# Transformer (完全版)

---

## 1. 时代背景：为什么需要 Transformer？

在 Transformer 诞生之前，自然语言处理 (NLP) 领域是**循环神经网络 (RNN)** 及其变体 (LSTM, GRU) 的天下。然而，RNN 架构存在两个难以根治的核心痛点，这正是面试官最喜欢考察的切入点：

1. **长距离依赖问题 (Long-Range Dependency Problem):** RNN 通过一个逐步更新的隐藏状态来传递信息。当序列很长时，早期的信息在传递过程中会不断被稀释，导致模型很难捕捉到相距遥远的词语间的语义关联（即**梯度消失/爆炸**问题）。
2. **计算效率低下，难以并行化:** RNN 的“循环”特性决定了它必须按时间步顺序计算——处理完 t-1 时刻才能处理 t 时刻。这与现代 GPU 追求大规模并行计算的特性背道而驰，导致模型训练异常耗时。

**Transformer 的革命性在于，它提出了一种全新的范式，通过引入自注意力机制 (Self-Attention) 彻底解决了以上两个问题：**

- **解决长距离依赖:** 注意力机制可以直接计算序列中**任意两个位置**之间的关联得分，无论它们相距多远，交互路径长度都是 O(1)，信息传递不再因距离而衰减。
- **实现高度并行化:** Transformer 摒弃了循环结构，所有输入词语的计算可以同时进行，极大地释放了 GPU 的潜力，使得训练前所未有的大模型成为可能。

> *面试官视角: 你能一句话总结 Transformer 最大的贡献吗？*
>
> *参考回答: Transformer 最大的贡献是提出了一个完全基于自注意力机制的模型架构，它摒弃了传统的循环和卷积结构，实现了高效的并行计算，并能出色地捕捉全局长距离依赖关系。这一架构创新为 BERT、GPT 等大规模预训练模型的诞生奠定了基础，开启了 NLP 的新纪元。*

## 2. Transformer 架构的庖丁解牛

标准的 Transformer 模型是一个 **Encoder-Decoder** 架构，最初被用于机器翻译任务。

## 2.1. 输入处理：一切的开始

在将文本送入模型之前，需要进行两步关键的预处理：

1. **词嵌入 (Token Embedding):** 将输入的文本序列通过分词器 (Tokenizer) 切分成一个个 Token，然后将每个 Token 映射为一个固定维度的向量。这个向量是词语的初始语义表示。常见的分词算法有 BPE (GPT 使用) 和 WordPiece (BERT 使用)。
2. **位置编码 (Positional Encoding):** 这是 Transformer 的一个标志性设计。由于自注意力机制本身是**位置无关**的（对输入序列进行任意打乱，输出结果不变），我们必须显式地为模型注入序列的顺序信息。
    - **为什么需要位置编码？** 如果没有位置编码，模型将无法区分 "我爱你" 和 "你爱我" 这两个语义完全不同的句子。
    - 它是如何工作的？ 原始论文使用了 sin 和 cos 函数的组合来生成固定的位置编码：

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

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

    其中 $pos$ 是词语在序列中的位置，$i$ 是嵌入向量的维度索引。

    - 面试核心问题：为什么 sin/cos 编码有效？

      答: 这种设计的精妙之处在于，对于任意固定的偏移量 $k$，$PE_{pos+k}$ 可以表示为 $PE_{pos}$ 的一个线性变换。这意味着模型可以轻易地学习到相对位置信息。例如，模型可以通过学习一个固定的变换矩阵，来理解“下一个词”或“前两个词”这样的相对概念，而不需要关心绝对位置 $pos$ 是多少。

    - 面试追问: 位置编码是必须用 sin/cos 函数吗？可以学习吗？

      答: 不是必须的。sin/cos 是一种固定的、绝对位置编码方案。后续的研究也提出了多种方案：

        - **可学习的绝对位置编码 (Learned Positional Embedding):** BERT 使用的方式。直接为每个位置初始化一个可学习的向量，让模型在训练中自己去学。优点是灵活，缺点是没见过超长序列时泛化能力可能受限。
        - **相对位置编码 (Relative Positional Encoding):** 不直接编码绝对位置，而是在计算注意力时，根据两个 Token 间的相对距离动态地调整注意力分数。这种方式在理论上对序列长度的泛化性更好。

## 2.2. 编码器 (Encoder)：深度理解输入上下文

编码器由 N 层相同的层堆叠而成，每一层包含两个核心子层：

### a. 多头自注意力机制 (Multi-Head Self-Attention)

这是 Transformer 的灵魂。它让模型在处理一个词时，能够同时“关注”到输入序列中的所有其他词，并动态计算它们对当前词的重要性。

- 核心概念: Q, K, V (查询, 键, 值)

  这个概念源于信息检索。你可以这样直观理解：

  - **Query (Q):** 当前词的查询请求 — “我想找什么样的信息？”
  - **Key (K):** 所有词的索引标签 — “我能提供什么信息？”
  - **Value (V):** 所有词的实际信息 — “我具体的内容是什么？”

  在实践中，Q, K, V 都是由输入嵌入（加上位置编码）分别乘以三个不同的可学习权重矩阵 $W_Q, W_K, W_V$ 得到的。

- **注意力计算三部曲：**

  a. **计算相似度（打分）**
  使用 Q 和 K 的点积来计算注意力分数。

  b. **缩放与归一化（Softmax）**

  注意力计算公式如下：

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

  - 面试核心问题：为什么注意力计算要除以 $\sqrt{d_k}$？

    答：假设 Q 和 K 的元素是均值为 0、方差为 1 的独立随机变量，则 $Q \cdot K$ 的方差为 $d_k$。当 $d_k$ 很大时，点积结果将落在一个较大的范围，导致 Softmax 输出非常接近于 one-hot，梯度几乎消失。因此我们通过除以 $\sqrt{d_k}$ 来缩放，使分布稳定，便于模型训练。

  c. **加权求和**
  将归一化后的注意力权重应用于所有 $V$ 向量，形成上下文感知的新表示。

- **多头 (Multi-Head) 的意义：**

  - **为什么需要“多头”？**
  单一注意力机制可能只能关注一种模式，比如语法或语义。多个注意力头让模型可以**并行学习多个关注模式**，如主谓一致、指代关系、语义相似等。

- “多头”是如何工作的？
  将 Q, K, V 分成 $h$ 个子空间，分别进行注意力计算，最后拼接这些输出并投影为原始维度：

  $$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

  其中每个头的计算如下：

  $$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- 面试官视角：多头注意力机制的好处？

  答：1）从多个子空间并行学习不同的注意力模式；2）增强模型的表达能力和泛化能力；3）捕捉长距离依赖、局部模式、语法和语义等多种特征。

### b. 前馈神经网络 (Feed-Forward Network)

在注意力之后，Transformer 会将每个位置的表示输入一个相同的前馈神经网络：

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- 作用：

  a. 引入非线性，增强表达能力；

  b. 对每个位置独立处理，进一步提炼信息；

  c. 注意力机制负责“关系建模”，前馈网络负责“内容变换”。

### c. 残差连接 (Add) 和层归一化 (Norm)

每个子层（注意力或前馈）外部都包裹了残差连接和层归一化：

$$\text{output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

- 为什么用残差连接？
  类似 ResNet，可以缓解梯度消失，让信息在深层网络中更容易传播，从而训练更深的模型。
- 为什么用 **LayerNorm** 而不是 **BatchNorm**？

  答：

  a. **BatchNorm** 的局限性：

  - 它对每个特征维度在一个 batch 中的所有样本做归一化（基于 batch 统计量：均值 & 方差）。

- 在 NLP 中，输入序列通常是 **变长的**，需要填充（padding），而填充会干扰 batch 的统计计算。
- 小 batch size 时，BatchNorm 效果也不稳定（常见于大模型训练）。

b. **LayerNorm 更适合 Transformer 的原因：**

- LayerNorm 是在一个样本内部做归一化（对一个 token 的所有特征维度），与 batch 大小无关。
- 它天然支持 **变长输入**，不受填充影响，更适合处理自然语言的序列数据。
- Transformer 模型训练稳定性好，很大程度归功于 LayerNorm。

c. **总结一句话**：LayerNorm 不依赖 batch 的统计信息，天然支持变长序列，是更适合 NLP 和 Transformer 的归一化方式。

> ***Q:** 那 CNN 为什么用 **BatchNorm**，而 **Transformer** 用 **LayerNorm**？*
>
> - *CNN 处理图像，输入大小统一、batch size 大，适合用 BatchNorm；*
> - *Transformer 处理文本，序列长度不一致、填充多，BatchNorm 不稳定；*
> - *所以，归一化方式的选择取决于数据结构和建模需求，不是谁更强，而是谁更适合。*

- 面试追问：**Pre-LN 与 Post-LN 的区别？**

  答：

  - **Post-LN（原始论文方式）**

    $$\text{output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

    - 优点：更符合残差结构直觉
    - 缺点：训练初期梯度不稳定，需 warm-up
  - **Pre-LN（现代主流方式）**

    $$\text{output} = x + \text{Sublayer}(\text{LayerNorm}(x))$$

    - 优点：训练更稳定，梯度更平稳，支持更大模型和学习率
    - GPT-2 / GPT-3 等模型采用了此方式

## 2.3. 解码器 (Decoder)：生成目标输出序列

解码器的结构与编码器类似，但多了一个关键的子层。解码器的每一层包含三个子层：

1. **带掩码的多头自注意力 (Masked Multi-Head Self-Attention):**
    - **“掩码 (Masking)” 的作用是什么？** 在生成任务中，模型在预测第 t 个词时，只能看到 t 之前已经生成的词，不能“偷看”未来的词。这个掩码机制通过将未来位置的注意力分数设置为一个极大的负数（经过 softmax 后变为0），来确保这种**因果依赖关系 (causality)**，防止信息泄露。
2. **编码器-解码器注意力 (Encoder-Decoder Attention / Cross-Attention):**
    - **这是连接编码器和解码器的桥梁。** 它的 **Q (查询)** 来自于解码器前一层的输出，而 **K (键) 和 V (值)** 则来自于**编码器的最终输出**。
    - **它的作用是什么？** 它允许解码器在生成每个词时，能够“关注”到输入序列（源语言句子）中最相关的部分。例如，在翻译任务中，当要生成英文单词 "cat" 时，解码器会通过这个机制重点关注输入法语句子中的 "chat"。它回答了“为了生成当前词，我应该关注输入序列的哪个部分？”这个问题。
3. **前馈神经网络 (Feed-Forward Network):** 与编码器中的作用相同。

同样，解码器的每个子层也都有残差连接和层归一化。

## 2.4. 解码器推理：贪心搜索 vs. 集束搜索 (Beam Search)

解码器的最终输出是一个词汇表的概率分布。如何根据这个分布生成最终的文本序列呢？

- **贪心搜索 (Greedy Search):** 最简单的方法。在每个时间步，都选择当前概率最高的那个词作为输出。
    - **优点:** 计算速度快，实现简单。
    - **缺点:** 过于短视，容易导致局部最优而非全局最优。例如，在第一步选择了一个看似最好但后续发展空间很小的词，可能会错过整体更优的句子。
- **集束搜索 (Beam Search):** 对贪心搜索的改进，是一种启发式搜索算法。
    - **工作原理:** 在每个时间步，不再只保留一个最优选择，而是保留 `k` 个（`k` 称为集束宽度，Beam Width）概率最高的候选序列。在下一个时间步，从这 `k` 个候选序列出发，分别生成下一个词，然后从所有可能的新序列中，再次选出 `k` 个总概率最高的序列。如此往复，直到生成结束符。
    - **优点:** 通过保留多个候选，降低了陷入局部最优的风险，生成的句子通常更流畅、更合理。
    - **缺点:** 计算成本更高，需要更多内存。`k` 值越大，效果越好，但计算开销也越大。

# 3. Vision Transformer (ViT)：让 Transformer "看见" 世界

ViT 的核心思想是：**尽可能地不修改原始 Transformer 架构，而是将图像数据转化为 Transformer 能处理的序列形式。**

## 3.1. 核心步骤

1. **图像分块 (Image Patching):** 将一张图像 (如 224x224) 切割成一系列固定大小的小块 (patches)，例如 16x16。这些小块就是图像的 "tokens"。
2. **块嵌入 (Patch Embedding):** 将每个展平 (flatten) 的图像块通过一个线性投影层，映射成固定维度的向量。
3. **位置编码:** 与 NLP 中的 Transformer 一样，为每个块嵌入向量添加可学习的位置编码，以保留其空间信息。
4. **[CLS] Token:** 在序列的开头加入一个特殊的可学习的 `[CLS]` (Classification) 令牌。这个令牌在经过 Transformer 编码器后，其对应的输出向量将被用作整个图像的全局表示，用于最终的分类任务。
5. **Transformer 编码器:** 将处理好的序列输入到标准的 Transformer 编码器中，通过自注意力机制捕捉图像块之间的全局关系。
6. **分类头 (Classification Head):** 将 `[CLS]` 令牌的最终输出送入一个简单的 MLP (多层感知机) 进行分类。

## 3.2. [CLS] Token 深度解析：信息如何汇总？

`[CLS]` Token 的设计借鉴自 BERT，它的作用是作为一个“信息汇总器”。

- **初始化:** 它被初始化为一个可学习的向量，不包含任何图像的初始信息。
- **信息流动:** 在 Transformer 的每一层中，`[CLS]` Token 和其他所有图像块 Token 一样，都参与自注意力计算。
    - 它会根据自己的 Q 向量去“查询”所有图像块的 K 向量，从而**主动地**从所有图像块中收集信息。
    - 同时，它的 K 和 V 向量也会被其他图像块查询，从而**被动地**将自己已经汇总的信息广播出去。
- **逐层聚合:** 经过一层层的自注意力计算，`[CLS]` Token 就像一个“漩涡中心”，不断地与其他所有图像块进行信息交换和聚合。到最后一层时，它的输出向量就包含了对整个图像序列的全局理解，成为了整个图像的最终表示。

### 3.3. ViT 面试核心问题

- **ViT 和 CNN 的根本区别是什么？**
    - **归纳偏置 (Inductive Bias) 不同。** 这是最核心的区别。
        - **CNN** 具有很强的归纳偏置：**局部性 (Locality)** 和 **平移不变性 (Translation Invariance)**。它默认图像中的像素关系是局部的（通过小卷积核实现），并且物体在图像中移动后仍然是同一个物体。这使得 CNN 在数据量较少时也能学得很好，数据效率高。
        - **ViT** 的归纳偏置很弱。它不对图像做任何先验假设，而是通过自注意力机制从数据中学习所有块之间的关系。这赋予了它捕捉全局关系的强大能力，但也意味着它**需要海量的数据**进行预训练才能学好，否则在小数据集上容易过拟合，效果不如 CNN。
- **ViT 的计算复杂度如何？**
    - 自注意力的计算复杂度是序列长度的平方，即 O(N2·d)，其中 N 是图像块的数量。对于高分辨率图像，N 会很大，导致计算成本急剧增加。这也是后续 Swin Transformer 等模型试图解决的关键问题。

## 4. Transformer 家族谱系与演进

原始的 Transformer 是一个 Encoder-Decoder 结构，但其强大的组件后来被拆分和独立发展，形成了三大主流分支：

1. **Encoder-Only (编码器架构，如 BERT):**
    - **特点:** 使用 Transformer 的编码器部分，能够同时看到上下文（双向）。
    - **核心思想:** 通过 Masked Language Model (MLM) 任务进行预训练，随机遮盖输入句子的一些词，然后预测这些被遮盖的词。
    - **适用场景:** 适合对输入文本进行**理解**的任务，如文本分类、情感分析、命名实体识别等。
2. **Decoder-Only (解码器架构，如 GPT):**
    - **特点:** 使用 Transformer 的解码器部分，只能看到前面的词（单向，自回归）。
    - **核心思想:** 通过标准的语言模型任务进行预训练，即根据前面的词预测下一个词。
    - **适用场景:** 适合**生成式**任务，如文本生成、对话系统、文章续写等。
3. **Encoder-Decoder (编码器-解码器架构，如 T5, BART):**
    - **特点:** 保留了完整的 Transformer 结构。

- **核心思想:** 将各种 NLP 任务都统一为“文本到文本 (Text-to-Text)”的格式。
- **适用场景:** 适合需要从一个序列映射到另一个序列的任务，如机器翻译、文本摘要等。

## 5. 训练细节与高级技巧

- **学习率调度 (Learning Rate Scheduler):** Transformer 的训练对学习率非常敏感。原始论文采用了一种特殊的学习率策略：在训练初期使用一个**线性预热 (Warm-up)** 阶段，让学习率从0线性增长到一个设定值，之后再让它按照**平方根倒数进行衰减**。这种“先增后减”的策略有助于在训练初期保持稳定，在后期更好地收敛。
- **标签平滑 (Label Smoothing):** 在分类任务中，传统的 one-hot 标签会鼓励模型对正确答案产生“过分自信”的预测（概率无限接近1）。标签平滑通过将一小部分概率（如0.1）从正确标签上分配给其他所有错误标签，来对模型进行“软化”。这相当于一种正则化手段，可以防止模型过拟合，提高泛化能力。

## 6. 宏观视角：模型缩放定律 (Scaling Laws)

这是一个由 OpenAI 等机构在训练大型 Transformer 模型时发现的重要规律。它指出：

在数据量足够大的情况下，语言模型的性能（通常用交叉熵损失来衡量）与模型大小 (N)、数据集大小 (D) 和训练所用的计算量 (C) 之间存在幂律关系。

这意味着模型的性能提升是**可以预测的**。我们可以通过在小模型上的实验结果，来推断出将模型、数据和计算量扩大10倍、100倍后，模型性能会达到什么水平。这个定律为设计和训练更大、更强的基础模型提供了理论指导，是现代大模型竞赛的基石。

## 7. 面试终极追问 (Q&A)

### Q1: Transformer 如何处理可变长度的输入？

**A:**

Transformer 模型要求同一个 batch 内所有输入序列长度一致，但实际数据往往长度不一。为了解决这一问题，Transformer 采用了以下两种方法：

1. **填充 (Padding)**：
   对所有序列进行填充，将短序列后面补充特殊的 `<PAD>` token，使得它们长度统一。
2. **注意力掩码 (Attention Mask)**：
   因为填充的 `<PAD>` token 并不包含有效信息，不能参与模型的注意力计算，Transformer 引入了 Attention Mask，用来屏蔽这些无效位置。

---

## Attention Mask 如何工作？

- Attention Mask 是一个与输入序列等长的二值矩阵，标记有效 token（1）和填充 token（0）。
- 在计算注意力分数（即 QK^T）后，模型会将填充 token 对应位置的分数加上一个极大的负数（如 -1e9），
  这样经过 softmax 归一化后，这些填充位置的注意力权重接近 0。
- 因此，填充 token 不会对最终的注意力输出产生影响，确保模型只关注有效 token。

---

## 为什么要这样做？

- 避免无效的填充 token 干扰模型学习和输出结果。
- 保证模型的注意力计算只在真实输入数据上进行，提升训练和推理的准确性。
- 支持批处理时可变长度序列的高效计算。

---

## 简单类比

可以把 Attention Mask 想象成“会议中空座位的‘勿扰’标志”，使得与会者（token）不会把注意力浪费在空位（padding）上，而只专注于真正的发言者（有效 token）。

---

## 总结

- **Padding** 统一序列长度；
- **Attention Mask** 屏蔽 padding，防止信息泄露和干扰；

- 这两者结合，保证 Transformer 能高效且准确地处理可变长度的序列数据。

---

## Q2: Transformer 中的参数主要集中在哪里？

**A:** 主要集中在以下两个部分：

1. **前馈神经网络 (Feed-Forward Network)**：
    - FFN 的中间层维度通常是模型维度的 4 倍；
    - 每层包含两个大的权重矩阵；
2. **嵌入层 (Embedding Layer)**：
    - 参数量 = 词表大小 × 模型维度；
    - 对于大词表，这部分参数非常可观。

> *相比之下，自注意力机制中的 $Q$、$K$、$V$ 权重矩阵参数量相对较小。*

---

## Q3: 有哪些为了解决 Transformer 计算瓶颈而设计的变体？

**A:** 常见优化 Transformer 计算效率的架构有：

- **Swin Transformer**：
    - 引入局部窗口注意力（Window Attention）；
    - 再通过 Shifted Window 实现跨窗口通信；
    - 计算复杂度显著下降，适合视觉任务。

> ***Swin Transformer 是怎么做的？***
>
> - *窗口划分：把输入序列（或图像特征）划分成多个固定大小的小窗口（比如 7x7 的小块），只在窗口内计算自注意力。*
> - *局部计算：每个窗口内部计算自注意力，减少了全局计算量。*
> - *窗口滑动：通过“Shifted Window”技术，滑动窗口位置，让不同窗口之间可以信息交流，保证全局信息传递。*
> - *结果：计算量大幅降低，适合图像等大尺寸输入。*

- **Longformer / BigBird**：
    - 引入稀疏注意力模式（如滑动窗口、全局注意力等）；

- 将原本的 $O(N^2)$ 注意力降低为近似线性的 $O(N)$；
- 适合长文本处理任务。

> ***Longformer / BigBird 是怎么做的？***
>
> - *稀疏注意力设计：并不是所有的token 都互相关注，而是只关注部分token，比如：*
>   - *每个token只看邻近一段滑动窗口内的token（局部关注）。*
>   - *加上一些“全局token”被所有token关注（全局关注）。*
>   - *加上一些随机关注（增强多样性）。*
> - *结果：避免了原本所有 token 两两注意力计算，复杂度从 $O(N^2)$ 降到 $O(N)$，适合超长文本。*

- **FlashAttention**：
  - 注意力机制的高效 GPU 实现；
  - 利用分块计算与高速缓存减少内存访问；
  - 在不改变数学本质的前提下，大幅提升显存利用与速度。

> - ***Swin 和 Longformer/BigBird** 主要“改模型结构”或“改注意力计算范围”，让计算少很多；*
> - ***FlashAttention** 主要“改计算实现”，用更高效的算法和硬件技巧提升速度和显存效率。*

---

## Q4: 多头注意力（Multi-Head Attention）的本质好处是什么？

**A:**

多头注意力的核心优势在于：

1. **多视角捕捉信息**
   模型通过多个“头”同时关注输入的不同部分，每个头学习不同的注意力模式，能更全面地理解句子。
2. **增强表达能力**
   每个头关注不同的语言特征，比如语法结构、词义关系或上下文依赖，使模型更细致地捕捉信息。
3. **提升泛化和鲁棒性**
   多个注意力头共同作用，减少单一注意力模式的局限，模型对各种输入更稳定，泛化能力更强。

简言之，多头注意力就像多双“眼睛”，让模型能同时从不同角度看问题，提升理解和生成效果。

---

## Q5: 为什么注意力计算要除以 $\sqrt{d_k}$？

**A:** 因为：

- Q 与 K 做点积时，数值的方差为 $d_k$；
- $d_k$ 越大，softmax 前的值越大 → 结果趋近 one-hot → 梯度消失；
- 因此，使用 $\sqrt{d_k}$ 进行缩放，保持梯度分布稳定，有助于训练。

---

## Q6: Pre-LN 和 Post-LN 的区别？

**A:**

| 名称 | 结构 | 优点 | 缺点 |
| --- | --- | --- | --- |
| **Post-LN** | `LayerNorm(x + Sublayer(x))` | 残差连接直观 | 训练初期梯度不稳定，需要 warm-up |
| **Pre-LN** | `x + Sublayer(LayerNorm(x))` | 稳定性更强，支持更大模型 | 理论解释较弱，结构稍复杂 |

> *GPT-2、GPT-3 等模型普遍使用**Pre-LN**。*

---

## Q7: Attention 本质上是不是一种可微分的 kNN（soft kNN）？

**A:** 是的，Attention 可以被视为一种 soft kNN 查询机制：

- Q 是查询向量；
- K 是数据库索引；
- Softmax(QK^T) 得到每个“邻居”的权重；
- 最终输出是所有 V 的加权和。

---

## Q8: Transformer 中的 LayerNorm 为什么优于 BatchNorm？

**A:**

- **BatchNorm** 依赖 batch 中多个样本的统计值，容易被填充位置干扰；
- **LayerNorm** 是对每个样本自身进行归一化，适用于 NLP 的变长输入；
- 独立于 batch size，稳定性更强。

---

## Q9: [额外加分] 你能举出几个现代 Transformer 的高效注意力变体吗？

**A:**

| 模型 | 技术 | 优点 |
| --- | --- | --- |
| **Performer** | 线性 Attention（基于核函数） | 近似 softmax，O(N) 时间复杂度 |
| **Reformer** | LSH + 可逆网络 | 低内存，低复杂度 |
| **Linformer** | 低秩投影 | 减少 K/V 维度，加快计算 |
| **RetNet** | 状态传播机制 | 模拟 RNN 收敛性，适合长序列 |

Q10: 讲讲你了解的、为解决 Transformer 计算瓶颈而设计的变体？

A:

- **Swin Transformer:** 针对 ViT 在视觉任务中的计算瓶颈，引入了**窗口化自注意力 (Windowed Self-Attention)**，只在局部窗口内计算注意力，并通过**移位窗口 (Shifted Window)** 机制实现跨窗口的信息交互，大大降低了计算量。
- **Longformer/BigBird:** 针对处理长文本的挑战，提出了**稀疏注意力机制**。它们将完整的 O(N2) 注意力替换为几种稀疏注意力的组合，如滑动窗口注意力（关注邻近词）、全局注意力（让少数重要词能关注全局）等，将计算复杂度从 O(N2) 降低到近似线性的 O(N)。
- **FlashAttention:** 这是一种 I/O 感知的注意力算法，它不直接实例化巨大的注意力矩阵，而是通过分块计算、利用 GPU SRAM 高速缓存等技巧，极大地减少了 GPU 显存的读写量，从而在不改变数学计算结果的前提下，大幅提升了注意力的计算速度和显存效率。

## Q11: GPT 的 Attention Mask 是怎么构建的？是否能同时支持 Padding 和 Causality？

答：

是的。GPT 等 Decoder-only Transformer 同时支持两种 Mask：

- **Causal Mask（因果掩码）**：确保每个 token 只能关注它前面的 token，自回归生成；
- **Padding Mask（填充掩码）**：用于屏蔽掉 padding token 的注意力分数，避免无效信息干扰。

最终的 Attention Mask 是这两个掩码的组合：先构造一个下三角矩阵（Causal），再在其中的 padding 部分加掩码。Softmax 后，屏蔽位置的权重会变为 0。

> ***GPT Attention Mask** 关键记忆点*
>
> *1. 双重掩码*
>
> - *Causal Mask：保证只看当前词前面的内容（自回归），防止信息泄露*
> - *Padding Mask：屏蔽填充 token，避免无效信息干扰*
>
> *2. 实现细节*
>
> - *先构造下三角矩阵（Causal Mask）*
> - *在 padding 位置加大负数掩码*
> - *Softmax 后被掩码位置权重为 0*
>
> *3. 为什么要用大负数掩码？*
>
> - *softmax 对大负数趋近于 0，确保无关注力*
>
> *4. 不做 **Padding Mask** 会怎样？*
>
> - *填充位置影响计算，导致性能下降*
>
> *5. **Encoder-Decoder** 也用类似 **Mask***
>
> - *Encoder 主要用 Padding Mask*
> - *Decoder 用 Causal + Padding Mask*
> - *Encoder-Decoder Attention 也屏蔽 padding*
>
> *6. 作用*
>
> - *保证关注合法有效信息，提升训练稳定性和生成质量*

---

## Q12: 为什么 Transformer 更擅长建模长序列？有没有缺陷？

答：

- **优势：**
    - **并行性高**：不依赖时间步，支持所有 token 同时处理；
    - **全局建模能力强**：自注意力机制能捕捉任意两点间的关系，路径长度为 O(1)，RNN 是 O(n)。
- **缺点：**
    - 注意力机制的复杂度为 **$O(n^2)$**，当序列变长时计算和显存开销非常大；
    - 缺乏 RNN 的“记忆”结构，对局部连续依赖建模不如某些递归网络自然。

---

## Q13: Self-Attention 和 Cross-Attention 的本质区别是什么？

答：

| 类型 | **QUERY 来源** | **KEY / VALUE 来源** | 使用位置 |
| --- | --- | --- | --- |
| Self-Attention | 当前层的自身输出 | 当前层的自身输出 | Encoder / Decoder |
| Cross-Attention | Decoder 的前层输出 | Encoder 的最终输出 | Decoder Only |

- **Self-Attention：** 模型学习输入序列内部的联系；
- **Cross-Attention：** 模型学习“输出”和“输入”之间的对应关系（如翻译时英文与法文对齐）。

---

## Q14: 词嵌入和位置编码如何结合？能否混淆？可以不用加法吗？

答：

- 标准做法是**直接相加**：

$$\text{Input} = \text{TokenEmbedding} + \text{PositionalEncoding}$$

- 二者不会“混淆”，而是共同构成词语的“含义 + 位置”信息；
- 但研究发现，加法可能不是最优的融合方式，还有一些替代设计：
    - 拼接 (concatenation)；
    - 加权融合（Gated Embedding）；
    - 动态位置编码（如 ALiBi、RoPE）。

## Q15: 多语言 Transformer 是怎么支持多语言的？

答：

- **共享词表**：用 BPE 或 SentencePiece 构造跨语言的统一子词词表；
- **共享 Transformer 权重**：不同语言的输入共享同一个模型；
- **语言提示 Token**：给每个句子添加如 `<EN>`、`<ZH>` 的语言标识；
- **优势**：模型可以迁移不同语言间的知识，提升低资源语言性能；
- **代表模型**：mBERT、XLM、mT5。

---

## Q16: Transformer 中的参数压缩方法有哪些？（剪枝 / 共享）

答：

- **参数共享（Parameter Sharing）**：
  - ALBERT：所有 Transformer 层参数共享；
  - T5：Encoder 和 Decoder 间权重共享；
  - 能显著减少参数数量，同时保持效果。
- **剪枝（Pruning）技术**：
  - 移除冗余的 Attention Head；
  - 对 FFN 的中间维度进行剪枝；
  - 使用 L0 Regularization 或 梯度稀疏化 策略选择重要通道。
- **应用场景**：移动端部署、大模型压缩、微调提速等。

---

## Q17: 除了 Warm-up 和 Label Smoothing，还有哪些训练技巧值得知道？

答：

- **Dropout**：用于 Attention 和 FFN 中防止过拟合；
- **Gradient Clipping**：裁剪梯度范数，避免梯度爆炸；
- **Weight Decay**：L2 正则化，提升泛化能力；
- **Mixed Precision 训练**：使用 FP16 训练加速计算并节省显存；

- **Checkpoint Averaging**：最后几轮模型平均，提升稳定性。

---

## Q18: BERT 与 GPT 在 Attention Mask 和结构设计上有何根本差异？

答：

| 特征 | BERT | GPT |
| --- | --- | --- |
| 架构类型 | Encoder-only | Decoder-only |
| Attention Mask | 双向（无因果掩码） | 单向（Causal 掩码） |
| 输入方式 | 整个句子（双向） | 左到右自回归输入 |
| 预训练任务 | Masked LM (MLM) | 自回归语言建模 |
| 应用任务 | 分类、理解、抽取类任务 | 文本生成、对话、代码生成等 |

---

## Q19: FlashAttention 到底“快”在哪里？和原 Attention 有什么数学上的区别？

答：

- **数学上没有区别**：FlashAttention 计算的 Attention 和原始公式结果**完全一致**；
- **核心优化在系统实现**：
    - 避免生成完整的 Attention 矩阵；
    - 使用 **块级 Softmax**，分段处理；
    - 最大限度利用 GPU SRAM（而不是慢速 HBM）；
- **实际效果**：在长序列任务上显著提升训练速度，降低显存消耗。

---

## Q20: 你能用一句话总结 Transformer 信息是如何在层间流动的吗？

答：

Transformer 中的信息通过多层堆叠的自注意力机制捕捉序列中任意位置的依赖，再结合位置编码、前馈网络和残差连接，实现了对序列全局语义的高效表达与传递。

---

## Q21: 假如你要优化 Transformer，你会从哪些方向入手？

答：

可考虑的优化方向包括：

- **结构优化：**
    - 替换 Attention（如 Performer, Linear Attention）；
    - 混合 CNN / RNN 特性（如 Conformer、gMLP）；
- **计算优化：**
    - 稀疏 Attention（Longformer, BigBird）；
    - FlashAttention / Memory-efficient Attention；
- **可解释性：**
    - 理解 Attention 权重与实际推理关系；
    - 利用显著性方法解释模型关注点；
- **多模态拓展：**
    - 融合图像、语音、代码等其他模态；
- **推理部署优化：**
    - 量化、剪枝、蒸馏、MoE 等。