

## 时间线

- 241023
- 241024 提取关键词、录音模拟、真实输出 80%够了
  - <https://www.tianjinsc.cn/default/2002321.html>
  - [https://github.com/DA-southampton/NLP\\_ability/tree/master/](https://github.com/DA-southampton/NLP_ability/tree/master/)

## Transformer发家史

为什么用三角函数表示位置编码?

Transformer为什么使用多头注意力机制? 为什么不使用一个头?

Transformer为什么Q和K使用不同的权重矩阵生成, 为什么不使用同一个值进行自身的点乘? 注意和上一个问题 多头注意力机制 的区别

Transformer在计算attention的时候为什么使用点乘 而不是加法?

两者计算复杂度和效果上有什么区别?

为什么在进行softmax之前需要对attention进行scaled (为什么除以dk的平方根) , 并使用公式

在计算attention score的时候如何对padding做mask操作?

为什么在进行多头注意力的时候需要对每个head进行降维? (可以参考上面一个问题)

大概讲一下Transformer的Encoder模块?

Transformer中decoder的结构

为何在获取输入词向量之后需要对矩阵乘以embedding size的开方? 意义是什么?

简单介绍一下Transformer的位置编码? 有什么意义和优缺点?

你还了解哪些关于位置编码的技术, 各自的优缺点是什么?

简单讲一下Transformer中的残差结构以及意义。

为什么transformer块使用LayerNorm而不是BatchNorm?

LayerNorm 在Transformer的位置是哪里?

简答讲一下BatchNorm技术, 以及它的优缺点。

LayerNorm在Transformer中的应用

简单描述一下Transformer中的前馈神经网络？使用了什么激活函数？相关优缺点？

Encoder端和Decoder端是如何进行交互的？（在这里可以问一下关于seq2seq的attention知识）

Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？

为什么需要decoder自注意力需要进行 sequence mask

Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？

Transformer的并行化体现在哪个地方？Decoder端可以做并行化吗？

Decoder端可以做并行化吗？

简单描述一下wordpiece model 和 byte pair encoding，有实际应用过吗？

Transformer训练的时候学习率是如何设定的？Dropout是如何设定的，位置在哪里？Dropout 在测试的需要有什么需要注意的吗？

引申一个关于bert问题，bert的mask为何不学习transformer在attention处进行屏蔽score的技巧？

请简要介绍 Transformer 模型以及它与传统的神经网络架构（如 RNN 和 LSTM）之间的主要区别。

Transformer 模型中的自注意力机制是如何工作的？可以解释一下“缩放点积注意力”吗？

在 Transformer 中，位置编码的作用是什么？为什么要引入位置编码？

请解释 Transformer 的多头注意力是如何实现并同时处理不同的信息子空间的。

在 Transformer 模型中，层归一化（Layer Normalization）和残差连接（Residual Connection）的目的是什么？

叙述Transformer架构

Transformer的优缺点、评价

为什么 Transformer 模型在处理序列数据方面比传统的 RNN 和 LSTM模型有效率得多？

可以描述一下在 Transformer 模型训练中常用的优化方法和训练技巧吗？

可以给出一些基于 Transformer 模型改进和变体的例子，以及它们的优点是什么吗？（如：BERT、GPT系列）

请讨论一下 Transformer 模型用于序列生成任务（如机器翻译、文本生成）时的输出序列是如何生成的。

在深度学习中，过拟合是一个常见问题。在实际工作中，你是如何防止 Transformer 模型过拟合的？

使用 Transformer 模型时可能会遇到的内存和计算问题有哪些？又是如何解决这些问题的？

Transformer 模型对于输入序列的长度有什么限制？如果要处理很长的序列，你会采取什么措施？

在某些情况下，Transformer 模型可能并不是最佳选择。可以讨论一下在什么类型的任务或场景中可能会偏好其他模型而不是 Transformer 吗？

Transformer 模型的解释性通常被认为是一个挑战，你是否有什么见解或方法来理解模型的决策过程？

最后，设想你需要在不同的硬件配置上部署 Transformer 模型。你会考虑哪些因素以及可能采用的优化策略？

面试：为什么要用带掩码masked？

面试：Q、K、V到底是什么？目的是什么？

Transformer 自注意力计算中，为什么Q和K要使用不同的权重矩阵进行线性变换投影，为什么不使用同一个变换矩阵，或者不进行变换？

说明Transformer中的三种mask机制

Transformer&CNN 感受野

预热？预训练微调？后训练？代码 逐行

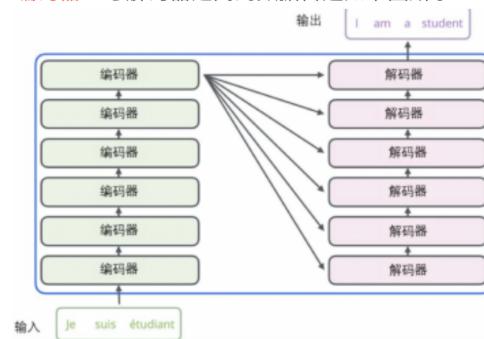
Transformer模型中，注意力计算后面使用了两个 FFN层，为什么第一个FFN层先把维提升，第二个FFN层再把维度降回原大小？

Transformer一个block 中最耗时的部分是哪个？

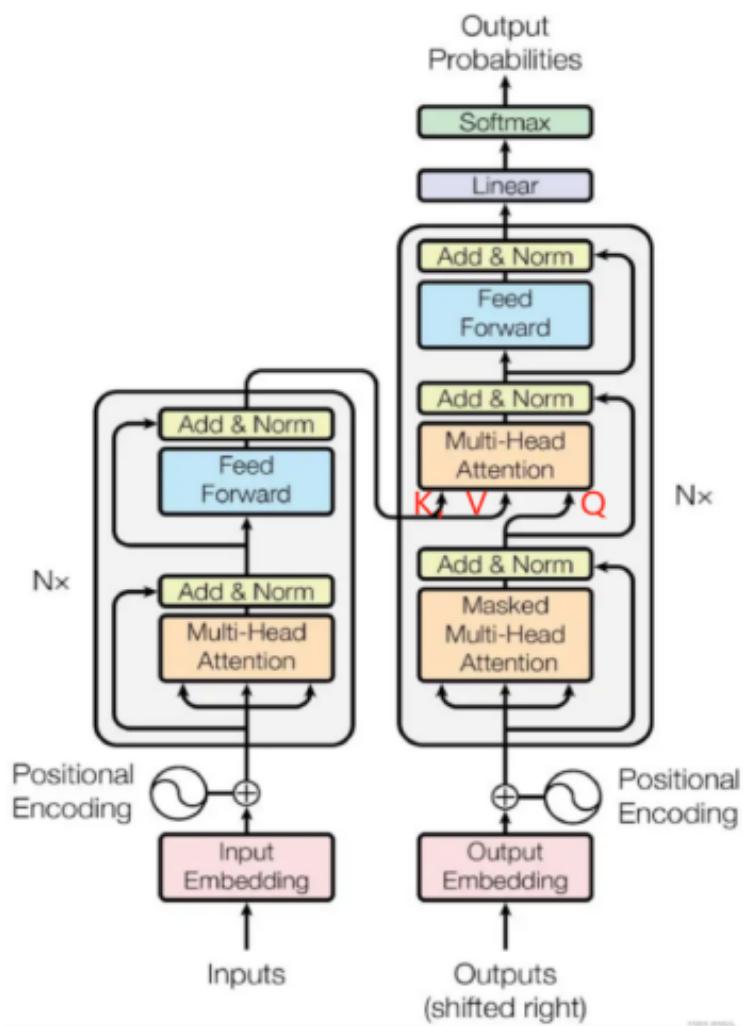
Transformer的并行化体现在哪个地方？Decoder端可以做并行化吗？

其基本架构如上图所示，图中的N在论文中等于6。

编码器与解码器之间的数据传递如下图所示：

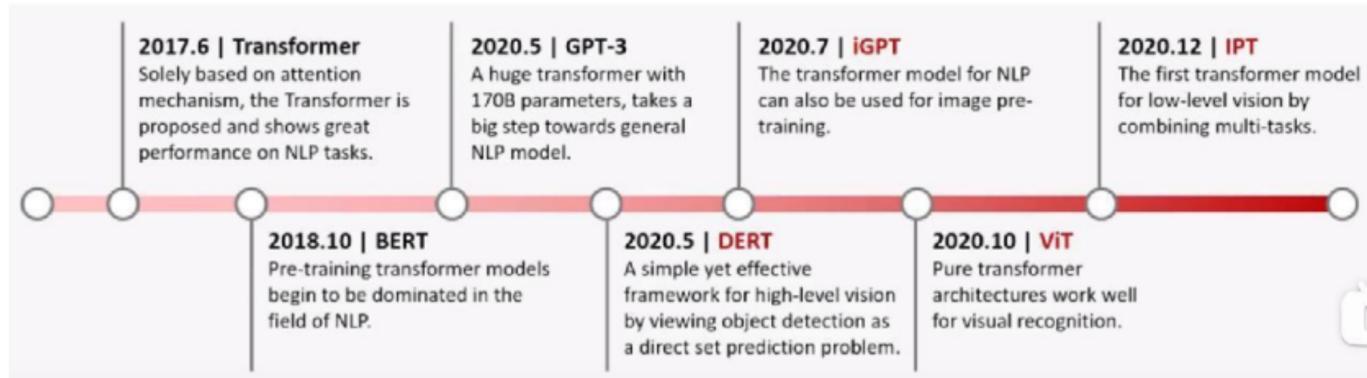


## 模型结构图：



# Transformer发家史

基于Transformer经典模型的发家史：



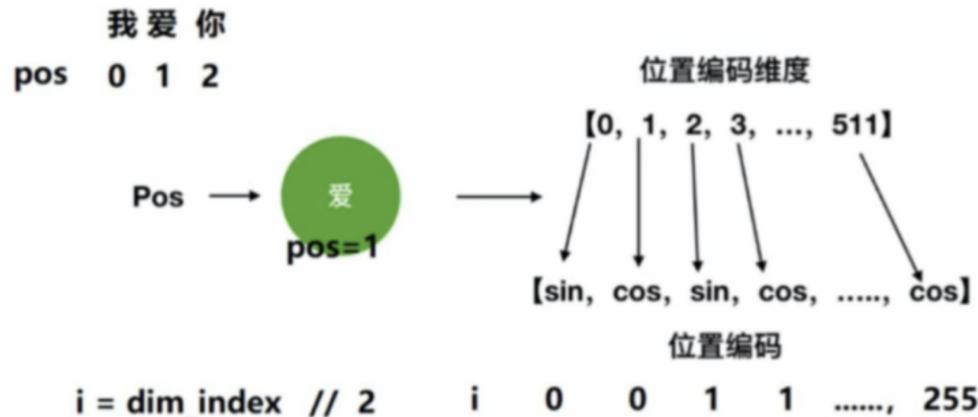
## 为什么用三角函数表示位置编码？

波长： $2\pi$ 到 $10000 * 2\pi$

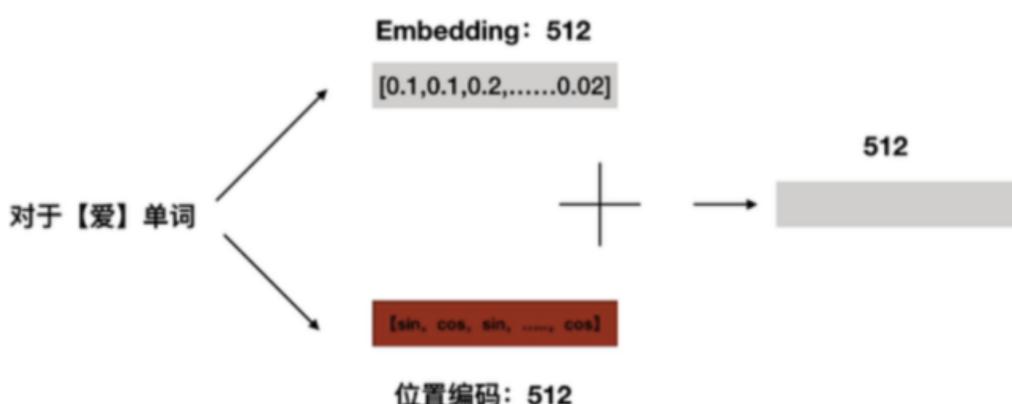
2) 编码过程

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

上式中，pos表示单词在句子中的位置， $2i$ 和 $2i+1$ 要整体看， $2i$ 表示向量的偶数位置， $2i+1$ 表示向量的奇数位置。如图所示：



这样，每个单词都能有一个唯一的512维位置编码，然后将其和Embedding相加：



注：单个的位置只能表示绝对位置信息！然后由于三角函数的特性，他们之间还可以表现出相对位置信息

### 3) 为什么位置嵌入是有用的?

因为借助三角函数的性质，可以表现出单词与单词之间的相对位置信息。

借助上述公式，我们可以得到一个特定位置的  $d_{model}$  维的位置向量，并且借助三角函数的性质

$$\begin{cases} \sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta \\ \cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta \end{cases} \quad (2)$$

我们可以得到：

$$\begin{cases} PE(pos + k, 2i) = PE(pos, 2i) \times PE(k, 2i + 1) + PE(pos, 2i + 1) \times PE(k, 2i) \\ PE(pos + k, 2i + 1) = PE(pos, 2i + 1) \times PE(k, 2i + 1) - PE(pos, 2i) \times PE(k, 2i) \end{cases} \quad (3)$$

可以看出，对于  $pos+k$  位置的位置向量某一维  $2i$  或  $2i+1$  而言，可以表示为， $pos$  位置与  $k$  位置的位置向量的  $2i$  与  $2i+1$  维的线性组合，这样的线性组合意味着位置向量中蕴含了相对位置信息。

如上图所示，位置为： $pos+k$  的单词可以分别用位置为  $pos$  和  $k$  的来计算出来，这样就表现出相对位置信息了。

## 为什么使用 $\sin$ 和 $\cos$ ?

1.  $\sin \cos$  函数是无限延长的，适合单词序列的长短变化

2.  $\sin \cos$  是具有周期性的， $pos + k$  的单词可以分别用位置为  $pos$  和  $k$  的来计算出来

## Transformer为什么使用多头注意力机制？为什么不使用一个头？

Transformer 使用多头注意力机制的主要原因

捕捉不同的特征：每个头可以学习和捕捉输入序列中的不同特征或模式。

多头注意力机制相当于 **多个不同的注意力的集成**。一个head相当于CNN中一个卷积核，多个head增强网络的容量和表达能力。提升了模型对不同位置的注意力。通过多头注意力机制，**不同的头关注的点是不同的**。比如有的头关注的是近距离的词，而有些头则是关注的更远的距离的词。给注意力层多个表示子空间。使用多头注意力机制我们就能得到多个查询/键/值的权重向量矩阵。Transformer默认使用8个注意力头。这样每个编码器就能得到8个矩阵。头数可调。因为每个初始的权重矩阵都是随机初始化的，所以经过模型训练之后将被投影到不同的表示子空间中。

增强模型的表达能力：多个头的并行计算可以丰富模型的表达能力，使其能够关注到输入的不同方面。

具体而言，多头注意力机制通过并行计算多个不同的注意力头，每个头有自己的一组权重矩阵，最后将这些头的输出拼接起来，再进行线性变换，从而综合各个头的信息。

3

(1) 多头保证了transformer可以注意到不同子空间的信息，捕捉到更加丰富的特征信息。

(2) 每个并行计算注意力权重，提高计算效率（参数总量和总计算量没有减少）

(3) 由于多头注意力机制能够从多个角度学习输入数据的特征，它有助于提高模型的泛化能力，使其能够更好地处理未见过的数据或任务。

(4) 在自然语言处理中，句子中的词语往往与距离较远的其他词语存在依赖关系。多头注意力机制允许模型在不同的头中关注序列的不同部分，从而更有效地捕获这些长距离依赖。

4

多头保证了transformer 可以注意到不同子空间的信息，捕捉到更加丰富的特征信息。其实本质上是论文原作者发现这样效果确实好。多头可以使参数矩阵形成多个子空间，矩阵整体size不变，只改变了每个head对应的维度大小，这样使矩阵对多方面信息进行学习，但计算量和单个head差不多。

5

多头可以使参数矩阵形成多个子空间，矩阵整体的size不变，只是改变了每个head对应的维度大小，这样做使矩阵对多方面信息进行学习，但是计算量和单个head差不多。

## Transformer为什么Q和K使用不同的权重矩阵生成，为什么不使用同一个值进行自身的点乘？注意和上一个问题 多头注意力机制 的区别

Q（查询）和K（键）使用不同的权重矩阵生成，是为了在计算注意力得分时能够捕捉到输入序列中不同的特征。如果使用同一个值进行自身的点乘，模型无法有效区分查询向量和键向量的不同特征，导致注意力机制失去灵活性和区分能力。因此，通过不同的权重矩阵生成Q和K，可以增强模型的表达能力，确保注意力机制能够更好地识别和利用输入序列中的信息。

## Transformer在计算attention的时候为什么使用点乘 而不是加法？两者计算复杂度和效果上有什么区别？

1. 捕捉相关性：点乘能够更好地捕捉查询（Q）和键（K）之间的相关性。点乘可以视为一种元素级别的加权求和，权重由Q和K的对应元素共同决定，这使得模型能够更精确地衡量它们之间的匹配程度。
2. 计算效率：虽然点乘和加法在单个元素操作的计算复杂度上相似，但在矩阵运算中，点乘可以利用现代硬件（如GPU）上的并行计算优势，实现高效的大规模运算。
3. 可扩展性：点乘天然支持扩展到多头注意力（Multi-Head Attention），这是Transformer架构中的一个重要特性。在多头注意力中，模型并行地执行多个点乘操作，然后将结果合并，以捕获不同子空间的信息。
4. 梯度传播：点乘在反向传播中具有更好的梯度传播特性。在深度学习中，梯度的传播对于模型的训练至关重要，点乘操作的梯度计算相对简单，有助于优化算法的稳定性和收敛速度。

5. 泛化能力：点乘作为一种通用的操作，可以更容易地泛化到不同的任务和模型架构中。加法虽然简单，但在捕捉复杂模式和关系方面可能不如点乘有效。

## 为什么在进行softmax之前需要对attention进行scaled（为什么除以dk的平方根），并使用公式

### 推导进行讲解

在Transformer模型中，自注意力（Self-Attention）机制的核心是计算一个查询（Query）与所有键（Key）的点积，然后通过softmax函数进行归一化，得到注意力分布。公式如下：

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

其中，V是值（Value）， $d_k$ 是键（Key）的维度。

为什么需要进行缩放（Scaling）？

1. 梯度消失/爆炸问题：在[深度学习](#)中，当模型很深时，梯度可能会随着层数的增加而变得非常小（梯度消失）或者非常大（梯度爆炸）。在自注意力中，如果不进行缩放，随着 $d_k$ 的增加， $QK^T$ 的结果可能会变得非常大，导致softmax函数的梯度变得非常小，进而导致梯度消失问题。
2. 稳定性：缩放操作提高了模型的稳定性。通过除以 $\sqrt{d_k}$ ，我们确保了即使在 $d_k$ 较大的情况下，softmax函数的输入也不会变得过大，从而使得梯度保持在一个合理的范围内。
3. 概率分布：softmax函数的输出是一个概率分布，其值的范围在0到1之间。如果不进行缩放，当 $d_k$ 较大时， $QK^T$ 的值可能会非常大，导致softmax函数的输出值远离0和1，这会使得模型难以学习到有效的注意力分布。

### 公式推导

假设我们有一个查询 $Q$ 和一个键 $K$ ，它们都是维度为 $d_k$ 的向量。我们计算它们的点积：

$$\text{score} = QK^T$$

在没有缩放的情况下，softmax函数的计算如下：

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

其梯度为：

$$\frac{\partial \text{Attention}}{\partial Q} = \text{softmax}(QK^T) \odot \left(V^T \frac{\partial \text{Loss}}{\partial V}\right)$$

其中， $\odot$ 表示Hadamard积（元素乘积）， $\frac{\partial \text{Loss}}{\partial V}$ 是损失函数对值  $V$  的梯度。

当  $d_k$  较大时， $QK^T$  的值可能会非常大，导致softmax函数的梯度非常小，因为softmax函数的梯度与输入值的差值成反比。因此，我们通过缩放来控制这个差值的大小：

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

这样，即使  $d_k$  较大， $\frac{QK^T}{\sqrt{d_k}}$  的值也不会过大，从而保持了梯度的稳定性。梯度变为：

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

通过这种方式，我们确保了即使在高维情况下，梯度也能保持在一个合理的范围内，从而有助于模型的训练和收敛。

## Attention 的计算公式中为什么要除以根号 dk?

当值变大的时候，softmax 函数会造成梯度消失问题，所以设置了一个softmax的temperature 来缓解这个问题。这里 temperature 被设置为了，也就是乘上。简单来说，就是需要压缩软 softmax 输入值，以免输入值过大，进入了softmax 的饱和区，导致梯度太小而难以训练。如果不进行 attention 值进行 scaling，也可以通过在参数初始化时将方差除以根号 d，同样可以起到预防 softmax 饱和的效果。

## 在计算attention score的时候如何对padding做mask操作？

对需要mask的位置设为负无穷，再对attention score进行相加

在计算注意力得分时，对padding进行mask操作目的是为了避免模型将注意力集中在填充位置上（因为这些位置不包含实际的有用信息）。具体做法是在计算注意力得分之前，对填充位置对应的得分加上一个非常大的负数（如负无穷），通过softmax后，这些位置的权重接近于零，从而不影响实际有效的序列位置。

## 注：什么是padding？

在处理自然语言时，输入的序列长度可能不同。为了让所有序列能够在一个批次中进行计算，我们会在较短的序列后面填充特殊的标记，通常是零（0）。这些填充标记就是padding。

### 注：为什么要对padding做mask操作？

如果不对padding做mask操作，模型可能会误把这些填充位置当作有效信息进行处理，从而影响注意力得分的计算，最终影响模型的性能。因此，需要在注意力计算时忽略这些填充位置。

## 为什么在进行多头注意力的时候需要对每个head进行降维？（可以参考上面一个问题）

在进行多头注意力时，需要对每个头进行降维，以保证每个头的计算复杂度不会过高，同时能够并行计算。将输入的维度分成多个头，可以让每个头处理更小维度的数据，从而降低单头的计算复杂度，减小参数量，提高计算效率。并且通过多个头的组合，增强模型的表达能力和鲁棒性。

- 降低计算复杂度

假设输入的维度是 $d$ ，每个头的输出维度也是 $d$ 。如果不进行降维，每个头的输出维度仍然是 $d$ ，那么在拼接多个头的输出时，最终的维度将是 $d * \text{num\_heads}$  ( $\text{num\_heads}$ 表示头的数量)。这样维度会变得非常大，计算复杂度和内存需求都大大增加。

通过对每个头进行降维，每个头的输出维度变为 $d / \text{num\_heads}$ 。这样，即使拼接多个头的输出，最终的维度也仍然是 $d$ ，保持了与输入相同的维度，避免了计算复杂度和内存需求的急剧增长。

- 保持模型的参数数量可控

模型的参数数量直接影响训练的难度和时间。如果每个头都不进行降维，那么模型的参数数量会大大增加，训练起来会非常困难。而对每个头进行降维，可以控制每个头的参数数量，从而使得整个模

型的参数数量保持在一个可控范围内。

- 维持信息的多样性

通过对每个头进行降维，可以确保每个头在一个更小的子空间中进行注意力计算。这意味着每个头可以在不同的子空间中学习到不同的特征，增加了模型的多样性和鲁棒性。最终的拼接结果融合了不同子空间的信息，使得模型能够更全面地理解输入数据。

## 大概讲一下Transformer的Encoder模块？

Transformer的Encoder模块由N层堆叠组成，每层包括两个子层：多头自注意力机制（Multi-Head Self-Attention）和前馈神经网络（Feed-Forward Neural Network）

每个子层后都接一个残差连接（Residual Connection）和层归一化（Layer Normalization）。输入首先通过嵌入层（Embedding），然后通过位置编码（Positional Encoding）加上位置信息，再依次经过各层编码器，最终输出编码后的序列表示。

## Transformer中decoder的结构

decoder也是由6个相同的层堆叠构成，即上图中 $N=6$ 。

每一层都有三个子层：

- 1、第一层是带掩码的多头自注意力机制（**Masked** multi-head self-attention mechanism）
- 2、第二层是全连接的前馈网络（feed forward network）
- 3、第三层是多头注意力机制，类似于在**encoder**的第一子层中实现的机制。在decoder，这种多头注意力机制接收来自前一个decoder层的查询，以及来自encoder输出的键和值。这允许decoder处理输入序列中的所有单词

# 为何在获取输入词向量之后需要对矩阵乘以embedding size的开方？意义是什么？

在获取输入词向量之后，需要对矩阵乘以embedding size的平方根，是为了保持向量的尺度稳定。Embedding的值通常是随机初始化的，乘以开方后的结果能保证在后续的点乘计算中，值的尺度不会过大或过小，从而有利于模型的训练稳定性。【不明白

2

embedding matrix的初始化方式是xavier init，这种方式的方差是 $1/\text{embedding size}$ ，因此乘以embedding size的开方使得embedding matrix的方差是1，在这个scale下可能更有利于embedding matrix的收敛。

# 简单介绍一下Transformer的位置编码？有什么意义和优缺点？

1

位置编码向量与输入embedding具有相同的维度（因此可以相加），并且使用正弦和余弦函数用以下的公式表示：

$$PE(pos, 2i) = \sin\left(\frac{pos}{2i}\right),$$
$$PE(pos, 2i) = \cos\left(\frac{pos}{2i}\right)$$

上式中，pos表示单词的位置，i表示单词的维度

即位置编码的每个维度对应于正弦曲线。波长上形成从 $2\pi$ 到 $10000 \cdot 2\pi$ 的几何级数【波长=周期；】

选择这个函数是因为我们假设它可以让模型很容易通过相对位置来学习，因为对于任何固定的偏移量k，

$PE_{pos+k}$  可以通过  $PE_{pos}$  线性表示

2

Transformer的位置编码（Positional Encoding）是为了给模型提供序列中各个位置的信息，因为Transformer本身不具备顺序信息。位置编码通过正弦和余弦函数生成，对不同位置生成不同的编码。

优点是能够显式地提供位置信息，易于计算，缺点是位置编码固定，不能根据上下文动态调整。

3

因为self-attention是位置无关的，无论句子的顺序是什么样的，通过self-attention计算的token的hidden embedding都是一样的，这显然不符合人类的思维。因此要有一个办法能够在模型中表达出一个token的位置信息，transformer使用了固定的positional encoding来表示token在句子中的绝对位置信息。

## 你还了解哪些关于位置编码的技术，各自的优缺点是什么？

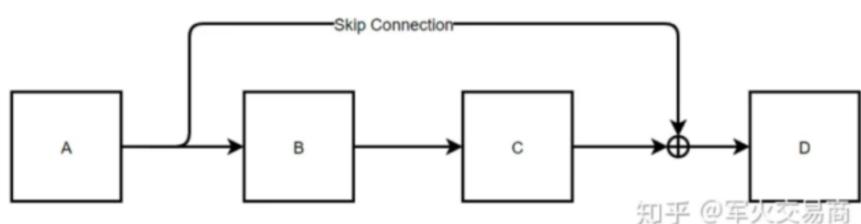
除了位置编码，其他位置表示技术还有：

- 可学习的位置编码（Learnable Positional Encoding）：位置编码作为可学习的参数，优点是灵活，能够根据数据调整，缺点是可能需要更多的训练数据。
- 相对位置编码（Relative Positional Encoding）：考虑到相对位置关系，优点是能够捕捉相对位置信息，适用于长序列，缺点是实现复杂度高。
- 混合位置编码（Hybrid Positional Encoding）：结合绝对和相对位置编码，优点是综合两者优点，缺点是实现复杂度增加。

相对位置编码（RPE）

1. 在计算attention score和weighted value时各加入一个可训练的表示相对位置的参数。
2. 在生成多头注意力时，把对key来说将绝对位置转换为相对query的位置
3. 复数域函数，已知一个词在某个位置的词向量表示，可以计算出它在任何位置的词向量表示。前两个方法是词向量+位置编码，属于亡羊补牢，复数域是生成词向量的时候即生成对应的位置信息。

## 简单讲一下Transformer中的残差结构以及意义。



如上图，表示一个残差网络结构图，在进行反向传播时，有如下推导：

$$\frac{\partial L}{\partial X_{A_{out}}} = \frac{\partial L}{\partial X_{D_{in}}} \frac{\partial X_{D_{in}}}{\partial X_{A_{out}}}$$

$X_{A_{out}}$   $X$ 是输入的数据  $A$ 是模块的处理

而有： $X_{D_{in}} = X_{A_{out}} + C(B(X_{A_{out}}))$

所以，

$$\frac{\partial L}{\partial X_{A_{out}}} = \frac{\partial L}{\partial X_{D_{in}}} \left[ 1 + \frac{\partial D_{in}}{\partial C} \frac{\partial X_C}{\partial X_B} \frac{\partial X_B}{\partial X_{A_{out}}} \right]$$

由式子可知，即使后面的连乘接近0，由于前面有个1，所以导数不会为0，这就是残差网络能很深的原因，因为残差网络缓解了梯度消失。

Transformer中的残差结构（Residual Connection）是在每个子层输出后，加入输入的原始信息，通过直接相加实现。这有助于缓解深层网络中的梯度消失问题，保证信息流的顺畅，促进训练过程的稳定和快速收敛。

3

encoder和decoder的self-attention层和ffn层都有残差连接。反向传播的时候不会造成梯度消失

## 为什么transformer块使用LayerNorm而不是BatchNorm？ LayerNorm 在Transformer的位置是哪里？

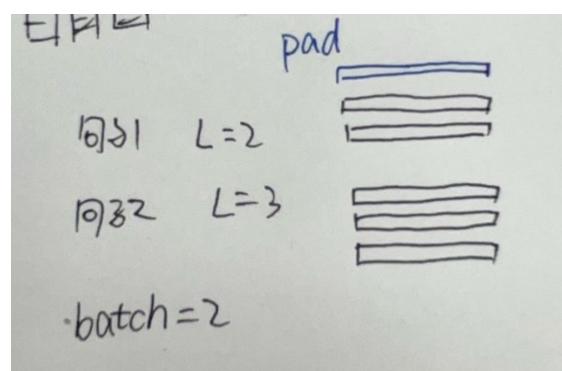
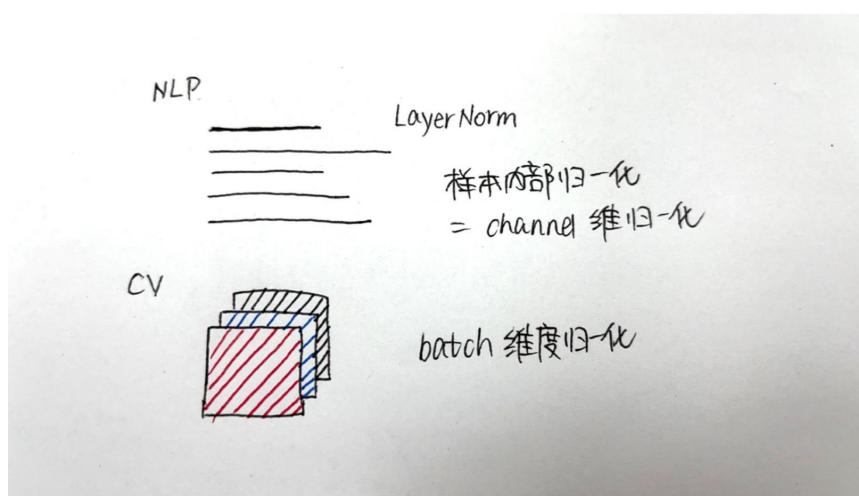
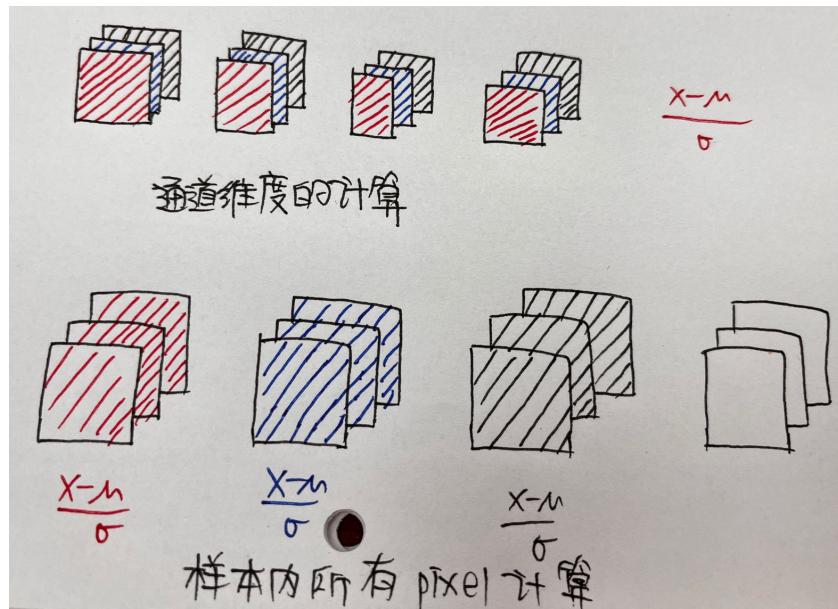
1

Transformer块使用LayerNorm而不是BatchNorm，因为LayerNorm在序列模型中表现更好。BatchNorm在处理变长序列和小批量数据时不稳定，而LayerNorm对每个样本独立进行归一化，更适合变长序列数据。LayerNorm通常位于每个子层的残差连接之后。

2

CV使用 BN 是认为 channel 维度的信息对 cv 方面有重要意义，如果对 channel 维度也归一化会造成不同通道信息一定的损失。而同理 nlp 领域认为句子长度不一致，并且各个 batch 的信息没什么关系，因此只考虑句子内信息的归一化，也就是LN。

LN是为了解决梯度消失的问题。



3

## 为什么 Transformer 用 layernorm 而不 batchnorm?

1. NLP数据中由于每条样本可能不一样长，会使用padding，如果对padding部分进行normalization, 对效果有负面影响
2. 直观来说，batchnorm会对同一个特征以batch为组进行归一化，而对于文本数据，同一个位置的token很可能是没有关联的两个token, 对这样一组数据进行归一化没有什么实际意义
3. BN会抹去同一样本所有位置特征的原有大小关系，而LN能保留这种同一样本内部特征的大小关系；《PowerNorm:Rethinking Batch Normalization in Transformersx》论文的实验也表明，在NLP数据使用 batchnorm，均值和方差相对 layernorm 会更加震荡，因此效果欠佳。

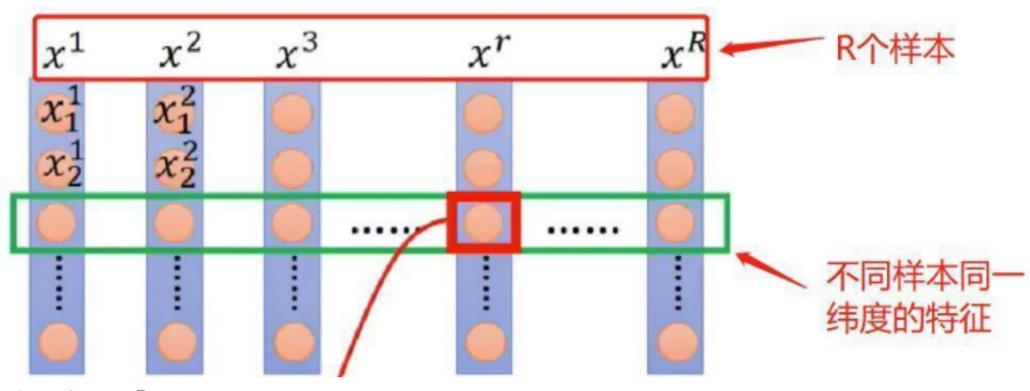
## 简答讲一下BatchNorm技术，以及它的优缺点。

BatchNorm（批量归一化）是对每个小批量数据进行归一化，减去均值除以标准差，再引入可学习的缩放和平移参数。

优点是加快训练速度，缓解梯度消失和爆炸问题。

缺点是在小批量或变长序列中效果不稳定，不适合序列模型。

BN的理解重点在于它是针对整个Batch中的样本在同一维度特征做处理！！！如下图所示：



BN的优点：

## ①解决内部协变量偏移

简单来说，在训练过程中，各样本之间可能分布不同，增大了学习难度，BN通过对各样本同一维度进行归一化缓解了这个问题。也有说，BN使损失平面更加光滑，从而加快收敛。

## ②缓解了梯度饱和问题，加快收敛。

### BN的缺点：

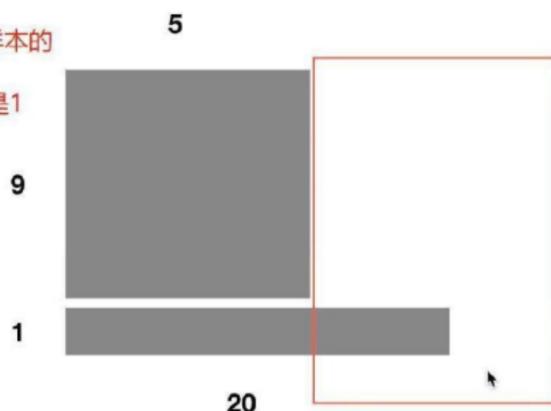
#### ①batch\_size较小时，效果较差

BN的过程，使用整个batch中样本的均值和方差来模拟全部数据的均值和方差，在batch\_size较小的时候，效果肯定不好。

#### ②对文本这种输入维度不一致的不友好

如图所示：

10个样本，其中9个样本的  
特征维度是5，  
一个样本的特征维度是1



举个最简单的例子，比如 batch\_size 为 10，也就是我有 10 个样本，其中 9 个样本长度为 5，第 10 个样本长度为 20。

那么问题来了，前五个单词的均值和方差都可以在这个batch中求出来从而模型真实均值和方差。但是第6个单词到底20个单词怎么办？

只用这一个样本进行模型的话，不就是回到了第一点，batch太小，导致效果很差。3) BN的使用场景

BN在MLP和CNN上使用的效果较好，在RNN这种动态文本模型上使用的比较差。

为啥BN在NLP中效果差：

### BN的使用场景

不适合RNN这种动态文本模型，有一个原因是因为batch中的长度不一致，导致有的靠后面的特征的均值和方差不能估算。

这个问题其实不是个大问题，可以缓解。我们可以在数据处理的时候，使句子长度相近的在一个batch，就可以了。所以这不是为啥NLP不用BN的核心原因。

BN在NLP中的应用，BN是对每个特征在batch\_size上求的均值和方差。记住，是每个特征。比如说身高，比如说体重等等。这些特征都有明确的含义。

但是我们想象一下，如果BN应用到NLP任务中，对应的是对什么做处理？是对每一个单词！也就是说，我现在的每一个单词是对应到了MLP中的每一个特征。也就是默认了在同一个位置的单词对应的是同一种特征，比如：“我/爱/中国/共产党”和“今天/天气/真/不错”

如何使用BN，代表着认为“我”和“今天”是对应的同一个维度特征，这样才可以去做BN。

大家想一下，这样做BN，会有效果吗？

不会有效果的，每个单词表达的特征是不一样的，所以按照位置对单词特征进行缩放，是违背直觉的。

## LayerNorm在Transformer中的应用

layner-norm 的特点是什么？layner-norm 做的是针对每一个样本，做特征的缩放。换句话讲，保留了N维度，在C/H/W维度上做缩放。

也就是，它认为“我/爱/中国/共产党”这四个词在同一个特征之下，所以基于此而做归一化。

这样做，和BN的区别在于，一句话中的每个单词都可以归到一个名字叫做“语义信息”的一个特征中（我自己瞎起的名字，大家懂就好），也就是说，layner-norm也是在对同一个特征下的元素做归一化，只不过这里不再是对应N（或者说batch size），而是对应的文本长度。

上面这个解释，有一个细节点，就是，为什么每个单词都可以归到“语义信息”这个特征中。大家这么想，如果让你表达一个句子的语义信息，你怎么做？

最简单的方法就是词语向量的加权求和来表示句子向量，这一点没问题吧。（当然你也可以自己基于自己的任务去训练语义向量，这里只是说最直觉的办法）。

sentence1	512维						
sentence2	512维	512维	512维				
sentence3	512维	512维	512维	512维			

简单描述一下Transformer中的前馈神经网络？使用了什么激活函数？相关优缺点？

Transformer中的前馈神经网络FeedForward由两个线性变换和一个激活函数组成，激活函数通常是ReLU。如下公式：max相当于Relu

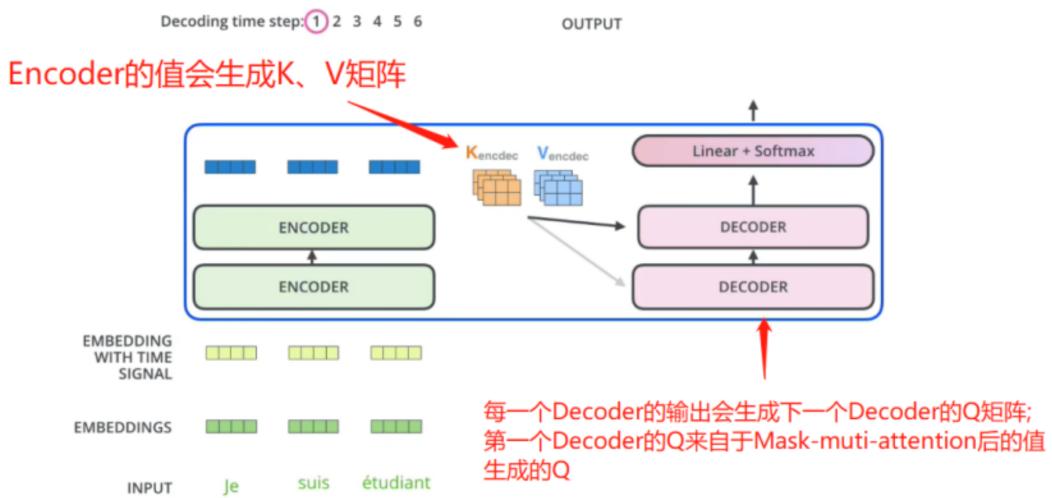
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

优点是增加模型的非线性表达能力，结构简单高效。缺点是ReLU可能导致部分神经元输出恒为零（死神经元），需要慎重选择超参数。

2

输入嵌入-加上位置编码-多个编码器层（每个编码器层包含全连接层，多头注意力层和点式前馈网络层（包含激活函数层））-多个解码器层（每个编码器层包含全连接层，多头注意力层和点式前馈网络层）-全连接层，使用了relu激活函数

# Encoder端和Decoder端是如何进行交互的？（在这里可以问一下关于seq2seq的attention知识）



如上图所示，每个Encoder会接受Q,K,V矩阵，其中，K，V来自与Encoder,Q来自与上一个Decoder.

之后就没什么好讲了。但需要明白的是：

在训练过程，Output(Shifted right) 表示的是GroundTruth；

在test过程，Output(Shifted right) 表示之前自己预测的序列。

Encoder端和Decoder端通过注意力机制进行交互。Encoder将输入序列编码成隐藏表示，Decoder通过多头注意力机制，将编码器的输出作为键和值，解码器的输出作为查询，计算注意力得分，从编码器的输出中提取相关信息，生成新的输出序列。

下面用一个更通俗的类比来解释Transformer中编码器（Encoder）和解码器（Decoder）之间的交互。想象一下，编码器和解码器是两个团队，它们要共同完成一个任务：把一种语言翻译成另一种语言。

## 1. 编码器团队（Encoder）：

- 编码器团队的任务是仔细阅读原始语言（比如英语）的句子，并理解它的意思。
- 每个团队成员（编码器层）都会贡献自己对句子的理解，最终形成一个整体的理解（隐藏状态）。

## 2. 解码器团队（Decoder）：

- 解码器团队的任务是根据编码器团队的理解，逐字逐句地把句子翻译成目标语言（比如法语）。

## 3. 交互的桥梁：注意力机制：

- 当解码器团队开始工作时，他们需要不断地与编码器团队沟通，以确保翻译的准确性。
- 他们通过一个特殊的“对讲机”（注意力机制）来沟通。解码器团队的每个成员（解码器层）都会问编码器团队：“在这个翻译步骤中，原文中的哪个部分最重要？”

## 4. 编码器团队的回答：

- 编码器团队会根据解码器团队的问题，给出一个“重要性评分”（注意力权重），告诉解码器团队在当前翻译步骤中，原文的哪些部分是重要的。

5. 解码器团队的翻译：

- 根据编码器团队给出的重要性评分，解码器团队会综合考虑这些信息，并决定下一个翻译出的词是什么。
- 这个过程会一直重复，直到整个句子被翻译完成。

6. 防止作弊的规则（掩码）：

- 在翻译过程中，有一个规则：解码器团队不能提前看到未来的词（不能作弊）。所以他们会用一个“遮盖布”（掩码）来确保在翻译当前词时，只能看到已经翻译出来的部分。

通过这种方式，Transformer模型中的编码器和解码器可以协同工作，完成复杂的任务，比如语言翻译、文本摘要等。编码器团队深入理解输入信息，而解码器团队则利用这些理解，一步步构建出高质量的输出。

（Seq2seq（序列到序列）模型中，注意力机制用来解决长序列依赖问题。传统的seq2seq模型在解码时只能使用Encoder的最后一个隐状态，这对于长序列可能效果不好。注意力机制通过计算Decoder的每个时间步与Encoder输出的所有时间步之间的相关性，动态地选择信息，提升了翻译效果。）

2

Transformer 中，encoder和decoder 是怎么进行交互的？

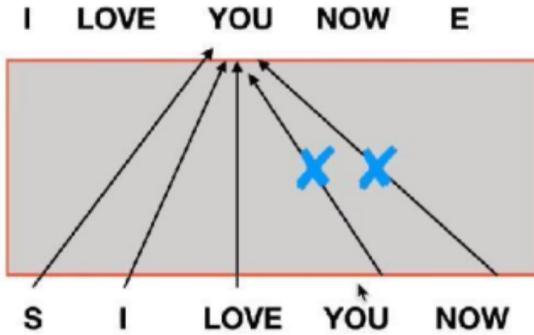
decoder 部分的输入，在每层中，先进行一次 self-attention;之后用 encoder 的输出作为 attention 计算中的K、V，decoder 的输入作为 Q，进行 Cross-attentionon

## Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？

Decoder阶段的多头自注意力需要进行sequence mask，以防止模型在训练时看到未来的单词。Encoder的多头自注意力没有这种限制。Sequence mask确保模型只关注已生成的部分，避免信息泄露，提高训练的效果。

# 为什么需要decoder自注意力需要进行 sequence mask

为什么需要mask



我的理解是，当我们的需要翻译的句子是：

“我现在爱你”，其Label为：“I LOVE YOU NOW”。

当前时刻是已经预测了“I LOVE”，正在预测“You”。此时我们就需要把Label中的“You NOW”给mask掉。

其原因就是在推理时，是没有Label的，所以我们要训练和推理过程之间没有gap，才能得到较好的结果。

而mask就是将从代码角度讲，就是把当前时刻之后所有单词mask掉就好了。

★ 需要提前知道的是：

在训练过程，Output(Shifted right) 表示的是GroundTruth；

在test过程，Output(Shifted right) 表示之前自己预测的序列。

这里探讨的是训练过程为什么需要Mask？

## Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？

Decoder有两层mha，encoder有一层mha，Decoder的第二层mha是为了转化输入与输出句长，Decoder的请求q与键k和数值v的倒数第二个维度可以不一样，但是encoder的qkv维度一样。

# Transformer的并行化体现在哪个地方？ Decoder端可以做并行化吗？

Transformer的并行化体现在注意力机制和前馈神经网络上，因为每个时间步的计算彼此独立。

Decoder端不能完全并行化，因为当前步的输出依赖于前一步的结果，但自注意力机制部分可以并行化。

2

Transformer的并行化主要体现在self-attention模块，在Encoder端Transformer可以并行处理整个序列，并得到整个输入序列经过Encoder端的输出，但是rnn只能从前到后的执行

## Decoder端可以做并行化吗？

训练的时候可以，但是交互的时候不可以

## 简单描述一下wordpiece model 和 byte pair encoding，有实际应用过吗？

没有

WordPiece Model和Byte Pair Encoding（BPE）都是子词分割技术。WordPiece将词分割成子词，提高模型的词汇覆盖率。BPE是合并最频繁的字对。

### WordPiece Model:

想象一下，你有一个工具箱，里面有各种各样的拼图碎片，每个碎片代表一个语素或词的一部分。WordPiece模型就像这个工具箱，它把单词分解成更小的、有意义的片段。

这样做好处是，即使工具箱里没有某个完整的单词，你也可以通过

拼凑这些小片段来表达这个单词的意思。在机器学习中，这可以帮助模型理解和生成新的、未见过的词汇。

### Byte Pair Encoding (BPE) :

BPE更像是一种编码技巧，它观察文本数据，找出最常见的字节对，然后把这些字节对合并成一个单一的单元。

比如，“power”和“ful”这样的词，如果它们经常一起出现，BPE就会把它们看作一个单元。这样做可以减少词汇表的大小，同时保持词汇的多样性。

实际应用：

这两种技术在自然语言处理（NLP）中非常实用，特别是在机器翻译、文本生成等任务中。它们帮助模型处理那些在训练数据中很少见或完全没见过的词汇。

## Transformer训练的时候学习率是如何设定的？Dropout是如何设定的，位置在哪里？Dropout在测试的需要注意什么需要注意的吗？

(1) 预热策略：Transformer通常使用预热（warm-up）策略和学习率衰减相结合的方法。在训练的第一部分迭代中，学习率逐渐增加，然后按照预定的方式逐渐减少。

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

其中， $d_{\text{model}}$  是模型的维度， $step\_num$  是当前训练步数， $warmup\_steps$  是预热步数。

(2) Transformer中使用Dropout层来防止过拟合，具体位置包括：

- 自注意力机制中的注意力权重计算后。
- 前馈神经网络的输出。
- 残差连接后的输出。

设定：通常Dropout概率设定为0.1，但可以根据具体任务和数据进行调整。

(3) 测试时：**不使用Dropout**：在测试或推理阶段，Dropout不再使用，即不会随机丢弃节点，而是使用所有节点参与计算。

LN是为了解决梯度消失的问题，dropout是为了解决过拟合的问题。在embedding后面加LN有利于embedding matrix的收敛。

# 引申一个关于bert问题， bert的mask为何不学习transformer在attention处进行屏蔽score的技巧？

## bert逐行 逐句

BERT的掩码设计目的是为了在预训练过程中让模型学习丰富的上下文表示，而不是为了防止信息泄漏，这与Transformer中attention mask的用途不同。

BERT中的mask：

预训练任务：BERT使用掩码语言模型（Masked Language Model, MLM）进行预训练，即在输入序列中随机选择一些单词进行掩码，然后让模型预测这些掩码位置的单词。

原因：独立于位置的预测：BERT的掩码操作是对输入的特定位置进行掩码，目的是让模型能够学习到每个单词的上下文表示，而不需要关注具体位置。

不同任务：BERT的设计目标是让模型学习到每个单词的上下文表示，而Transformer的attention掩码（mask）主要用于在序列生成中防止信息泄漏，如自回归模型中防止预测未来的单词。

2

BERT和transformer的目标不一致，bert是语言的预训练模型，需要充分考虑上下文的关系，而transformer主要考虑句子中第i个元素与前i-1个元素的关系。

请简要介绍 Transformer 模型以及它与传统的神经网络架构（如 RNN 和 LSTM）之间的主要区别。

Transformer 是一种基于自注意力机制的深度学习架构，它主要由编码器和解码器组成。每个编码器和解码器都由若干个相同的层构成，每层中都包含多头注意力和全连接的前馈网络。Transformer 的关键优势在于自注意力机制，它允许模型直接计算序列中任何两个位置之间的依赖关系，无需通过递归。与 RNN/LSTM 相比，Transformer 可以并行处理序列中的所有元素，这使得训练速度快得多。此外，Transformer 能够更好地处理长距离依赖问题，因为它不像 RNN 或 LSTM 那样受到梯度消失的限制。（Transformer 为什么 更好地处理长距离依赖问题）

## 2

论文提出了一种新的神经网络架构，称为Transformer，它完全依赖于注意力机制而不是循环或卷积神经网络进行序列转换。与传统的循环和卷积模型相比，这种新的架构具有更高的并行性，而不会牺牲质量，因为它能够在机器翻译中达到最先进的性能，并减少训练时间和计算成本。论文在两个机器翻译任务上进行了实验，英德翻译任务的BLEU得分为28.4，在英法翻译任务上实现了单模型的BLEU得分41.8，超过了现有文献中的最佳结果。研究人员还表明，Transformer在其他任务上也具有良好的泛化能力，比如英语短语分析。这种模型架构由一系列相同的层组成，每个层都具有多头自注意力和逐点全连接前馈网络，通过层归一化和残余连接实现了高效训练。注意力机制基于缩放的点积方法，模型还包括嵌入层、位置编码和softmax函数。总体而言，Transformer代表了神经机器翻译领域中一个令人期待的全新方向。与循环和卷积模型相比，Transformer架构提高了计算效率的主要原因如下：

1. 并行化能力更强：传统的循环模型在处理长序列时存在无法并行化的问题，而Transformer通过完全依赖注意力机制来计算输入和输出的表示，可以实现更高程度的并行计算。

2. 常数操作次数：传统的卷积模型和循环模型在处理远距离依赖关系时需要进行不同数量的操作，这会导致学习远距离依赖变得更加困难。而Transformer通过注意力机制可以将操作次数减少为常数次，从而更好地学习远距离的依赖关系。
3. 减少了内存限制：传统的循环模型由于其顺序计算的特性，在处理长序列时受到内存限制而难以进行批量化处理。而Transformer的并行计算和注意力机制可以在长序列上更好地进行批量处理，从而减少了内存限制。

## 原文

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states  $h_t$ , as a function of the previous hidden state  $h_{t-1}$  and the input for position  $t$ . This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU[16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or

output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

### 3

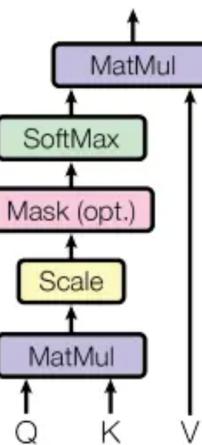
Transformer 是一种基于自注意力机制的深度学习架构，它主要由编码器和解码器组成。每个编码器和解码器都由若干个相同的层构成，每层中都包含多头注意力和全连接的前馈网络。Transformer 的关键优势在于自注意力机制，它允许模型直接计算序列中任何两个位置之间的依赖关系，无需通过递归。与RNN/LSTM 相比，Transformer 可以并行处理序列中的所有元素，这使得训练速度快得多。此外 Transformer 能够更好地处理长距离依赖问题，因为它不像 RNN 或 LSTM 那样受到梯度消失的限制

## Transformer 模型中的自注意力机制是如何工作的？可以解释一下“缩放点积注意力”吗？

自注意力机制允许模型在处理一个序列的每个元素时，考虑到序列中的所有位置，它通过计算每个元素对序列中其他所有元素的注意力分数，并基于这些分数对输入元素进行加权求和，从而捕获全局的上下文信息。具体来说，这是通过计算序列中每个元素的点积 (query 和 key 的点积) 来实现的再对点积进行缩放。然后应用 softmax 函数，将这些数值转换成概率分布形式 (attention weights)。这些概率用来加权相应的值 (value) 向量，最终的输出是这些加权的总和。通过这种机制，每个序列元素都能够参考到整个序列的信息。

Scaled Dot-Product Attention 是一种用于注意力机制的函数，可以将查询(query)和一组键值对(key-value pairs)映射到一个输出。它通过计算查询与所有键的点积，并将结果除以一个标量因子来获得对值的权重，然后通过 softmax 函数将这些权重归一化。在Transformer模型中，这个函数被用于多头注意力机制(Multi-Head Attention)中的一个子模块，用来计算注意力权重并获得最终的输出值。Scaled Dot-Product Attention是基于点积的注意力机制的一种改进，通过引入缩放因子 $1/\sqrt{dk}$ ，解决了点积注意力函数在高维空间中可能出现的梯度问题。

Scaled Dot-Product Attention



**在 Transformer 中，位置编码的作用是什么？为什么要引入位置编码？**

transformer 其他结构没有考虑到单词之间的顺序信息，而单词的顺序信息对于语义是非常重要的

由于 Transformer 中缺少任何对输入序列元素顺序的隐式建模(如递归网络中的时间步长)，因此需要通过位置编码来给模型提供关于元素位置的信息。通常这种编码是通过将每个位置的正弦和余弦函数的值加到对应的嵌入向量中，这些正弦和余弦函数的频率遵循几何级数，

这允许模型即使对于长序列也能区分不同位置。

## 请解释 Transformer 的多头注意力是如何实现并同时处理不同的信息子空间的。

在Transformer 的上下文中，多头注意力机制指的是并行计算多组自注意力（即头）。每个头学习序列的不同方面，这样一来，模型可以同时在不同的子空间学习信息。实际上，这是通过为每个头使用不同的、可学习的线性变换(权矩阵) 来实现的，用以产生相应 query、key 和 value。每个头的输出然后被拼接并再次线性变换形成最终的输出。

## 在 Transformer 模型中，层归一化（Layer Normalization）和残差连接（Residual Connection）的目的是什么？

- 残差连接：解决神经网络的“退化现象”
- 层归一化：使得训练的模型更稳定，并且起到加快收敛速度的作用

每个子层输出的是

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

其中  $\text{Sublayer}(x)$  就是子层计算的结果

为了保证残差连接，模型所有的子层（sub-layers）、embedding 层输出的都是 512 维，即  $d_{model}=512$

层归一化是指对每一个子层输出的激活值沿特征维度进行归一化。这有助于稳定训练过程并加快收敛速度。残差连接，又称作残差跳过(Skip connections)通过将输入直接加到子层的输出上(在层归一前)，这使得即使是更深的网络，每层的梯度也能够更加稳定地传播并且减轻梯消失的问题(防止梯度消失帮助深层网络训练)。

## 叙述Transformer架构

Transformer 是一种神经网络架构，最早由谷歌在 2017 年的论文《Attention is All You Need》中提出它引入了自注意力机制，并被广泛应用于序列到序列seq2seq 任务。

相比于传统的循环神经网络RNN，Transformer摒弃了顺序处理的方式，充分利用了并行计算，从而减少了训练时间。

### ★ 整体结构：

Transformer由 Encoder和 Decoder两个部分组成，每个部分包含6个 block

Encoder负责将输入句子编码为表示向量，而Decoder 则负责生成目标语言的翻译

### ★ 输入表示：

Transformer 中单词的输入表示由单词Embedding 和位置 Embedding 相加得到。单词Embedding 可以通过 word2Vec、Glove 等方法训练得到，也可以在 Transformer中训练得到。位置 Embedding 用于表示单词在句子中的位置，因为 Transformer 不采用RNN 的结构，需要保存单词的顺序信息.

### ★ 自注意力机制：

transformer 的关键部分是 Self-Attention，它在Encoder 和 Decoder 中都有应用。

Self-Attention 结构包括查询 (Q) 、键(K)和值(V)的计算。通过线性变

换， Self-Attention将输入(单词表示向量矩阵 X) 转换为Q、K、V

### ★工作流程：

获取输入句子的每个单词的表示向量X， 将x传入Encoder， 经过6个Encoderblock得到句子的编码信息矩阵C将C传递到Decoder， 逐步预测翻译后的单词。

## Transformer的优缺点、评价

Transformer不适用哪些场景和任务？

- 对于需要捕捉局部特征或具有明显时空结构的任务， 传统的CNN可能更合适。
- 对于实时或流式处理任务， 需要逐步处理序列数据的 RNN或LSTM可能更优， 因为Transformer的全局自注意力机制需要整个序列数据一次性提供。
- 此外， Transformer 参数量大， 对于数据量较小的任务可能容易过拟合。

优点

(1) 虽然Transformer最终也没有逃脱传统学习的套路， Transformer只是一个全连接（或者是一维卷积）加Attention的结合体。但是其设计已经足够有创新， 因为其抛弃了在NLP中最根本的RNN或者CNN并且取得了非常不错的效果， 算法的设计非常精彩， 值得每个深度学习的相关人员仔细研究和品味。

(2) Transformer的设计最大的带来性能提升的关键是将任意两个单词的距离是1， 这对解决NLP中棘手的长期依赖问题是非常有效的。

(3) Transformer不仅仅可以应用在NLP的机器翻译领域， 甚至可以不局限于NLP领域， 是非常有科研潜力的一个方向。

(4) 算法的并行性非常好， 符合目前的硬件（主要指GPU）环境。

缺点：

- (1) 粗暴的抛弃RNN和CNN虽然非常炫技，但是它也使模型丧失了捕捉局部特征的能力，RNN + CNN + Transformer的结合可能会带来更好的效果。
- (2) Transformer失去的位置信息其实在NLP中非常重要，而论文中在特征向量中加入Position Embedding只是一个权宜之计，并没有改变Transformer结构上的固有缺陷。
- (3) 有些rnn轻易可以解决的问题transformer没做到，比如复制string，或者推理时碰到的sequence长度比训练时更长（因为碰到了没见过的position embedding）

## 为什么 Transformer 模型在处理序列数据方面比传统的 RNN 和 LSTM 模型有效率得多？

transformer模型依赖的是自注意力机制，这使得模型可以在处理数据时并行化操作，要知道 RNN 或LSTM 必须按时间步骤顺序处理数据。此外，自注意力允许模型不受顺序长度限制地对所有位置之间的依赖进行建模，而传统的递归网络在长序列上常常面临着梯度消失或爆炸的问题。

## 可以描述一下在 Transformer 模型训练中常用的优化方法和训练技巧吗？

## 可以给出一些基于 Transformer 模型改进和变体的例子，以及它们的优点是什么吗？（如：BERT、GPT系列）

请讨论一下 **Transformer** 模型用于序列生成任务（如机器翻译、文本生成）时的输出序列是如何生成的。

在深度学习中，过拟合是一个常见问题。在实际工作中，你是如何防止 **Transformer** 模型过拟合的？

为了防止过拟合，可采用不同的策略，如增加数据集规模、数据增强、引入dropout(在训练时随机丢弃部分神经元)，以及正则化方法。另外，还可以使用提前停止策略来避免在验证集上的性能不再提升时继续训练从而防止在训练数据上过度拟合。

使用 **Transformer** 模型时可能会遇到的内存和计算问题有哪些？又是如何解决这些问题的？

**Transformer** 处理特别长的序列时会遇到内存和计算需求增大的问题，因为自注意力的复杂度与序列长度的平方成正比。为解决这一问题，可以采用各种策略例如梯度累积(分多个小批次累积梯度后再执行一次参数更新)，降低批量大小(减少同步计算的序列数量)，或使用16位浮点数进行混合精度训练。这能减少内存使用、加速训练，同时对模型精度的影响微乎其微。【什么是混合精度训练】

**Transformer** 模型对于输入序列的长度有什么限制？如果要处理很长的序列，你会采取什么措施？

在某些情况下，**Transformer** 模型可能并不是最佳选择。可以讨论一下在什么类型的任务或场景中可能会偏好其他模型而不是 **Transformer** 吗？

**Transformer 模型的解释性通常被认为是一个挑战，你是否有什么见解或方法来理解模型的决策过程？**

**最后，设想你需要在不同的硬件配置上部署 Transformer 模型。你会考虑哪些因素以及可能采用的优化策略？**

## **面试：为什么要用带掩码masked？**

论文原文是这么说的：第  $i$  个位置的预测结果，仅仅依赖与第  $i$  个位置之前的输出。

咱们解释一下：因为在机器翻译中，是一个先后顺序的过程，即翻译第  $i$  个单词时，我们只能看到第  $i-1$  及其之前的单词。通过 Masked 操作可以防止知道  $i+1$  个及以后的单词

与 encoder 类似，decoder 子层之间通过残差连接 (residual connection)，然后就是层归一化 (layer normalization)

位置编码也被添加到 decoder 的输入 embedding 中，其方式与之前 encoder 的方式相同

## **面试：Q、K、V到底是什么？目的是什么？**

**Transformer 自注意力计算中，为什么 Q 和 K 要使用不同的权重矩阵进行线性变换投影，为什么不使用同一个变换矩阵，或者不进行变换？**

实际上X的线性变换，为了提升模型拟合能力

- 如果 Q和K一样，则矩阵乘积的结果是一个对称矩阵，这样减弱了模型的表达能力。如果Q和K一样，乘积结果的对称矩阵中，对角线的值会比较大，导致每个位置过分关注自己。
- 使用不同的投影矩阵，参数增多，可以增强模型表达能力，简单来说，使用不相同的Q, K, V可以保证在不同空间进行投影，提高表达能力和泛化能力。

## 说明Transformer中的三种mask机制

两种mask，一种是padding mask，用于处理不定长的输入;一种是sequence mask，用于防止未来信息被泄露。

## Transformer&CNN 感受野

- CNN感受野相对更小，所以需要搭建很多层CNN，才能注意到全局信息。而transformer的全局感受能力更强
- Transformer全局信息更加丰富，仅用几层就能达到很大的感受野
- Encoder只有一个Multi-Head Attention层，而Decoder有两个，且Decoder的第一个Multi-Head Attention采用了mash操作  
Decoder第二个 Multi-Head Attention 层的K, V矩阵使用 Encoder 的编码信息矩阵进行计算，而Q使用上一个 Decoder block 的输出计算
- 功能性来说：encoder的功能是将输入序列编码成高维表示；而 Decoder则将高维表示转化为目标序列，方便下游任务
- 感受野：Encoder的感受野是输入序列中某个位置的上下文信息；而Decoder同时关注输入序列和已生成的序列

## 预热？预训练微调？后训练？代码 逐行

**Transformer模型中，注意力计算后面使用了两个 FFN 层，为什么第一个FFN层先把维提升，第二个FFN层再把维度降回原大小？**

提升维度;类似svm kernel，通过提升维度可以识别一些在低维无法识别的特征

提升维度;更大的可训练参数，提升模型的容量

降回原维度：方便多层注意力层和残差模块进行拼接，而无需进行额外的处理

**Transformer一个block 中最耗时的部分是哪个？**

feedword，参数量大。

**Transformer的并行化体现在哪个地方？Decoder端可以做并行化吗？**

Transformer的并行化主要体现在Encoder端，因为Encoder处理的每个位置可以独立于其他位置进行。而Decoder由于自回归的性质，在训练时可以部分并行化(通过 masking技术)，但在推理时必须逐步进行，不能完全并行化。