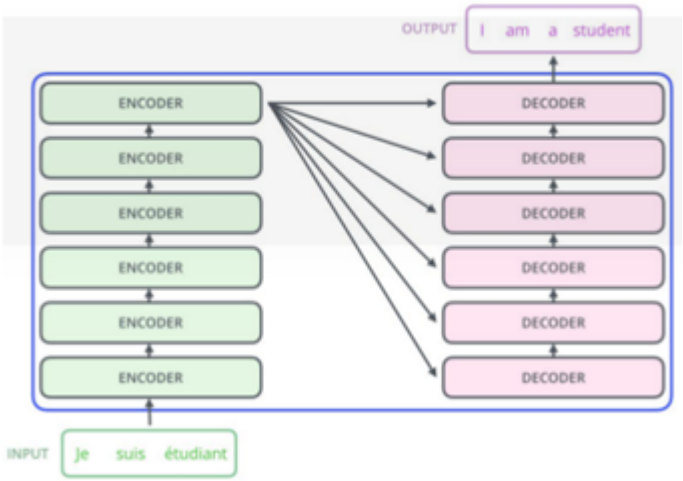
Transformer

Attention Is All You Need. Ashish Vaswani. NIPS 2017. [PDF]

Transformer是Google于2017年提出的序列建模模型，一开始应用于机器翻译上，它摒弃了RNN和CNN，采用纯注意力计算，大幅提高了序列建模的有效长度，并且也提升了在机器翻译任务上的成绩。这一工作为后来预训练的崛起奠定了扎实的基础。

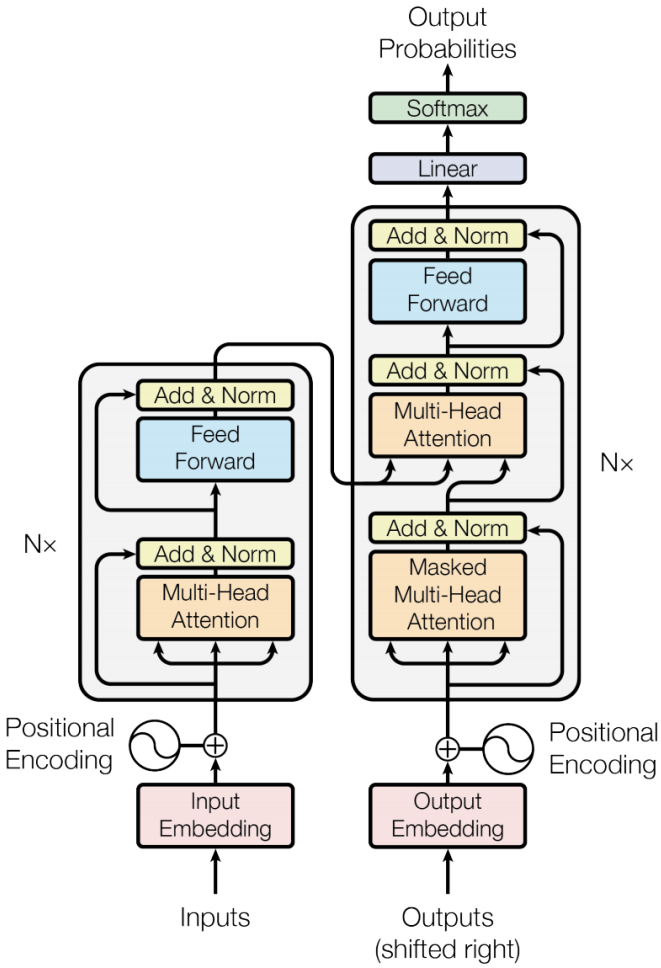
1 模型

Transformer的模型架构非常新颖。如下图所示，整个模型分为Encoder和Decoder两个部分，二者分别由六个相同的Encoder层和Decoder层堆积而成。



Transformer整体架构

而在Encoder层和Decoder层的内部，则是包含了**多头注意力层**（Multi-Head Attention）、**前向传播层**（Feed Foward）和**残差归一化层**（Add&Norm），如下图所示：



Transformer Encoder层和Decoder层内部结构

每个Encoder层内部包含了Multi-Head Attention和Point-Wise Fully Connected Feed-Forward Network两个子层，而在每个子层的输入和输出都设置了残差连接，并加上了一个层归一化（Layer Normalization），于是对于每一个子层都有：

$$LayerNorm(x + sublayer(x))$$

每个Decoder层和Encoder层包含的子层差不多，只是Decoder层多了一个Masked Multi-Head Attention子层，这里的Masked的作用就是防止在解码时注意力“看到”了当前预测位置后面的字符。

接下来将按照自底向上的顺序详细介绍Encoder层和Decoder层内部的各个模块。

2 输入和位置编码

由于摒弃了RNN和CNN，序列每个位置上仅仅输入token embedding会不可避免地缺失位置信息，于是Transformer把token embedding和位置编码（Positional Embedding）加在一起作为最终的Input embedding，三者的维度都是 d_{model} .

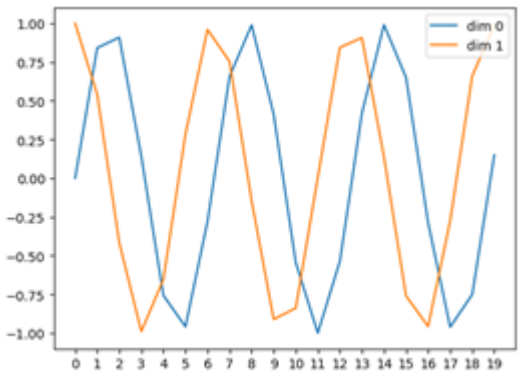
对于输入的embedding，Transformer使用的是预训练的token embedding，Encoder和Decoder共享同一个Embedding Matrix；

而对于位置编码，Transfromer采用固定的位置编码，计算公式为:

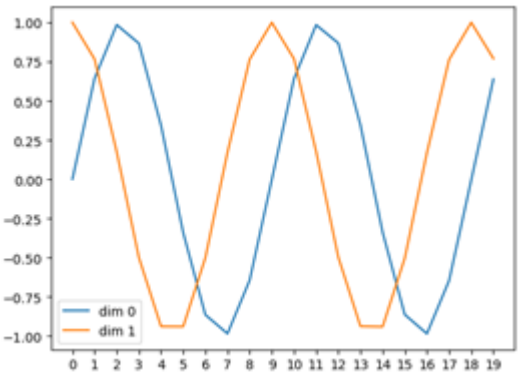
$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}})$$

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

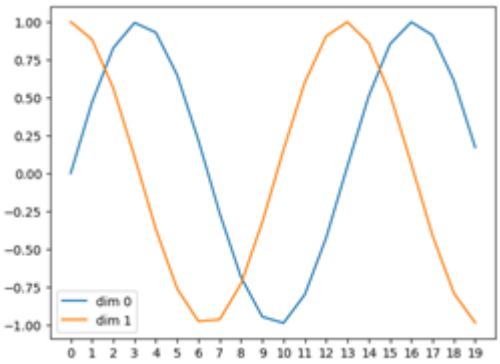
其中pos表示序列位置，i表示维度。由此可见，在同一个维度上（即i一定），位置嵌入的取值是通过一个正弦或者余弦函数计算得到的。只不过随着维度的增大，正弦（余弦）函数的周期从 2π 增大到 $10000 \cdot 2\pi$ ，并且相邻两个维度（0和1、2和3，依次类推）上的函数的周期相同。公式不是很直观，可以用代码画出来。



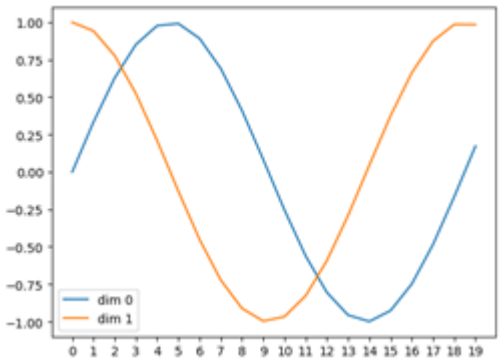
位置嵌入在第0维和第1维上的取值



位置嵌入在第20维和第21维的取值



位置嵌入在第40维和第41维上的取值



位置嵌入在第60维和61维上的取值

上面四个图中的横坐标均表示位置，由此可见位置编码在奇数维度上的取值可以看成离散化的余弦曲线，在偶数维度上的取值可以看成是离散化的正弦曲线。当 $2i = d_{model}$ 时，周期最大，这里的 d_{model} 是模型中每个编解码层的输入输出的统一长度，也是token embedding的维度。

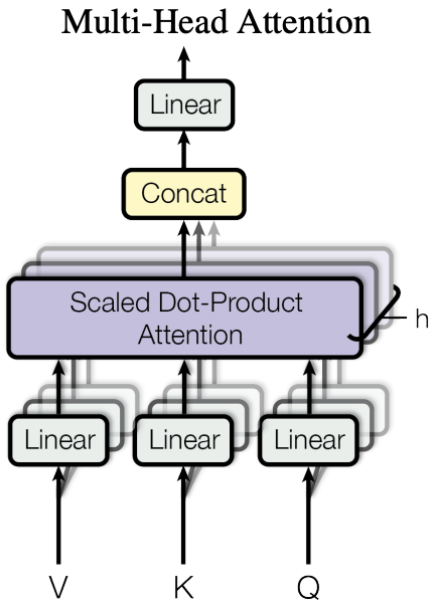
位置编码这种设计的目的是为了让模型更好地学习到相对位置。对于位置偏移量 k ， PE_{pos+k} 都可以由 PE_{pos} 的线性函数表示。不妨假设当前维度是偶数，令 $m = 10000^{-2i/d_{model}}$ ，则

$$\begin{aligned} PE_{pos+k} &= sin\big(m \cdot (pos + k)\big) \\ &= sin(m \cdot pos)\, cos(m \cdot k) + cos(m \cdot pos)\, sin(m \cdot k) \end{aligned}$$

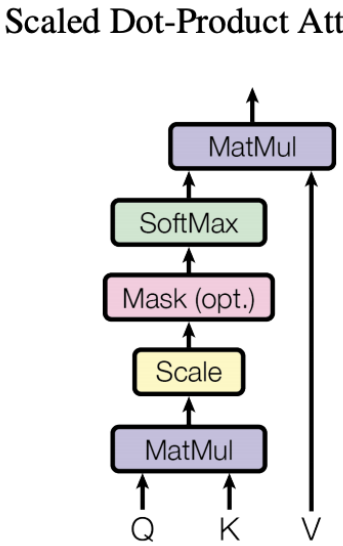
其中 $\cos(m \cdot k)$ 、 $\sin(m \cdot k)$ 都是常数， $\cos(m \cdot pos)$ 就是下一个维度相同位置上的位置编码取值。

3 多头注意力

多头注意力是整个Transformer的核心。Transformer之所以能实现高效的并行计算，多头注意力功不可没。多头指的将当前的输入分成多个相同的计算分支，最后每个分支的计算结果将被拼接在一起作为完整的输出，如下图(a)所示。其中的核心部分自然是缩放点乘注意力（Scaled Dot-Product Attention），如下图(b)所示。它和Luong提出的全局注意力一样，只不过这里的符号表示稍有变动。回顾基于注意力的编解码模型，图中的K、V就是Encoder中的隐藏状态（也称source hidden state），而Q是Decoder中的隐藏状态（target hidden state）。注意力首先基于Q、K计算出每个K的注意力权重，然后再将其作用在V上计算加权和。



多头注意力的多头分支计算和合并



缩放点乘注意力

值得注意的是，缩放点乘注意力有三点改进。第一是**加入了缩放来控制数值的取值范围**，避免QK相乘后数值过大降低了计算效率；第二是**加入了注意力遮掩机制**。这个是Decoder层才有的（即Masked Multi-Head Attention和Multi-Head Attention的区别所在），目的是为了避免计算力时，覆盖到了当前解码位置后面的字符。实现办法就是在对注意力分数进行归一化（Softmax）之前，将后面的当前解码位置之后的注意力分数全部设置为负无穷大，这样归一化之后的注意力权重就为零；第三，**用矩阵计算代替向量计算**，将所有source hidden state和target hidden state组成Q、K、V三个矩阵，在编码时，Q、K、V是相同的，包含所有source hidden state，是一种自注意力计算；在解码时，仅Q发生改变，它变成由所有target hidden stated组成的矩阵。

整个计算过程如下所示：

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$
$$MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O$$
$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

线性层参数 $(W_i^Q, W_i^K \in \mathbb{R}^{d_{model}\times d_k}$，$W_i^V \in \mathbb{R}^{d_{model}\times d_v})$ 首先将维度为 d_{model} 的输入向量 Q 、 K 、 V 分别压缩到 d_k 、 d_k 、 d_v .随后各个分支的缩放点乘注意力计算出结果 $head_i$ ，最后经过拼接和线性层的计算得到输出。其中隐含条件是 $d_{model} = h \times d_v$.

由于设置了残差连接，所以模型内部各个层的输入输出的维度始终维持在 d_{model} .

4 位置级前向传播

位置级前向传播（Position-wise Feed-Forward）是两层全连接网络，中间用一个ReLU激活层连接。它作用在输出的每个位置上。

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

W_1 将 d_{model} 先扩张到2048维，然后再压回到512.

5 总结

1. **优点**：Transformer的序列建模能力（尤其是在长序列上）超越了RNN、CNN，凭借的就是注意力机制。和逐步传播相比，它直接将任意两个位置的隐藏状态连接起来，所以长期依赖信息全都可以捕捉到；另外，多头注意力的并行计算有效缓解了注意力的高复杂度 $O(n^2)$ ；
2. **缺点**：Transformer存在上下文分裂问题，可参考Transformer-XL.

近几年，不仅仅是GPT、BERT等预训练模型将其作为基本网络层，机器翻译、阅读理解和命名实体识别等其他任务模型也开始将其作为基本的特征提取器。由此可见，Transformer俨然已经成为新的序列建模大杀器。