q×T_k	一个查询对全部键的分数或权重
V	3×2	T_k×d_v	每个键位置对应的值
O	3×2	T_q×d_v	每个查询读取得到的结果

这里特意把一般长度写为 T_q 和 T_k。自注意力通常二者相等，交叉注意力则不必相等。始终必须满足：K 的位置数等于 V 的位置数。因为每个权重对应一个键，就应当有同位置的值供它混合。

读代码时，每遇到一次矩阵乘法，先写形状。T_q×d_k 乘 d_k×T_k 得 T_q×T_k；再乘 T_k×d_v 得 T_q×d_v。这样你能发现大量程序即使运行成功也可能藏着的语义错误，例如把查询长度和批大小碰巧设成了相同数字而掩盖轴用错的问题。

4.5 为什么除以 √d_k

一个直觉是：向量维数越多，点积包含的相加项越多，分数的波动可能变大。为了说得更精确，做一个简化假设：q 和 k 的各分量相互独立，均值为 0，方差为 1。这个假设用于理解尺度，不是在断言真实训练始终满足它。

令 z=q·k。每一项 q_r k_r 的均值为 0、方差为 1；在独立假设下，d_k 项相加的方差是 d_k。于是：

$$\mathrm{Var}(z) = d_k, \qquad \mathrm{Var}\left(\frac{z}{\sqrt{d_k}}\right) = 1$$

为什么控制波动有帮助？比较 softmax([1,0]) 与 softmax([10,0])：前者约为 [0.731,0.269]，后者约为 [0.999955,0.000045]。分数整体放大后，分布容易变得非常尖锐，一些梯度变小。这不是说尖锐分布永远不好，而是我们不希望“只是换了维度”就无意中大幅改变初始分布的尺度。缩放是常见的稳健设计，不是对一切训练问题的保证。

4.6 softmax 的数值稳定与参考实现

数学上的指数函数没有计算机浮点数的上限。如果直接算 exp(1000)，通常会溢出。softmax 有一个非常实用的性质：同一行的所有分数减去同一个常数，结果不变。因为分子、分母同时乘上相同因子，该因子会约掉。

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

取 c 为这一行最大值，最大的指数就变成 exp(0)=1，其余指数不超过 1。本例第 1 行从 [0.707107,0,0.707107] 变成 [0,-0.707107,0]，指数约为 [1,0.493069,1]，归一化仍是刚才的权重。极小指数可能下溢到 0，但这通常比溢出稳妥。

下面的 NumPy 片段只演示无 mask 的有限输入。下一章会解释为什么“减最大值”仍不能解决整行都是负无穷的问题。

```
import numpy as np

Q = np.array([[1, 0], [0, 1], [1, 1]], dtype=np.float64)
K = Q.copy()
V = np.array([[1, 0], [0, 2], [3, 1]], dtype=np.float64)

scores = Q @ K.T / np.sqrt(Q.shape[-1])
shifted = scores - scores.max(axis=-1, keepdims=True)
exp_scores = np.exp(shifted)
weights = exp_scores / exp_scores.sum(axis=-1, keepdims=True)
out = weights @ V

assert np.allclose(weights.sum(axis=-1), 1.0)
assert np.allclose(out[0], [1.6044483707, 0.7966637220])
print(np.round(weights, 6))
print(np.round(out, 6))
```

axis=-1 表示沿最后一维，即键的位置维进行归一化。keepdims=True 保留长度为 1 的轴，让逐行减法和除法的广播关系清楚。实际使用 PyTorch 时，通常直接调用其稳定实现 torch.softmax(scores, dim=-1)，不用自己展开指数。

4.7 参数究竟从哪里学来

刚才把 Q、K、V 当作固定数字，是为了分离“会算”和“会训练”。在模型中，投影矩阵 W_Q、W_K、W_V 是参数；输入 X 是当前样本的表示；Q、K、V 和 A 是该样本前向计算的中间结果。换一个句子，A 就会变化。训练时通常不会给每个注意力权重一个人工标准答案，而是根据最终任务的损失，通过反向传播调整投影矩阵及其他参数。

例如模型预测下一 token 错了，损失对输出、权重、分数和投影参数都可能产生梯度。经过许多样本后，参数可能学会让有用的信息影响预测。注意力热力图显示的是某层某头的信息混合系数，并不自动等于最终预测的因果解释：V 的内容、其他头、残差路径、其他层都参与了计算。可视化是观察工具，不能仅凭“这格颜色深”就下结论说“模型完全因为这个词作答”。

4.8 小练习 答案与误解纠正

练习 1。 本例第 2 行权重为何是 [0.197776,0.401112,0.401112]？不重新计算指数，解释它与第 1 行的关系。

答案。 第 2 行分数是第 1 行交换前两项的结果。softmax 的分母没有变化，分子也只交换位置，因此权重交换前两项。用这些权重混合 V，得到 [1.401112,1.203336]；输出不只是把第 1 行输出的两维交换，因为 V 的结构并非对应对称。

练习 2。 如果一个查询对 4 个键的分数完全相等，它的注意力输出是什么？

答案。 每个权重为 1/4，输出是 4 个值向量的算术平均。与分数相等于 0、5 还是 -9 无关。若某些位置被 mask，则应当只在剩余合法位置上归一化。

练习 3。 Q 为 5×8、K 为 7×8、V 为 7×6，输出形状是什么？能不能把 V 改成 8×6？

答案。 分数为 5×7，输出为 5×6。V 改为 8×6 不合法，因为 7 个键对应的权重无法与 8 个值逐项对齐。这说明输出位置数由 Q 决定，输出每个位置的宽度由 V 决定。

练习 4。 把所有 V 都设为同一个向量 [2,-1]，不改 Q、K，输出会怎样？

答案。 每一行输出都是 [2,-1]，因为权重和为 1。这个反例表明：只看 A 不足以知道输出携带了什么信息，必须同时知道 V。

需要纠正的三句话。 “Q 就是问题中的某个词”不准确，它是每个位置的投影向量；“注意力只选择一个词”不准确，标准形式是连续加权混合；“自注意力意味着只看自己”不准确，“自”指 Q、K、V 来自同一序列，合法范围内的其他位置也可以参与。

第5章 多头注意力 批处理与遮罩

	键位置 1	2	3	4
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1

行是查询位置，列是键位置。

1 表示允许读取，0 表示禁止读取。

禁止位置的分数在 softmax 前设为负无穷。

例如第2个位置只能看第1和第2个位置。

图5 1 因果掩码的可见范围

5.1 学习目标与引例

如果只用一组权重混合信息，不同关系可能在一次平均中挤在一起。多头注意力让模型用多组独立投影分别计算，再把各头结果组合。读完本章，你应当会追踪批大小、序列长度、头数和每头宽度这四个轴；会写因果遮罩和 padding 遮罩；能诊断“代码没报错但偷看未来”以及“loss 突然变成 NaN”这两类重要问题。

可以想象你在读“红色的球放在蓝色盒子旁边”时，既关心名词与颜色，也关心物体之间的位置关系。但多头并没有被人工分配“颜色头”“位置头”。训练后某些头可能呈现可描述的规律，也可能混合多个功能、冗余或难以解释。可学习的多组表示才是机制，具体语义分工只是可能出现的现象。

5.2 多头不是把一句话分成几段

记头数为 h。常见配置中每头宽度 d_h=d_model/h，并且 Q、K、V 每头宽度都取 d_h。假设 d_model=8、h=2，则每个 token 依然由两个头都处理；每个头使用一个 4 维投影表示，而不是第 1 个头看前半句、第 2 个头看后半句。

第 r 个头有自己的参数，先做投影，再用第 4 章的注意力计算：

$$H_r = \text{Attention}(XW_Q^{(r)}, XW_K^{(r)}, XW_V^{(r)})$$

各 H_r 都有 T 行。把同一个位置的所有头输出沿特征轴拼起来，再做输出投影：

$$\text{MHA}(X) = \text{Concat}(H_1, \ldots, H_h)W_O$$

拼接不会把 T 变成 hT，它把特征宽度由 d_h 变成 h×d_h。W_O 随后让不同头的信息进行可学习的线性组合，并把输出宽度恢复到 d_model，方便与输入做残差相加。

实际代码常先执行一个大投影 q_proj(x)，输出宽度仍为 d_model，再把最后一维拆为 h × d_h。这与分别设置 h 组较小投影在数学上可以等价：大矩阵的不同列块正好对应各头参数。因此“先投影后拆头”并不意味着每个头只接触原始 X 的一部分维度；大投影通常已经让每个输出维度读取了所有输入特征。

5.3 四个轴的完整旅行

现在加上批大小 B。在一个 batch 中，放 B 条长度统一为 T 的序列。假设 B=2、T=4、d_model=8、h=2、d_h=4，按以下顺序跟踪：

步骤	一般形状	本例形状	操作的含义
输入 X	B×T×d_model	2×4×8	每条序列的 token 表示
大投影 Q	B×T×(h d_h)	2×4×8	生成全部头的查询
拆开特征轴	B×T×h×d_h	2×4×2×4	标明每头的特征范围
交换中间两轴	B×h×T×d_h	2×2×4×4	每个 batch、每个头独立做注意力
QK 转置相乘	B×h×T×T	2×2×4×4	查询位置与键位置的两两分数
与 V 相乘	B×h×T×d_h	2×2×4×4	每个头读取得到的值
交换回去	B×T×h×d_h	2×4×2×4	同位置的各头重新相邻
合并头与宽度	B×T×d_model	2×4×8	拼接各头
输出投影	B×T×d_model	2×4×8	融合头信息

本例中有好几个数字碰巧都是 4，所以测试时最好再使用 B=2、T=5、d_model=12、h=3，这样错误轴更容易暴露。程序里的 `transpose(-2, -1)` 只交换最后两个轴：对 K 从 B×h×T×d_h 变成 B×h×d_h×T，而不是把四个轴全倒过来。

下面展示关键变形，其中 `q_proj` 等代表已建立的线性层。

```
B, T, D = x.shape
assert D % num_heads == 0
head_dim = D // num_heads

q = q_proj(x).reshape(B, T, num_heads, head_dim).transpose(1, 2)
k = k_proj(x).reshape(B, T, num_heads, head_dim).transpose(1, 2)
v = v_proj(x).reshape(B, T, num_heads, head_dim).transpose(1, 2)

scores = q @ k.transpose(-2, -1) / (head_dim ** 0.5)
# 此处先加入下一节介绍的 mask 再做 softmax。
weights = scores.softmax(dim=-1)
heads = weights @ v
joined = heads.transpose(1, 2).contiguous().view(B, T, D)
out = out_proj(joined)
```

`transpose` 往往改变张量的步长，而没有立即重新排列底层数据；`contiguous()` 在需要时整理为连续布局，使之后的 `view` 明确可行。也可以在恰当位置用 `reshape`，让框架必要时复制。对初学者而言，先保证轴的语义正确，再考虑减少拷贝。

5.4 为什么必须遮住未来

用序列“我 喜欢 学习”训练下一个 token 预测。如果模型在“我”的位置就能看到后面的“喜欢”，它可能直接复制答案。训练损失很低，但实际生成时未来 token 不存在，这样的模型没有学到正确任务。

因果遮罩规定：第 i 个查询只能访问 j≤i 的键。允许对角线是因为当前位置的输入 token 已经给定，模型要预测的是它后面的 token。用加性 mask 表示，合法位置加 0，不合法位置加负无穷：

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A = \mathrm{softmax}_{row}(S + M)$$

因为 exp(-∞)=0，被遮住位置的权重为 0，其余合法位置重新归一化。注意遮罩必须在 softmax 之前参与计算。如果 softmax 后直接把非法权重清零却不重新归一化，行和会小于 1，已不是这里定义的注意力。

把第 4 章的例子加上因果遮罩，会得到很直观的变化：第 1 行只能看自己，权重变成 [1,0,0]，输出就是 [1,0]；第 2 行权重约为 [0.330238,0.669762,0]，输出约为 [0.330238,1.339523]；第 3 行可看三个位置，保持原结果。这是一个适合写成单元检查的例子。

遮罩的关键是“数据可见性”，并不是把训练改成从左到右一次只运行一行。矩阵里所有合法的注意力关系可以同时计算。第 8 章会把它与生成时的顺序依赖仔细区分。

5.5 padding 遮罩与损失遮罩解决不同问题

同一批文本长度不同，为了组成规则张量，常补上 PAD。例如两条序列分别长 4 和 2，右侧补齐后：

```
样本 A：今天 天气 很好 EOS
样本 B：你好 EOS  PAD  PAD
```

padding key mask 禁止任何查询从无意义的 PAD 键位置读取信息。若 token_ids 形状为 B×T，则合法键标记可写为 (token_ids != pad_id)[:, None, None, :]，形状 B×1×1×T。这个形状能广播到 B×h×T×T：每个头、每个查询都使用对应样本的合法键列表。

但禁止读取 PAD 并不意味着 PAD 查询位置的输出自动为零。PAD 位置可能仍读取前面真实 token，后续线性层的偏置与残差也可能产生非零值。通常还需要在损失中忽略标签为 PAD 的位置；若要对整句表示求平均，则平均池化也必须忽略 PAD。三个问题要分清：注意力 mask 管“能读谁”，损失 mask 管“哪些预测参与训练”，池化 mask 管“哪些位置参与汇总”。

在交叉注意力里，查询长度是目标长度 T_t，键和值的长度是源长度 T_s，分数是 B×h×T_t×T_s。此时 padding key mask 来自源序列，而不是目标序列；把二者写反，恰好长度相同的测试可能发现不了。

5.6 两个数值陷阱 整行遮挡与负无穷乘法

第一个陷阱是某个查询所有键都被遮住。softmax 面对 [-∞,-∞] 时，稳定实现要减去最大值 -∞，于是出现 -∞-(-∞)，结果是 NaN。此时数学上也没有可归一化的合法分布：分母为 0。把 NaN 强行替换成 0 会隐藏上游的 mask 错误，不应当成为默认补救。

哪些场景容易出现？全 PAD 的样本；左侧 padding 与因果遮罩组合后，最前面的 PAD 查询没有任何合法历史；交叉注意力遇到空源序列；或者布尔 mask 的真假语义写反。教学实验采用“非空序列、右侧 padding、有效序列带起始 token”等约定，通常更容易检查。其他实现可以跳过无效查询，或规定它们的输出并明确保证不进入损失，但要有完整设计。

第二个陷阱是用 mask.float() * -inf 构造加性 mask。你以为 False 变成 0，所以会得到 0；浮点规则却使 0×(-∞)=NaN。更可靠的写法是创建布尔合法矩阵，再用 masked_fill 把非法位置设为负无穷。

下面定义自己的接口：allowed=True 一律表示可以访问。代码只支持每个查询至少有一个合法键，遇到问题会尽早报错。

```
import torch

def attention_with_allowed(q, k, v, allowed):
    # q: [B, h, Tq, d], k: [B, h, Tk, d]
    # v: [B, h, Tk, dv]
    # allowed 必须能广播到 [B, h, Tq, Tk]
    scores = q @ k.transpose(-2, -1) / (q.shape[-1] ** 0.5)
    allowed = torch.broadcast_to(allowed.bool(), scores.shape)
    if not allowed.any(dim=-1).all():
        raise ValueError("存在没有任何合法键的查询，请检查数据与 mask")
    weights = scores.masked_fill(~allowed, float("-inf")).softmax(dim=-1)
    return weights @ v, weights

# token_ids  [B  T]；教学约定为非空序列与右侧 padding
T = token_ids.shape[1]
causal = torch.ones(T, T, dtype=torch.bool, device=token_ids.device).tril()
valid_keys = (token_ids != pad_id)[:, None, None, :]
allowed = causal[None, None, :, :] & valid_keys
```

不要把本书的布尔约定不加检查地套到所有库函数上。不同接口可能用 True 表示“允许”，也可能表示“屏蔽”。使用内置注意力时，应直接核对目标版本的参数文档，并用三位置小例子验证上三角确实为零。大负数如 -1e9 也不是所有数据类型都能安全表示；它对全遮挡行还可能产生看似正常的均匀分布，掩盖逻辑错误。

5.7 算力和显存为什么随长度增长很快

普通密集注意力需要比较每个查询和每个键。自注意力的注意力矩阵有 T^2 个元素，B 个样本、h 个头则有 $B\times h\times T^2$ 个权重。长度从 512 增到 1024，权重元素数增长为 4 倍。按朴素算法计，QK 和 AV 的核心算术量合计大致与 $B\times T^2\times d_model$ 成正比，还要加上线性投影与 FFN 的计算。

这不意味着整个模型的运行时间总严格增长 4 倍：硬件利用率、投影、FFN、批大小、内存读写都在影响结果。一些高效注意力实现也能避免完整存储 $T\times T$ 矩阵，但精确密集注意力的两两交互并未凭空消失。先分清“保存多少中间量”与“做多少运算”，以后再学加速方法会清楚得多。

在固定 d_model 下增加头数，若每头宽度相应减小，Q/K/V/O 的主要参数量不会按头数成倍增长。但注意力权重张量的头维变大，每头表示变窄，硬件效率和表示能力也会变化，所以“头越多越强”并无普遍保证。

5.8 小练习 答案与误解纠正

练习 1。 B=3、T=7、d_model=24、h=4，拆头后的 Q、注意力矩阵、拼接后结果各是什么形状？

答案。 每头宽度为 6，Q 为 $3\times4\times7\times6$，注意力矩阵为 $3\times4\times7\times7$，拼接后为 $3\times7\times24$。token 位置数没有变化；头数只切分特征表示与对应注意力计算。

练习 2。 一个长度为 4 的因果序列，第 3 个查询允许访问几个键？为什么允许访问自身？

答案。 按从 1 开始编号，允许前 3 个键。当前位置的输入 token 已知，这个位置的输出被用来预测下一个 token。目标右移与因果 mask 必须配套理解，不能机械地把对角线也遮掉。

练习 3。 为什么已经设置 padding key mask 后，还要在交叉熵中忽略 PAD 标签？

答案。 key mask 只是阻止从 PAD 读取信息，不会删除 PAD 查询位置及其输出。若把这些输出也计入损失，模型仍会为了大量补齐位置学习预测 PAD，改变任务权重。损失应只平均有效标签位置。

练习 4。 怎样检验因果 mask 真的生效，而不仅是查看它的三角形？

答案。 在评估模式下，固定一个输入，将位置 i 之后的 token 改掉，比较前 i 个位置的输出。正确的因果模型应在浮点误差范围内保持它们不变。比较时要关闭 dropout，并保证位置编号和其他输入条件一致。这个测试检查的是实际信息泄漏，比只看 mask 的形状更有力。

需要纠正的三句话。 “每个头负责一个固定语法规则”不是结构保证；“padding mask 会删除补齐位置”不对，它控制的是可见关系；“只要没有 NaN，mask 就正确”也不对，泄漏未来时计算通常完全有限，却训练了错误任务。

第6章 把零件装成一个 Transformer 层

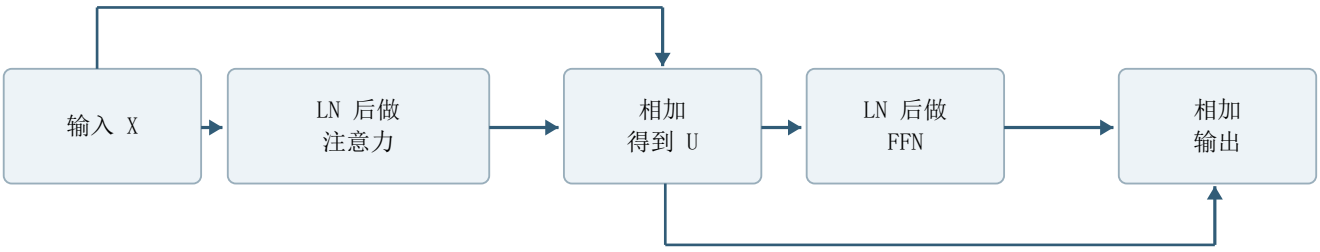


图6 1 Pre LN 模块中的两条残差路径

6.1 学习目标与本章路线

注意力负责让不同位置交换信息，但单独一条注意力公式还不是完整 Transformer。读完本章，你应当能画出一个完整层的数据流，说明前馈网络、残差连接、归一化分别解决什么问题；能区分原始论文的结构与本书实验使用的常见变体；能解释位置编码为什么必要，以及正弦编码与 RoPE 在哪里参与计算。

继续想象一行 token 表示。经过注意力，它吸收了别的位置的信息。下一步，我们还希望它在自己的特征之间做更复杂的变换；同时，深层网络不能在每一层都把已有表示全部丢掉，数值尺度也不能任其漂移。这几个需求分别引出前馈网络、残差连接和归一化。它们是协同工作的设计，而不是每个零件都单独承担一种确定的语言能力。

6.2 前馈网络 在每个位置内部加工特征

位置前馈网络通常简写为 FFN。最基本形式是两次线性变换，中间插入非线性激活：

$$\mathrm{FFN}(x) = \phi(xW_1 + b_1)W_2 + b_2$$

$$W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}, \qquad W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$$

例如 d_model=128、d_ff=512，一个 token 先从 128 维扩展到 512 维，再压回 128 维。对 B×T×128 的输入，最后得到的仍是 B×T×128。中间维度较宽，让网络有空间构造更多非线性特征组合；它不是在 token 轴上增加新词。

“位置前馈”意味着同一层的所有位置共享同一组 W_1、W_2，但每个位置各自计算。FFN 不会在这一步直接读取隔壁 token；不过它处理的输入已经可能通过注意力吸收了上下文，所以也不能说 FFN 的输出与上下文无关。不同 Transformer 层通常拥有各自的 FFN 参数，并不默认共享。

为什么需要非线性？若中间没有激活，两层可合并成一层：

$$(xW_1 + b_1)W_2 + b_2 = x(W_1W_2) + (b_1W_2 + b_2)$$

这样“扩展再压缩”仍只是一个线性映射。ReLU 把负输入变为 0，GELU 则使用较平滑的方式调节输入；加入非线性后，通常不能再把两层压成一个线性层。现代变体还可能使用门控前馈网络，例如一个分支产生内容，另一个分支调节各维通过程度。本教材先掌握两层 FFN，再学习门控设计，不把不同版本混写成唯一标准。

不计偏置时，两层 FFN 的参数量约为 2×d_model×d_ff。若 d_ff=4d_model，则约为 8d_model²；常见标准多头注意力中四个投影的参数量约为 4d_model²。这个估算提醒你：Transformer 的大量参数也在 FFN 中，不应把所有能力和开销都归到注意力。

6.3 残差连接 保留一条直接通路

残差连接把子层的结果与子层输入相加：

$$y = x + F(x)$$

这里 F 可以是注意力子层，也可以是 FFN。相加要求两项形状一致，所以各子层最终通常都回到 d_model 维。它允许子层学习“在现有表示上补充什么”，也保留了直接传递原表示的路径。如果某个位置暂时不需要大改，F(x) 接近 0 就能让输出接近输入。

从梯度看，残差路径提供了恒等项：

$$\frac{\partial y}{\partial x} = I + \frac{\partial F}{\partial x}$$

I 是单位矩阵，可理解为沿直接路径传播的贡献。这有助于深层网络优化，但不能据此宣称梯度永远不会消失或爆炸，因为其他分支、归一化、优化器和数值精度仍有影响。残差也不是“复制旧答案再投票”，它是特征向量的逐元素相加。

常见实现把 dropout 放在子层输出与残差相加之间。训练时随机屏蔽并适当缩放部分元素，减少对特定路径的依赖；评估和生成时关闭随机丢弃。model.eval() 负责切换这类模块行为，torch.no_grad() 负责不记录梯度，二者目的不同，推理时经常一起使用。

6.4 LayerNorm 与 RMSNorm 在哪个维度归一化

对一个 token 的 d 维向量 x，LayerNorm 计算该向量各维的均值与方差，再做归一化和可学习仿射变换：

$$\mu = \frac{1}{d}\sum_{r=1}^{d} x_r, \qquad \sigma^2 = \frac{1}{d}\sum_{r=1}^{d}(x_r - \mu)^2$$

$$\mathrm{LN}(x)_r = \gamma_r \frac{x_r - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta_r$$

ε 是小正数，避免零方差时除以 0；γ、β 是可学习的缩放与偏置。这里方差除以 d，不是统计课上估计总体方差时常见的 d-1。对形状 B×T×d 的隐藏表示，常见 Transformer LayerNorm 沿最后一个特征维独立进行，并不把不同样本或不同 token 的数值混在一起算均值。

用 x=[1,3] 做一次计算，μ=2，方差=1；忽略很小的 ε，设 γ=[1,1]、β=[0,0]，输出就是 [-1,1]。若 x=[5,5]，两维减均值后都是 0，加入 ε 后依旧输出 [0,0]，而不是 NaN。学到的 β 可以让最终结果不再为零均值，所以“经过 LayerNorm 后永远均值为 0”只有在讨论仿射变换之前才合适。

RMSNorm 采用均方根尺度，不先减均值。常见无偏置形式为：

$$\mathrm{RMSNorm}(x)_r = \gamma_r \frac{x_r}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \epsilon}}$$

对 [1,3]，忽略 ε 且 γ 全为 1，分母是 √5，输出约为 [0.447214,1.341641]。它与 LayerNorm 的 [-1,1] 显然不同：RMSNorm 控制均方根尺度，但不移除整体均值。它的设计可追溯到 Root Mean Square Layer Normalization。不要把模型的 LayerNorm 随手改成 RMSNorm 后直接加载旧权重并期待行为保持不变，这通常改变了计算函数。

LayerNorm 与 BatchNorm 也不同。BatchNorm 的常见用法会利用 batch 等轴上的统计量，而 Transformer 的逐 token LayerNorm 对 batch 大小没有同样的统计依赖。理解归一化时，首先问“对哪些轴取均值”，比只记名称更可靠。

6.5 Post-LN 与 Pre-LN 只移动位置也会改变网络

原始 Transformer 采用子层之后归一化的结构，常称 Post-LN。省略 dropout，一个 encoder 层可写成：

$$u = \mathrm{LN}_1(x + \mathrm{MHA}(x)), \qquad y = \mathrm{LN}_2(u + \mathrm{FFN}(u))$$

Pre-LN 把归一化移到子层之前：

$$u = x + \mathrm{MHA}(\mathrm{LN}_1(x)), \qquad y = u + \mathrm{FFN}(\mathrm{LN}_2(u))$$

这不是把同一组运算交换顺序后结果不变。LN 是依赖当前输入统计量的非线性变换，放在相加之前或之后，会改变前向输出和梯度路径。Pre-LN 常在整个堆叠结束后再加一个最终归一化。本书建议从这种容易实现的结构练习，同时在读论文时明确标注作者用的是哪一种。关于两者优化性质的分析可读 On Layer Normalization in the Transformer Architecture；它不是“某一种在所有任务永远更好”的结论。

以下是结构示意，attn 约定返回一个与输入同形状的张量，并使用前一章约定的 mask：

```
# Pre-LN 层的前向结构示意 不是独立完整模型
def forward(self, x, allowed):
    x = x + self.dropout(self.attn(self.norm1(x), allowed))
    x = x + self.dropout(self.ffn(self.norm2(x)))
    return x
```

层堆叠时，第 1 层输出成为第 2 层输入，依此类推。越往上层，并不是向量一定变成某个可命名的概念；可以确定的是它经过了更多次信息交换和非线性变换。不要把“底层语法、中层语义、高层推理”当作普遍适用的硬性架构说明。

6.6 为什么还要告诉模型顺序

“小狗追小猫”和“小猫追小狗”包含同一组词，但顺序改变了意思。没有任何位置信号或顺序限制的自注意力，对输入进行同样的行排列，输出也只会跟着进行同样排列。这叫排列等变性：它能处理一组向量间的关系，却没有额外坐标知道哪个原本在前、哪个原本在后。

更直观地说，若同一 token 的输入向量完全相同，而它周围只是同一批向量换了排法，那么仅靠“与每个向量做匹配并求和”无法读出顺序。因果 mask 本身提供了部分方向与可见性信息，但并不等同于完整的位置表示。标准实践仍通常显式提供位置机制。

一种简单做法是使用可学习位置向量 P_p，输入位置 p 处的表示为词嵌入加位置嵌入：

$$x_p = E[token_p] + P_p$$

词嵌入表达 token 身份，位置嵌入提供位置相关的可学习信号；训练中二者联合适应。相加并不要求每一维永久承担一种人类能解释的含义。预先设置的位置表通常有支持长度，未经处理的超范围位置不能直接索引；位置表里存在某个条目，也不代表模型已经在相应长距离分布上训练充分。

6.7 正弦位置编码 不同速度的波形

正弦编码不训练一个逐位置查表，而是用固定公式生成。对偶数 d_model，位置 p 的第 2i、2i+1 两维为：

$$PE(p,2i) = \sin\left(\frac{p}{10000^{2i/d_{model}}}\right)$$

$$PE(p,2i+1) = \cos\left(\frac{p}{10000^{2i/d_{model}}}\right)$$

这里 p 从 0 开始，i 从 0 到 d_model/2-1。若 d_model=4，四维是 sin(p)、cos(p)、sin(p/100)、cos(p/100)。位置 0 得 [0,1,0,1]；位置 1 得约 [0.841471,0.540302,0.010000,0.999950]。较快变化的维度提供细粒度变化，较慢的维度提供更长尺度变化，组合后形成位置相关的编码。

为什么使用成对的 sin 与 cos？由加角公式，同一频率在 p+Δ 处的两维可以写成 p 处两维的线性组合，组合系数只取决于 Δ。这提供了与相对位移有关的结构。不过，公式能够生成任意 p，并不证明训练后的模型就能可靠处理任意长的输入；数值编码可计算与模型长度外推有效是两回事。

教学实现可以先只支持偶数维，明确限制，避免悄悄漏掉一维：

```
import math
import torch

def sinusoidal_positions(length, dim, device="cpu"):
    if dim % 2:
        raise ValueError("本教学实现要求 dim 为偶数")
    pos = torch.arange(length, device=device).float()[:, None]
    frequency = torch.exp(
        torch.arange(0, dim, 2, device=device).float()
        * (-math.log(10000.0) / dim)
    )
    angles = pos * frequency[None, :]
    pe = torch.zeros(length, dim, device=device)
    pe[:, 0::2] = angles.sin()
    pe[:, 1::2] = angles.cos()
    return pe
```

6.8 RoPE 把位置放进 Q K 的旋转

RoPE 的常见用法不是给输入再加一个位置向量，而是在注意力分数计算前，根据位置旋转 Q 和 K 的二维坐标对。先只看二维，旋转矩阵为：

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

把本小节的向量暂写成列向量。位置 m 的 q 变成 R(mθ)q，位置 n 的 k 变成 R(nθ)k。它们的点积为：

$$(R(m\theta)q)^T(R(n\theta)k) = q^T R((n-m)\theta)k$$

关键在于旋转角从两个绝对位置 m、n 变成了相对差 n-m。高维情况下，把每头特征拆成多个二维对，给不同对安排不同频率，逐对旋转。这使位置关系直接进入 QK 点积；常见实现并不对 V 做同样旋转。旋转不改变单个向量的长度，但会改变处在不同位置的向量之间的点积。

例如 q=k=[1,0]，同位置时点积为 1；若相对角度是 π/2，点积变为 0。这只是展示机制的特殊例子，真实查询与键不一定相同，多频率也会共同作用，不能把它理解为“距离越远权重必定越小”。RoPE 的来源可读 RoFormer。本书的重点是看懂旋转后的点积如何带上相对位置，而非背诵所有长上下文扩展方案。

使用缓存生成时，新 token 的位置编号必须接续历史位置，不能每一步都重置为 0。更改 RoPE 的频率或缩放方式也会改变模型实际看到的位置关系；扩展上下文通常需要配套配置、训练或验证，不能仅把“最大长度”改大就宣称问题已解决。

6.9 小练习 答案与误解纠正

练习 1。 d_model=64、d_ff=256，忽略偏置，FFN 有多少参数？同一层输入长度从 10 变成 20，参数量是否翻倍？

答案。 参数量为 64×256+256×64=32768。长度增加不改变参数量，因为所有位置共享参数；计算量与中间激活量会增加。要区分模型参数和随输入改变的中间张量。

练习 2。 忽略 ε 和可学习仿射变换，对 x=[2,2]，LayerNorm 与 RMSNorm 各输出什么？

答案。 严格计算 LayerNorm 不能忽略 ε，因为方差为零；保留 ε 后输出 [0,0]。RMSNorm 的分母为 2，输出 [1,1]。这道题故意说明“忽略 ε”只适合分母本来非零时的近似推导。

练习 3。 输入 B×T×128 经多头注意力后变成 B×T×64，能直接残差相加吗？

答案。 不能，特征宽度不一致。应通过输出投影恢复为 128 维，或明确另设残差投影。标准教学层保持输入输出同宽，最容易排查形状错误。

练习 4。 给 q、k 同时增加相同的位置偏移 c，RoPE 点积中的相对角会改变吗？

答案。 在相同频率配置下，(n+c)-(m+c)=n-m，因此上述旋转点积的相对位置部分不变。这个性质并不意味着整个模型在所有上下文变化下输出不变，因为 token 内容、可见范围等也可能改变。

需要纠正的三句话。 “FFN 每个位置一套参数”不对，同一层通常共享；“LayerNorm 是把整句压到统一长度”不对，它通常归一化每个 token 的特征；“有了位置编码就能无限长输入”不对，可计算位置值不等于模型具有可靠的长度外推能力。

第7章 原始 encoder-decoder 完整走完一次翻译

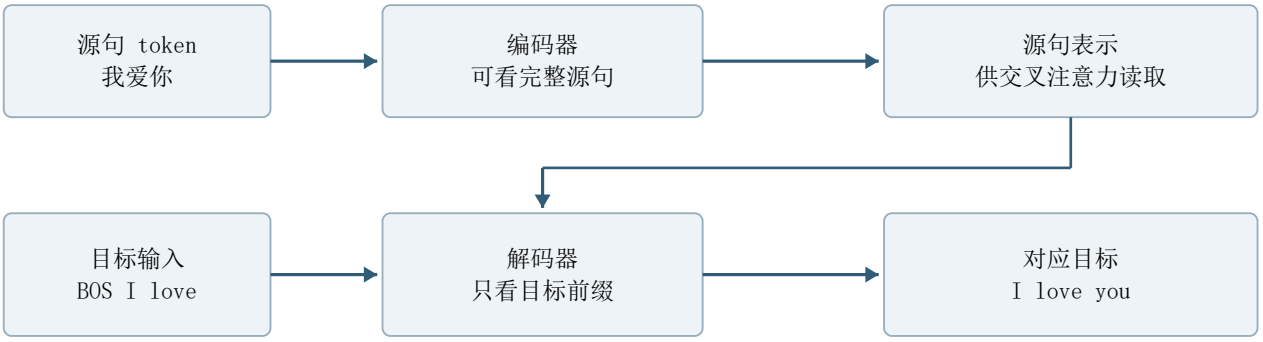


图7 1 编码器解码器的训练输入与目标

7.1 学习目标与一条教学样本

现在将前面的零件连接成一台完整模型。学习目标是：沿一条翻译样本追踪从 token 到概率、从损失到生成的全过程；分清三种注意力的 Q、K、V 来源；用自己画的对齐表解释 BOS、EOS、右移与 teacher forcing。读懂这一章后，你应当能判断一份序列到序列代码是否在训练真正想解决的任务。

教学样本采用如下人为分词，实际模型如何分 token 留到下一章说明：

```
源句：我 / 喜欢 / 茶
目标：I / like / tea
```

我们用 BOS 表示目标序列开始，用 EOS 表示结束，用 PAD 表示批处理补齐。它们是词表里的特殊 token ID，不是需要模型从自然语言中猜出来的空格。具体模型可能采用不同的起始约定，本章固定使用 BOS，只为把逻辑讲清。

7.2 encoder 先把源序列变成上下文表示

源序列经过分词、转 ID、词嵌入、位置处理，成为形状 B×T_s×d_model 的张量。教学样本可把 EOS 加在源句后，T_s=4。然后通过若干 encoder 层，每层都有自注意力与 FFN，并配合残差和归一化。最终得到源端表示 H_src，形状仍是 B×T_s×d_model。

encoder 的每个有效位置通常可以访问源句全部有效位置，因此“喜欢”的表示可以同时利用“我”和“茶”。padding 位置仍须屏蔽。整个源句在翻译开始前已经给定，所以这里查看源句后面的词，并不构成对目标答案的泄漏。

H_src 不等于一个固定长度“句子向量”。这里保留了 T_s 个位置，每个位置都有 d_model 维表示。后面的 decoder 可以根据不同生成阶段，读取不同源位置的组合。对初学者而言，把 encoder 理解成“生成一组可反复读取的上下文表示”，比理解成“把整句压缩成一个数字包”更准确。

7.3 decoder 每层有三项主要计算

原始 encoder-decoder 结构的历史定义见 Attention Is All You Need。在 decoder 中，先对目标前缀做因果自注意力，再用交叉注意力读取源端表示，最后做 FFN。每个子层都有自己的残差与归一化。若沿用 Post-LN 写法，单层示意为：

$$u = \mathrm{LN}_1(z + \mathrm{MaskedSelfAttention}(z))$$
$$v = \mathrm{LN}_2(u + \mathrm{CrossAttention}(u, H_{src}))$$
$$y = \mathrm{LN}_3(v + \mathrm{FFN}(v))$$

decoder 的自注意力先整理“到目前为止已经有的目标端信息”；交叉注意力再根据当前目标表示向源端读取信息。交叉注意力的 Q 来自 decoder 当前子层输入，K 和 V 来自 encoder 最终输出，各自还要经过可学习投影。不是把源句、目标句的矩阵直接逐元素相加。

注意力类型	Q 来自哪里	K、V 来自哪里	通常允许访问哪些位置
encoder 自注意力	当前 encoder 表示	同一 encoder 表示	全部有效源位置
decoder 因果自注意力	当前 decoder 表示	同一 decoder 表示	当前及之前有效目标位置
decoder 交叉注意力	当前 decoder 表示	encoder 最终输出	全部有效源位置

如果目标长度为 4、源长度为 6，decoder 自注意力每头分数为 4×4，交叉注意力为 4×6。交叉注意力不需要沿源位置施加“目标因果三角形”，因为源句已完整给定，源位置与目标位置也不存在一一编号对应。翻译可能改变词序，哪个源位置有用由参数和当前表示决定。

7.4 右移目标 输入与答案差一格

训练时，我们已知完整参考译文，但必须把问题构造成“根据过去预测下一 token”。因此把目标拆成两行：

时间位置	1	2	3	4
decoder 输入	BOS	I	like	tea
监督标签	I	like	tea	EOS
此位置可见的目标输入	BOS	BOS、I	BOS、I、like	BOS、I、like、tea

第 1 个位置读取 BOS 和源句，预测 I；第 2 个位置读取 BOS、I 和源句，预测 like；最后一个位置预测 EOS。输入和标签有相同长度，但错开一个 token。BOS 给第一个预测提供起点，EOS 让模型学习什么时候结束。

从完整 ID 序列 `[BOS, I, like, tea, EOS]` 出发，可以令 decoder 输入为 `full[:, :-1]`、标签为 `full[:, 1:]`。关键不是一定要写这两句代码，而是每个 logit 与它应该预测的下一个 token 正确对齐。有些模型库会在内部右移标签，这时调用者不能再重复移一次。

只右移而不做因果 mask 仍然会泄漏答案。例如预测 I 的第 1 个位置，若可以读到第 2 个输入位置，那里就是 I。只做因果 mask 却不右移，也可能变成“看到当前 token，再预测当前 token”的复制任务。因此要同时核对对齐关系与可见范围。

7.5 teacher forcing 为什么可以同时算很多位置

训练时，把正确历史 token 作为 decoder 输入的做法称为 teacher forcing。模型在预测 tea 时使用参考译文中的 I、like，而不是等自己先生成出 I、like。这样整条目标输入从一开始就已知，各位置可以一次前向传播同时计算。

“同时计算”不等于“每个位置看到了所有答案”。因果 mask 确保第 i 个位置只从合法历史提取信息；第 i+1 个位置虽然也在同一块显卡上计算，但它的内容不能倒流回来。并行是计算安排，可见性是数学依赖，二者并不矛盾。

条件概率的分解把训练目标写得很清楚。源句记作 x，包含 EOS 的目标 token 序列记作 y_1 到 y_T：

$$p(y \mid x) = \prod_{t=1}^{T} p(y_t \mid y_{<t}, x)$$

$$\mathcal{L} = -\frac{1}{N_{valid}} \sum_{t \in valid} \log p(y_t \mid y_{<t}, x)$$

这里 valid 只包含有效目标标签，N_valid 是有效标签数。负对数把多个条件概率的乘积变成可相加的损失。模型若给正确 token 更高概率，这一项损失就更小。teacher forcing 使用正确前缀来计算这些条件概率，正好对应上述最大似然训练目标，并不是不正当的“作弊”。

7.6 从隐藏向量到词表概率

decoder 最终隐藏状态仍是 B×T_t×d_model，不直接是一串文字。输出线性层把每个位置映射到词表大小 |Vocab|：

$$logits = H_{dec} W_{vocab} + b_{vocab}$$

$$W_{vocab} \in \mathbb{R}^{d_{model} \times |Vocab|}$$

每个 logit 是候选 token 的未归一化分数；softmax 后得到词表上的分布。不要把这里“沿词表维的 softmax”与注意力里“沿键位置维的 softmax”混为一谈：前者用于预测 token，后者用于混合上下文。

计算交叉熵时，PyTorch 的相关损失函数接收 logits，并在内部进行稳定计算，通常不应先手动 softmax 再传入。下面只展示对齐好的 logits 与标签如何组成损失：

```
import torch.nn.functional as F

# logits  [B  Tt  vocab_size] labels  [B  Tt]
loss = F.cross_entropy(
    logits.reshape(-1, logits.shape[-1]),
    labels.reshape(-1),
    ignore_index=pad_id,
)
```

这要求 batch 中至少有一个非 PAD 的有效标签。常规数据管道应拒绝空目标或全忽略 batch。若不同样本长度差异很大，按 token 平均与先按句平均再按 batch 平均，给每个样本的权重不同，应根据实验定义保持一致。

7.7 真正翻译时 目标未来还不存在

推理时只有源句，没有参考译文。先计算一次 H_src，目标前缀从 BOS 开始。运行 decoder，取最后一个位置的词表分布，根据某种解码策略选择下一个 token，追加到前缀，再重复。直到生成 EOS，或达到预设的最大生成长度。

```
源句编码一次 → H_src
前缀 [BOS]              → 预测 I
前缀 [BOS, I]           → 预测 like
前缀 [BOS, I, like]     → 预测 tea
前缀 [BOS, I, like, tea] → 预测 EOS → 停止
```

这是理想输出路径，实际模型也可能第一个词就生成错误，之后便是在错误前缀上继续。训练时经常看到正确历史，生成时看到自己的历史，这种分布差异常被称为 exposure bias。认识这种差异有助于解释：验证集 teacher-forced loss 下降，不保证每一条自由生成样例都会更好。应同时观察损失与真实生成质量。

最简单的解码是每一步选概率最大的 token，叫贪心解码。它不保证整句联合概率最大，因为当前最优选择未必导向后续最优序列。束搜索保留多个候选前缀；采样则按分布随机选取，并可能配合温度等控制。模型输出概率，解码策略根据概率选择 token，二者要分别讨论。

初学实现可每次重算整个目标前缀，逻辑简单但有重复工作。更高效的实现保存各层历史 K、V，即 KV cache，让新 token 直接读取过去的键和值；encoder 输出及交叉注意力的源端 K、V 也可复用。缓存节省重复计算，不会让所有未知未来 token 一次同时变成已知。

7.8 一个检查训练代码的五步方法

遇到翻译程序，先挑出一条极短样本打印 ID 对应的 token；其次打印 decoder 输入与标签两行，人工核对错位；再用三角形 mask 检查未来不可见；随后检查 source padding 进入交叉注意力 mask、target padding 进入标签忽略；最后尝试让一个极小 batch 过拟合，再进行自由生成。这个顺序从数据关系开始，通常比一上来盲目扩大模型更有效。

单 batch 过拟合只是管道检查：它说明模型至少有能力记住这几个例子，不能证明泛化。若记不住，应首先查标签错位、优化器是否更新、梯度是否存在、学习率、mask 和损失；若能记住训练样本却不能处理新样本，再讨论数据规模、正则化与模型能力。

7.9 小练习 答案与误解纠正

练习 1。 目标是 We / study / models。写出 decoder 输入和标签，并说明 EOS 出现在哪一行。

答案。 输入为 [BOS, We, study, models]，标签为 [We, study, models, EOS]。EOS 是最后一个要预测的标签；本例不需要把 EOS 放入 decoder 输入再让它预测别的 token。批处理时若还补 PAD，其位置不计入损失。

练习 2。 预测目标第 2 个 token 时，可以读取源句最后一个词吗？可以读取目标第 3 个 token 吗？

答案。 可以读取完整有效源句，因为它是条件输入；不可以读取尚未生成的目标未来。区分源端已知信息与目标端待预测信息，是判断泄漏的核心。

练习 3。 输入、标签都采用 [BOS, I, like, tea]，并用了因果 mask，哪里错了？

答案。 对角线允许读取当前输入，所以第 2 个位置已经看到 I，却又被要求预测 I，任务退化为复制。应当把标签右对齐为下一 token，并让末尾预测 EOS。

练习 4。 源长度为 5，当前目标前缀长度为 3，每头交叉注意力矩阵是什么形状？哪个轴做 softmax？

答案。 形状是 3×5，沿最后的 5 个源键位置做 softmax。每个目标查询形成一组读取源端的分布。不是沿三个目标位置归一化。

需要纠正的三句话。 “encoder 把一句话变成一个向量”并非本章结构；“teacher forcing 让模型看到了当前标签”不对，正确实现只提供先前目标 token；“训练能并行，所以生成能一次得到整句”不对，通常的自回归生成还依赖逐步确定的前缀。

第8章 三条架构路线与 token 的真实面貌

8.1 学习目标 把结构 目标 用途分开

你已经看过完整 encoder-decoder，但很多模型只保留其中一种堆叠。本章要建立一张清晰地图：encoder-only、decoder-only、encoder-decoder 的主要区别是什么；BERT、GPT、T5 分别怎样训练；文字怎样变成 token；为什么训练、处理提示词、生成新 token 的并行方式不相同。

先分开三层问题。架构决定模块如何连接、哪些位置可以读取哪些位置；训练目标决定哪些输出要与哪些标签比较；应用方式决定如何把用户任务转换成模型输入输出。相同模块可以配合不同目标，某种架构也不被永远绑定到一种用途。以下比较的是典型配置，不是不可突破的边界。

8.2 encoder-only 与 BERT 利用两侧上下文

BERT 使用 Transformer encoder 堆叠。原始预训练包含遮蔽语言建模 MLM 与下一句预测 NSP；MLM 选取约 15% 的 token 位置作为预测目标，其中 80% 替换为 [MASK]，10% 换成随机 token，10% 保持原 token，损失计算在被选位置。它通过这种目标学习双向上下文表示。以上概述来自 BERT 原论文，后续模型不一定沿用 NSP 或相同遮蔽方案。

用教材自拟例子说明：小明 喜欢 [MASK]，每天 都 去 泳池，模型可以同时利用空缺左右的线索。这里 mask 指把输入内容替换或扰动，不能与第 5 章的 attention mask 混为一谈。[MASK] 位置通常仍作为有效位置参加注意力，它并不是被整行整列删除。

拿到每个位置的上下文表示后，可以为句子分类连接一个分类头，为序列标注在每个位置连接标签头，为抽取式问答预测答案起止位置。表示如何汇总、训练数据是什么，仍会影响效果。不能因为模型叫 encoder，就认为直接平均其输出必然得到优秀的语义检索向量。标准 BERT 的 MLM 也不等同于左到右生成训练目标，不能把它当成直接替换 GPT 的续写接口。

8.3 decoder-only 与 GPT 在一个序列中继续写

典型 GPT 路线使用带因果自注意力的 Transformer 堆叠，通过先前 token 预测下一个 token。这里的 decoder-only 通常没有第 7 章那套读取独立 encoder 输出的交叉注意力。早期代表可读 Improving Language Understanding by Generative Pre-Training。名称延续了 Transformer decoder 的因果生成特点，不代表完整复制原始翻译 decoder 的全部子层。

在教材的教学任务中，可以把“输入提示”和“待输出文本”放进一个序列。例如：

```
翻译：我喜欢茶。 英文： I like tea EOS
```

模型在预测回答部分时，通过同一条序列的自注意力读取此前提示。若监督微调只希望优化回答，可以在损失里忽略提示位置，但仍让回答位置读取提示。这再次说明“忽略损失”与“禁止注意力读取”不是同一种操作。

因果模型在处理一个已给定的完整前缀时，最后一个 token 的表示可以利用所有更早 token，所以它也能基于一整段输入做分类或判断。不能从“只能看左边”推出“无法理解整句”；准确说法是序列中每个位置的上下文范围受方向限制。能否完成具体任务，还取决于训练和使用方式。

8.4 encoder-decoder 与 T5 把任务写成文本到文本

T5 采用 encoder-decoder，并把多种任务统一成文本输入到文本输出。其代表性预训练方式会移除连续片段，用不同的特殊哨兵 token 标识缺口，decoder 生成缺失片段。参考 T5 原论文。理解这条路线时，最重要的不是模型名字里的数字，而是“encoder 处理输入，decoder 条件生成输出”的分工。

例如，本教材构造一个片段恢复样本：

```
原文：今天 小王 在 图书馆 认真 读书
输入：今天 <extra_id_0> 在 图书馆 <extra_id_1> 读书
目标：<extra_id_0> 小王 <extra_id_1> 认真 <extra_id_2>
```

哨兵 token 告诉模型缺口与输出片段如何对应；不同缺口采用不同标记。这里展示内容组织，不展开具体库内部的 EOS 与 decoder 起始 ID 设置。实际使用时应读取模型配置，不应假设所有模型的开始 token 都叫 BOS。

文本到文本不仅能表示翻译，也能把情感分类写成“输入评论，输出积极或消极”，把摘要写成“输入文章，输出摘要”。统一输出形式方便复用训练管道，但并不让所有任务的评价方法相同：分类可看准确率，生成任务还需要评估内容是否正确、是否遗漏、是否符合要求。

8.5 一张架构对照表

维度	encoder-only 的典型配置	decoder-only 的典型配置	encoder-decoder 的典型配置
代表路线	BERT	GPT	T5、原始翻译 Transformer
主要输入组织	一条或一对输入序列	提示与续写在一个序列中	源序列和目标序列分开
自注意力可见性	通常双向	通常因果	encoder 双向，decoder 因果
独立交叉注意力	通常无	通常无	decoder 读取 encoder 输出
典型预训练目标	遮蔽 token 恢复等	下一 token 预测	条件片段恢复等
常见输出方式	分类、每 token 标签或表示	自回归续写	条件自回归生成
教学重点	利用全句信息建立表示	正确右移与因果依赖	三类注意力与源目标对齐

这张表可用于定位一份模型代码。不要把它读成性能排名，也不要只凭参数量相同就认定哪个模型更好。比较需要固定任务、数据、计算预算和评价方式，架构只是其中一个因素。

8.6 token 不一定是一个词 也不一定是一个汉字

神经网络不直接接收自然语言字符串。tokenizer 先把文本分成离散片段，再映射到整数 ID；嵌入层根据这些 ID 查出可学习向量。可以把完整路径写为：

```
字符串 → token 片段 → token ID → 嵌入向量 → Transformer
```

token 可以是字、词、子词、字节片段或其他离散单位。教学字符模型可能把“学习”拆成“学”“习”；某个真实 tokenizer 可能把它作为一个 token，也可能采用其他片段。英语单词也可能被切成多个子词。不要用一个固定经验比例估计所有语言、所有 tokenizer 的 token 数；需要精确长度时，就用对应 tokenizer 实际编码。

只按完整词建词表会遇到生僻词、新词和拼写变化。子词方法尝试在“词表不能无限大”与“尽量覆盖新字符串”之间折中。BPE 的基本思路是从较小单位开始，逐步合并训练语料中频繁相邻的单位，直到达到某个词表规模或合并次数。常见方法还包括 WordPiece、Unigram；它们不能仅因都叫子词就视为相同算法。进一步阅读可看 BPE 在神经机器翻译中的论文 与 SentencePiece 论文。

用一个人为小语料理解“合并”：假设若干字符串中 a b 经常相邻，训练 tokenizer 时可以把这对合并成 ab；下一轮再统计新的相邻单位。最终编码新文本时，按已学规则使用有限词表组合表示。这里并不需要人为规定每个片段都有独立语义，常见片段只是统计上有用的离散单位。字节级方案可以覆盖任意字节序列，但某些字符可能消耗多个 token；覆盖能力与编码效率也要区分。

token ID 只是编号。ID 为 100 的 token 不比 ID 为 10 的 token“语义大 10 倍”，二者差值也没有语言距离意义。模型利用的是对应嵌入向量。如果模型权重把 ID 100 学成某个片段，而你更换 tokenizer 使它指向另一个片段，即使词表大小不变，语义对应也会错乱。因此预训练权重、tokenizer、特殊 token 配置应当配套使用。

学习型 tokenizer 的拟合数据也应遵守数据划分：自建实验时，用训练集学习词表和合并规则，再编码验证集、测试集，避免提前利用测试文本。使用公开预训练 tokenizer 时，要记录它的来源与版本，而不是把它误称为“只在我这个训练集上学到的”。

8.7 四种经常被混淆的“并行”

第一种是不同样本之间的批并行。无论是 encoder 还是 decoder，多个独立样本通常都可以组成 batch；批处理怎样补齐、是否按长度分组会影响效率。

第二种是已知输入内部的 token 并行。encoder 的输入整段已知，许多位置的线性投影、注意力和 FFN 可以并行计算。teacher forcing 下，decoder 的正确输入序列也已知，即使有因果 mask，单层中的多个位置仍可一起计算。层与层之间通常仍有依赖，第 2 层必须使用第 1 层的结果，不能据此说整座网络所有运算都同时完成。

第三种是提示词处理，常称 prefill。给生成模型一段已知提示，例如 300 个 token，这些 token 可以在一次前向传播中处理，并建立相应的历史缓存。对因果模型，提示内部仍遵守各位置的因果可见性。

第四种是新增 token 的生成，常称 decode。下一步输入包含上一步选出的 token，因此标准自回归生成通常逐步推进。假设用户提示是“今天天气”，模型先选择“很好”，才能基于这个新前缀继续选择后面的 token。如果“很好”恰好是两个 token，则它本身也可能需要两个生成步骤；步骤数取决于 tokenizer，不取决于你眼中有几个词。

阶段	未来目标 token 是否已作为正确输入准备好	常见计算方式
teacher-forced 训练	是，但因果 mask 限制每个位置读取范围	多个位置并行预测、统一算损失
已知提示 prefill	提示本身已知，回答未知	并行处理提示位置并准备缓存
标准自回归 decode	否	每步确定下一个 token，再继续

KV cache 保存各层先前位置的 K、V；它不保存未来的标准答案。缓存增长通常与历史长度线性相关，具体大小还取决于层数、batch、KV 头数、每头维度与数据类型。若不使用缓存，重算前缀会重复大量工作；使用缓存则需要正确维护位置、mask 和缓存顺序。初学时先让无缓存模型正确，再验证缓存结果与其一致，是合理顺序。

8.8 从任务出发选择第一个练习

如果想观察每个输入位置如何结合两侧上下文，可以做一个小型 encoder 分类器或简单遮蔽恢复；如果想亲眼看到模型从一个前缀不断续写，先做小型 decoder 字符语言模型；如果想验证源序列与目标序列长度可以不同，并看到交叉注意力发挥作用，可以做反转序列、复制带变换的序列，再进入翻译。

这些练习的价值在于暴露不同结构问题。字符语言模型不需要先训练复杂 tokenizer，方便把注意力放在损失、mask 和采样；人工序列变换任务能自动生成标准答案，适合验证 encoder-decoder 管道；真实语言任务能进一步暴露数据质量、长尾词、评价歧义等问题。练习模型与成熟大模型的差别不仅在参数规模，还在数据、优化、训练预算与后续训练流程。

8.9 小练习 答案与误解纠正

练习 1。 BERT 的 [MASK] 与因果 attention mask 是同一个东西吗？

答案。 不是。前者是输入内容中的特殊 token，用来构造恢复任务；后者是注意力计算中的可见性规则，用来禁止读取某些位置。[MASK] 位置通常仍然能读取其他有效位置。

练习 2。 为什么 decoder-only 没有独立 encoder，也能按提示翻译？

答案。 可把源句、任务指令与目标输出组织在同一序列中；目标生成位置通过因果自注意力读取此前源句和指令。是否学会翻译还取决于训练，而非输入格式本身就保证成功。

练习 3。 换一个词表大小相同的 tokenizer，能否直接沿用旧嵌入？

答案。 通常不能直接认为等价。形状相同只满足计算要求，ID 与片段的语义对应可能完全不同。必须维护匹配关系，或进行明确的词表映射、参数处理与再训练验证。

练习 4。 训练时一次产生 100 个位置的 logits，是否证明推理时可以一次生成 100 个未知 token？

答案。 不能。训练输入中已有正确历史 token；生成时这些历史必须由前面步骤确定。标准自回归模型需要逐步选择新 token。特定的加速或非自回归方法会改变计算流程，但不能从常规训练并行直接推出它们。

练习 5。 一个生成模型输出句子正确，但你报告的 token 准确率与另一个模型不可直接比较，可能是什么原因？

答案。 两个模型可能使用不同 tokenizer，同一句话的 token 边界、数量和编号不同；还可能采用不同对齐、忽略规则或数据集。应先统一比较定义，生成任务通常还需要文本层面的评价。

需要纠正的三句话。 “token 就是词”不对；“decoder-only 不能处理分类”不是架构定律；“Transformer 的所有计算都并行”不对，要分别考虑样本、序列位置、网络层与未知输出步骤之间的依赖。

第 9 章 从零实现一个小型 Transformer

本章目标：把公式对应到一段能读懂的程序。学完后，你应能说明输入是什么、每一步张量形状如何变化、模型为什么不能看到未来，以及每类参数从哪里得到学习信号。

9.1 我们究竟要造什么

我们要造一个“读前面的字，猜下一个字”的小模型。输入“小明在读”，模型给词表中的每个字符一个分数；训练希望“书”的分数相对较高。模型本身不会直接输出一句完整的话，也没有一个装着标准答案的数据库。它首先输出一组数，程序再把这组数转成下一个字符，反复调用模型才形成续写。

这个模型属于 decoder-only Transformer：由带因果约束的自注意力层和前馈网络组成，没有单独的编码器，也没有读取另一段编码结果的交叉注意力。这里保留字符嵌入、位置嵌入、多头注意力、残差连接、LayerNorm、GELU 和语言模型输出层。它采用 Pre-LN，即进入注意力或前馈网络之前先做归一化。它是帮助理解机制的教学变体，不等于逐行复现 2017 年原始编码器-解码器架构。

字符是最直观的 token。例如“小明读书。”可变成五个整数。真实大模型常使用子词等分词方式；本实验先把分词器这个变量拿开，让你集中看网络。整数只是索引：“书”被编号为 20、“水”被编号为 21，并不意味着二者比编号为 200 的字符更相似。相似性要在训练得到的向量中观察。

配套文件都在教材目录的 配套实验/ 中。先读 attention_handcalc.py，再读 data.py 与 tiny_transformer.py，最后读 train.py。初学时不要试图一次记住所有代码；先沿着一批输入追踪张量，就能把网络理解为一条明确的数据通路。

9.2 先把数据做对 文档 词表 窗口

配套 corpus.py 提供本教材原创的二十篇微型短文，其中十六篇用于训练、四篇用于验证。训练原文共有 1397 个字符，验证原文共有 368 个字符。它们只是方便运行的玩具数据。语言风格相近、数量极少，即使模型生成了一个通顺句子，也不能证明它具备一般中文能力。

先按完整文档划分，再从文档内部取窗口。假设一篇文章产生了几百个高度重叠的小片段，如果取完片段才随机分成训练和验证，模型可能在训练中看过验证片段的大部分内容，验证分数就会过于乐观。本实验的窗口不跨越文档边界，完整文档也不会同时出现在两份数据中。不过，文档不重叠并不保证语义独立；这些短文共享人物和句式，结果仍需要谨慎解释。

CharTokenizer 只读取训练文本来建立词表，把训练中出现过的字符和换行排序，再分配编号。编号 0 保留给未知字符，解码时显示为“□”。这样，验证集中的新字符不会被提前用于决定词表。本例词表共有 358 类，验证原文中有 5 个训练词表外字符。把不同新字符折叠到同一个未知类别会损失信息，这也是本实验不能用来比较真实语言模型成绩的原因之一。

设上下文长度为 T=4。从一篇文档中取连续五个字符“小明在读书”，构造：

```
原片段：小 明 在 读 书
输入 x：小 明 在 读
目标 y：明 在 读 书
```

输入位置 0 对应目标“明”，位置 1 对应目标“在”。每个位置都贡献一次预测任务，这能充分利用训练样本。程序在数据端执行 x=sequence[:, :-1]、y=sequence[:, 1:]。**右移对齐只做一次。**如果又在模型里切掉一位，就会错把“当前字”对到“后面第二个字”。模型代码明确要求传入的 targets 已经完成对齐。

一批有 B 个窗口，每个窗口 T 个输入字符，张量 x 的形状是 [B,T]，内容类型是整数索引。本例默认 B=16、T=32。不同样本长度相同，因此不需要 padding；这不是说 Transformer 永远不需要 padding，而是我们为第一份实现主动选了更简单的数据组织方式。

9.3 嵌入与位置 先给每个字符一张数值名片

nn.Embedding(V,D) 可以理解为一张有 V 行、D 列的可训练表。给它某个字符的编号，就取出对应的一行。V 是词表大小，D 是模型宽度。默认 D=64，所以一批输入经过字符嵌入后变为 [B,T,64]。这张表开始时是小随机数，随后和其他网络参数一起更新。

模型还需要知道字在哪里。本实验用另一张可训练表表示位置 0 到 T-1：

```
positions = torch.arange(length, device=token_ids.device)
x = token_embedding(token_ids) + position_embedding(positions)
```

位置嵌入的形状是 [T,D]，相加时自动扩展到每个批次。两个向量相加后，结果仍是 [B,T,D]。同一个字符出现在不同位置，会得到不同的初始表示。位置表中的数也由训练学得，不能因为代码能接收整数 1000，就假定只训练了 32 个位置的模型懂得位置 1000。

这里的两张嵌入表是全模型共享的，不是每一层重新查一次原始字符。第一层得到嵌入表示，后面的层则逐层处理已经包含上下文信息的表示。

9.4 多头注意力 按形状读懂实现

默认有 H=4 个头，每头宽度 Dh=D/H=16。因此 D 必须能被 H 整除。多头不是四份完整的 64 维向量各算一次，而是把投影后的总宽度分成四个 16 维子空间。也不要预先给头命名为“语法头”“事实头”；它们是否形成某种可观察模式，要通过训练和分析判断。

首先，用一个线性层同时产生 Q、K、V，再切开：

```
q, k, v = qkv(x).chunk(3, dim=-1)
q = q.reshape(B, T, H, Dh).transpose(1, 2)
```

qkv 的输出形状为 [B,T,3D]，切开后各是 [B,T,D]。重排后，各自成为 [B,H,T,Dh]。K 和 V 做同样的重排。这里没有把字符顺序打乱，只是把“头”单独放到一个维度，方便一次并行计算所有头。

接着计算 scores=q @ k.transpose(-2,-1) / sqrt(Dh)。转置只交换 K 的最后两个维度，使 [T,Dh] 与 [Dh,T] 相乘，结果是 [T,T]；带上批次和头就是 [B,H,T,T]。scores 中第 i 行、第 j 列，是位置 i 的查询与位置 j 的键之间的匹配分数。它不是概率，可以是负数，也不要求总和为 1。

对象	默认示例形状	这个维度表示什么
字符编号	[16,32]	十六个样本，每个三十二个字
隐藏表示	[16,32,64]	每个字一个六十四维向量
分头后的 Q/K/V	[16,4,32,16]	四个头，每头十六维
注意力分数与权重	[16,4,32,32]	每个查询位置看哪些键位置
各头输出	[16,4,32,16]	加权汇总的值向量
合并后的输出	[16,32,64]	回到模型统一宽度

在分数上应用因果遮罩后，沿最后一维做 softmax。**最后一维是“当前查询可以看哪些键”**，因此每一行归一化。若误写成沿查询维度归一化，程序可能仍能运行，但算的已不是所需注意力。

之后是 `weights @ v`：每个查询位置按它的权重汇总各位置的值向量。把头维度换回去、拼成 D 维，再乘输出投影矩阵，就得到与输入同形状的结果。转置后张量的内存布局可能不连续，所以代码使用 `contiguous().view(...)` 合并维度，避免把“看起来形状合适”误当成“底层布局一定合适”。

9.5 因果遮罩 提前封住偷看答案的路

位置 i 可以看 0 到 i，包括自己；不能看 i+1 及以后。自己不算未来，因为位置 i 的任务是根据已见到的 x_i 去预测 x_(i+1)。默认采用如下可见性矩阵：

```
允许看见：
1 0 0 0
1 1 0 0
1 1 1 0
1 1 1 1
```

不能看的分数在 softmax **之前**改成负无穷，指数之后它们贡献 0。不要在 softmax 之后简单把部分权重清零就算结束：那样剩余权重通常不再和为 1。也不能只把未来分数置成普通的 0，因为 exp(0)=1，它们依然会得到正权重。

```
scores = scores.masked_fill(~allowed[:T, :T], float("-inf"))
weights = torch.softmax(scores, dim=-1)
```

本实验的 `allowed=True` 表示可见，取反后交给 `masked_fill`。迁移到库接口时必须重新核对语义：PyTorch 的 `scaled_dot_product_attention` 中布尔 `attn_mask=True` 表示允许；`nn.MultiheadAttention` 的布尔 `attn_mask=True` 和 `key_padding_mask=True` 表示屏蔽。两者不能直接照搬同一张布尔表。SDPA 官方文档、MultiheadAttention 官方文档

此外，不能把某一整行都屏蔽掉。普通 softmax 面对全是负无穷的行，可能产生未定义数值。这里每个位置至少允许看自己，因此不会出现空的可见集合。

9.6 残差 归一化与前馈网络

注意力负责跨位置汇总信息；前馈网络则在每个位置上，用共享的一套参数处理该位置的特征。本例 FFN 先把宽度从 D 扩到 4D，经过 GELU 非线性，再缩回 D。若没有中间非线性，两次线性映射可以合成一次线性映射，就无法获得这层非线性变换的表达能力。

默认 D=64，因此前馈路径是 64→256→64。每个位置计算时使用相同权重，但输入不同，所以结果不同。FFN 本身不直接读取别的位置；它能利用上下文，是因为输入表示已经通过注意力吸收了上下文。

本例一个 Block 的完整骨架只有两行：

```
x = x + attention(layer_norm_1(x))
x = x + ffn(layer_norm_2(x))
```

外面的 `x + ...` 是残差连接。它保留一条直接传递已有表示的路径，让子层学习对表示的补充。残差两端必须有相同形状，这也是注意力合头后和 FFN 最后都回到 D 维的原因。

LayerNorm 在本例中针对每个样本、每个位置的 D 个特征做归一化，再用可学习缩放和平移调整结果。它不把整段序列混在一起求均值，所以不会通过统计未来位置而破坏因果性。Pre-LN 是一种便于训练的常见组织方式，但它不能自动解决学习率过大、坏数据或错误 mask 等问题。

Dropout 在训练时随机丢弃部分连接贡献，评估时关闭。本例既对注意力权重使用 dropout，也在子层输出上使用 dropout。要注意，softmax 后、dropout 前的权重每行和为 1；某次经过 dropout 的随机结果不要求仍然恰好为 1。测试返回的是 dropout 前权重。

9.7 从隐藏表示得到下一字分数

经过两个 Block，再做最后一次 LayerNorm，隐藏表示仍然是 [B,T,D]。输出线性层把 D 维映射为 V 维，于是得到 [B,T,V] 的 logits。logit 是尚未归一化的实数分数；某个字符的绝对分数本身没有“百分之几”的意义，概率要由同一位置所有字符的相对分数决定。

训练时保留所有位置的 logits，因为每个位置都有目标。生成时一般只取最后一个位置的 logits，因为我们想知道已有前缀后面该接什么。不要把这两种使用方式混淆：训练并不是每次只利用一句话的最后一个字。

本实现没有把输入嵌入和输出层权重绑定，也没有 KV cache、旋转位置编码或混合精度。这些都可以后续学习。眼下最重要的是你能在代码中指出哪一行负责跨位置通信、哪一行制造非线性、哪一行防止偷看未来。

9.8 用性质检查 胜过只看“能运行”

test_model.py 检查输出形状、损失是否为标量、所有参数梯度是否有限、一次优化是否改变权重、生成能否跨过上下文窗口。最关键的检查是因果性：给模型两段长度相同且前五个 token 一样的输入，只修改第五个位置之后的内容；在关闭 dropout 后，前五个位置的输出应保持一致。再将后面所有 token 删掉，前缀输出也应一致，允许很小的浮点误差。

这种测试比“损失能下降”更有针对性。一个偷看答案的网络，训练损失可能下降得尤其快。检查还会确认注意力上三角恰为 0、每行权重和为 1，并核对数据端右移对齐。先通过这些检查，再相信训练曲线。

本章练习

1. B=8、T=20、D=96、H=6 时，分头后的 Q 和注意力权重分别是什么形状？

2. 为什么位置 i 可以看到输入 x_i，却不能看到 x_(i+1)？

3. 把 FFN 中间的 GELU 删除，且暂时忽略 dropout，会丢失什么能力？

4. 一个模型能正常反向传播，是否说明 mask 正确？写出更可靠的检查。

5. 输入“春天来了”，希望预测后续字符时，为什么训练通常不应设 targets=input？

参考答案:

1. 每头宽度为 96/6=16。Q 是 [8,6,20,16]，权重是 [8,6,20,20]。

2. 该位置的目标是下一字符 x_(i+1)。x_i 已经是当前前缀的一部分，x_(i+1) 则是正在预测的答案。

3. 两个线性层的组合仍是仿射变换，失去中间非线性提供的表达能力；扩大再缩小维度本身不能代替激活函数。

4. 不能。应在 eval 模式中固定前缀、改动未来输入，比较前缀 logits；也要检查权重矩阵的未来区域为 0。

5. 因果遮罩允许看到当前位置。若目标就是当前位置的字符，网络容易学到复制输入，而非预测下一字符。应取连续 T+1 字符，前 T 个作为输入、后 T 个作为目标。

第 10 章 让模型学习 并让它逐字生成

本章目标：理解损失如何把“猜错”变成参数更新，区分训练与评估，并读懂温度、top-k 和 KV cache 的作用。完成本章后，你应能解释一条训练日志，而不仅是按下运行按钮。

10.1 交叉熵 把正确答案的概率变成学习信号

假设下一字只有“书、水、树”三种候选，模型给出的概率分别是 0.7、0.2、0.1，真实答案是“书”。该位置的交叉熵为 $-\ln(0.7)\approx 0.357$。若模型只给“书”0.1 的概率，损失就变为 $-\ln(0.1)\approx 2.303$。概率越偏离正确答案，惩罚越大；把概率从很小提高一点，损失可能明显下降。

真实模型先给 logits z，再计算 $p_j=\exp(z_j)/\Sigma_k \exp(z_k)$。将 softmax 与负对数合起来，可以写成 $\text{loss}=\text{logsumexp}(z)-z_{\text{target}}$。程序使用框架提供的数值稳定实现，避免自己先求很大的指数再取对数。

因此代码把**原始 logits** 传给 F.cross_entropy，而不是先 softmax 后再传进去。目标是正确字符的整数编号，类型为长整数。把 [B,T,V] 的 logits 展成 [B×T,V]，把 [B,T] 的目标展成 [B×T]，就得到 B×T 道分类题的平均损失。这只改变形状，不再移动标签位置。PyTorch 交叉熵文档

对某一位置，交叉熵关于 logit 的梯度有一个有用结论：$\partial \text{loss}/\partial z_j=p_j-1(j\text{为正确类别})$。正确类别对应的梯度通常为负，梯度下降会倾向提高它的 logit；错误类别对应正梯度，会倾向降低它们的 logit。网络内部的链式法则把这些信号继续传回输出层、注意力、FFN 和嵌入表。实际共享参数会同时影响很多位置，因此不能把一次更新理解成只改单个答案的分数。

本实验共有 V=358 个类别。假如每次都均匀猜测，正确字概率为 1/358，交叉熵就是 $\ln(358)\approx 5.8805$。这个数是可计算的基线，不是本次训练测得的结果。随机初始化模型的实际损失一般接近而不严格等于它；训练能否优于基线需要运行验证。

10.2 一次训练更新 究竟发生了什么

设参数集合为 θ，模型前向计算得到损失 L。训练希望找到使平均损失更小的参数，但不知道直接跳到哪里合适，因此先用梯度估计局部变化方向，再迈出一小步。最简单的梯度下降写成 $\theta \leftarrow \theta-\eta\nabla L$，$\eta$ 是学习率。学习率不是“学习能力”：过大可能震荡或发散，过小可能进展缓慢。

配套代码中每一步包含以下过程：

```
optimizer.zero_grad(set_to_none=True)
logits, loss = model(x, y)
loss.backward()
clip_grad_norm_(model.parameters(), 1.0)
optimizer.step()
```

第一行清理上次保存的梯度，因为 PyTorch 默认会累加梯度。第二行根据当前参数算预测与误差。第三行做反向传播，计算损失对参数的导数。第四行限制梯度整体范数，避免某次异常大更新；它不能修复错误数据或错误公式。第五行才真正修改参数。调用 backward() 本身并不会把模型变得更好，参数更新发生在优化器执行时。

训练使用 AdamW。可以先把它理解为：除了当前梯度，优化器还维护梯度的一阶和二阶移动统计量，以调整每个参数的更新尺度；权重衰减则以独立方式使参数适度收缩。本实现对矩阵参数使用衰减，对 bias 和 LayerNorm 的一维参数不衰减。不要把 AdamW 误认为“自动选好学习率”，它依然需要设置学习率和其他超参数。AdamW 官方文档

每步只用一小批随机窗口，而不是遍历全部训练数据。日志中的 step 指优化器更新次数，不等于 epoch。由于有放回抽样，同一个窗口可能被抽到多次，也可能暂时没有被抽到。固定随机种子能使同一环境中的实验更容易复现，但软件版本、设备和某些运算差异仍可能使结果不完全相同。

10.3 为什么训练可以一次处理全部位置

训练时，正确文本已经知道，整段输入可以一次送入模型。通过因果遮罩，各位置只能用自己的前缀，却能同时计算各自的下一字预测。这通常称为 teacher forcing：输入前缀来自真实训练文本，而非来自模型上一步随手生成的字符。

例如训练输入“小明在读”，四个位置分别预测“明、在、读、书”。第四个位置可以使用前三个位置的信息，但第一个位置不能越过遮罩去看到“明”。所以“并行计算多个位置”并不等于“允许偷看未来”。

生成时没有后续真值，必须先生成一个字，把它接到输入里，再预测下一个字。因此生成过程具有逐步依赖。一个早期错误可能改变后面的上下文，进而连续影响后续输出。这解释了为何较好的训练损失并不保证任意长的自由续写都可靠。

10.4 评估要同时处理模式 梯度和数据

model.train() 把模型切换到训练模式，model.eval() 切到评估模式。本例最直接的差别是 dropout 是否工作。eval() 不会自动禁止梯度记录，也不会冻结参数；它只调整那些区分训练与评估的模块行为。Module.eval 文档

评估中还使用 torch.no_grad()，避免为不需要反向传播的计算保存梯度图。它也不会自动关闭 dropout。因此你通常会同时看到 model.eval() 和 with torch.no_grad():。评估完成后，代码恢复原先的训练模式，防止后续训练一直处于关闭 dropout 的状态。no_grad 文档

本实验训练 CE 和验证 CE 都在固定窗口、eval 模式下计算。这样两者更容易比较。如果拿“开启 dropout 的随机训练批次损失”和“关闭 dropout 的验证损失”硬比，差异中混入了评估规则变化。这里按每个 batch 的目标 token 数加权汇总，而不是不管 batch 大小直接平均若干个 batch 均值。

评估窗口步长等于上下文长度，所以所计算的目标字符不重叠；每篇文档的不足整窗尾段会被舍去，文档第一个字没有前缀也不作为目标。报告的未知字符比例则针对全部文档含后加换行，二者统计范围不完全相同。开始认真比较实验时，这些细节都要写进记录。

困惑度 PPL 定义为平均交叉熵的指数：PPL=exp(CE)，这里对数底为自然底数 e。若 CE=2.0，则 PPL≈7.39。可以粗略理解为模型的平均不确定程度，但不要把它解释为“每次恰好在 7.39 个字中选择”。不同词表、分词方式、数据分布或评估范围下的 PPL 不能直接比较。这个字符模型的 PPL，也不能与子词大模型排行榜上的数字直接比较。

10.5 按顺序运行 先流程 再学习 再续写

进入 配套实验/ 目录，在安装好 PyTorch 的 Python 环境中执行：

```
python test_model.py
python train.py --smoke-test --device cpu
python train.py --steps 400 --device cpu --seed 42 --save outputs/model.pt --log outputs/train.csv
```

第一条运行结构检查。第二条使用更小配置训练三步，确认从数据到反向传播、评估和生成的流程能走通。第三条才进行较长的教学训练，保存最后一步模型和评估 CSV。四百步不是保证成功的神奇数字，三步冒烟训练也不要求输出通顺。运行速度由你的电脑决定。

模型文件保存配置、词表、参数、训练步数和种子；它不包含优化器的完整状态，所以不是用于精确续训的检查点。文件保存的是最终一步，而非自动选择的最佳验证步。读取它续写可运行：

```
python generate.py --checkpoint outputs/model.pt --prompt 小明 --max-new-tokens 80 --temperature 0.8 --top-k 20
--device cpu
```

配套 验证记录.md 记录本次交付真正运行过的检查、实际环境以及实测范围；请以它为准，区分静态语法检查、结构测试和真实训练。你的电脑、依赖版本和实验设置可能不同，不应把别人的运行数字当作自己的结果。你运行后产生的 CSV 才是自己的实测记录，具体准备方式见附件 README.md。

10.6 生成 模型给概率 程序决定怎么选

生成时，先把提示词变成整数，得到 [1,T] 输入。取最后位置 logits，调整采样策略，选出一个新字符，再拼接到提示后面。重复八十次，就新增八十个字符。输出中若出现“□”，表示选中了未知类别或输入中含训练词表外字符；它不是一个神秘的特殊语义符号。

最简单的选择方式是贪心：每次取分数最大的字符。它容易复现，也可能陷入重复。随机采样按概率抽取，有时更丰富，也可能选到不合适的字符。二者改变的是如何使用已有模型，并没有重新训练它。

温度 τ 在 softmax 前把 logits 除以 τ。τ 小于 1 时分布更尖，常见候选更占优势；τ 大于 1 时分布更平，低概率候选更可能被选中。例如两个 logits 为 2 和 1，温度 1 时前者概率约 0.731；温度 0.5 时约 0.881；温度 2 时约 0.622。温度改变概率分布，不改变字符原有的分数排序。代码要求温度严格大于 0，不能直接用除以 0 实现贪心。

Top-k 先保留分数最高的 k 个候选，把其余候选的 logit 设为负无穷，再在保留集合中重新归一化。top-k=0 在本程序中特别约定为不使用截断。top-k=1 则只剩一个候选，效果等同于贪心选择。对同一个模型同时改变温度与 k，会混入两个因素；做比较时先固定一个，只改另一个。

配套代码没有显式结束 token，按指定新增长度停止，换行只是普通字符。超过上下文长度后，模型只使用最近 T 个字符，并把这段窗口位置重新编号为 0 到 T-1。这是为了让最小程序持续生成的简化方式，会丢掉更早的信息；不代表模型自动掌握了超长上下文，也不等价于完整历史上的持续位置编码。

10.7 KV cache 为什么能加速 又为什么占内存

当前程序每生成一个字，都重新计算整个保留前缀的各层表示。因果性告诉我们：对于同一历史位置，在模型、位置编号和上下文不变且关闭 dropout 时，后来新增的字不会改变它已经算出的键和值。因此可以把过去的 K、V 保存起来，下一步只为新字计算新的 Q、K、V，再用新查询读取历史缓存。这就是 KV cache 的基本想法。

若已有 L 个位置，单层普通自注意力的整段分数矩阵有 L×L 项，主要注意力乘法量约为 $O(L^2D)$。使用缓存后的一个解码步，只需要新查询与 L 个键匹配，注意力部分约为 $O(LD)$；新 token 的线性投影与 FFN 仍需计算，不能说所有成本都变成线性的常数。首次读取长提示的 prefill 也仍需要处理整段提示。

标准多头注意力中，缓存两份向量 K、V，每层、每个样本、每个历史位置各保存 D 个数，所以缓存元素数约为 2×层数×B×L×D。例如本例两层、批量 1、L=32、D=64，共 8192 个元素；若每个元素占 4 字节，仅这部分约 32 KiB。真实模型层数、宽度和并发批量更大时，这部分可能很可观。缓存用内存换重复计算的减少，不会让模型本身更聪明。

本代码没有实现缓存，也不能在“丢弃旧窗口并把位置重置”后原样复用所有旧缓存，因为位置变了，对应表示也会改变。先把普通生成读懂，再学习缓存的位置管理、滑动窗口和不同注意力结构，会更容易避免隐蔽错误。

10.8 看训练曲线时 先提出可检查的解释

训练损失下降、验证损失也下降，说明模型在当前评估规则下对这些验证文本的预测有所改善；不是“理解能力全面提高”的直接证据。训练损失继续下降、验证损失转而上升，常见解释是过拟合：模型越来越适应训练样本，但没有同样改善新样本预测。

若两者都不下降，先检查输入与标签是否对齐、输出是否真的是 logits、优化器是否包含模型参数、是否调用了 step()、学习率是否合理。若出现 NaN 或 Inf，先查是否有整行被 mask、学习率是否过大、数据和梯度是否有限。先排除实现错误，再增加模型规模。

若损失异常地接近零，也要检查是否泄漏答案：训练验证是否共享文档、是否缺少因果遮罩、targets 是否误设为 input。低得惊人的损失不是跳过检查的理由。

若输出不停重复某个片段，可能与语料很小、模型过拟合或采样过于集中特征有关。先保留同一模型，比较固定提示下的几组采样；再结合验证曲线判断是否值得调整训练。不要凭一个“看起来不错”的样本挑选结果，也不要凭一个坏样本就断定全部训练失败。

验证集可以帮助选模型，但反复根据它改配置之后，它就参与了研究决策。正式结论还需要未参与调参的测试集。本教材只有训练与验证两部分，适合练流程，不给出独立测试成绩。保留配置、版本、种子、指标和失败记录，比只保存最漂亮的续写更有价值。

本章练习

1. 真实类别的概率从 0.25 提高到 0.5，单位置交叉熵分别是多少？

2. 为什么 loss.backward() 之后还要 optimizer.step()？为什么下一批之前要清理梯度？

3. 验证时只写 torch.no_grad()，模型仍是 train 模式，会有什么问题？

4. 一个字符模型 PPL=20，另一个子词模型 PPL=10，能直接判断后者更好吗？

5. 每次生成都截取最后 32 字并重置位置时，能直接复用按旧位置得到的 KV cache 吗？

6. 训练 CE 从 3.0 降至 1.0，验证 CE 从 3.2 升至 3.5，下一步最应该先记录和检查什么？

参考答案:

1. 分别为 $-\ln(0.25)\approx1.386$ 和 $-\ln(0.5)\approx0.693$。正确类别概率更高，损失更小。

2. backward 计算并存储梯度，step 用梯度更新参数。默认梯度会累加，因此普通独立 batch 训练应先清理；有意做梯度累积则另有安排。

3. Dropout 仍然随机工作，评估具有额外噪声，且与实际推理模式不一致。no_grad 管梯度记录，eval 管模块模式。

4. 不能。两者的 token 单位与预测问题不同，还需要一致数据和可比评估定义。

5. 不能直接这么做。位置编号变化会改变位置嵌入和后续表示，旧缓存不再对应新窗口的计算。

6. 保留两条完整曲线和配置，检查划分与对齐后考虑过拟合；比较较早训练步的验证结果，避免只因训练 CE 更低就继续加步数。由于验证集极小，也要承认估计存在波动。

第11章 从原始 Transformer 走向现代模型

本章是基础课程的延伸地图。你不必一次记住全部名称；先弄清楚每项改动解决什么问题，再挑一个动手验证。下面介绍的是几类已经形成清晰原理的代表性方法，不是对全部前沿工作的罗列。

11.1 先把改动放回正确的位置

现代模型往往仍然重复“注意力、前馈网络、残差连接、归一化”这些模块，但具体实现可以不同。读一份模型配置时，先问五件事：token 怎么表示？位置信息放在哪里？每一层怎样计算？怎样训练？怎样生成？不要把所有新名词都当作一种全新的 Transformer。

改动位置	代表方法	主要问题
位置表示	RoPE	让注意力分数携带相对位置关系
层内计算	RMSNorm、SwiGLU	调整归一化和前馈网络
自回归推理	KV cache、MQA、GQA	减少重复计算或缓存开销
注意力实现	FlashAttention	减少大矩阵在显存中的搬运
适配与部署	LoRA、蒸馏、量化	分别减少微调参数、迁移知识、降低数值存储精度
外部信息	RAG	先检索资料，再依据资料生成

11.2 RoPE 给查询和键加上旋转

假设一个头的维度为偶数。把查询向量的相邻两个坐标看作二维向量，对第 m 个位置施加角度 mθ 的旋转：

```
q'_m = R(mθ) q_m
k'_n = R(nθ) k_n
R(a) = [[cos a, -sin a], [sin a, cos a]]
```

每对坐标使用不同的 θ。关键不是“把向量转得好看”，而是旋转后点积中出现相对位移：

```
(R(mθ)q)^T(R(nθ)k) = q^T R((n-m)θ) k
```

为什么？旋转矩阵的转置等于反方向旋转；先转 -mθ、再转 nθ，就等于转 (n-m)θ。于是，把同一对内容从位置 (2,5) 一起平移到 (12,15)，这一对旋转带来的相对位置关系不变。这是代数性质，不是说整张网络的输出必然不变。

常见语言模型实现把 RoPE 用在投影后的 Q、K 上，而不是简单地加到输入向量上。它保留了每个二维子向量的长度，但改变点积中的位置关系。RoPE 可以为训练范围之外的位置计算旋转角度，**这并不保证模型能可靠处理任意长文本**：外推仍取决于训练分布、频率设置和后续适配。原理见RoFormer 原论文。

11.3 RMSNorm 与 SwiGLU 注意力之外同样有设计空间

LayerNorm 会减去均值，再除以标准差。RMSNorm 常用的形式是：

```
rms(x) = sqrt(mean(x²) + ε)
yᵢ = gᵢ • xᵢ / rms(x)
```

其中 g 是可学习缩放，ε 避免分母为零。RMSNorm 不减均值。对向量 [3,4]，均方根约为 3.536；在 g=[1,1]、忽略 ε 时，输出约为 [0.849,1.131]。输出均值并不为零。因此，“用了归一化就一定中心化”是错误的。论文讨论了这种简化，但不能据此推断你的任务一定更快或更准。RMSNorm 原论文

普通前馈网络可以写作 $\phi(xW_1)W_2$。一种不含偏置的 SwiGLU 写法是：

```
SwiGLU(x) = [SiLU(xWg) ⊙ (xWu)] Wd
SiLU(z) = z • sigmoid(z)
```

⊙ 表示逐元素相乘。一条分支生成内容，另一条分支调节各通道的通过程度。“门”只是理解方式；SiLU 的输出不被限制在 0 到 1。若隐藏宽度同为 f，普通两层前馈的主要参数约为 2df，SwiGLU 三个矩阵约为 3df。公平比较时应缩小后者的隐藏宽度，例如用约 2f/3，使参数量接近；否则效果变化可能来自更大的模型。GLU Variants 原论文

11.4 KV cache 已经读过的历史不必反复重算

生成“今天”之后再生成下一个 token 时，朴素做法会把整个前缀再次送入模型。在因果模型中，旧位置无法看见后来新增的位置；推理模式下，旧位置各层的 K、V 因而可以复用。第一次处理提示词叫 prefill；之后每次加入一个新 token 的过程叫 decode。

使用缓存时，每层保存历史 K、V。新增 token 只计算自己的 Q、K、V，把新 K、V 接在缓存后，用新 Q 读取全部可见 K、V。通常不用缓存历史 Q，因为下一步只需要新位置的查询。缓存改变的是计算复用方式，模型仍逐 token 生成。

对标准多头注意力，忽略额外元数据，缓存字节数近似为：

```
2 × 层数 L × 批量 B × 已缓存长度 T × KV头数 Hkv × 每头维度 dh × 每元素字节数 s
```

开头的 2 代表 K 和 V。例如 L=12、B=1、T=1024、Hkv=8、dh=64、s=2，结果是 25,165,824 字节，即 24 MiB。这只是示例缓存，未计模型权重、临时张量和内存管理开销。

缓存不是让长文本完全“免费”。每生成一个新 token，仍需读取可见历史；整个生成过程仍有累积的注意力计算。测试缓存时，应比较相同前缀下的 logits，而非只比较采样文字：微小数值差异可能被随机采样放大。位置编号必须连续；如果用截断窗口并把位置从零重排，就不能直接与保留旧位置的缓存路径宣称等价。

11.5 MQA 和 GQA 让多个查询头共享键和值

标准 MHA 中，假设有 8 个查询头，也有 8 个 K 头和 8 个 V 头。MQA 保留 8 个查询头，但它们共享 1 组 K、V。GQA 则在两者之间，例如 8 个查询头分成 2 组，每组共享一组 K、V。因此 Hq 和 Hkv 不必相等；常见均匀分组要求 Hq 能被 Hkv 整除。

在其他条件相同、缓存采用紧凑存储时，Hkv 从 8 降为 2，KV 缓存大小降至四分之一。查询头数没有跟着减为 2，模型的总参数和总耗时也不会简单降至四分之一。把共享 K、V 真正复制成 8 份保存，会抵消缓存收益。把训练好的 MHA 任意改为 GQA 也不能保证输出保持不变；转换后通常需要适配训练。参见MQA 原论文与GQA 原论文。

11.6 FlashAttention 公式相同 搬运方式不同

手写注意力常先生成 T×T 的分数矩阵，再生成同样大的概率矩阵，最后乘 V。GPU 在大容量显存和较小的片上高速存储之间搬运这些中间结果，也会花时间。

FlashAttention 通过分块计算、在线 softmax 和反向时的重计算，避免把完整注意力矩阵长期存到显存中。它属于**精确注意力算法**：目标数学运算仍是标准注意力，不靠丢弃部分关系近似结果。浮点运算顺序变化后，结果不一定逐位相同。标准稠密版本的运算量仍随 T^2 增长；减少显存占用和读写，不等于把注意力计算变成线性。FlashAttention 原论文

初学者先写透明的普通实现以理解维度，再学习优化接口。调用 PyTorch 的 SDPA 接口，也不代表一定启用了 FlashAttention；后端选择受硬件、数据类型、形状和软件版本影响。性能报告应写清实际后端。PyTorch SDPA 官方教程

11.7 LoRA RAG 蒸馏 量化分别改变什么

方法	改变的对象	是否必须训练	不能由它自动保证的事
LoRA	为冻结权重加入低秩可训练增量	是	事实更新完整、显存随可训练参数同比减少
RAG	给输入增加检索到的资料	应用可直接搭建；原论文也训练检索/生成组件	检索结果正确、答案完全忠于资料
蒸馏	让学生学习教师的输出或中间表示	是	学生保留教师全部能力
量化	用较低位宽表示数值	可训练后量化，也可量化感知训练	所有设备都加速、质量完全不变

采用列向量记法，若原线性层为 $y=W_0x$，W_0 的形状为 dout×din，LoRA 可写为 $W=W_0+(\alpha/r)BA$，其中 A 为 r×din，B 为 dout×r。训练 A、B，冻结 W_0；新增参数为 r(din+dout)。例如 din=dout=512、r=8，新增 8192 个参数，而完整 W_0 有 262144 个参数。基础权重仍要存储，许多中间激活仍要计算；“只训练少量参数”不等于“整个模型只占少量内存”。LoRA 原论文

RAG 的典型应用流程是“问题→检索相关片段→把片段和问题交给生成器→附证据作答”。原始 RAG 论文有特定的概率模型和训练方案；今天常见的检索后拼接提示词，只共享它的核心思路，不能自动称为完整复现。应分别评估有没有找到证据、有没有正确使用证据。RAG 原论文

蒸馏中，教师对“猫、狗、汽车”给出的软概率，比只有“猫”一个标签多表达了类别相似性。温度升高可使分布更平缓。学生学习这种分布，也可同时学习真实标签；教师犯错时，学生可能继承错误。知识蒸馏原论文

量化可粗略理解为用整数和缩放因子近似浮点数，例如 $w\approx s\cdot q$。它引入舍入和可能的截断误差；离群值会使简单量化困难。LLM.int8() 提供了针对大模型离群特征的混合精度思路，不等同于把所有参数直接强制转换为整数。LLM.int8() 原论文

11.8 ViT 多模态与对话模型

ViT 把图像切成 patch，再将每块展平、线性投影为 token。例如 224×224 的 RGB 图片切成 16×16 的块，得到 14×14=196 个 patch；每块原始维度为 16×16×3=768。投影后的维度由模型设定，不必等于 768。加入位置信息后，Transformer 编码器就能处理这串视觉 token。ViT 原论文

多模态没有唯一结构。CLIP 用图像编码器和文本编码器将配对图文映射到可比较的表示空间，通过对比学习提高配对相似度；它本身不是一个逐字回答问题的聊天模型。另一类系统把视觉表示接入语言生成器，需要另外设计连接模块与训练目标。不要把“用了 Transformer”理解为“所有模态天然互通”。CLIP 原论文

同样，预测下一 token 的预训练模型不自动等于好用的助手。指令微调让模型学习“用户请求→合适回复”的例子；偏好优化进一步使用人类或其他来源的比较信号。InstructGPT 展示了一条包含监督微调、奖励模型和强化学习的路线，但它不是所有对话模型必须采用的唯一方案。InstructGPT 原论文

本章自查。 用一句话解释 RoPE、KV cache、GQA、FlashAttention 各自改变什么。若四句话都写成“让模型更快”，说明还需要回到相应小节，补上具体对象与条件。

第12章 十个分级实验 从看懂数字到独立研究

12.1 先学会做可解释的实验

前五项可从 CPU 开始；第六至第八项是自行修改代码的进阶任务；第九项的真实性能部分适合有兼容 GPU 时选做；第十项先在已有小模型上练习。配套目录为 Transformer教材/配套实验/。本章既包含配套脚本的使用方向，也包含需要读者继续实现的拓展，不能把全部十项都视为一键完成的程序。

最先运行 python attention_handcalc.py，它只需 Python 标准库；然后按配套说明准备 PyTorch，运行 python test_model.py 检查模型结构，再以 python train.py --smoke-test --device cpu 做最小流程检查。完整训练从 train.py 开始，生成从 generate.py 开始，具体参数以配套 README 为准。配套程序覆盖计算演示、结构检查、训练与生成；实验中的缓存、RoPE、GQA、LoRA 等修改需要自行实现。

每次实验保存：代码版本、环境版本、随机种子、数据划分、模型配置、训练 token 数、学习率、训练和验证指标，以及失败记录。比较模型时先固定数据、验证集和训练预算，再只改一个因素。若改变批量或序列长度，仅固定步数会使训练 token 总量变化。若测试集被你反复查看并用于调参，它就实际变成了验证集。

结果表先留空，跑完再填写。本书的预期现象和通过标准是教学设计，**不是已经测得的准确率或速度承诺**。小数据实验可以帮助排错和形成假设，不能单独证明某结构在大规模模型上更优。对正式比较，至少换几个种子检查方向是否稳定；种子相同也不能保证跨设备、跨版本完全一致。

实验1 亲手追踪一次注意力 入门 CPU

目标与假设： 证明注意力输出是 V 的加权组合；分数越高通常权重越大，但权重不是未经归一化的相似度。

操作： 取三个二维 token，自己指定 Q、K、V。依次打印 QK^T、除以 $\sqrt{2}$ 的分数、逐行 softmax、最后输出。先不加 mask，再加入含对角线的下三角因果 mask。使用一个只允许读取自身的 mask 作额外对照。

控制与指标： 固定输入，关闭 dropout；检查概率每行求和、被遮挡概率最大值、手算与程序输出最大绝对差。

预期与诊断： 只看自身时，输出应等于对应的 V。若概率沿错误方向归一化，检查 softmax 的维度；若遮挡后仍能读未来，检查是在 softmax 前屏蔽分数，还是仅在最后乘零。

通过标准： FP32 小张量下，以 1e-5 左右为起点设置误差容限；行和接近 1、未来权重为 0，并能解释每个中间张量。这里不是要求训练出好模型。

实验2 用扰动检测“偷看未来” 入门 CPU

目标与假设： 正确的因果模型在位置 t 的输出不受 t 之后输入改变的影响。

操作： 把模型设为评估模式。构造两条序列，它们前四个 token 相同，后四个不同。比较前四个位置的 logits。随后故意移除因果 mask，再比较一次。还要测试目标是否比输入向右错开一位，例如输入 abcd、目标 bcde。

控制与指标： 同一份权重、同样的位置编号、关闭 dropout、不更新参数；记录前缀 logits 的最大绝对差。用多个随机输入，避免偶然相同。

预期与诊断： 正确模型的前缀差异只应有很小数值误差；故意解除遮挡后通常出现变化。若两次都不变，可能错误地把注意力输出置零或测试输入没有有效差别；若正确版本变化明显，检查 mask 广播维度和归一化是否跨 token。

通过标准： 正确版本通过前缀不变性检查；故障版本能被检查捕获。此实验比只看训练损失更能发现信息泄漏。

实验3 让小模型记住一个固定批次 基础 CPU

目标与假设： 容量足够的小网络，在无正则干扰的简单固定样本上，应能显著降低训练损失。

操作： 使用短小、固定、无随机打乱的字符序列，只训练一个批次；先把 dropout 和权重衰减关闭。打印更新前后一个参数的差异，并保存损失曲线。不要把这个批次用于评价泛化。

控制与指标： 固定初始化、输入和标签；指标为交叉熵、token 正确率、梯度是否有限、参数是否改变。将均匀预测的交叉熵 ln|V| 作为可解释参照。

预期与诊断： 损失应较初始值明显下降。完全不降时，依次检查标签错位、是否调用反向传播、是否更新了同一个模型、学习率是否为零、梯度是否被断开。发散时先降低学习率，再查数值稳定性。

通过标准： 可以在预设的简单批次上达到接近完整记忆，并解释失败时的排查过程。若仍有随机或冲突标签，不要求零损失。

实验4 训练一个字符语言模型 基础 CPU

目标与假设： 完整的训练、验证、保存和生成流程能形成闭环；训练集拟合改善不必然意味着验证集改善。

操作： 运行配套小型 pre-LN、decoder-only 模型，用本地教学语料训练。阅读数据切分代码，确认先划分语料，再各自切窗口。每隔固定间隔评估，保留验证损失较好的权重；同时保留训练前与训练后的同提示词样本。

控制与指标： 固定字符表、划分和评估 token；记录训练交叉熵、验证交叉熵和同一 tokenization 下的困惑度。文本样本用于解释现象，不能替代数值评价。

预期与诊断： 模型可能学习局部格式和常见组合，但小语料不产生可靠的通用问答能力。训练与验证同时不降，先回到实验3；训练下降而验证恶化，考虑过拟合、语料分布差别和重复样本。

通过标准： 能重新载入权重、复算验证指标、生成文本，并说清本地玩具语料限制了哪些结论。

实验5 采样改变了什么 基础 CPU

目标与假设： 温度和 top-k 改变输出分布，不改变已经学到的参数。

操作： 固定一个训练好的检查点和十个提示词，分别用贪心、温度 0.7/1.0/1.3，以及合理的 top-k 生成等长文本。每个随机设置重复几次；不要只展示最顺眼的一段。若词表小于 k，应把 k 截到词表大小。

控制与指标： 同权重、同提示词、同生成长度；记录重复字符或重复 n-gram 的比例、不同样本数量、人工观察到的格式完整性。生成质量没有一个简单指标能完全概括。

预期与诊断： 高温通常更分散，低温通常更集中；它们不保证更正确。若温度变化毫无影响，检查是否仍直接取 argmax；若 NaN，检查是否除以零、非法 logits 或空候选集合。

通过标准： 给出整组结果而非精选样本，并说明为什么更有变化的文字不必更符合任务。

实验6 位置表示的有无与替换 进阶 需扩展

目标与假设： 在对顺序敏感的任务上，显式位置信息会影响可学习性；但不能据一个简单任务声称无位置模型完全不能处理顺序。

操作： 在双向编码器小任务中判断三符号串是 abc 还是 cba，保持符号集合相同，用置换不变的平均池化读取结果；比较无位置和加入位置的版本。再在字符语言模型中比较学习式位置与 RoPE，记录训练长度内外的表现。

控制与指标： 第一部分固定池化方式、均衡标签和全部组合划分；第二部分固定宽度、层数、训练 token 预算，记录参数差异。观察分类正确率、各长度的验证损失。

预期与诊断： 第一部分无位置的双向模型加平均池化无法区分两种排列；若明显能区分，检查输入或数据是否偷偷携带位置。因果 mask 本身带有结构信息，所以不要把第一部分的结论直接套到因果语言模型。

通过标准： 说明任务为什么需要顺序；对长度外推单独测试，并接受 RoPE 也可能失败。

实验7 缓存路径和完整路径是否等价 进阶 需扩展

目标与假设： 在相同前缀、相同位置与确定性评估条件下，正确 KV cache 应保留模型输出。

操作： 为每层实现缓存。先用完整前缀得到最后位置 logits，再逐 token 增量输入，比较每一步结果。测试前缀长度 1、2、7、16；然后加上非空提示词的 prefill。先不使用滑动窗口或位置重置。

控制与指标： 关闭 dropout，用 FP32 起步；记录最大绝对误差、缓存形状、缓存字节数，最后才测耗时。测试包含超过一层的模型，避免只检查第一层。

预期与诊断： logits 应在数值容限内接近。第一步就错，检查初始化；第二步起错，检查拼接轴；越往后越错，检查位置编号和 mask 对齐。单 token 查询对应长 K 时，不能盲目套用“左上角下三角”的默认 mask。

通过标准： 所有预设长度的等价性检查通过，实际缓存大小与公式一致；小 CPU 样例即使没有速度收益，也不能据此判定数学实现错误。

实验8 一次只替换一个模块 进阶 需扩展

目标与假设： RMSNorm、SwiGLU、GQA 的作用可以通过受控对照分析，而不是凭一次运行评判。

操作： 复制基线配置，分别只把 LayerNorm 改为 RMSNorm、普通前馈改为参数量接近的 SwiGLU、MHA 改为查询头数相同的 GQA。每种版本从头训练，不直接拿基线权重硬替换后宣布公平比较。

控制与指标： 同数据、优化器、token 预算和多个种子；报告参数量、验证损失、缓存理论量与实测量。一次只改变一个模块；最后才尝试组合。

预期与诊断： GQA 的缓存量应按 KV 头数下降；小语料上质量差异可能很小或方向不稳定。SwiGLU 参数明显增加时，重新计算隐藏宽度；GQA 缓存不减时，检查是否把共享张量复制保存。

通过标准： 写出对照表与限制，并用“本设置下观察到”陈述结论。能够报告没有明显收益，同样算完成实验。

实验9 普通注意力与优化实现 选做 GPU性能部分

目标与假设： 实现方式能在保持近似相同数值结果的同时改变耗时和峰值显存；实际收益依赖配置。

操作： 先用普通实现和 PyTorch SDPA 对齐输出；随后在兼容设备上逐步增加序列长度，记录实际后端。先做预热；GPU 计时前后同步；明确测的是仅前向还是前向加反向。改变实现时固定其他条件。官方教程

控制与指标： 同 B、T、头数、维度、dtype、mask 和 dropout；分别记录误差、耗时中位数、重复波动、峰值已分配显存。推理时显式把 SDPA 的 dropout_p 设为 0。

预期与诊断： 长序列可能更能显示显存差异；小张量可能被启动开销主导。若不加速，检查是否回退到其他后端；若误差很大，优先查 mask 语义和 dropout，而非简单放宽容限。

通过标准： 正确性先通过，性能表能复现，并准确区分“调用了 SDPA”和“使用了 FlashAttention 后端”。不要求达到论文中的加速倍数。

实验10 给已训练小模型做 LoRA 适配 选做 CPU或小GPU

目标与假设： 少量可训练参数能改变模型在新分布上的行为，但收益取决于基模型是否已学到有用表示。

操作： 先完成小模型预训练，再准备另一组不含测试样本的本地格式化文本。比较冻结基线、全量微调、只给 Q/V 投影加 LoRA 三组。新增矩阵可采用 A 随机、B 为零的初始化，使初始增量为零。保留一份原领域验证集检查遗忘。

控制与指标： 同基线权重、同新数据、同 token 预算；分别调参时记录搜索范围。报告可训练参数数、目标领域损失、原领域损失及格式正确率。显存比较要实测，不能用参数比例代替。

预期与诊断： LoRA 可能改善目标格式，也可能因基模型太弱而收效有限。损失不变时检查 A/B 是否进入优化器；基座参数改变时检查冻结；两矩阵都初始化为零可能阻断起步梯度。

通过标准： 证明基座被冻结、适配器确实更新，并给出收益与遗忘的共同评价。完成后可另开小项目比较检索增强；不要把微调和检索混成一个无法分析的改动。

12.2 一页实验报告模板

每次用下面七个问题收尾：我提出了什么可检验假设？基线是什么？只改了什么？哪些变量被固定？指标怎样计算？失败案例有哪些？这个实验不能说明什么？最后保留代码、配置和原始输出，而不只保留一张曲线。一个解释清楚的负结果，比未经核实的漂亮数字更有学习价值。

第13章 推荐论文 按问题阅读 而不是按热度收藏

13.1 三遍阅读法

第一遍读摘要、主要图、结论，写出“问题-方法-证据”各一句。第二遍只追踪一个关键公式和对应张量形状；看不懂的符号先建表。第三遍读实验设置与消融，检查它比较了什么、没比较什么。无需第一天就逐行啃完整篇论文。

下列 16 篇按建议阅读顺序分层。“年份”以首次公开预印本或技术报告年份为主；因此 BERT 写 2018、RoFormer 写 2021、ViT 写 2020，即使它们的会议或期刊版本在随后年份发表。书目信息及原始链接于 2026-09-29 核验。复现小题是本教材设计的缩小练习，不代表能够复现论文的大规模指标。

13.2 必读 建立完整主线

P01 | Vaswani 等，2017，Attention Is All You Need。 原论文

解决问题：如何主要依靠注意力完成序列到序列建模。先读模型结构图及第3节 Model Architecture，再读第5节训练设置；最后回看实验。先修：矩阵乘法、softmax、交叉熵。小题：逐层标注一个两层编码器-解码器的张量形状，并区分两处自注意力和一处交叉注意力。不要把本书的 pre-LN 小解码器直接当成原论文的逐项复现。

P02 | Devlin 等，2018，BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding。 原论文

解决问题：怎样通过双向上下文预训练得到可迁移表示。先读第3节 BERT，关注预训练与微调的区分，再看消融部分。先修：编码器、mask、分类头。小题：构造一句话的遮蔽语言建模输入与标签，说明为什么它不同于从左到右预测。了解原始 BERT 的下一句预测任务，但不要把它视为所有编码器模型的必要组成。

P03 | Radford 等，2019，Language Models are Unsupervised Multitask Learners。 官方技术报告

解决问题：大规模语言建模能否迁移到多种任务。先读第2节方法与模型，再读第4节 Generalization vs Memorization。先修：因果语言模型、tokenization、概率链式法则。小题：给同一个小模型设计三个不同格式的提示词，记录成功与失败；说明玩具模型不能验证报告的大规模结论。

13.3 进阶 理解模块与计算成本

P04 | Su 等，2021，RoFormer: Enhanced Transformer with Rotary Position Embedding。 原论文

解决问题：让注意力中的位置关系以旋转形式表达。先读二维情形的推导，再读高维推广；先不纠结全部实验。先修：点积、sin/cos、矩阵转置。小题：计算两个位置同步平移前后的旋转点积，并检查误差。把数学上的位置可计算性与模型的长度泛化分开。

P05 | Zhang 与 Sennrich，2019，Root Mean Square Layer Normalization。 原论文

解决问题：归一化是否必须减均值。先读 RMSNorm 的定义与性质，再读实验比较条件。先修：均值、方差、LayerNorm。小题：对同一小张量分别计算 LayerNorm 和 RMSNorm，比较输出均值及均方根；用代码检验输入乘正数前后的结果，并说明 ε 带来的微小差别。

P06 | Shazeer，2020，GLU Variants Improve Transformer。 原论文

解决问题：门控前馈网络能否改善 Transformer。先读各变体公式和参数预算说明，再看结果表。先修：线性层、激活、逐元素乘法。小题：实现 SwiGLU，匹配普通前馈网络的近似参数量，再做一次小规模对照。不要把“多一组矩阵”带来的容量增加误认为门控的独立收益。

P07 | Shazeer，2019，Fast Transformer Decoding: One Write-Head is All You Need。 原论文

解决问题：增量解码时，多组 K、V 的读写代价过高。先读增量推理分析，再对照多头与多查询伪代码。先修：多头拆分、KV cache、内存带宽。小题：对固定层数与长度计算 MHA 和 MQA 的缓存字节数，再画出查询头与 KV 头的对应关系。

P08 | Ainslie 等，2023，GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints。 原论文

解决问题：在 KV 共享程度和模型质量之间选择中间方案，并研究已有检查点的转换。先读方法部分与第一张图，再看不同组数的结果。先修：P07。小题：把 8 个查询头分成 1、2、4、8 组，计算缓存量；解释为什么均值合并权重后还需要继续训练，不能仅凭形状兼容宣称等价。

P09 | Dao 等，2022，FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness。 原论文

解决问题：标准注意力的大型中间矩阵带来显存与读写开销。先读第2节硬件背景、第3.1节分块算法，再读定理中的计算量和内存量。先修：注意力、稳定 softmax、GPU 存储层次。小题：把一行 softmax 分两块，保存每块最大值与指数和，合并得到完整结果；不要将“精确”解释为浮点逐位相同。

13.4 选读 选择一个方向继续深入

P10 | Hu 等，2021，LoRA: Low-Rank Adaptation of Large Language Models。 原论文

解决问题：怎样减少任务适配时需要更新和保存的参数。先读方法图、低秩参数化和秩的实验。先修：矩阵秩、微调、冻结参数。小题：在一个线性层上实现 BA 增量，计算新增参数，验证合并前后的输出一致性。再检查只保存适配器时，是否同时记录了基模型版本。

P11 | Lewis 等，2020，Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks。 原论文

解决问题：把参数内知识与可检索的外部文档结合。先读总体图及 RAG-Sequence、RAG-Token 两种形式，再看检索相关实验。先修：条件概率、向量检索、编码器-解码器。小题：用十份自写文档设计问答集，分别衡量检索是否命中和答案是否有证据；基础练习不要求复现论文的联合训练。

P12 | Hinton 等，2015，Distilling the Knowledge in a Neural Network。 原论文

解决问题：把大型模型或集成的行为迁移给较小模型。先读温度与软目标的说明，再看一个实验。先修：softmax、交叉熵。小题：对固定 logits 比较温度 1 与 3 的概率，并让小分类器分别学习硬标签和教师分布。它不是 Transformer 论文，但提供理解模型压缩的重要基础。

P13 | Dettmers 等，2022，LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale。 原论文

解决问题：如何在存在离群特征时进行较低位宽矩阵计算。先读量化背景与混合精度分解，再看误差分析。先修：浮点与整数表示、线性层。小题：给小矩阵加入离群值，比较逐张量与逐通道缩放的重建误差。这个练习说明误差来源，并不完整复现 LLM.int8() 的内核。

P14 | Dosovitskiy 等，2020，An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale。 原论文

解决问题：图像能否直接变成 patch 序列交给 Transformer。先读方法图和第3节，再关注数据规模相关实验。先修：编码器、图像张量、分类。小题：把随机图片切块并还原，验证像素排列完全一致；再写 patch embedding。不要由小数据上的单次结果断言 Transformer 必然超过卷积网络。

P15 | Radford 等，2021，Learning Transferable Visual Models From Natural Language Supervision。 原论文

解决问题：利用图文配对学习可迁移的视觉表示。先读对比学习示意图与伪代码，再读零样本评估方法。先修：图像编码器、文本编码器、向量相似度、交叉熵。小题：为三对手工向量构造 3×3 相似度矩阵，说明行方向和列方向各自把什么当分类目标。

P16 | Ouyang 等，2022，Training language models to follow instructions with human feedback。 原论文

解决问题：语言模型怎样更好地遵循用户意图。先读三阶段流程图、数据收集和人工评估，再读限制。先修：预训练、监督微调；强化学习公式可以后补。小题：为同一个任务写清评分标准，盲评多组输出并记录分歧；先学会定义偏好数据，不急于复现完整强化学习训练。

13.5 配套官方学习资源

PyTorch Learn the Basics适合补张量、自动求导、优化与模型保存。按本书进度学，不必先把整个文档读完。PyTorch 安装页用于按操作系统和计算平台选择安装方式；软件版本变化时，以页面与本机环境为准。SDPA 官方教程留到实验9。原论文与作者官方实现用于核验思想；旧仓库的历史环境未必适合直接安装到当前机器。

附录A 中英术语速查

中文	英文/缩写	本书中的含义
张量	Tensor	有形状的数值数组；维度顺序影响计算意义
词元	Token	分词器定义的基本单位，不必等于一个字或词
词表	Vocabulary	token 与整数编号的映射集合
嵌入	Embedding	把离散编号映射成可学习向量
隐藏状态	Hidden state	模型某一层中 token 的表示
批量	Batch	一次参与计算的一组样本
参数	Parameter	通常由训练更新的模型数值
超参数	Hyperparameter	学习率、层数等由训练设置决定的量
查询/键/值	Query/Key/Value	用于匹配和聚合的三种投影表示
自注意力	Self-attention	Q、K、V 来自同一序列的表示
交叉注意力	Cross-attention	Q 与 K/V 来自不同序列或表示源
因果遮罩	Causal mask	阻止当前位置读取未来位置
填充遮罩	Padding mask	阻止无效填充位置参与相应计算
头	Head	一组并行的注意力表示子空间
残差连接	Residual connection	把子层输入加回输出
前馈网络	Feed-forward network, FFN	对各 token 分别应用的通道变换
归一化	Normalization	调整数值尺度的操作，具体方法不完全相同
分数	Logit	概率归一化之前的模型输出
交叉熵	Cross-entropy	衡量目标类别所获概率的损失
梯度	Gradient	损失对参数局部变化的敏感方向
反向传播	Backpropagation	沿计算图计算梯度的过程
训练步	Step	通常指一次参数更新；梯度累积时需另行说明
轮次	Epoch	一遍训练数据；流式采样时不一定自然定义
泛化	Generalization	在未用于训练的数据上表现良好
消融	Ablation	移除或替换某部分，观察作用

中文	英文/缩写	本书中的含义
困惑度	Perplexity, PPL	常用自然对数交叉熵的指数
预训练	Pretraining	在较通用数据与目标上学习基础参数
微调	Fine-tuning	为新任务或分布继续训练模型
检查点	Checkpoint	保存的权重及可能包含的训练状态
上下文窗口	Context window	一次计算中可见或允许使用的序列范围
预填充	Prefill	一次处理已有提示词并构建相关状态
解码	Decoding	从输出分布逐步选取 token 的过程
缓存	KV cache	自回归推理中保留历史各层 K、V
推理	Inference	使用训练好的参数进行预测或生成
吞吐	Throughput	单位时间处理的工作量，需说明 token 口径
延迟	Latency	一次请求或某一步等待的时间

附录B 八周学习建议

以下以每周约 6-8 小时安排，只是规划建议，按理解程度调整。每周都留出“从空白纸讲一遍”和“独立改一次代码”的时间。

周	主要任务	可以交给别人检查的成果
第1周	Python 数组、张量形状、向量点积、线性层	手算一个矩阵乘法并用代码核对
第2周	分类概率、交叉熵、梯度与训练循环	一张损失曲线及逐行解释
第3周	token、嵌入、单头与多头注意力	完成实验1、2，画出维度变化
第4周	位置、残差、归一化、编码器和解码器	不看书讲清完整数据流；读 P01
第5周	训练小语言模型、数据划分、调试	完成实验3、4；保留一次失败分析
第6周	生成、温度、验证、架构对比	完成实验5；读 P02、P03
第7周	RoPE、缓存或一个现代模块	从实验6-8选一项，写受控对照报告
第8周	选择研究方向与论文复现小题	一份可复查的项目报告和后续问题

若第三周仍常把 B、T、D 弄混，延长这一周通常比赶着读十篇论文有效。达成“能解释、能实现、能测试、能承认边界”四项，比按时读完更重要。

附录C 综合练习与参考答案

题1： B=2、T=5、D=12、H=3 时，拆头后的 Q 和注意力分数分别是什么形状？

答： 每头维度 dh=4。常见排列的 Q 为 [2,3,5,4]，分数 QK^T 为 [2,3,5,5]；每头输出为 [2,3,5,4]，合并头恢复 [2,5,12]。矩阵转置只交换 K 最后的长度和特征两个轴，不是把整张四维张量全部倒序。

题2： 有三个等概率 token，正确 token 的交叉熵与困惑度是多少？

答： 交叉熵为 $-\ln(1/3)=\ln 3\approx 1.099$，困惑度 $\exp(\ln 3)=3$。跨分词器或不同评估文本直接比较困惑度通常不成立，因为预测单位或分布改变了。

题3： 为什么训练时能一次计算多个位置，生成时却通常逐 token 进行？

答： 训练时完整真实序列已知，使用右移目标和因果遮罩，可以并行计算每个位置的条件预测；生成时后续 token 尚未确定，要把已选 token 加入前缀再预测。并行计算训练位置不代表允许读取未来答案。

题4： 注意力图中某个词的权重很大，能否断言它就是模型答案的唯一原因？

答： 不能。权重只描述特定层、特定头的一次聚合；输出还取决于 V 的内容、其他头、残差与后续层。可以把图作为线索，再通过修改输入、消融和对照检查实际影响。

题5： L=4、B=2、T=256、Hkv=4、dh=16，缓存使用 FP32。理论 KV 缓存多大？

答： $2\times4\times2\times256\times4\times16\times4=1{,}048{,}576$ 字节，即 1 MiB。若其他条件不变，改为 Hkv=1，则为 256 KiB；这不包含模型权重。

题6： 把 LoRA 的 A、B 都初始化为零为什么可能训练不起来？

答： 增量为 BA。对 A 的梯度带有 B，对 B 的梯度带有 A；两者同时为零时，这一分支起步时两边的梯度都可能为零。常见做法令一边随机、一边零，既使初始增量为零，又给其中一边留下可更新的梯度路径。

题7： 验证损失突然好得离谱，应先庆祝还是排查？

答： 先排查：训练验证是否共享重叠窗口或重复文本；是否在验证数据上更新了参数；标签是否在同一位置直接暴露；是否把 padding 计成大量容易预测的 token；指标平均分母是否正确。全部排除后再解释收益。

题8： 序列长度翻倍，普通稠密注意力的分数元素数、KV cache 大小各怎样变化？

答： 在 B、头数和维度不变时，分数元素数由 T^2 变成 $4T^2$；缓存按 T 增长，所以翻倍。不能把“注意力矩阵二次增长”套到所有模型内存，也不能因 FlashAttention 不保存完整矩阵就断言它只需线性运算。

题9： 想让模型回答经常更新的内部手册，首先比较什么方案？

答： 先建立能检索手册并引用依据的基线，独立测检索命中与答案忠实度；如果任务还要求固定表达形式或领域行为，再比较微调。LoRA 可用于适配行为，RAG 可提供外部资料；它们可以组合，但应先分开评估作用。

题10： 一个改动只在一个随机种子上比基线略好，报告应怎么写？

答： 写明本次配置、样本量、种子和差值，保留初步结论，并补充多个种子或更充分对照。不能把一次小差异直接写成方法普遍优越；也不能只报告最好的一次而隐藏其他运行。

附录D 一次真实训练记录与曲线解读

本附录记录配套程序在 CPU 上的一次实际运行，供你核对流程和练习分析。数据是随书提供的微型中文语料，结果只说明这一教学设置的表现，不代表通用语言能力，也不是你更换设备和软件后必须逐位复现的标准答案。

D.1 运行条件与已完成的检查

运行日期为2026年9月29日，使用 Python 3.12.14、PyTorch 2.14.0+cpu，随机种子42，CPU计算线程数2。模型使用2层、4个头、宽度64、上下文32、批量16、dropout 0.1、学习率0.001，训练400步。模型有147968个可训练参数，词表358类。

在训练前，8项结构测试全部通过，包括因果前缀不受未来输入影响、权重归一化、张量形状、梯度有限、标签只错位一次、窗口不跨文档、参数更新、生成跨过窗口边界，以及保存加载后输出一致性等检查。部分测试同时覆盖多个性质。

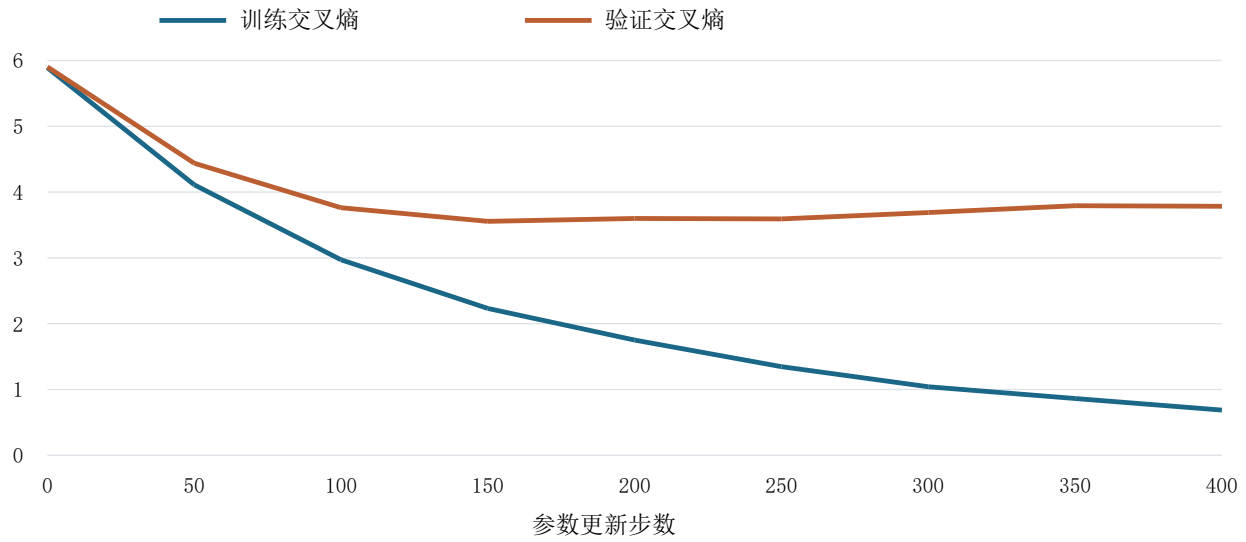
400步训练完成后，最终模型已成功保存，并通过独立生成入口重新加载。固定同样的提示词、采样参数和采样种子，重新加载后得到的文本与训练结束时一致。这里只核验 CPU 路径；没有据此声称 GPU、其他版本或所有超参数组合均已验证。

D.2 记录的数据

训练与验证交叉熵均在 eval 模式下、各自固定评估窗口上计算。训练评估包含1024个目标 token，验证评估包含256个目标 token。完整高精度记录见配套实验中的 示例训练记录.csv。下表为便于阅读而四舍五入的值。

更新步数	训练交叉熵	验证交叉熵	验证困惑度
0	5.8893	5.9020	365.77
50	4.1103	4.4385	84.65
100	2.9709	3.7620	43.04
150	2.2315	3.5549	34.98
200	1.7506	3.5988	36.55
250	1.3465	3.5919	36.30
300	1.0413	3.6885	39.99
350	0.8628	3.7924	44.36
400	0.6867	3.7838	43.98

D.3 怎样读这两条曲线



图D 1 同一次训练中的两条交叉熵曲线 数值越低越好

刚初始化时，交叉熵接近均匀猜测的 ln(358)≈5.8805，这与输出尚未学到有效规律相符。前150步中，两条曲线一起下降；之后训练交叉熵继续下降，而验证交叉熵总体回升。这是过拟合的一个直接例子。

在本次每50步采样的记录点中，第150步验证交叉熵最低。不能因此断言真正最佳步数恰好是150，也不能认为150步对别的数据都最佳。当前脚本保存最终模型，未自动保存最佳验证检查点。你可以把“按验证损失保存最佳模型并比较结果”作为一个小改进。

生成文本能出现类似训练语料的短句，但仍会混合不连贯片段。即使某几句话很通顺，也应检查是否复用了训练短句。生成示例不能代替独立验证指标；验证集又很小，因此指标本身也需要结合数据范围解释。

可以进一步做三个对照：缩小模型；在同样划分规则下增加独立训练文本；比较较早检查点与最后检查点。每次只改变一个因素，并记录实际训练 token 数。先解释曲线为什么变化，再考虑增加层数或使用更复杂结构。