Transformer相关——（8）Transformer模型

发表于 2021-08-18 | 更新于 2021-08-19 | 深度学习
| 字数总计: 1,702 | 阅读时长: 6分钟 | 阅读量: 2498

Transformer相关——（8）Transformer模型

引言

千呼万唤始出来，前面做了那么多Transformer内部相关模块扩展和铺垫，现在让我们正式地来看一下Transformer模型。

这一篇会对前面的各个模块是如何在Transformer中结合的，也就是会对Transformer的结构和运行机制进行介绍。虽然各个模块都拆解完毕了，但是连接和运行机制还是有设计的嗷，又是亿个小细节~

Transformer模型由N个Encoder层和N个Decoder组合而成，接下来分别介绍单个Encoder层和Decoder层内部的结构和运行机制，然后会介绍Encoder和Decoder之间是如何交互的。

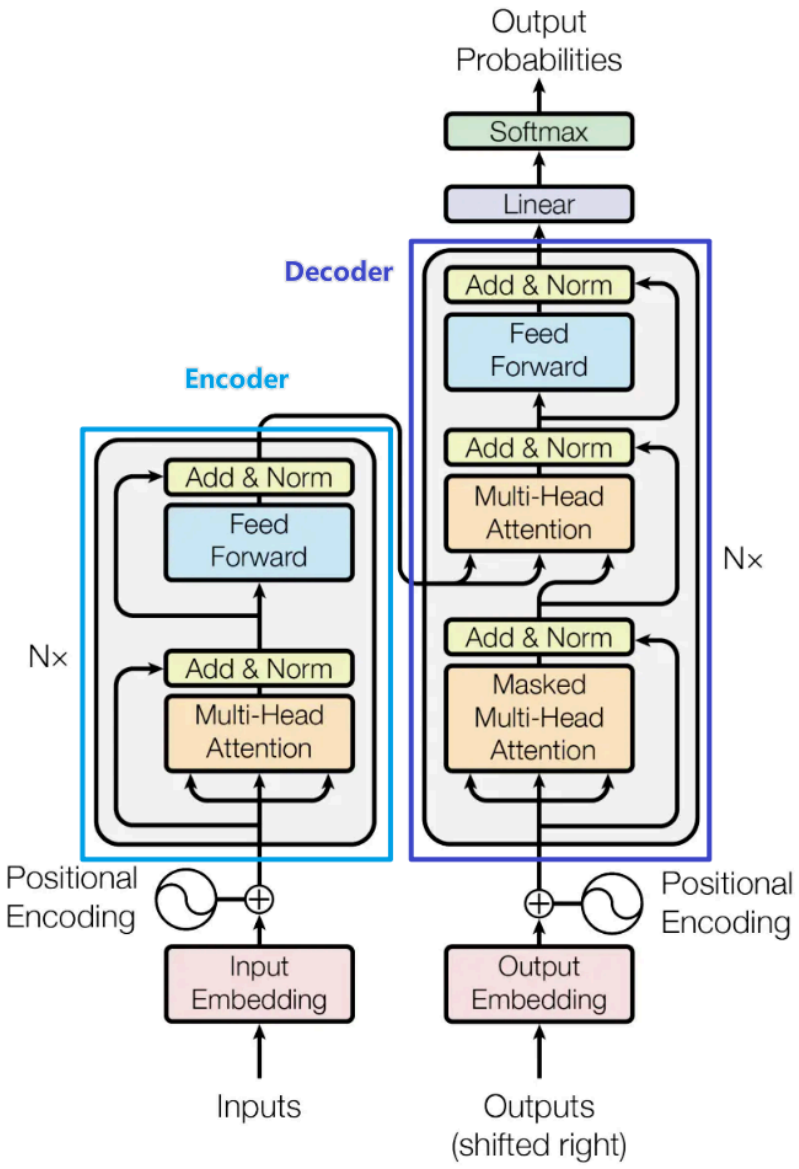
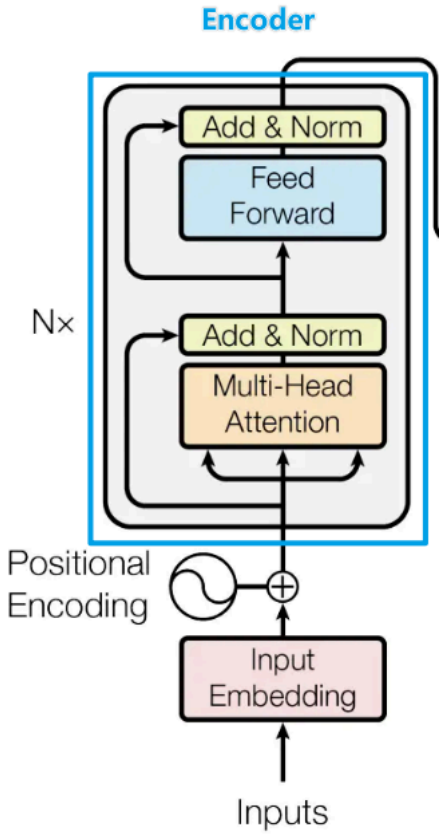


Figure 1: The Transformer - model architecture.

Encoder层



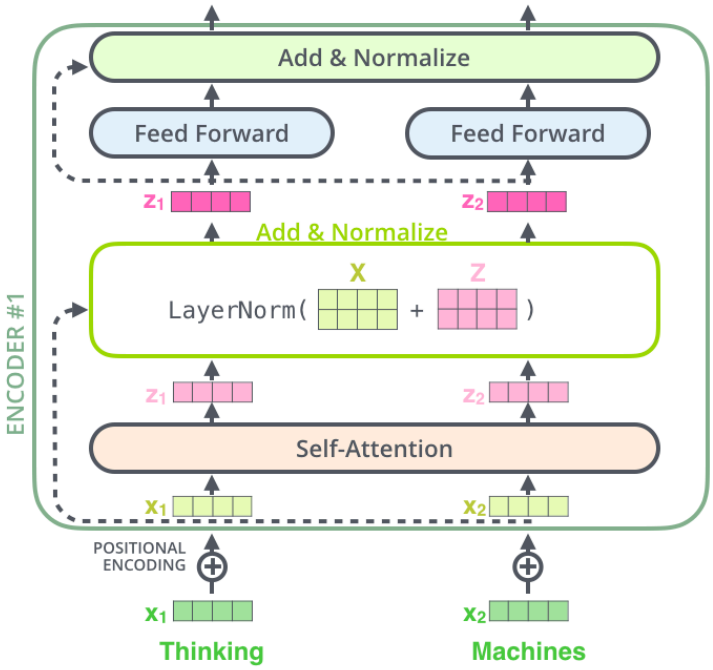
1 我们先来看Encoder部分，从最下方的$Inputs$开始，这里就输入了一个序列$a_1, a_2, \ldots a_N$（比如在NLP中，输入了一个句子），然后获得每一项a_i的embedding（嵌入），这里的embedding其实是a_i的特征向量。

2 接着利用前面提到的位置编码position encoding方式对序列的各个位置进行编码，并把位置编码向量与序列的特征向量embedding直接相加，得到下一层（Multi-Head Attention）的输入X'。

3 可以看到Multi-Head Attention模块的Q、K、V来自于同一个输入X'（模块输入的三个箭头来源相同），所以它是一个Multi-Head self-Attention，多头自注意力机制模块，这在之前的Attention机制中说到过。Transformer的attention score的计算方法采用的是缩放点积相关性：$\alpha_{i,j} = \frac{(q^i \cdot k^j)}{\sqrt{d}}$，$d$是输入信息的维度。**为什么要对点积进行缩放？**参考：transformer中的attention为什么scaled? - TniL的回答

4 经过多头自注意力机制模块的输出下一步经残差模块和Normalization模块（Add&Norm），这可以缓解深层次网络梯度弥

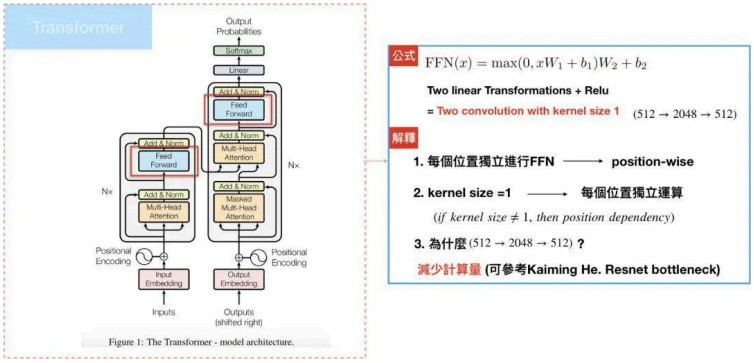
散、网络退化等问题。



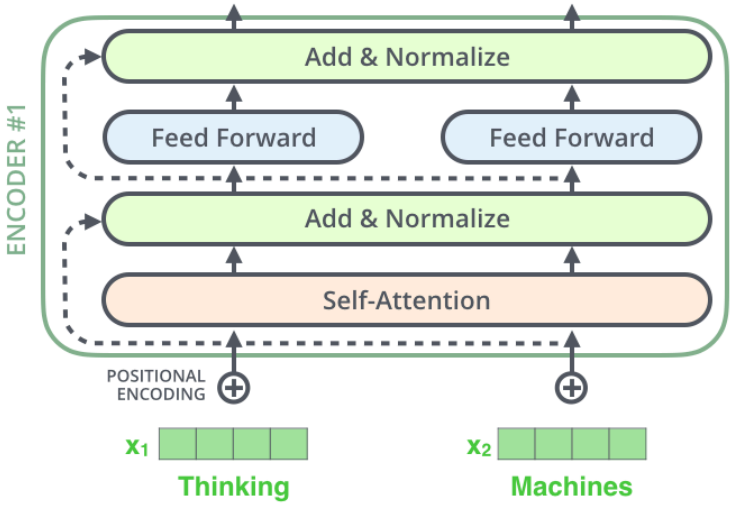
5 经残差+正则化模块后，将进入位置前馈网络 (**position-wise** feed-forward network, FFN) 层，同样该层的输出也需要再经过残差+正则化模块。

这里需要特别补充的是，**位置前馈网络层对于 Transformer 实现良好性能至关重要。**

> **Position-wise Feed-Forward network是一个全连接网络，**包含**两个线性变换和一个非线性函数**（ReLU）。公式如下：$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$
>
> 这个线性变换在不同的位置都是一样的，并且在不同的层之间使用不同的参数。



position-wise Feed-Forward network的细节如下图所示，这个position-wise可以看作**两个核大小为1x1的一维卷积层**。



> 研究者观察到**简单地堆叠 self-attention 模块会导致等级崩溃问题以及 token 均匀性归纳偏差，而前馈层是缓解此问题的重要构建块之一。**

这里需要强调一下，我们可以看到Encoder部分都是关注上下文信息的，而self-attention是支持并行化的，因此**Encoder可以并行处理输入的序列**，然后**输出一整个序列的embedding**。

Decoder层

我们暂时先不关注图中Encoder是如何和Decoder结合的，先看一下Decoder是如何运作的。

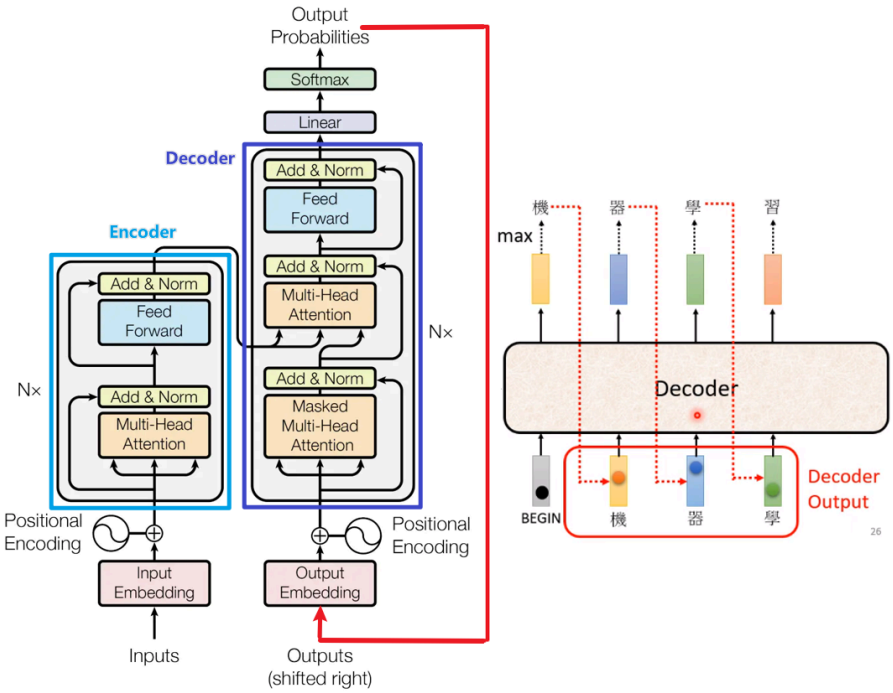
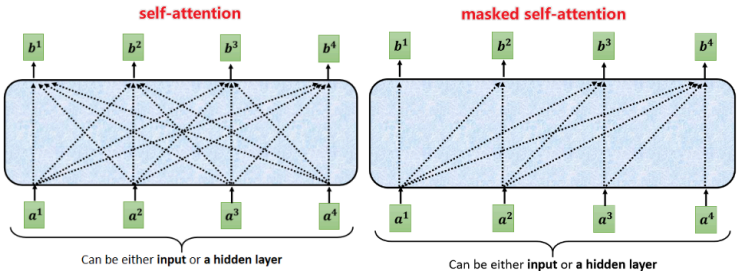


Figure 1: The Transformer - model architecture.

1 我们从原框架的右下角开始看（上图Decoder的最下面），这里输入是一个$Outputs$，**Decoder的第一个输入来自于目标序列**，在NLP中，常常会在输入句子的开头加上一个表示开始的字符（比如$[CLS]/[BEGIN]$等等），这个字符经过embedding后就是Decoder的第一个输入。

结合右图，可以发现Decoder的机制类似于RNN，是**串联**的，前一个输出会作为后一项的输入，所以我在原始的Transformer框架中，加了一个红色箭头表示其输出又作为Decoder的输入，对$Outputs$进行一个解释。

2 Decoder的输入进入Masked Multi-Head Attention模块，可以看到该模块三个箭头的输入均来自于同一个embedding，所以是一个self-Attention模块，额外加了mask机制，self-Attention和masked self-attention的差别如下图所示，当前预测位置只能获取已经输出了的信息，即b_1只能获取a_1的信息，b_2只能获取a_1, a_2的信息.....以此类推。

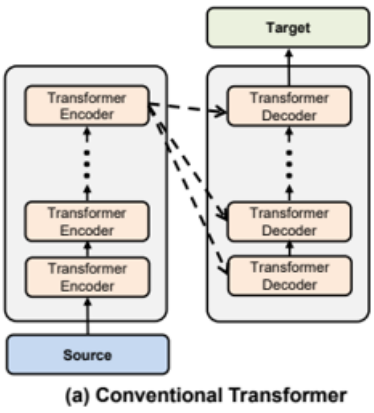


3 接下来经残差+正则化模块后，输入另外一个Multi-Head Attention模块，注意该模块的输入箭头，**有两个箭头来自于Encoder的输出，一个箭头来自于Decoder上一层的输出。该Attention模块为Encoder-Decoder的cross attention模块**（我们下面详细说）。

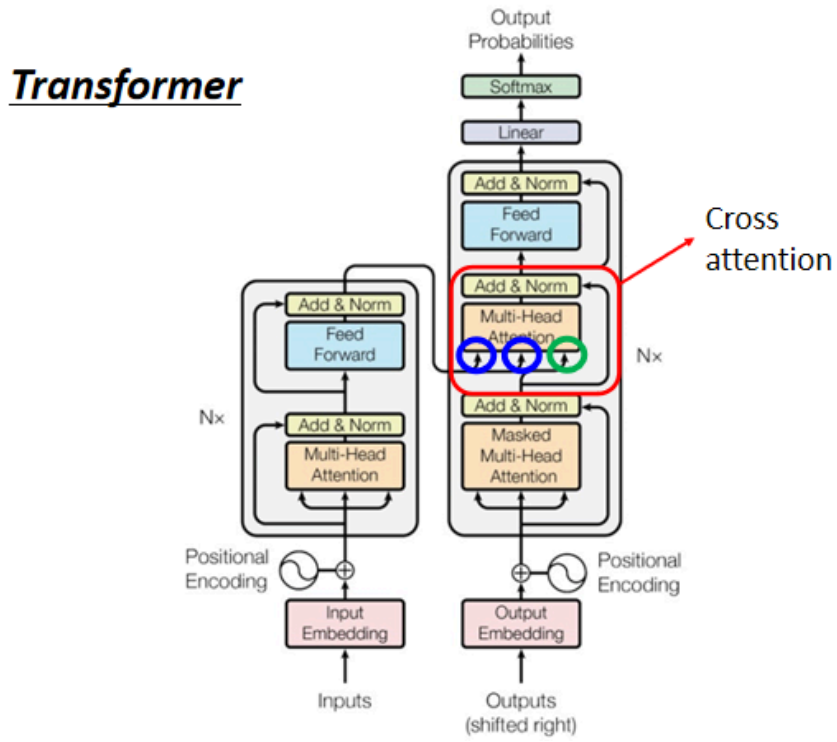
4 经过该cross attention模块后，再经过与Encoder类似的残差+正则化模块、FFN模块+残差+正则化模块，最后接上与下游任务相关的线性层（比如分类、线性回归等），逐步获取序列的输出。

Encoder-Decoder交互-cross attention

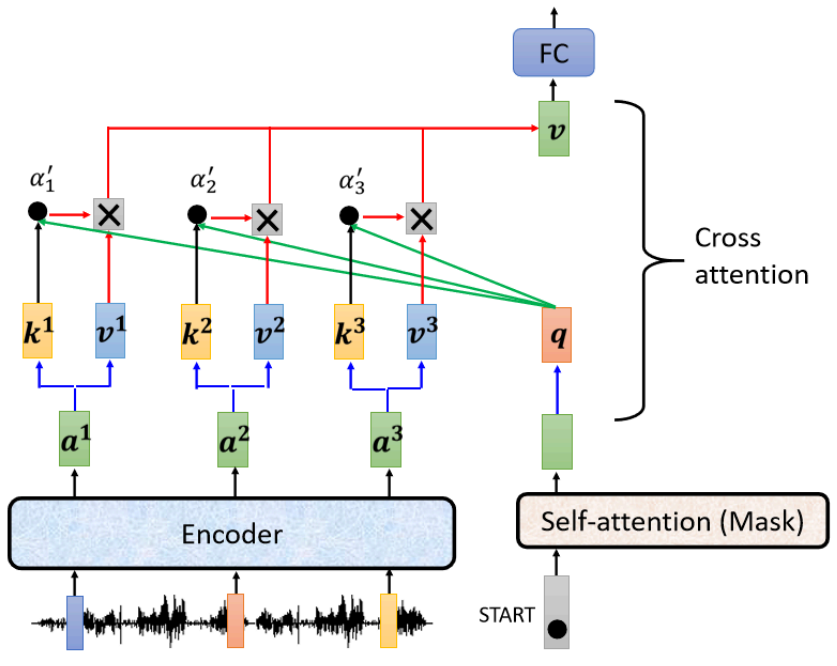
Encoder-Decoder之间交互存在多种形式，传统Transformer中Encoder和Decoder交互的方式是：第N个Encoder层最后的输出与每一层Decoder进行交互，如下图所示：



在$Target_0$经过Masked Multi-Head Attention、Add&Norm模块后（我们这里暂时记为O_0'）。可以看到下图，O_0'**（绿圈所示箭头）和来自Encoder的输出（两个蓝圈所示箭头）作为下一多头自注意力模块的输入。**



在详解Attention机制时，提到Q、K、V三项可以来自不同矩阵，选择不同的Q、K、V就形成了不同的attention变形，比如当Q=K=V时，就是self-attention机制，那么这里Encoder-Decoder交互的Cross Attention实际上就是**将Decoder内该模块上一层的输出作为Q，而Encoder最后一层的输出（一整个序列的embedding）作为K和V（K=V）**。可以直观理解为，哪个Key可以更好地回答这个Query。



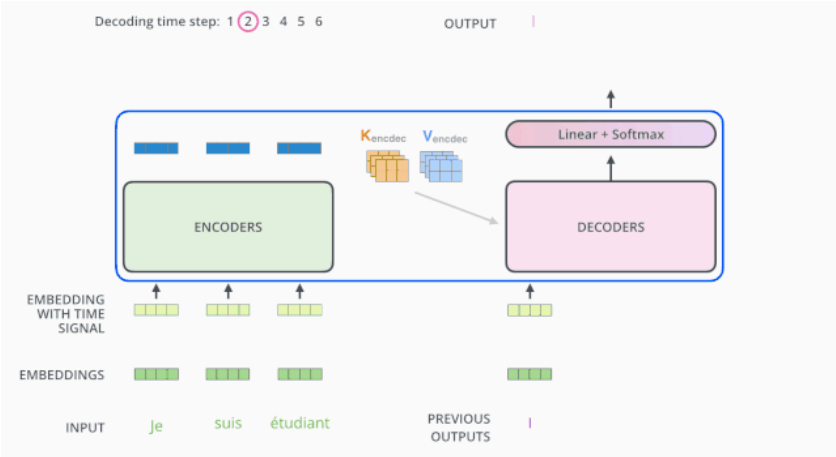
以"机器学习"这句话为例：

1. "BEGIN"为Q，相当于找到字典中谁最可能是这句话的开始，"机"这个字的概率最大，输出"机"；
2. "BEGIN"和"机"经self-attention为Q，下一个字最可能是"器"那么输出"器"；
3. "BEGIN"、"机"、"器".....以此类推。

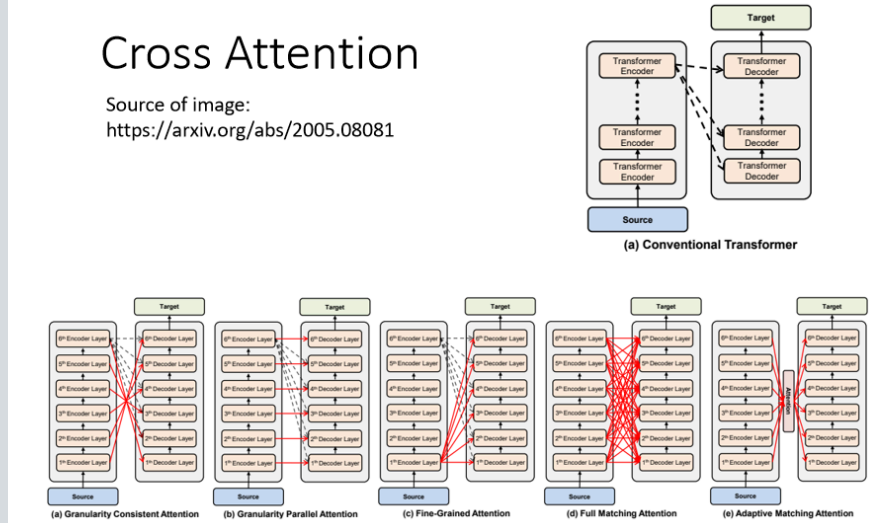
那啥时候结束呢？通常是在句子末尾加一个结束字符表示该句子结束（比如$[END]/[SEP]$等）。

当输出为结束字符，就结束输出。

下面这个动图也可以帮助理解Decoder的机制：



一个Transformer中是有好几层Encoder层和Decoder层的，如下图所示。**Decoder 每一层不是一定要看Encoder 最后一层的输出，可以有各种各样的连接方式。这都属于Cross Attention。**



参考文献

Transformer架构详解

(强推)李宏毅2021春机器学习课程

李宏毅老师机器学习课程笔记

2.2-图解transformer.md