面经—Transformer篇
字节一面

1. transformer encoder结构, 位置编码用的是什么 除了正余弦位置编码还有什么别的
2. 具体讲multi head attention怎么做的以及公式 为什么要除根号下dk
3. transformer decoder和encoder有哪些不同的地方
4. Cross4 attention中q k v分别来自哪里?
5. 带mask的self attention是什么样的mask
6. 为什么transformer中用的是ln不是bn
7. 交叉熵损失函数公式 为什么分类问题只能用交叉熵不能用mse

代码题:

1. 交叉熵损失函数公式写出来
2. 求y=wx+b梯度
3. 两个数组的子序列求最大点积 有点难 用二维dp才行

快手star二面

1. 写了self attention
2. 为什么要除根号下dk，能不能不是根号下dk是别的？(从数学原理上答)
3. multi head attention怎么做的
4. **具体讲Multi-Head Attention怎么做的以及公式，为什么要除根号下dk?**
5. **Transformer Decoder和Encoder有哪些不同的地方?**
6. **Cross Attention中Q、K、V分别来自哪里?**
7. **带Mask的Self-Attention是什么样的Mask?**
8. **为什么Transformer中用的是LayerNorm而不是BatchNorm?**
9. **交叉熵损失函数公式，为什么分类问题只能用交叉熵不能用MSE?**

代码题:

1. **交叉熵损失函数公式写出来:**
2. **求 $y = wx + b$ 的梯度:**
3. **两个数组的子序列求最大点积:**

快手star二面

1. **写了Self-Attention**
2. **为什么要除根号下dk，能不能不是根号下dk是别的?**
3. **Multi-Head Attention怎么做的?**

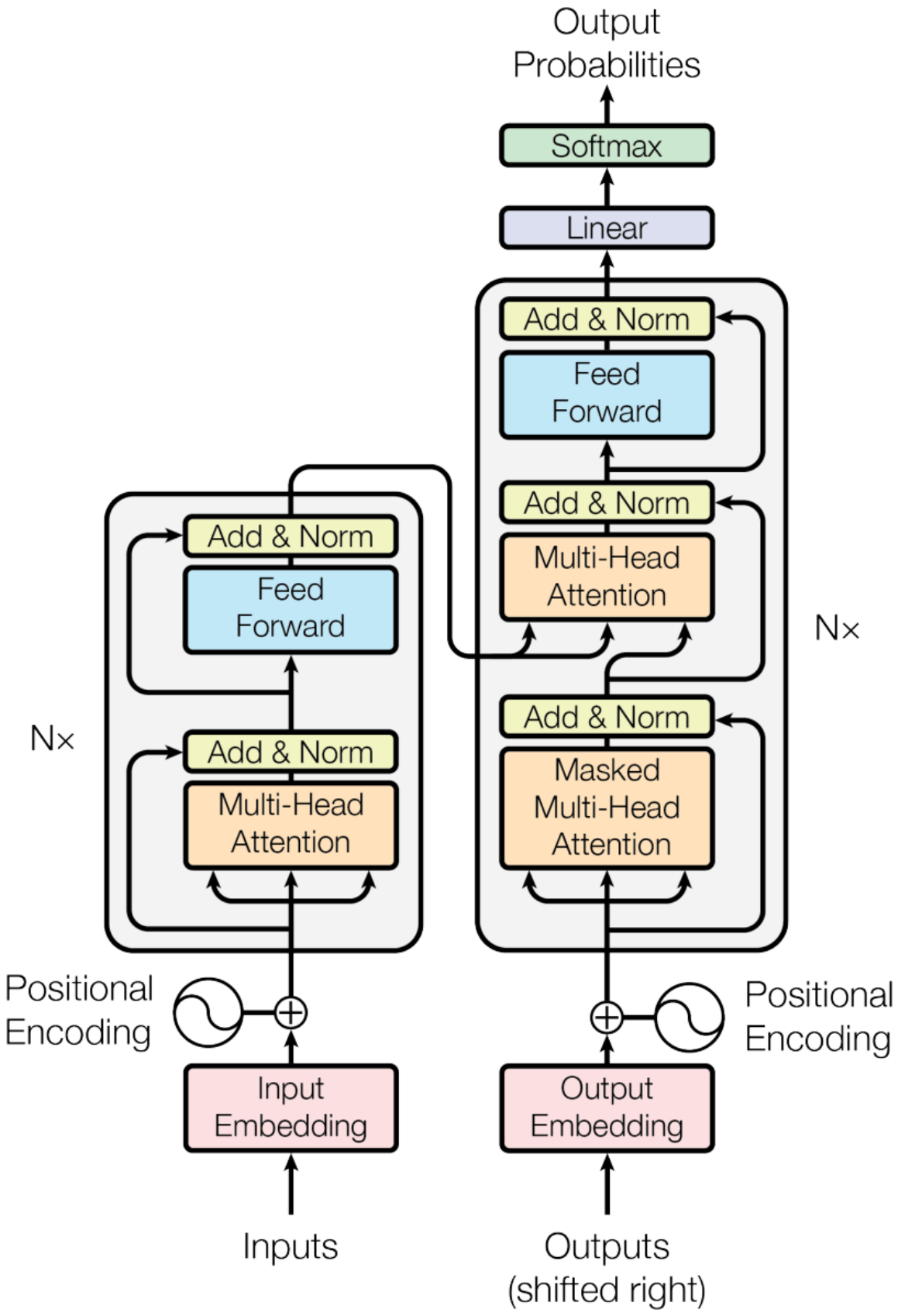
🔷 **Transformer 面经篇**

1. 基础概念
2. 进阶机制
3. 实际应用
4. 优化与改进
5. 场景题 / 开放题

面经—Transformer篇

参考文档:

https://zhuanlan.zhihu.com/p/338817680


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

字节一面

1. transformer encoder结构, 位置编码用的是什么 除了正余弦位置编码还有什么别的

- **Transformer Encoder结构：**
  Transformer Encoder 由6个相同的层组成，每个层包含两个子层：
  - **Multi-Attention层**：计算输入的每个词与其它词的关系。
  - **Feed-Forward Neural Network层**：独立应用于每个位置的全连接层，通常包括ReLU激活。
  - 每个子层后都带有 **残差连接** 和 **LayerNorm**。
- **残差连接：**
- **位置编码：**
  Transformer本身不处理序列的顺序，因此需要将位置信息通过位置编码加入到输入中。最常见的是 **正余弦位置编码**，其公式为：
  $PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$
  $PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$
  - **除了正余弦位置编码，还有什么别的：**
    - **Learnable Position Embeddings**：这种方式通过学习一个位置向量，而不是使用固定的公式，适用于较小序列长度的任务。
    - **Rotary Position Embedding (RoPE)**：一种基于旋转的编码方式，常用于多模态Transformer。

2. 具体讲multi head attention怎么做的以及公式 为什么要除根号下dk

Multi-Head Attention：

- **步骤**：
  1. **输入**：输入为三个矩阵 Q（Query）、K（Key）和 V（Value），它们通过线性变换从输入嵌入生成。
  2. **Attention机制**：对每个头，计算 $Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$。
  3. **头连接**：每个头的结果拼接起来，再通过一个线性层映射到输出。
- **公式**： $Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

Multi-Head Attention代码:

```
1
```

- **2.1内积（Dot Product）的增长速度**

  当 Q 和 K 的维度 d_k 很大时，内积 QK^T 的值会非常大。这是因为向量内积的结果是所有对应元素乘积的和，随着维度的增加，数值增长会变得非常快。

  对于两个高维向量 Q 和 K，它们的内积为：$QK^T = \sum_{i=1}^{d_k} Q_i K_i$

  如果 d_k 很大，每个 Q_i 和 K_i 可能都比较大，那么它们的乘积之和（内积）也会非常大。特别是当 Q 和 K 具有相同的分布时，内积值增长的速度是非常快的。
- **2.2 Softmax的数值范围问题**

Softmax 函数的定义是：$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

其中，x_i 是输入的元素（在 Attention 中，输入是 $\frac{QK^T}{\sqrt{d_k}}$）。Softmax 的主要作用是将输入值转换成概率分布。Softmax 会对每个输入计算指数函数 e^{x_i}，然后对所有输入的指数结果求和，再除以总和。当输入的值（即 $\frac{QK^T}{\sqrt{d_k}}$）很大时，指数函数 e^{x_i} 会非常大。举个例子，如果 QK^T 的某个值为 1000，经过指数运算后会变得非常大，而其他较小的值（例如 1 或 -1）会几乎变成零。这使得 Softmax 的输出会非常不平衡，即大部分权重会集中在一个位置，导致梯度计算时，只有一个位置的梯度值较大，其他位置的梯度几乎为零。

- **2.3 *梯度消失问题***

Softmax 函数的梯度为：$\frac{\partial \text{softmax}(x_i)}{\partial x_j} = \text{softmax}(x_i)(\delta_{ij} - \text{softmax}(x_j))$

> 其中，δ_{ij} 是 Kronecker delta（当 $i = j$ 时为1，否则为0）。
>
> Kronecker delta (δ_{ij}) 是一种数学符号，通常用于描述两个变量是否相等。它的定义如下：
>
> $$\delta_{ij} = \begin{cases} 1, & \text{如果 } i = j \\ 0, & \text{如果 } i \neq j \end{cases} \tag{1}$$
>
> 换句话说，δ_{ij} 是一个矩阵的元素，在 $i = j$ 时等于1，在 $i \neq j$ 时等于0。在 Softmax 的梯度中，δ_{ij} 是用来指示当我们计算梯度时是否在同一个类别的位置。如果 $i = j$，说明我们是在对角线位置计算梯度，也就是自己与自己的关系；如果 $i \neq j$，则是计算不同类别之间的关系。
>
> 当 x_i 很大时，$\text{softmax}(x_i)$ 会非常接近1，而其他类别的 $\text{softmax}(x_j)$（当 $j \neq i$）会非常接近于0。因此，公式中的 δ_{ij} 和 $\text{softmax}(x_j)$ 的关系决定了梯度的大小。
>
> - **对于 $i = j$**：$\text{softmax}(x_i)$ 趋近于1，$\text{softmax}(x_j)$ 趋近于0。因此，梯度为：
>
> $$\frac{\partial \text{softmax}(x_i)}{\partial x_i} = \text{softmax}(x_i)(1 - \text{softmax}(x_i)) \approx 1 \times (1 - 1) = 0 \tag{2}$$
>
> - **对于 $i \neq j$**：$\text{softmax}(x_i)$ 趋近于1，而其他的 $\text{softmax}(x_j)$ 值接近0，因此梯度为：
>
> $$\frac{\partial \text{softmax}(x_i)}{\partial x_j} = \text{softmax}(x_i)(0 - \text{softmax}(x_j)) \approx 1 \times 0 = 0 \tag{3}$$

- 因此，当 x_i 很大时，$\text{softmax}(x_i)$ 会非常接近1，而其他的 $\text{softmax}(x_j)$ 会接近0。这个现象导致了 **Softmax** 函数在某个类别的得分非常大时，输出的概率几乎全部集中在这个类别，其他类别的概率几乎为0，从而导致了梯度的消失问题。
- 这会导致学习变得困难，因为梯度更新非常小，尤其是在多层网络中，梯度消失会使得权重更新变慢，甚至停止学习。

- **2.4 为什么除以 $\sqrt{d_k}$ 可以解决问题?**

为了防止这种情况，Transformer 通过除以 $\sqrt{d_k}$ 来**缩放**内积结果，使得内积的值不会过大，从而避免了 Softmax 的梯度过小。

- **数学解释**：

  通过除以 $\sqrt{d_k}$，我们有效地减少了 QK^T 的数值范围，使得其结果更加平衡。特别是对于大规模的模型，随着 d_k 的增大，内积的值增长得更快，所以通过 $\sqrt{d_k}$ 来缩放是有效的，使得内积值保持在合理的范围内。

  这就像是给输入加了一个“标准化”，让 Softmax 的输出不至于变得极端，从而避免梯度消失问题。

例如，如果 d_k 很大， QK^T 的值可能会达到上万，通过 $\sqrt{d_k}$ 的缩放，使得结果更稳定，Softmax 的梯度能够更加均匀地传播，促进模型更有效地学习。

- **3.1 为什么会选择 $\sqrt{d_k}$ 而不是其他缩放因子?**

  在 Attention 机制中， $\frac{QK^T}{\sqrt{d_k}}$ 是为了避免内积 QK^T 的值过大。首先，回顾一下内积的计算过程：
  $QK^T = \sum_{i=1}^{d_k} Q_i K_i$

  随着 d_k （Query 和 Key 的维度）增大，内积的数值会变得非常大，这会影响 Softmax 的梯度计算。由于 Softmax 的梯度依赖于输入值的差异，如果输入值的差异过大，梯度会非常小（接近0），导致梯度消失问题。

  期望值分析：假设 Q 和 K 是从均值为零、方差为 $1/d_k$ 的分布中采样的（假设是高斯分布），那么它们的内积 QK^T 的期望值大致是零，方差大约为 d_k。所以，内积的结果随着 d_k 的增大而增大。

  - 为了避免内积的数值变得过大，我们通过 $\sqrt{d_k}$ 来缩放，使得内积的方差不会随着 d_k 的增大而变得过大，从而确保它保持在合适的范围。

  统计学原理：从**中心极限定理**来看，多个独立的随机变量的和（即内积）会随着维度的增加而变得更加分布集中，标准差增加的速度是 $\sqrt{d_k}$。因此，使用 $\sqrt{d_k}$ 来缩放内积结果是自然的选择，因为它能确保内积的数值稳定在合理范围内，不会过大或过小。

  避免梯度爆炸或消失：如果没有缩放，内积 QK^T 会随着 d_k 增大而增大，导致 Softmax 的输出非常不平衡。这样，梯度几乎全部集中在最大值类别，其他类别的梯度几乎为零，造成 **梯度消失**。如果缩放因子过小（例如除以 d_k 或不缩放），则内积的数值会过小，导致 **梯度爆炸**。而 $\sqrt{d_k}$ 确保了数值范围合理，使得梯度既不会过小也不会过大，从而稳定训练过程。

3. transformer decoder和encoder有哪些不同的地方

- **Encoder**：
  - 输入：处理原始的输入序列。
  - 只有 **Self-Attention** 层：每个位置的表示考虑了输入序列中所有位置的信息。
- **Decoder**：
  - 输入：接收Encoder的输出以及已经生成的目标序列（通过 **Masked Self-Attention**）作为输入。
  - **Masked Self-Attention**：Decoder中的Self-Attention层使用掩码，防止模型在生成过程中查看未来的词。
  - **Cross-Attention**：Decoder有一个额外的Cross-Attention层，用于接收来自Encoder的上下文信息。

4. Cross4 attention中q k v分别来自哪里?

5. 带mask的self attention是什么样的mask

6. 为什么transformer中用的是ln不是bn

7.交叉熵损失函数公式 为什么分类问题只能用交叉熵不能用mse

代码题：

1. 交叉熵损失函数公式写出来

2. 求y=wx+b梯度

3. 两个数组的子序列求最大点积 有点难 用二维dp才行

快手star二面

1.写了self attention

2.为什么要除根号下dk，能不能不是根号下dk是别的？(从数学原理上答)

3. multi head attention怎么做的

---

2. 具体讲Multi-Head Attention怎么做的以及公式，为什么要除根号下dk?

---

3. Transformer Decoder和Encoder有哪些不同的地方?

-

---

4. Cross Attention中Q、K、V分别来自哪里?

- **Q (Query)**：来自Decoder的上一层的输出。
- **K (Key)** 和 **V (Value)**：来自Encoder的输出。

在 **Cross Attention** 中，Decoder的查询 Q 与Encoder的键 K 和值 V 进行交互，生成Decoder的注意力输出。这使得Decoder能够在生成时根据Encoder的上下文来调整其输出。

---

5. 带Mask的Self-Attention是什么样的Mask?

- **Mask**：用于 **Decoder** 的Self-Attention，以确保每个位置的查询只能与当前及之前的位置进行交互（防止模型看到未来信息）。
  - **Mask矩阵**：对未来位置的计算加以掩盖，通常使用负无穷来替代无效的元素，使得这些元素在Softmax后得到0权重。
  - **示例**：
    对于位置 t，掩码会使得计算中的 $t+1, t+2, \ldots$ 被屏蔽掉。

---

6. 为什么Transformer中用的是LayerNorm而不是BatchNorm?

- **LayerNorm**（层归一化）适用于Transformer，因为它是在每个位置上进行归一化的，而 **BatchNorm** 是对整个batch做归一化。
  - **Transformer的特性**：输入序列的长度可变，BatchNorm的依赖于批次大小，而LayerNorm对每个样本独立地进行标准化，这更适合处理变长输入。
  - **训练时问题**：Transformer中每层的计算并不依赖于批次维度，因此BatchNorm在此场景下不如LayerNorm稳定。

---

7. 交叉熵损失函数公式，为什么分类问题只能用交叉熵不能用MSE?

- **交叉熵损失函数**公式：
  $L = -\sum_{i=1}^{N} y_i \log(p_i)$
  其中，y_i 是真实标签的概率分布（一般是 one-hot 编码），p_i 是预测的概率。
- **为什么分类问题用交叉熵而不用MSE**：
  - **MSE**适用于回归任务，它通过最小化误差平方和来调整预测值。然而，**分类问题**要求模型输出的是每个类别的概率，交叉熵损失能度量预测分布与真实分布之间的差异，优化目标是最大化正确类的概率。
  - **MSE**会倾向于给每个类别赋予相似的预测值，可能导致模型产生错误的概率输出（例如，非概率分布）。而交叉熵损失会使得模型更加关注正确类别，避免了这种情况。

代码题：

1. 交叉熵损失函数公式写出来：

```python
import torch
import torch.nn as nn

# 交叉熵损失函数
loss_fn = nn.CrossEntropyLoss()
output = model(input)  # 模型输出
target = labels  # 真实标签
loss = loss_fn(output, target)
```

2. 求 $y = wx + b$ 的梯度：

```python
import torch

w = torch.tensor([2.0], requires_grad=True)  # 权重
x = torch.tensor([3.0], requires_grad=True)  # 输入
b = torch.tensor([1.0], requires_grad=True)  # 偏置

y = w * x + b  # 线性函数
y.backward()  # 计算梯度

print(w.grad)  # 输出 w 的梯度
print(b.grad)  # 输出 b 的梯度
print(x.grad)  # 输出 x 的梯度
```

3. 两个数组的子序列求最大点积：

```python
# 求两个数组的子序列最大点积，使用二维DP
def max_dot_product(nums1, nums2):
    n = len(nums1)
    dp = [[0] * (n+1) for _ in range(n+1)]

    for i in range(1, n+1):
        for j in range(1, n+1):
            dp[i][j] = max(dp[i-1][j-1] + nums1[i-1] * nums2[j-1], dp[i-1][j], dp[i][j-1])

    return dp[n][n]

# 示例
nums1 = [1, 2, 3]
nums2 = [3, 2, 1]
print(max_dot_product(nums1, nums2))  # 输出 10
```

快手star二面

1. 写了Self-Attention

- 请参考前面讲解的Self-Attention的计算公式和步骤。

2. 为什么要除根号下dk，能不能不是根号下dk是别的?

- **数学原理**：除以 $\sqrt{d_k}$ 是为了稳定计算。在Attention中，内积值会随着词向量维度增大而增大，导致Softmax的梯度消失。通过根号下dk的除法，使得内积值在合适的范围内。

3. Multi-Head Attention怎么做的?

- 如第2部分所述，Multi-Head Attention将输入的Q、K、V分成多个头，通过多个子空间计算不同的Attention，每个头的结果拼接后通过线性变换输出。

希望这些答案能帮助你准备面试！如果需要进一步解释某些部分，随时告诉我。

你要的 **Transformer 面经篇** 我帮你整理成一份面试常见问答清单，分为 **基础概念 → 进阶机制 → 实际应用 → 优化改进 → 场景题** 五个部分，适合算法岗、AIGC岗、大模型岗等面试。

🔷 Transformer 面经篇

1. 基础概念

- **Q1:** Transformer 的整体架构是什么？和 RNN/CNN 的区别?
  - Encoder-Decoder 架构；基于 Attention 而非循环/卷积；并行计算、长依赖捕捉更强。
- **Q2:** 为什么需要位置编码 (Positional Encoding)?
  - Transformer 不具备顺序建模能力，位置编码通过 sin/cos 或 learnable embedding 注入序列顺序信息。
- **Q3:** Self-Attention 的计算公式?
  - $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$
- **Q4:** 为什么要除以 $\sqrt{d_k}$?
  - 防止 QK^T 内积值过大，导致 softmax 梯度消失。

---

2. 进阶机制

- **Q5:** Multi-Head Attention 的好处?
  - 多头可以从不同子空间学习不同的关系，提升模型表达能力。
- **Q6:** Residual + LayerNorm 为什么重要?
  - 残差保证梯度流动，LayerNorm 保持数值稳定，加速收敛。
- **Q7:** Encoder 和 Decoder 的区别?
  - Encoder: Self-Attention（双向）；Decoder: Masked Self-Attention（单向）+ Cross-Attention（和 Encoder 交互）。
- **Q8:** 为什么 Decoder 需要 Mask?
  - 防止模型泄漏未来信息，保证自回归生成。

---

3. 实际应用

- **Q9:** Transformer 在 NLP 以外的应用?
  - CV（ViT）、语音（Speech-Transformer）、推荐系统（Sequential RecSys）、强化学习、时间序列预测等。
- **Q10:** ViT 和 CNN 的区别?
  - ViT 把图像分 patch，当作 token 输入，依赖全局 Attention；CNN 擅长局部感受野。

- **Q11:** 在推荐系统中如何应用 Transformer?
  - 用户行为序列建模（Self-Attention 替代 RNN），支持长序列兴趣建模（DIN/DIEN → SASRec → BERT4Rec）。

---

4. 优化与改进

- **Q12:** Transformer 的缺点?
  - 计算复杂度 $O(n^2)$，长序列难处理，推理慢。
- **Q13:** 有哪些长序列改进方法?
  - Sparse Attention (Longformer)、低秩近似 (Linformer)、因果卷积 (Performer)、Recurrence (Transformer-XL)。
- **Q14:** 参数优化和训练技巧?
  - Mixed Precision、梯度累积、Warmup 学习率、正则化 (Dropout/Label Smoothing)。
- **Q15:** 为什么要用 Pre-Norm 代替 Post-Norm?
  - Pre-Norm 稳定梯度，适合深层模型；Post-Norm 在浅层时表现更好。

---

5. 场景题 / 开放题

- **Q16:** 如果面试官问："如何用 Transformer 改进推荐系统排序模型？"
  - 答：可以把用户行为序列作为输入，使用 Self-Attention 捕捉长序列依赖，融合多模态信息（图文/视频 embedding），最后用多任务学习优化点击率/停留时长。
- **Q17:** 如果问："Transformer 为什么能做 CoT（思维链）推理？"
  - 答：多层 Attention 可逐层抽取、组合信息；训练时通过 CoT 数据增强，让模型学会逐步生成推理链条。
- **Q18:** 如果问："如何优化 Transformer 在大规模训练时的效率？"
  - 答：分布式训练（DP/TP/PP）、ZeRO 优化、MoE 稀疏化、FlashAttention 提速。

📌 **总结：**

- **基础题**（架构、公式、位置编码）必问。
- **进阶题**（多头注意力、Mask、LayerNorm）考察理解深度。
- **应用题**（NLP 以外、推荐系统、CV）考察知识面。
- **优化题**（长序列、高效训练）考察工程能力。
- **场景题**（开放式）考察解决实际问题的能力。