一文了解Transformer全貌（图解Transformer）

绝密伏击
《推荐系统技术原理与实践》作者，欢迎知友京东购买。

自2017年Google推出Transformer以来，基于其架构的语言模型便如雨后春笋般涌现，其中Bert、T5等备受瞩目，而近期风靡全球的大模型ChatGPT和LLaMa更是大放异彩。网络上关于Transformer的解析文章非常大，但本文将力求用浅显易懂的语言，为大家深入解析Transformer的技术内核。

前言

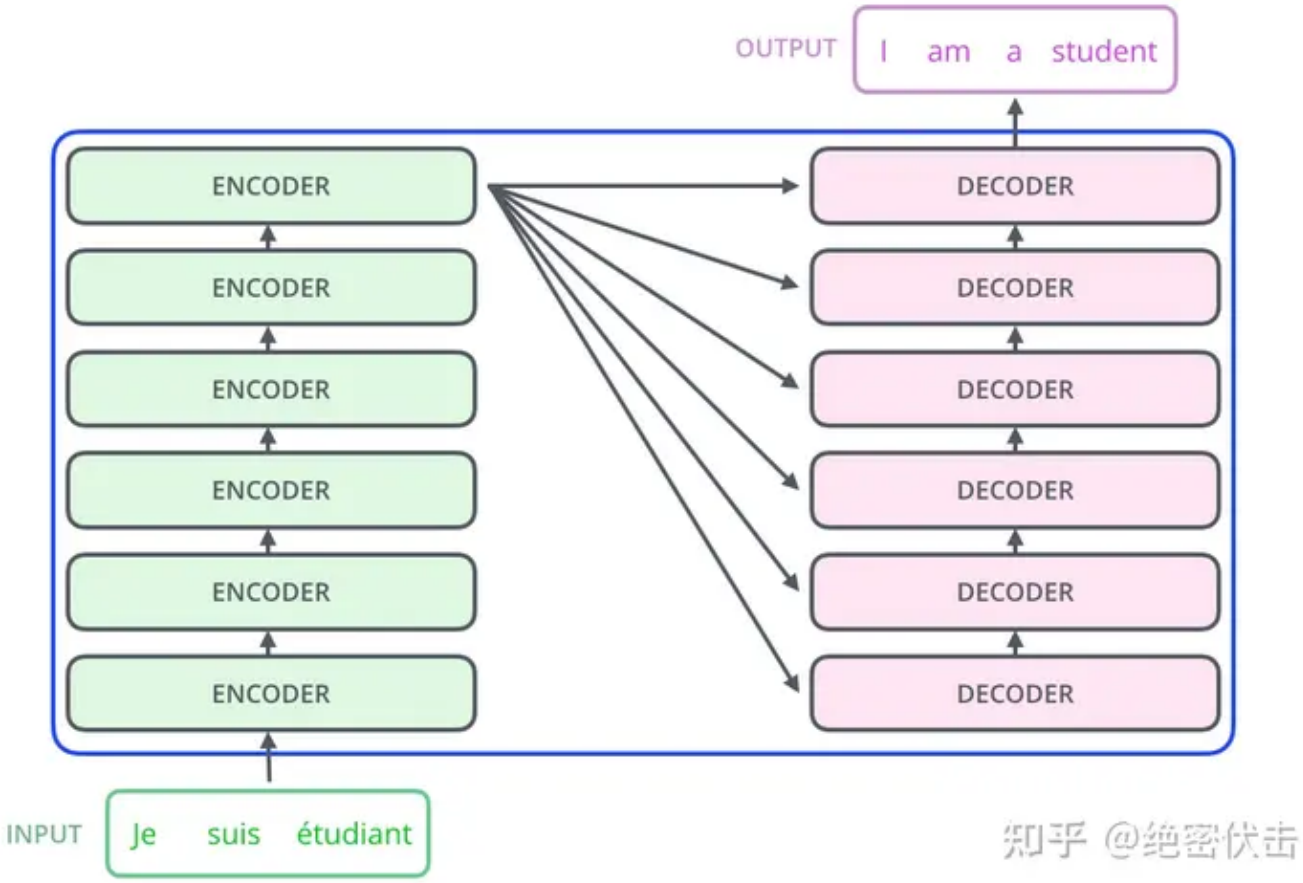
Transformer是谷歌在2017年的论文《Attention Is All You Need》中提出的，用于NLP的各项任务，现在是谷歌云TPU推荐的参考模型。网上有关Transformer原理的介绍很多，在本文中我们将尽量模型简化，让普通读者也能轻松理解。

1. Transformer整体结构

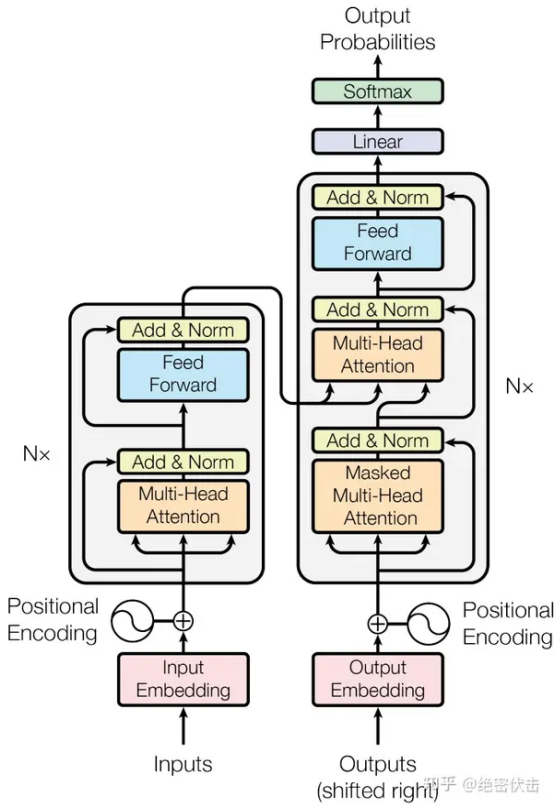
在机器翻译中，Transformer可以将一种语言翻译成另一种语言，如果把Transformer看成一个黑盒，那么其结构如下图所示：

将法语翻译成英语

那么拆开这个黑盒，那么可以看到Transformer由若干个编码器和解码器组成，如下图所示：



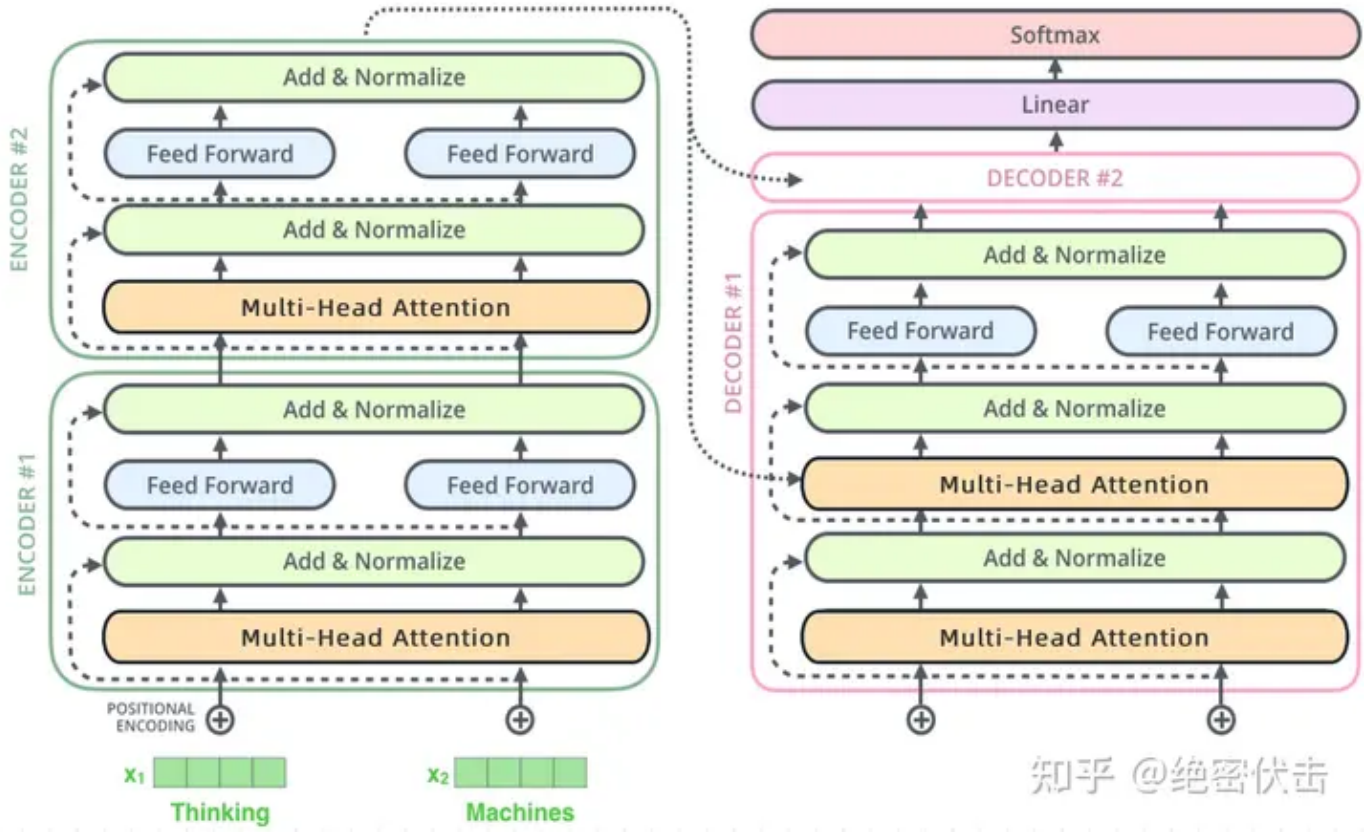
继续将Encoder和Decoder拆开，可以看到完整的结构，如下图所示：



Transformer整体结构（引自谷歌论文）

可以看到Encoder包含一个Muti-Head Attention模块，是由多个Self-Attention组成，而Decoder包含两个Muti-Head Attention。Muti-Head Attention上方还包括一个 Add & Norm 层，Add 表示残差连接 (Residual Connection) 用于防止网络退化，Norm 表示 Layer Normalization，用于对每一层的激活值进行归一化。

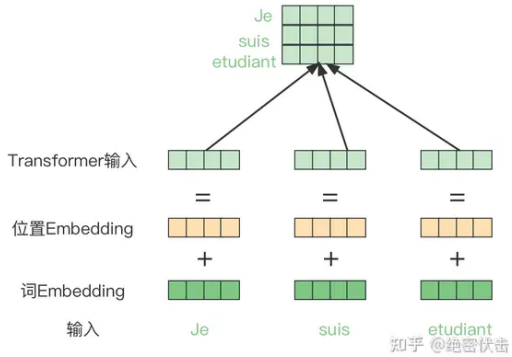
假设我们的输入包含两个单词，我们看一下Transformer的整体结构：



Transformer整体结构（输入两个单词的例子）

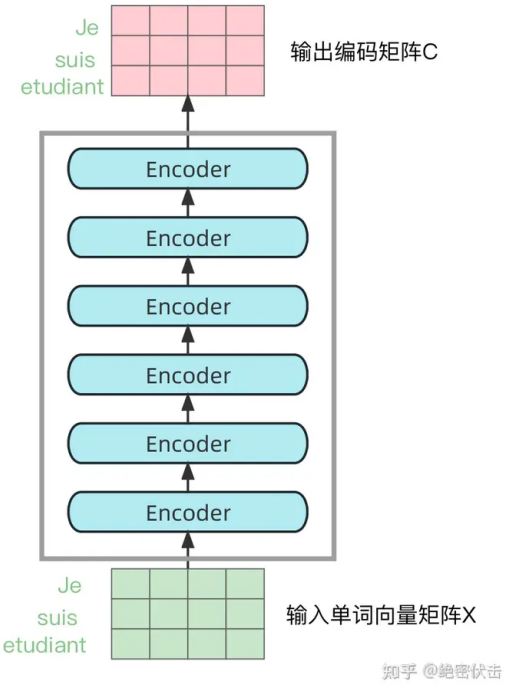
为了能够对Transformer的流程有个大致的了解，我们举一个简单的例子，还是以之前的为例，将法语"Je suis etudiant"翻译成英文。

第一步：获取输入句子的每一个单词的表示向量 $\boldsymbol{x}$ ， $\boldsymbol{x}$ 由单词的Embedding和单词位置的Embedding 相加得到。



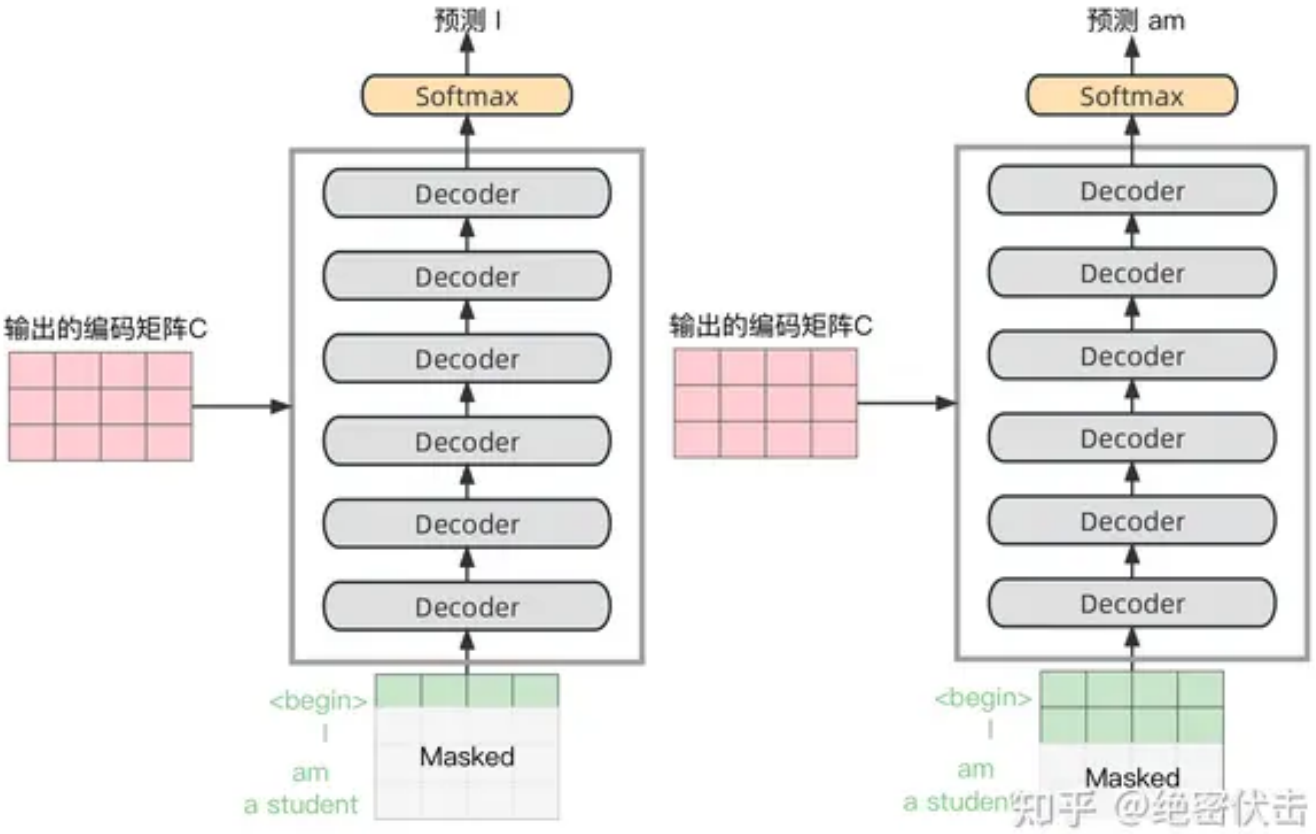
Transformer输入表示

第二步：将单词向量矩阵传入Encoder模块，经过N个Encoder后得到句子所有单词的编码信息矩阵 C ，如下图。输入句子的单词向量矩阵用 $X \in \mathbb{R}^{n \times d}$ 表示，其中 n 是单词个数， d 表示向量的维度（论文中 $d = 512$ ）。每一个Encoder输出的矩阵维度与输入完全一致。



输入X经过Encoder输出编码矩阵C

第三步：将Encoder输出的编码矩阵 C 传递到Decoder中，Decoder会根据当前翻译过的单词 $1 \sim i$ 翻译下一个单词 $i + 1$ ，如下图所示。

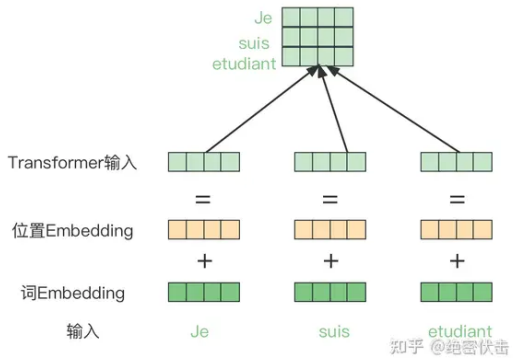


Transformer Decoder预测

上图Decoder接收了Encoder的编码矩阵，然后首先输入一个开始符 "<Begin>"，预测第一个单词，输出为"I"；然后输入翻译开始符 "<Begin>" 和单词 "I"，预测第二个单词，输出为"am"，以此类推。这是Transformer的大致流程，接下来介绍里面各个部分的细节。

2. Transformer的输入表示

Transformer中单词的输入表示由**单词Embedding**和**位置Embedding**（Positional Encoding）相加得到。



Transformer输入表示

2.1 单词Embedding

单词的Embedding可以通过Word2vec等模型预训练得到，可以在Transformer中加入Embedding层。

2.2 位置Embedding

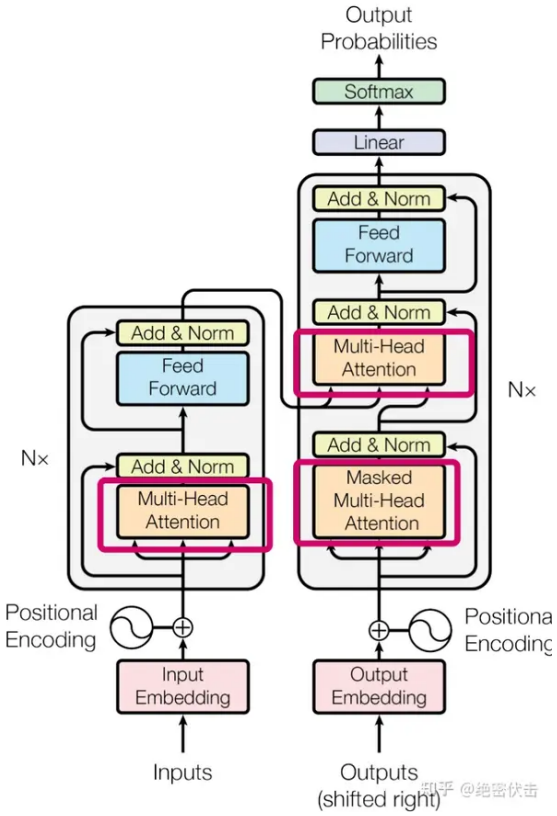
Transformer 中除了单词的Embedding，还需要使用位置Embedding 表示单词出现在句子中的位置。**因为 Transformer不采用RNN结构，而是使用全局信息，不能利用单词的顺序信息，而这部分信息对于NLP来说非常重要。**所以Transformer中使用位置Embedding保存单词在序列中的相对或绝对位置。

位置Embedding用 PE 表示， PE 的维度与单词Embedding相同。 PE 可以通过训练得到，也可以使用某种公式计算得到。在Transformer中采用了后者，计算公式如下：

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

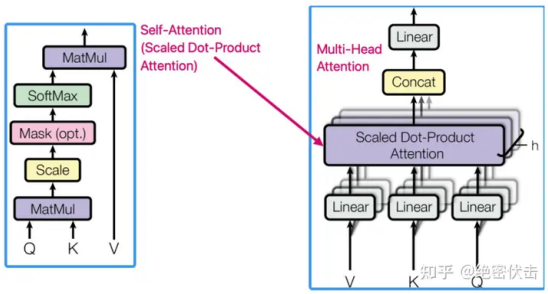
其中， pos 表示单词在句子中的位置， d 表示 PE 的维度。

3. Multi-Head Attention（多头注意力机制）



Transformer内部结构

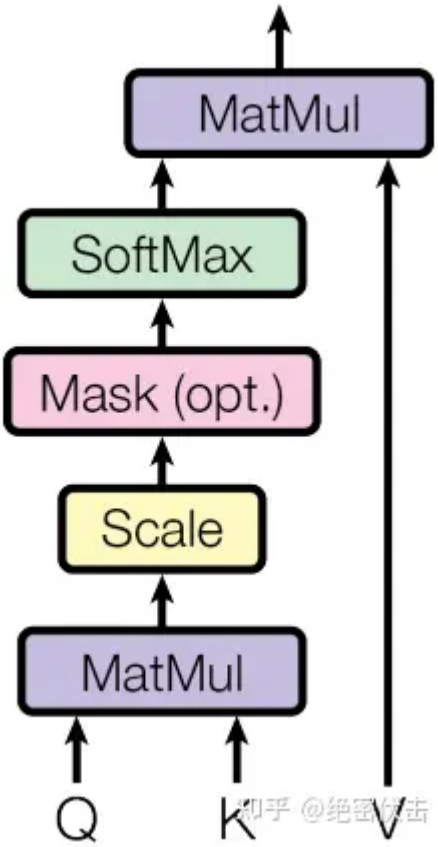
上图是Transformer的内部结构，其中红色方框内为**Multi-Head Attention**，是由多个**Self-Attention**组成，具体结构如下图：



Self-Attention和Multi-Head Attention

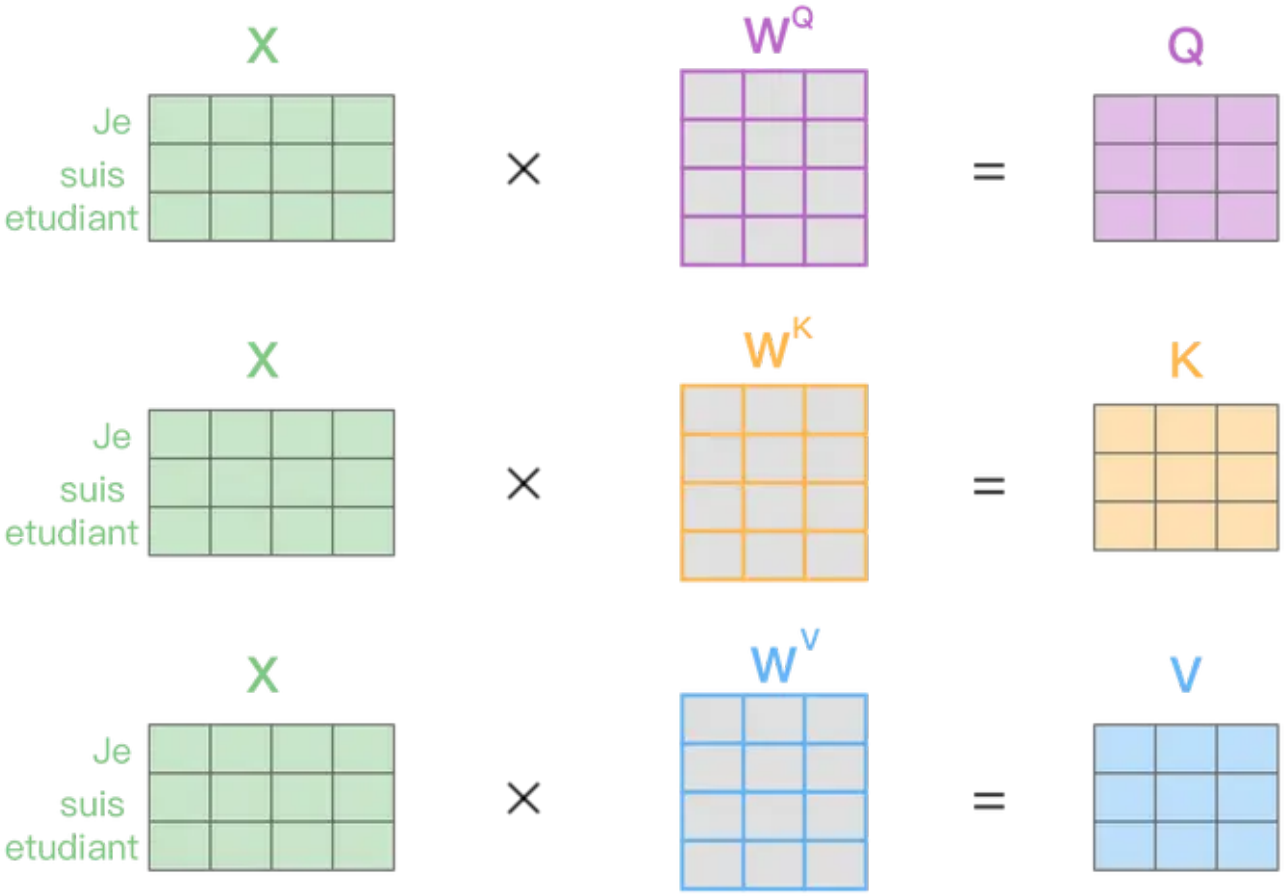
因为**Self-Attention**是Transformer的重点，所以我们重点关注 Multi-Head Attention 以及 Self-Attention，首先介绍下Self-Attention的内部逻辑。

3.1 Self-Attention结构



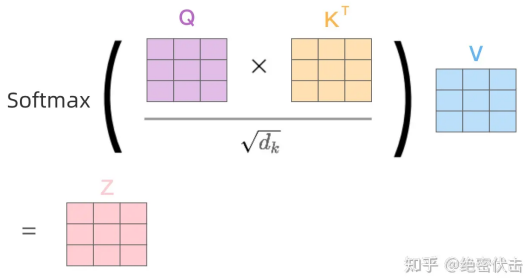
Self-Attention结构

上图是Self-Attention结构，最下面是 $\boldsymbol{Q}$ (查询)、 $\boldsymbol{K}$ (键值)、 $\boldsymbol{V}$ (值)矩阵，是通过输入矩阵 $\boldsymbol{X}$ 和权重矩阵 $\boldsymbol{W^Q}, \boldsymbol{W^K}, \boldsymbol{W^V}$ 相乘得到的。



Q,K,V的计算

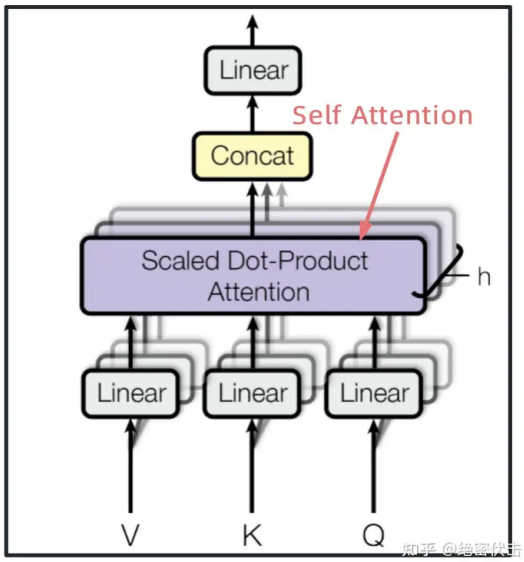
得到 $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ 之后就可以计算出Self-Attention的输出，如下图所示：



Self-Attention输出

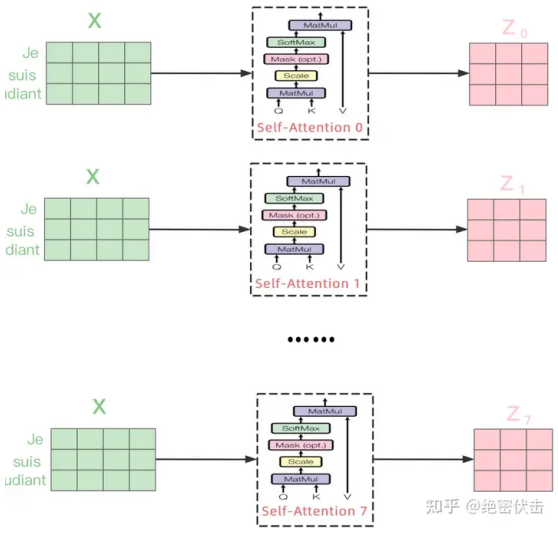
3.2 Multi-Head Attention输出

在上一步，我们已经知道怎么通过Self-Attention计算得到输出矩阵 $\boldsymbol{Z}$ ，而Multi-Head Attention是由多个Self-Attention组合形成的，下图是论文中Multi-Head Attention的结构图。



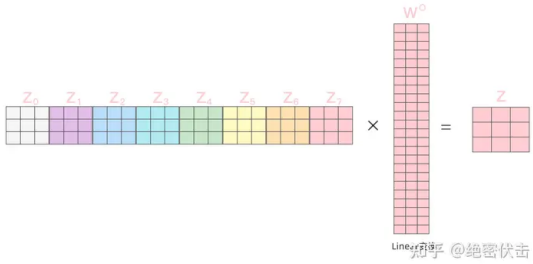
Multi-Head Attention

从上图可以看到Multi-Head Attention包含多个Self-Attention层，首先将输入 $\boldsymbol{X}$ 分别传递到 $\boldsymbol{h}$ 个不同的Self-Attention中，计算得到 $\boldsymbol{h}$ 个输出矩阵 $\boldsymbol{Z}$ 。下图是 $\boldsymbol{h}=8$ 的情况，此时会得到 8 个输出矩阵 $\boldsymbol{Z}$ 。



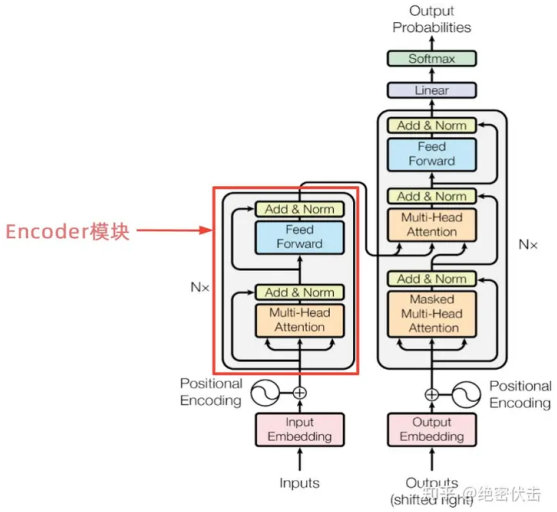
多个Self-Attention

得到8个输出矩阵 $\boldsymbol{Z_0}\sim\boldsymbol{Z_7}$ 后，Multi-Head Attention将它们拼接在一起（Concat），然后传入一个Linear层，得到Multi-Head Attention最终的输出矩阵 $\boldsymbol{Z}$ 。



Multi-Head Attention输出

4. 编码器Encoder结构



Transformer Encoder模块

上图红色部分是Transformer的Encoder结构， N 表示Encoder的个数，可以看到是由Multi-Head Attention、Add & Norm、Feed Forward、Add & Norm组成的。前面已经介绍了Multi-Head Attention的计算过程，现在了解一下Add & Norm和 Feed Forward部分。

4.1 单个Encoder输出

Add & Norm是指残差连接后使用LayerNorm，表示如下：

$$\textbf{Add \& Norm}: \mathbf{LayerNorm}\left(\boldsymbol{X}+\mathbf{Sublayer}\left(\boldsymbol{X}\right)\right) \quad (2)$$

其Sublayer表示经过的变换，比如第一个Add & Norm中Sublayer表示Multi-Head Attention。

Feed Forward是指全连接层，表示如下：

$$\mathbf{FFN}\left(\boldsymbol{X}\right)=\max\left(0,\boldsymbol{X}\boldsymbol{W}_1+b_1\right)\boldsymbol{W}_2+b_2 \quad (3)$$

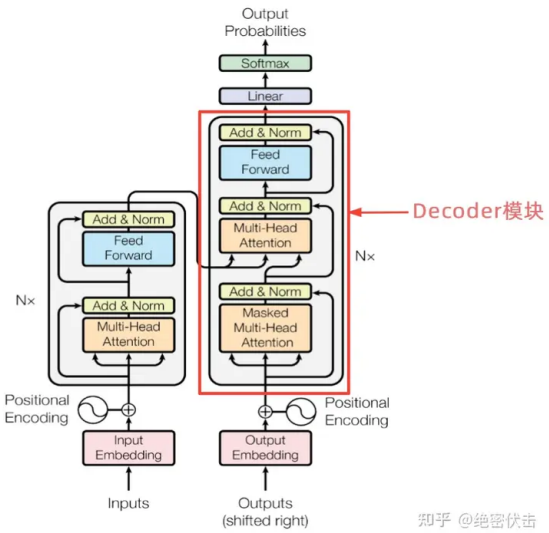
因此输入矩阵 $\boldsymbol{X}$ 经过一个Encoder后，输出表示如下：

$$\begin{aligned}\boldsymbol{O}&=\mathbf{LayerNorm}\left(\boldsymbol{X}+\textbf{Multi-Head-Attention}\left(\boldsymbol{X}\right)\right)\\ \boldsymbol{O}&=\mathbf{LayerNorm}\left(\boldsymbol{O}+\mathbf{FFN}\left(\boldsymbol{O}\right)\right)\end{aligned} \quad (4)$$

4.2 多个Encoder输出

通过上面的单个Encoder，输入矩阵 $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ ，最后输出矩阵 $\boldsymbol{O} \in \mathbb{R}^{n \times d}$ 。通过多个Encoder叠加，最后便是编码器Encoder的输出。

5. 解码器Decoder结构



Transformer Decoder模块

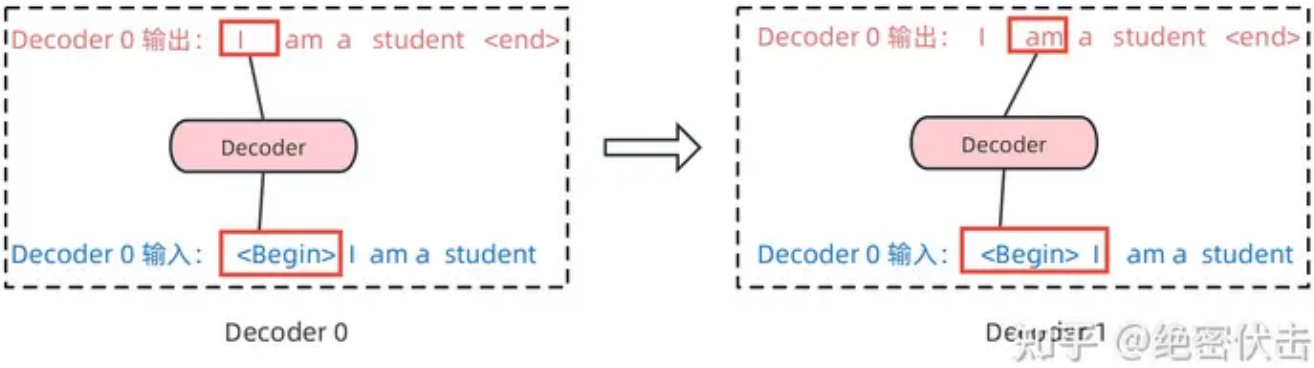
上图红色部分为Transformer的Decoder结构，与Encoder相似，但是存在一些区别：

包含两个Multi-Head Attention
第一个Multi-Head Attention采用了Masked操作
第二个Multi-Head Attention的 $\boldsymbol{K}, \boldsymbol{V}$ 矩阵使用Encoder的**编码信息矩阵** $\boldsymbol{C}$ 进行计算，而 $\boldsymbol{Q}$ 使用上一个 Decoder的输出计算
最后有一个Softmax层计算下一个翻译单词的概率

5.1 第一个Multi-Head Attention

Decoder的第一个Multi-Head Attention采用了Masked操作，因为在翻译的过程中是顺序翻译的，即翻译完第 i 个单词，才可以翻译第 $i+1$ 个单词。通过 Masked 操作可以防止第 i 个单词知道 $i+1$ 个单词之后的信息。下面以法语"Je suis etudiant"翻译成英文"I am a student"为例，了解一下 Masked 操作。

在Decoder的时候，需要根据之前翻译的单词，预测当前最有可能翻译的单词，如下图所示。首先根据输入"<Begin>"预测出第一个单词为"I"，然后根据输入"<Begin> I" 预测下一个单词 "am"。



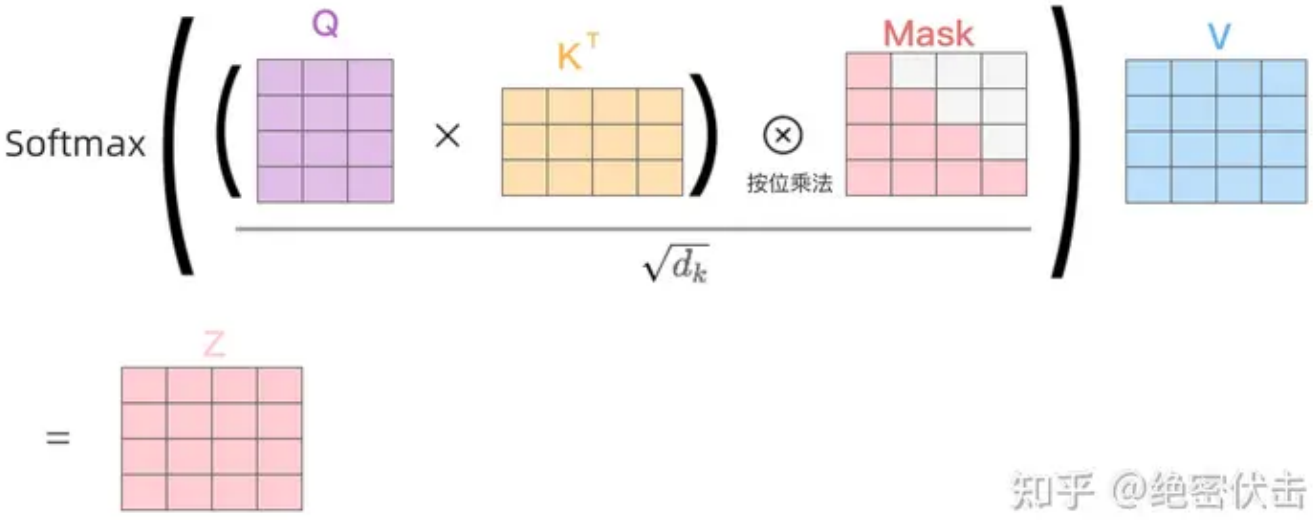
Decoder预测（右图有问题，应该是Decoder 1）

Decoder在预测第 i 个输出时，需要将第 $i+1$ 之后的单词掩盖住，**Mask操作是在Self-Attention的Softmax之前使用的，**下面以前面的"I am a student"为例。

第一步：是Decoder的输入矩阵和**Mask**矩阵，输入矩阵包含"<Begin> I am a student"4个单词的表示向量，**Mask**是一个 4×4 的矩阵。在**Mask**可以发现单词"<Begin>"只能使用单词"<Begin>"的信息，而单词"I"可以使用单词"<Begin> I"的信息，即只能使用之前的信息。

输入矩阵与Mask矩阵

第二步：接下来的操作和之前Encoder中的Self-Attention一样，只是在Softmax之前需要进行Mask操作。



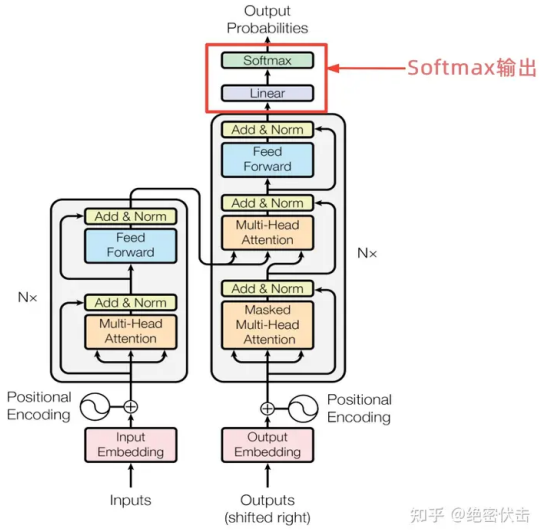
Mask Self-Attention输出

第三步：通过上述步骤就可以得到一个Mask Self-Attention的输出矩阵 Z_i ，然后和Encoder类似，通过Multi-Head Attention拼接多个输出 Z_i 然后计算得到第一个Multi-Head Attention的输出 Z ，Z 与输入 X 维度一样。

5.2 第二个Multi-Head Attention

Decoder的第二个Multi-Head Attention变化不大， 主要的区别在于其中Self-Attention的 K, V 矩阵不是使用上一个Multi-Head Attention的输出，而是使用**Encoder的编码信息矩阵** C 计算的。根据Encoder的输出 C 计算得到 K, V ，根据上一个Multi-Head Attention的输出 Z 计算 Q 。这样做的好处是在Decoder的时候，每一位单词（这里是指"I am a student"）都可以利用到Encoder所有单词的信息（这里是指"Je suis etudiant"）。

6. Softmax预测输出



Softmax预测输出

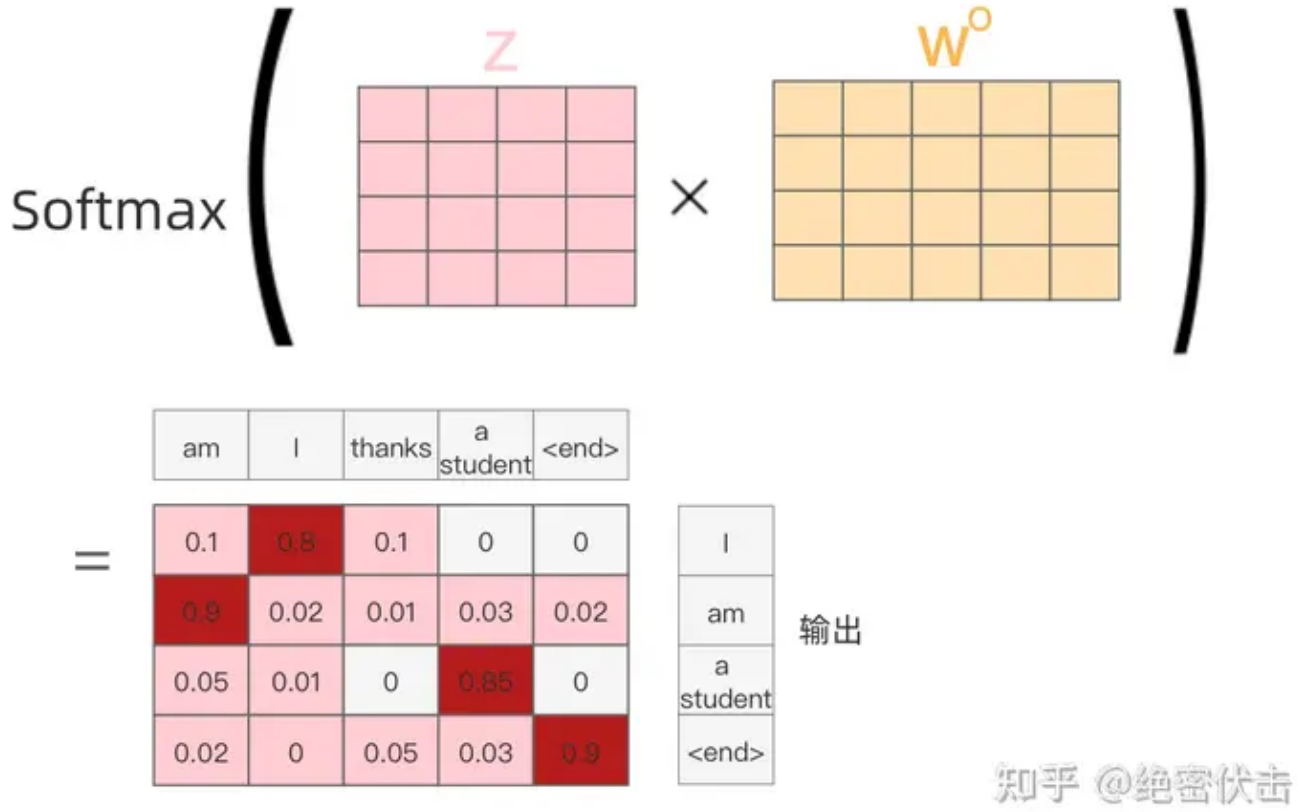
编码器Decoder最后的部分是利用 Softmax 预测下一个单词，在Softmax之前，会经过Linear变换，将维度转换为词表的个数。

假设我们的词表只有6个单词，表示如下：

词	am	I	thanks	a student	<end>
编码	0	1	2	3	4

词表

因此，最后的输出可以表示如下：



Softmax预测输出示例

总结

Transformer由于可并行、效果好等特点，如今已经成为机器翻译、特征抽取等任务的基础模块，目前ChatGPT特征抽取的模块用的就是Transformer，这对于后面理解ChatGPT的原理做了好的铺垫。

代码实现

绝密伏击：OPenAI ChatGPT（一）：Tensorflow实现Transformer

参考

初识CV：Transformer模型详解（图解最完整版）

数据汪：BERT大火却不懂Transformer？读这一篇就够了

The Illustrated Transformer

忆臻：搞懂Transformer结构，看这篇PyTorch实现就够了（上）

The Annotated Transformer

arxiv.org/pdf/1706.0376

青空栀浅：图解Transformer

Ph0en1x：Transformer结构及其应用详解--GPT、BERT、MT-DNN、GPT-2

大师兄：ChatGPT/InstructGPT详解

张俊林：ChatGPT会取代搜索引擎吗

张俊林：放弃幻想，全面拥抱Transformer：自然语言处理三大特征抽取器（CNN/RNN/TF）比较