Attention Is All You Need

“Attention Is All You Need” 一文自从发表以来，犹如一声春雷，对以 RNN 为代表的深度网络模型产生了很大的影响，尤其是机器翻译等 NLP 工作提供了新的思路。这里就简单记录一下阅读体验。文章链接：https://arxiv.org/abs/1706.03762

摘要

目前主要的序列转化模型，都是基于 RNN/CNN ，使用端对端的 Encoder - Decoder 框架。最好的模型中，引入了 Attention 机制。本文提出了一个全新、简单的网络结构，Transformer，它仅仅使用了 Attention 机制。在机器翻译的任务和相关评测上取得了更好的结果。实验中，也显示出 Transformer 对于英语相关的成分分析任务上，无论数据集的大小，都具有更好的泛化能力。

简介

RNN / LSTM / GRU 现在是建立序列化模型的标配，尤其是在语言模型研究和机器翻译领域。典型的循环网络会根据输入和输出符号的位置信息进行运算，把一个位置作为循环的一步，每一步得到的一个隐变量 h_t 是上一个隐变量 h_{t-1} 和当前位置输入 t 的一个函数。输入、输出信息的序列化的特性，却成为训练并行化的一个阻碍，虽然也有一些相关工作对此进行了一些改进，然而本质的问题依旧存在。

Attention 机制现在已经成为序列化模型、转化中的一个内在组成部分了，旨在解决序列中长距离依赖的问题。本文据此提出了 Transformer 模型，规避了循环而完全只依赖于注意力机制，为输入和输出序列刻画全局的依赖信息。Transformer 不仅能够并行训练，也将机器翻译的质量带向了新的 the-state-of-the-art。

背景与相关工作

注意到，有一些相关工作如 ConvS2S、ByteNet，使用 CNN 的结构来构建输入块，可以并行地计算每个隐藏特征。但是为了考虑到长距离的依赖问题，相关的计算代价与输入输出对应的长度和位置关系呈线性、对数增长关系，长距离的对应关系依旧是性能的瓶颈。而 Transformer 通过一种 **多头 Attention** (Multi-Head Attention) 则将代价降到常数级别。

Self-attention，是一种内部自我的注意力机制，通过对序列中的不同位置上作用 Attention，计算出序列的表示。在阅读理解、摘要抽取、文本续写甚至是一些没有具体任务的无监督文本表示任务中，也取得了巨大的成功。

端到端的记忆网络，是基于循环网络加上 Attention 的结构，而不是在循环中完成序列对齐的。这在问题回答、语言模型任务中也是十分常见的方法。

本文提出的 Transformer 模型，是第一个仅仅依靠 self-attention 计算输入、输出序列的表示的转化模型。

模型结构

与传统的神经网络的序列化建模方式一致，也是基于 Encoder-Decoder 的结构，如下图 1 所示。

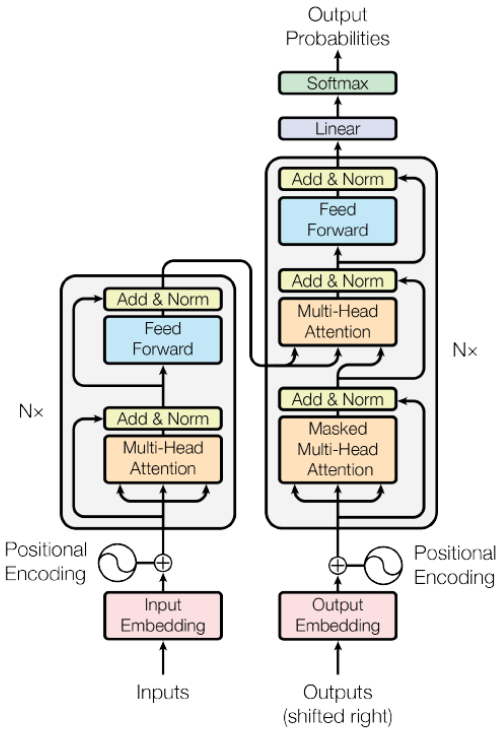


Figure 1: The Transformer - model architecture.

Encoder - Decoder

1. Encoder 部分是 N = 6 个相同的层堆叠得到，其中每一层由两个子层——首先是一个多头 Attention 层，第二层就是一个简单的 position-wise 的全连接前向网络。在两个子层之间，都使用残差连接，并且加上了 normalization 操作。具体来说，每一个子层的输出是 $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$。在这个模型中，取所有的 embedding 维数为 512。
2. Decoder 部分也是类似的，是 6 个相同的层堆起来的，只是每一层里有三个子层，前两个是多头 Attention，最后一个是前馈全连接层。注意这里的第二子层中是与 encoder 段做了 attention 的。此外，第一个 self-attention 子层中使用了 Mask 操作抹掉了未来的信息，使得对当前位置输出的预测仅与已经预测出的序列相关。

Attention

注意力函数可以认为是一个将 query 向量、key-value 向量映射为输出向量的一个操作。这个输出向量是对 values 向量的一个加权和，权重由 query 向量和对应的 key 通过某种复杂的函数运算得到。

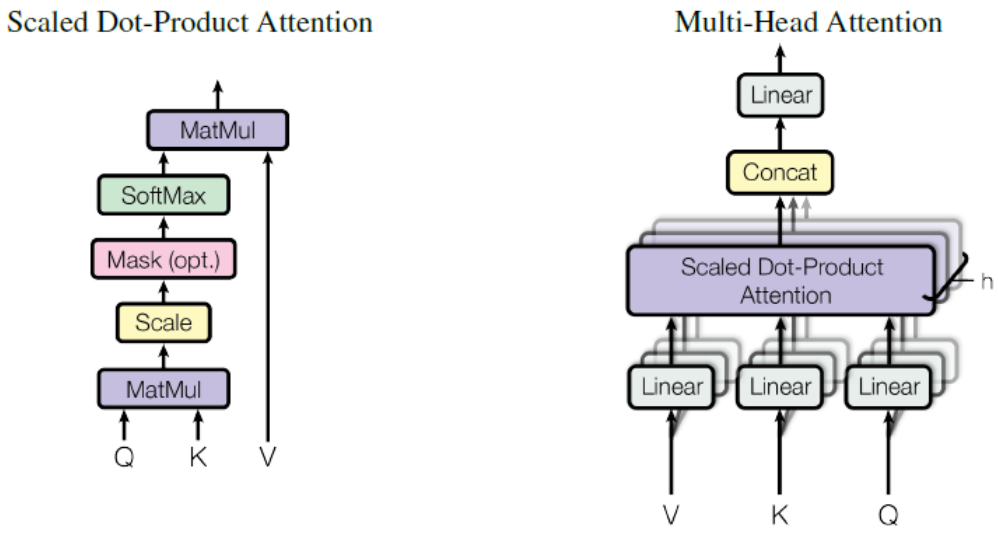
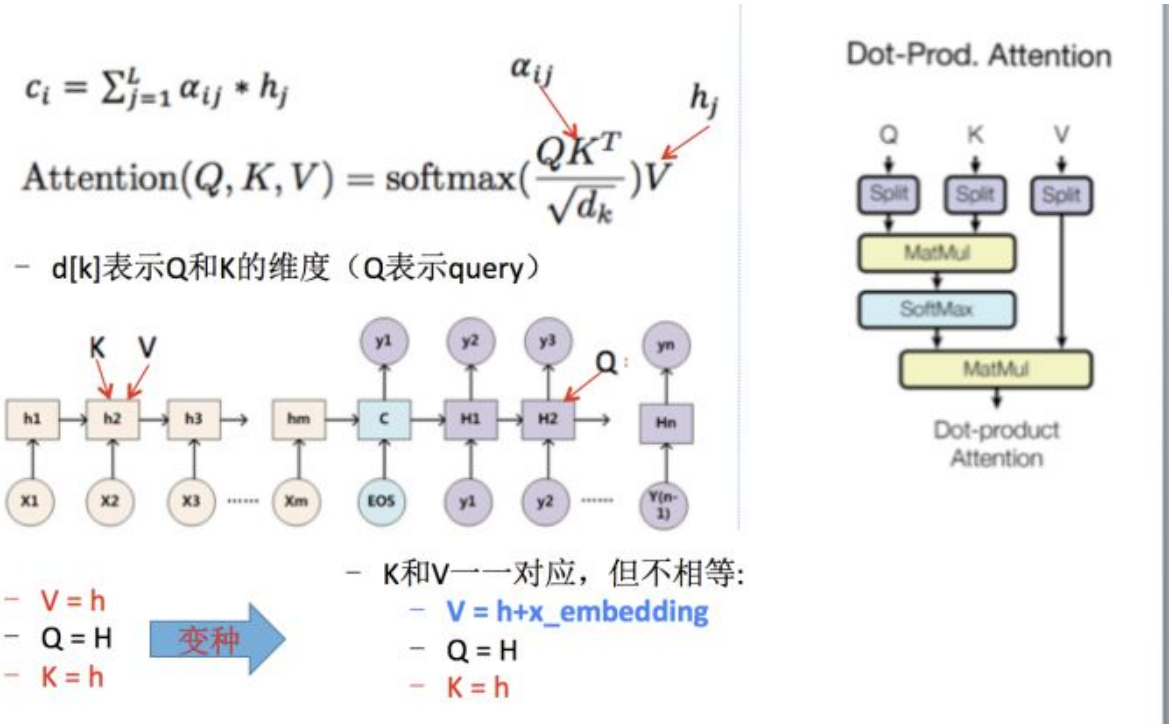


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

这里有一张帮助理解的图，来自知乎。还有另外一个知乎专栏，介绍 Attention 函数的本质。



Scaled Dot-Product Attention

本文提出的这种 Attention 方式，如图 2 的左图示意，点乘是比较基本的做法，多了一个 Scale 操作。输入包括 d_k 维的 query 和 keys，d_v 维的 values，计算 query 和所有 keys 的点乘，然后除以 $\sqrt{d_k}$（体现了 Scaled），通过 softmax 得到每个 value 的权重。 在实际操作中，Attention 是在一批的 queries 上同时进行的，将这些 queries 并在一起形成一个矩阵 Q，keys-values 也并在一起形成了矩阵 K 和 V，那么 Attention 的输出矩阵可以如下计算：

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

这里通过 $\sqrt{d_k}$ 进行了调节，使得内积不至于太大，当 d_k 较大的时候，相比常用的 additive attention，能够避免在 softmax 后带来的值为 0 从而陷入到梯度较小的区域中去。

Multi-Head Attention

这应该是本文的核心，与 CNN 的做法有点像。

因为发现在对 queries、keys 和 values 进行 h 次不同的线性映射后效果更好，并且可以并行运算。从 d_{model} 维分别映射到 d_k, d_k, d_v 维，最终得到的是一个 d_v 维的输出向量。这里 multi 的意思大概就是指的可以从不同表示的子空间，给出不同位置上的信息。

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

这里的 $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$。实验中设置的是 $h = 8$，$d_k = d_v = d_{model}/h = 64$。由于这里的每个"头"的维度比使用一个头的传统做法都要小，又可以并行，所以和使用的单头 Attention 的计算代价是相当的。这些头得到的输出拼接起来，就得到了最终的输出。如图 2 的右图示意。

模型中的 Attention 应用

在本文的 Transformer 模型中有三种不同的方式来使用 multi-head attention。

- 在 encoder-decoder attention 层，queries 来自之前的 decoder 层，keys 和 values 则来自于 encoder 层的输出，使得在 decoder 的每个位置，都可以注意到输入序列的每个位置上的信息。这种做法和典型的序列化 encoder-decoder attention 机制是一致的；
- encoder 层上使用 self-attention 层。在一个 self-attention 层中，keys、values 和 queries 都来自同一个地方，即上一个 encoder 层的输出信息，本层的 encoder 在每个位置，可以通过上一层来注意每个位置的信息；
- decoder 层上使用，也是类似的。不同之处在于，会把某些位置信息设置成 $-\infty$，达到一个 mask 的效果，保证 softmax 之后出现一些非法的连接。（比如说是序列之后的信息不应该对序列之前位置的 attention 有影响，当前位置的 values 只能与之前位置相关）

注：自注意力机制，可以简单理解为 $\text{MultiHead}(X, X, X)$，$X$ 即为输入的序列。

Point-wise Feed-Forward Networks

Transformer 模型中除了attention 层，每一个 encoder、decoder 层中都包含一个全连接前馈网络，在这一堆 6 层中都起到相同的效果。

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

就是进行了两次线性变换，中间加了一个 ReLU 激活函数。

Embedding and Softmax

常规操作，将输入序列的词转换为 d_{model} 维的向量；在预测时，也是通过 softmax 来找概率最大的下一个词。一个 Trick，Embedding 层的权重乘上 $\sqrt{d_{model}}$。

Positional Encoding

至此，发现 Transformer 并不能很好捕捉到序列本身的顺序信息，即如果打乱句子中单词的顺序，Attention 的结果是不变的。所以需要加入一些序列中的顺序信息。本文中把“序列编码”加到输入的 embedding 和输出端上，这个序列编码也是具有 d_{model} 维数的向量，这里不是向量拼接，而是直接的向量加法。本文中采用了正弦、余弦函数来编码：

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

这里的 pos 表示位置，i 是维数下标。选择这样一个映射方式，是希望能够通过相对的位置信息对信息进行 attention，因为 PE_{pos+k} 能够表示为一个关于 PE_{pos} 的线性函数。（一开始不理解，后来看了一些相关的分析和讨论，指出：
$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta, \cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$）虽然也有相关工作使用别的一些编码方式，通过实验发现效果基本一样，为了能够让模型更好地推断测试集上的序列长度而不是局限于训练时遇到的最长长度，选择了正弦曲线。

这不像是 Trick 一类的操作了，因为这里融合了很重要顺序信息！

为什么 Self-Attention

这一部分通过与 RNN、CNN 结构的模型进行比较，给出了使用 Self-attention 的三个主要因素，分别由表中的三列给出数据：

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. 每一层的计算复杂度；
2. 可并行化执行程度，由最少必要序列上的操作数量给出。Self-Attention 层在一个序列上的操作是常数级别的，而 RNN 是 $O(n)$ 的，也因此 self-attention 要比 RNN 快。
3. 长距离依赖的路径长度，事实上这是很多序列建模问题的关键性挑战。Self-attention 将序列两两比较，虽然每一层计算的代价变为 n^2 ，可是能够捕捉到全局的联系。

为了解决特别长的序列问题，self-attention 中可以加上一些限制，只考虑输入位置上宽度为 r 的一个窗口，最大的路径长度增加为 $O(n/r)$，如何进行改进是未来的工作了。

Self-Attention + MultiHead 的优势还在于相对于 RNN、CNN 模型具有更好的可解释性，在本文的附录中给出了一些 case study，可以了解一下。

模型实验

实验设置

主要进行了机器翻译相关的实验，成为新的 the-state-of-the-art。数据集采用的是 WMT-2014 的英-德、英-法数据集，实际实验是在 8 块 NVIDIA P100 GPU 上跑的，一个比较小的模型中，每一个训练 step 耗时 0.4s，12h 完成训练，大的模型则是 3.5 days。实验采用了 Adam 优化器，也比较常规，加入了一些正则项 Tricks：

- 残差 Dropout，$P_{drop} = 0.1$
- 标签平滑策略，允许一定的扰动，虽然模型缺少了一定的确定性，但确实提高了 BLEU

实验结果

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

比当前最好的 ConvS2S 集成模型 BLEU 值在英-德上提高的较明显，英-法在较大的模型中才有提升，其他情况下，小的模型性能有所下降，大的模型才超过 ConvS2S，但是从花费的时间上来说，降低为原来的 1/10，还是能够看好的。

模型分析

通过对模型中一些超参的修改，观察模型性能的变化，设计 Multihead 数量、d_k 的设置、模型参数规模、dropout、标签平滑的扰动值等。

- 文中指出，Multihead 如果只使用 1 个，那么 BLEU 将会有 0.9 的下降，而如果太多，也没有性能的进一步提升，所以这里默认值设置为 8；
- d_k 对模型的性能有较大的影响，大一些则好一点，这也意味着简单的点乘可能并不是十分合适的做法；
- 总的来说，越大的参数规模模型的性能会好一些；
- dropout 和标签平滑都起到了缓解过拟合的问题；
- 使用其他的位置信息 embedding，基本上没有太大的性能影响

英文短语成分分析

为了证明 Transformer 可以推广在其他的任务中，这里选择了英文成分句法分析。现有这方面的研究，使用 RNN 来做序列化的标注（这里输出的序列将比输入序列更长，且具有严格的规则限制），在较小规模的数据集上效果并不好。这里通过对 Transformer 模型的参数调整，使用 4 层的结构，在 Wall Street Journal 数据集上取得了不错的成绩，可与一些 RNN 模型媲美。

总结与点评

本文一经推出即受到热宠，在许多任务中都可以融合 self-attention、multihead 的思想。本文最大的贡献在于抛弃了 CNN 以及 RNN 做机器翻译的固有思维，仅用了 attention 来对句子进行编码和解码，同时提升了模型的效率和性能。从实验结果中来看，似乎能够对句子的成分结构、语义有一定程度上的“理解”。作者也提到，这项工作必将对神经网络的有着里程碑式的启发，又将催生出一种新的套路。

我读完这篇文章，感觉所谓的 multihead、self-attention 透露出 CNN 的味道，multihead 就像是 CNN 的多个卷积核，最后还做一次拼接，这里面用到的残差网络的方法，也是来源于 CNN 的。在处理序列化顺序信息上，引入了位置编码的方式，但感觉还是会有些生硬。就机器翻译这一任务来说，语序并不是十分强调的，更多的语序其实被蕴含在了句子的语法信息中。如果用来做序列化标注，不知道会怎么样，是不是可以避开也不得而知。

本文的口气很大，过分强调了 ”Attention“，标题给人一种霸气的感觉，颇有大象无形的意思。我认为应该还会有很多工作会把这一种机制融合到 RNN、CNN 中，不太可能会真的只用 attention。以推荐系统视角下，用户的网页浏览记录顺序来说，通过 self-attention 或许可以找到不同商品之间的依赖关系，也是可以尝试用来预测下一个物品的，这和 RNN 或者是 seq2seq 是一致的。