【Transformer】关于Transformer，面试官们都怎么问

机器学习算法与自然语言处理 2020-04-06

以下文章来源于NewBeeNLP ，作者Adherer

NewBeeNLP
永远有料，永远有趣

公众号关注“ML_NLP”
设为“星标”，重磅干货，第一时间送达！

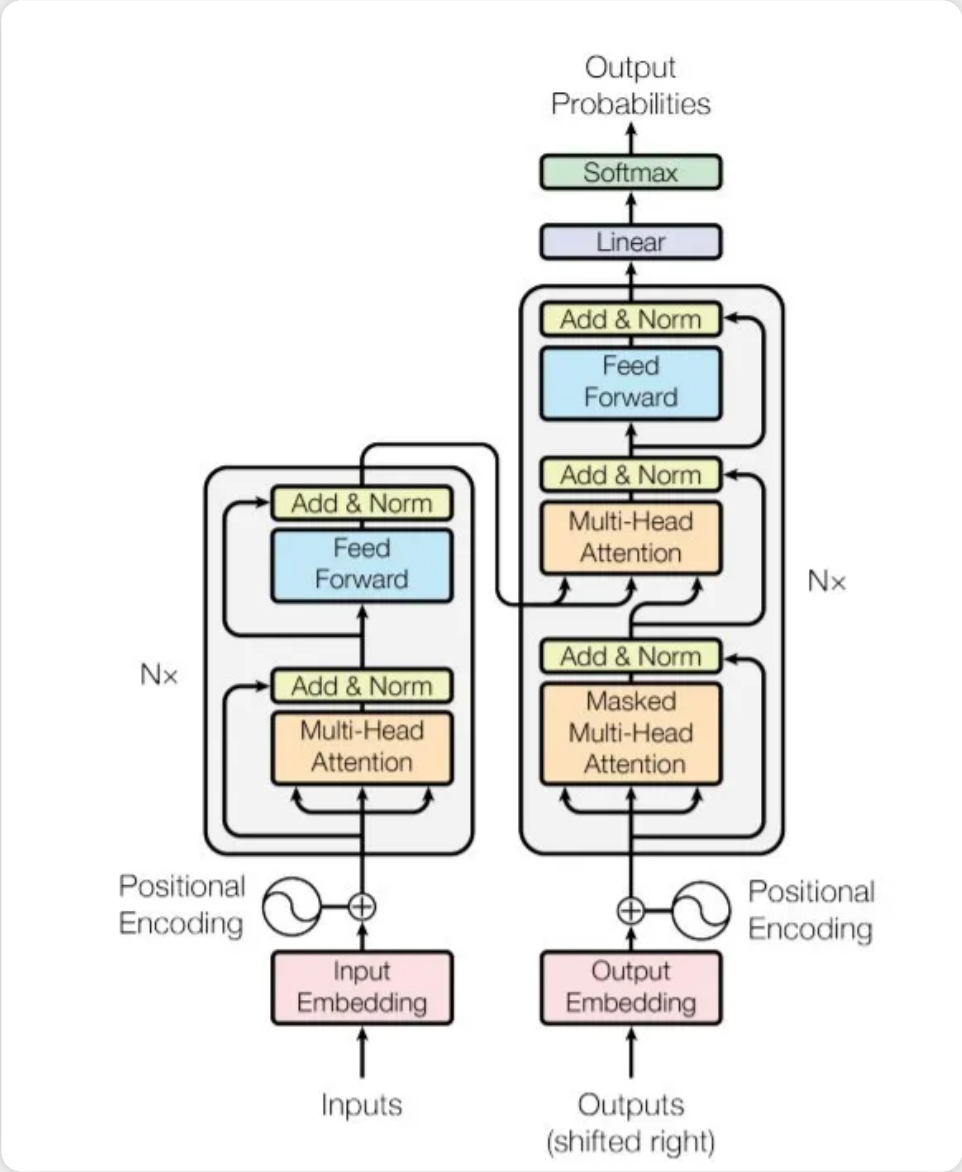
> 作者 | Adherer
> 编辑 | NewBeeNLP

面试锦囊之知识整理系列，持续更新中

写在前面

前些时间，赶完论文，开始对 Transformer、GPT、Bert 系列论文来进行仔仔细细的研读，然后顺手把站内的相关问题整理了一下，但是发现站内鲜有回答仔细的~所以自己就在网上针对每个问题收集了一些资料，并做了整理，有些问题还写了一些自己的看法，可能会有纰漏，甚至还有错误，还请大家赐教 ☺

模型总览：



Transformer模型总览

1.Transformer 的结构是什么样的?

Transformer 本身还是一个典型的 encoder-decoder 模型，如果从模型层面来看，Transformer 实际上就像一个 seq2seq with attention 的模型，下面大概说明一下 Transformer 的结构以及各个模块的组成。

1.1 Encoder 端 & Decoder 端总览

- Encoder 端由 N(原论文中「**N=6**」)个相同的大模块堆叠而成，其中每个大模块又由「**两个子模块**」构成，这两个子模块分别为多头 self-attention 模块，以及一个前馈神经网络模块；

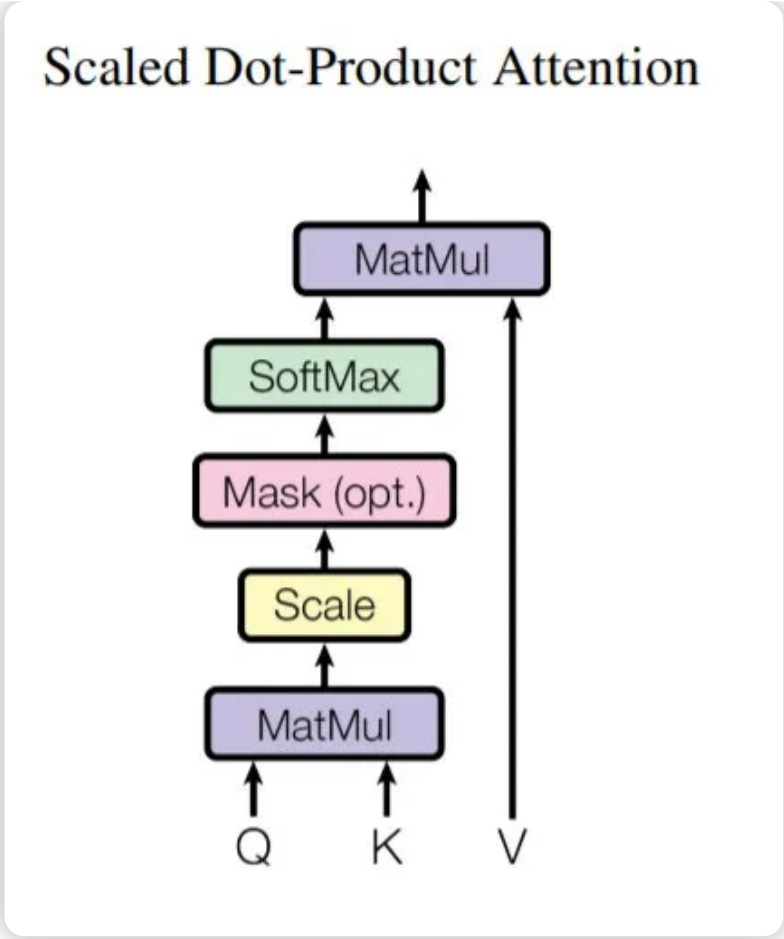
- **「需要注意的是，Encoder 端每个大模块接收的输入是不一样的，第一个大模块(最底下的那个)接收的输入是输入序列的 embedding(embedding 可以通过 word2vec 预训练得来)，其余大模块接收的是其前一个大模块的输出，最后一个模块的输出作为整个 Encoder 端的输出。」**

- Decoder 端同样由 N(原论文中**「N=6」**)个相同的大模块堆叠而成，其中每个大模块则由**「三个子模块」**构成，这三个子模块分别为多头 self-attention 模块，**「多头 Encoder-Decoder attention 交互模块」**，以及一个前馈神经网络模块；
  - 同样需要注意的是，Decoder端每个大模块接收的输入也是不一样的，其中第一个大模块(最底下的那个)训练时和测试时的接收的输入是不一样的，并且每次训练时接收的输入也可能是不一样的(也就是模型总览图示中的"shifted right"，后续会解释)，其余大模块接收的是同样是其前一个大模块的输出，最后一个模块的输出作为整个Decoder端的输出
  - 对于第一个大模块，简而言之，其训练及测试时接收的输入为：
    - 训练的时候每次的输入为上次的输入加上输入序列向后移一位的 ground truth(例如每向后移一位就是一个新的单词，那么则加上其对应的 embedding)，特别地，当 decoder 的 time step 为 1 时(也就是第一次接收输入)，其输入为一个特殊的 token，可能是目标序列开始的 token(如)，也可能是源序列结尾的 token(如)，也可能是其它视任务而定的输入等等，不同源码中可能有微小的差异，其目标则是预测下一个位置的单词(token)是什么，对应到 time step 为 1 时，则是预测目标序列的第一个单词(token)是什么，以此类推；
      - 这里需要注意的是，在实际实现中可能不会这样每次动态的输入，而是一次性把目标序列的embedding通通输入第一个大模块中，然后在多头attention模块对序列进行mask即可
    - 而在测试的时候，是先生成第一个位置的输出，然后有了这个之后，第二次预测时，再将其加入输入序列，以此类推直至预测结束。

1.2 Encoder 端各个子模块

「1.2.1 多头 self-attention 模块」

在介绍 self-attention 模块之前，先介绍 self-attention 模块，图示如下：

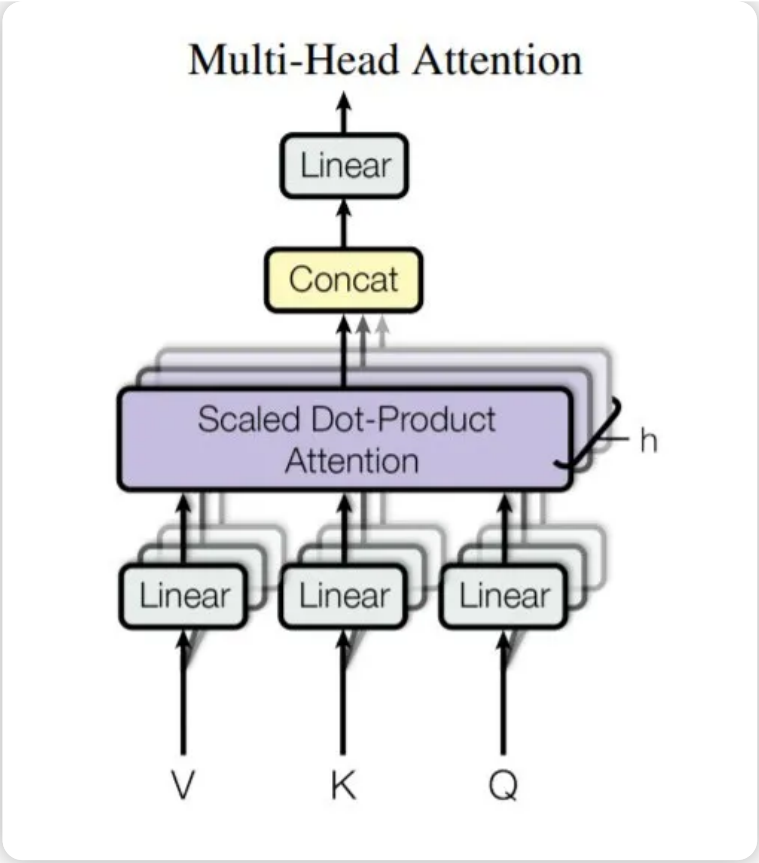


self-attention

上述 attention 可以被描述为**「将 query 和 key-value 键值对的一组集合映射到输出」**，其中 query，keys，values 和输出都是向量，其中 query 和 keys 的维度均为d_k，values 的维度为d_v(论文中$d_k = d_v = d_{\text{model}}/h = 64$)，输出被计算为 values 的加权和，其中分配给每个 value 的权重由 query 与对应 key 的相似性函数计算得来。这种 attention 的形式被称为“Scaled Dot-Product Attention”，对应到公式的形式为：

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

而多头 self-attention 模块，则是将Q, K, V通过参数矩阵映射后(给Q, K, V分别接一个全连接层)，然后再做 self-attention，将这个过程重复h(原论文中$h = 8$)次，最后再将所有的结果拼接起来，再送入一个全连接层即可，图示如下：



multi-head attention

对应到公式的形式为：

$$MultiHead(Q,K,V) = Concat(head_1,\ldots,head_{\mathrm{h}})W^O$$
$$\text{where head}_{\mathrm{i}} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

其中$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

「1.2.2 前馈神经网络模块」

前馈神经网络模块(即图示中的 Feed Forward)由两个线性变换组成，中间有一个 ReLU 激活函数，对应到公式的形式为：

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

论文中前馈神经网络模块输入和输出的维度均为$d_{\text{model}} = 512$，其内层的维度$d_{ff} = 2048.$

1.3 Decoder 端各个子模块

「1.3.1 多头 self-attention 模块」

Decoder 端多头 self-attention 模块与 Encoder 端的一致，但是**「需要注意的是 Decoder 端的多头 self-attention 需要做 mask，因为它在预测时，是“看不到未来的**

序列的”，所以要将当前预测的单词(token)及其之后的单词(token)全部 mask 掉。」

「1.3.2 多头 Encoder-Decoder attention 交互模块」

多头 Encoder-Decoder attention 交互模块的形式与多头 self-attention 模块一致，唯一不同的是其Q, K, V矩阵的来源，其Q矩阵来源于下面子模块的输出(对应到图中即为 masked 多头 self-attention 模块经过 Add & Norm 后的输出)，而K, V矩阵则来源于整个 Encoder 端的输出，仔细想想其实可以发现，这里的交互模块就跟 seq2seq with attention 中的机制一样，目的就在于让 Decoder 端的单词(token)给予 Encoder 端对应的单词(token) “更多的关注(attention weight)”

「1.3.3 前馈神经网络模块」

该部分与 Encoder 端的一致

1.4 其他模块

「1.4.1 Add & Norm 模块」

Add & Norm 模块接在 Encoder 端和 Decoder 端每个子模块的后面，其中 Add 表示残差连接，Norm 表示 LayerNorm，残差连接来源于论文Deep Residual Learning for Image Recognition[1]，LayerNorm 来源于论文Layer Normalization[2]，因此 Encoder 端和 Decoder 端每个子模块实际的输出为：LayerNorm $(x+\operatorname{Sublayer}(x))$，其中Sublayer $(x))$为子模块的输出。

「1.4.2 Positional Encoding」

Positional Encoding 添加到 Encoder 端和 Decoder 端最底部的输入 embedding。Positional Encoding 具有与 embedding 相同的维度d_{model}

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

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

，因此可以对两者进行求和。

具体做法是使用不同频率的正弦和余弦函数，公式如下：

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

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

其中pos为位置，i为维度，之所以选择这个函数，是因为任意位置PE_{pos+k}可以表示为PE_{pos}的线性函数，这个主要是三角函数的特性：

$$\sin(\alpha+\beta)=\sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta)$$
$$\cos(\alpha+\beta)=\cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta)$$

需要注意的是，Transformer 中的 Positional Encoding 不是通过网络学习得来的，而是直接通过上述公式计算而来的，论文中也实验了利用网络学习 Positional Encoding，发现结果与上述基本一致，但是论文中选择了正弦和余弦函数版本，**「因为三角公式不受序列长度的限制，也就是可以对 比所遇到序列的更长的序列 进行表示。」**

2.Transformer Decoder 端的输入具体是什么?

见上述 Encoder 端 & Decoder 端总览中，对 Decoder 端的输入有详细的分析

3.Transformer 中一直强调的 self-attention 是什么? self-attention 的计算过程? 为什么它能发挥如此大的作用? self-attention 为什么要使用 Q、K、V，仅仅使用 Q、V/K、V 或者 V 为什么不行?

3.1 self-attention是什么?

「self-attention」，也叫 **「intra-attention」**，是一种通过自身和自身相关联的 attention 机制，从而得到一个更好的 representation 来表达自身，self-attention 可以看成一般 attention 的一种特殊情况。在 self-attention 中，$Q=K=V$，序列中的每个单词(token)和该序列中其余单词(token)进行 attention 计算。self-attention 的特点在于**「无视词(token)之间的距离直接计算依赖关系，从而能够学习到序列的内部结构」**，实现起来也比较简单，值得注意的是，在后续一些论文中，self-attention 可以当成一个层和 RNN，CNN 等配合使用，并且成功应用到其他 NLP 任务。

3.2 关于 self-attention 的计算过程?

问题 1 中有详细的解答

3.3 关于 self-attention 为什么它能发挥如此大的作用

在上述 self-attention 的介绍中实际上也有所提及，self-attention 是一种自身和自身相关联的 attention 机制，这样能够得到一个更好的 representation 来表达自身，在多数

情况下，自然会对下游任务有一定的促进作用，但是 Transformer 效果显著及其强大的特征抽取能力是否完全归功于其 self-attention 模块，还是存在一定争议的，参见论文：How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures[3]，如下例子可以大概探知 self-attention 的效果：

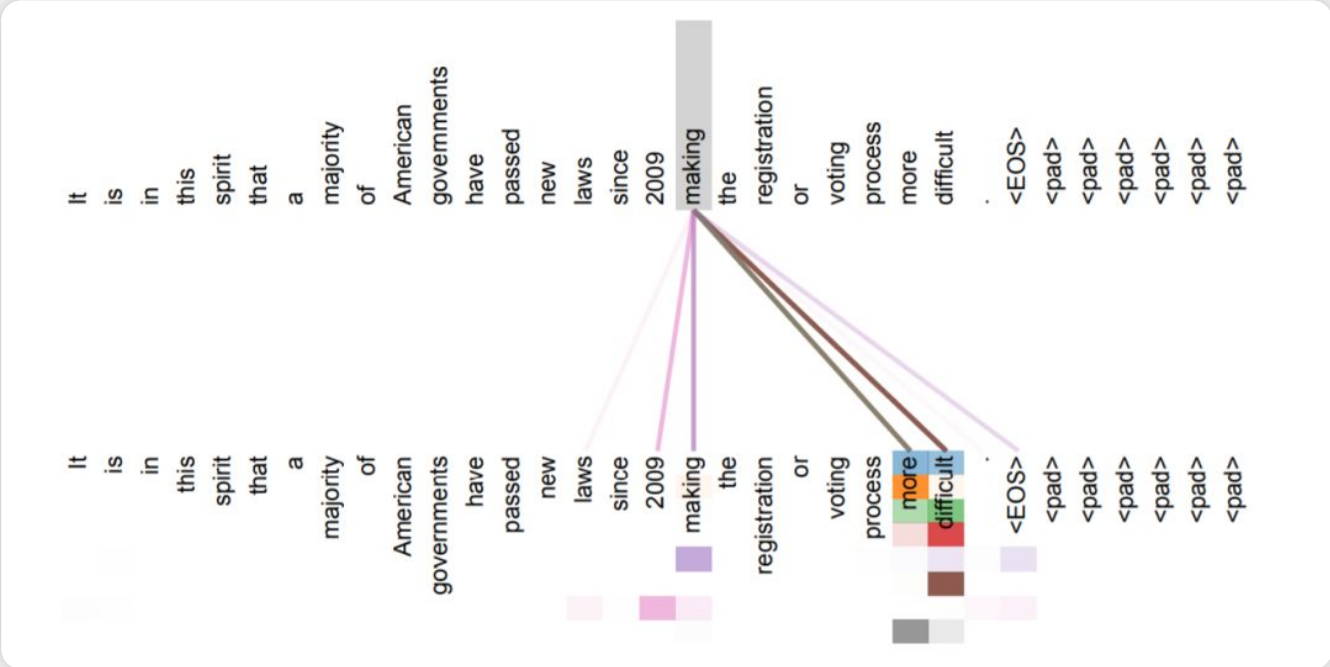


图 1 可视化 self-attention 实例

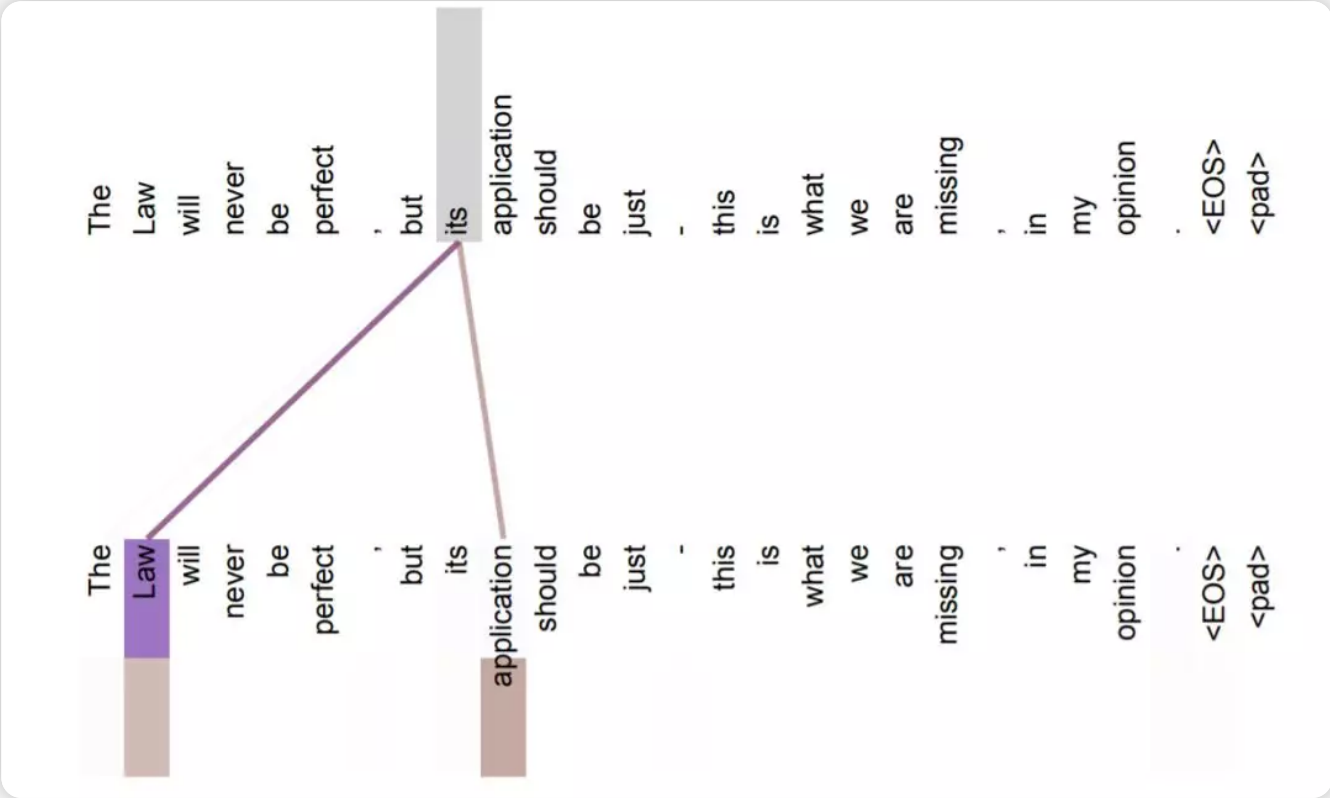


图 2 可视化 self-attention 实例

从两张图（图 1、图 2）可以看出，self-attention 可以捕获同一个句子中单词之间的一些句法特征（比如图 1 展示的有一定距离的短语结构）或者语义特征（比如图 1 展示的 its 的指代对象 Law）。

很明显，引入 Self Attention 后会更容易捕获句子中长距离的相互依赖的特征，因为如果是 RNN 或者 LSTM，需要依次序序列计算，对于远距离的相互依赖的特征，要经过若干时间步步骤的信息累积才能将两者联系起来，而距离越远，有效捕获的可能性越小。

但是 Self Attention 在计算过程中会直接将句子中任意两个单词的联系通过一个计算步骤直接联系起来，所以远距离依赖特征之间的距离被极大缩短，有利于有效地利用这些特征。除此外，Self Attention 对于增加计算的并行性也有直接帮助作用。这是为何 Self Attention 逐渐被广泛使用的主要原因。

3.4 关于 self-attention 为什么要使用 Q、K、V，仅仅使用 Q、V/K、V 或者 V 为什么不行?

这个问题我觉得并不重要，self-attention 使用 Q、K、V，这样三个参数独立，模型的表达能力和灵活性显然会比只用 Q、V 或者只用 V 要好些，当然主流 attention 的做法还有很多种，比如说 seq2seq with attention 也就只有 hidden state 来做相似性的计算，处理不同的任务，attention 的做法有细微的不同，但是主体思想还是一致的，不知道有没有论文对这个问题有过细究，有空去查查~

「其实还有个小细节，因为 self-attention 的范围是包括自身的(masked self-attention 也是一样)，因此至少是要采用 Q、V 或者 K、V 的形式，而这样“询问式”的 attention 方式，个人感觉 Q、K、V 显然合理一些。」

4.Transformer 为什么需要进行 Multi-head Attention? 这样做有什么好处? Multi-head Attention 的计算过程? 各方论文的观点是什么?

4.1 Why Multi-head Attention

原论文中说到进行 Multi-head Attention 的原因是将模型分为多个头，形成多个子空间，可以让模型去关注不同方面的信息，最后再将各个方面的信息综合起来。其实直观上也可以想到，如果自己设计这样的一个模型，必然也不会只做一次 attention，多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用**「多个卷积核」**的作用，直观上讲，多头的注意力**「有助于网络捕捉到更丰富的特征/信息」**。

4.2 关于 Multi-head Attention 的计算过程

在 1 中也有详细的介绍，但是需要注意的是，论文中并没有对 Multi-head Attention 有很强的理论说明，因此后续有不少论文对 Multi-head Attention 机制都有一定的讨论，一

些相关工作的论文如下(还没看，先攒着)

4.3 Multi-head Attention 机制相关的论文：

A Structured Self-attentive Sentence Embedding[4]

Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned[5]

Are Sixteen Heads Really Better than One?[6]

What Does BERT Look At? An Analysis of BERT's Attention[7]

A Multiscale Visualization of Attention in the Transformer Model[8]

Improving Deep Transformer with Depth-Scaled Initialization and Merged Attention[9]

5.Transformer 相比于 RNN/LSTM，有什么优势？为什么？

5.1 RNN 系列的模型，并行计算能力很差

RNN 系列的模型T时刻隐层状态的计算，依赖两个输入，一个是T时刻的句子输入单词X_t，另一个是$T-1$时刻的隐层状态的输出S_{t-1}，这是最能体现 RNN 本质特征的一点，RNN 的历史信息是通过这个信息传输渠道往后传输的。而 RNN 并行计算的问题就出在这里，因为t时刻的计算依赖$t-1$时刻的隐层计算结果，而$t-1$时刻的计算依赖$t-2$时刻的隐层计算结果，如此下去就形成了所谓的序列依赖关系。

5.2 Transformer 的特征抽取能力比 RNN 系列的模型要好

上述结论是通过一些主流的实验来说明的，并不是严格的理论证明，具体实验对比可以参见：

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[10]

但是值得注意的是，并不是说 Transformer 就能够完全替代 RNN 系列的模型了，任何模型都有其适用范围，同样的，RNN 系列模型在很多任务上还是首选，熟悉各种模型的内部原理，知其然且知其所以然，才能遇到新任务时，快速分析这时候该用什么样的模型，该怎么做好。

6.Transformer 是如何训练的？测试阶段如何进行测试呢？

6.1 训练

Transformer 训练过程与 seq2seq 类似，首先 Encoder 端得到输入的 encoding 表示，并将其输入到 Decoder 端做交互式 attention，之后在 Decoder 端接收其相应的输入(见 1 中有详细分析)，经过多头 self-attention 模块之后，结合 Encoder 端的输出，再经过 FFN，得到 Decoder 端的输出之后，最后经过一个线性全连接层，就可以通过 softmax 来预测下一个单词(token)，然后根据 softmax 多分类的损失函数，将 loss 反向传播即可，所以从整体上来说，Transformer 训练过程就相当于一个有监督的多分类问题。

- 需要注意的是，**「Encoder 端可以并行计算，一次性将输入序列全部 encoding 出来，但 Decoder 端不是一次性把所有单词(token)预测出来的，而是像 seq2seq 一样一个接着一个预测出来的。」**

6.2 测试

而对于测试阶段，其与训练阶段唯一不同的是 Decoder 端最底层的输入，详细分析见问题 1。

7.Transformer 中的 Add & Norm 模块，具体是怎么做的？

见 1 其他模块的叙述，对 Add & Norm 模块有详细的分析

8.为什么说 Transformer 可以代替 seq2seq？

这里用代替这个词略显不妥当，seq2seq 虽已老，但始终还是有其用武之地，seq2seq 最大的问题在于**「将 Encoder 端的所有信息压缩到一个固定长度的向量中」**，并将其作为 Decoder 端首个隐藏状态的输入，来预测 Decoder 端第一个单词(token)的隐藏状态。在输入序列比较长的时候，这样做显然会损失 Encoder 端的很多信息，而且这样一股脑的把该固定向量送入 Decoder 端，Decoder 端不能够关注到其想要关注的信息。

上述两点都是 seq2seq 模型的缺点，后续论文对这两点有所改进，如著名的Neural Machine Translation by Jointly Learning to Align and Translate[11]，虽然确确实实对 seq2seq 模型有了实质性的改进，但是由于主体模型仍然为 RNN(LSTM)系列的模型，因此模型的并行能力还是受限，而 transformer 不但对 seq2seq 模型这两点缺点

有了实质性的改进(多头交互式 attention 模块)，而且还引入了 self-attention 模块，让源序列和目标序列首先“自关联”起来，这样的话，源序列和目标序列自身的 embedding 表示所蕴含的信息更加丰富，而且后续的 FFN 层也增强了模型的表达能力(ACL 2018 会议上有论文对 Self-Attention 和 FFN 等模块都有实验分析，见论文：How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures[12])，并且 Transformer 并行计算的能力是远远超过 seq2seq 系列的模型，因此我认为这是 transformer 优于 seq2seq 模型的地方。

9.Transformer 中句子的 encoder 表示是什么？如何加入词序信息的？

Transformer Encoder 端得到的是整个输入序列的 encoding 表示，其中最重要的是经过了 self-attention 模块，让输入序列的表达更加丰富，而加入词序信息是使用不同频率的正弦和余弦函数，具体见 1 中叙述。

10.Transformer 如何并行化的？

Transformer 的并行化我认为主要体现在 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，在 self-attention 模块，对于某个序列$x_1, x_2, \ldots, x_n$，self-attention 模块可以直接计算$\boldsymbol{x}_i, \boldsymbol{x}_j$的点乘结果，而 RNN 系列的模型就必须按照顺序从x_1计算到x_n。

11.self-attention 公式中的归一化有什么作用？

首先说明做归一化的原因，随着d_k的增大，$q \cdot k$点积后的结果也随之增大，这样会将 softmax 函数推入梯度非常小的区域，使得收敛困难(可能出现梯度消失的情况)

为了说明点积变大的原因，假设q和k的分量是具有均值 0 和方差 1 的独立随机变量，那么它们的点积$q \cdot k = \sum_{i=1}^{d_k} q_i k_i$均值为 0，方差为，因此为了抵消这种影响，我们将点积缩放$\frac{1}{\sqrt{d_k}}$，对于更详细的分析，参见(有空再来总结，哈哈~)：transformer 中的 attention 为什么 scaled?[13]

写在后面

17 年提出的 Transformer 模型，在当时确实引起了很大的轰动，但是到现在事后看来，Transformer 模型也确实能力很强，但是我觉得并不像论文题目说的那样《attention is all you need》，反而我觉得论文最大的贡献在于它第一次做到了在自然语言处理任务中把网络的深度堆叠上去还能取得很好的效果，而机器翻译恰好也是一个目前数据量非常丰富且问题本身难度不大的一个任务了，这样充分发挥了 Transformer 的优势。另外，self-attention 其实并不是 Transformer 的全部，实际上从深度 CNN 网络中借鉴而来的 FFN 可能更加重要。所以，理智看待 Transformer，面对不同的任务，选择最合适自己任务的模型就好了~[14][15][16][17][18][19][20][21][22][23][24

本文参考资料

[1]**Deep Residual Learning for Image Recognition:** https://arxiv.org/abs/1512.03385

[2]**Layer Normalization:** https://arxiv.org/abs/1607.06450

[3]**How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures:** http://aclweb.org/anthology/P18-1167

[4]**A Structured Self-attentive Sentence Embedding:** https://arxiv.org/abs/1703.03130

[5]**Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned:** https://arxiv.org/abs/1905.09418

[6]**Are Sixteen Heads Really Better than One?:** https://arxiv.org/abs/1905.10650

[7]**What Does BERT Look At? An Analysis of BERT's Attention:** https://arxiv.org/abs/1906.04341

[8]**A Multiscale Visualization of Attention in the Transformer Model:** https://arxiv.org/abs/1906.05714

[9]**Improving Deep Transformer with Depth-Scaled Initialization and Merged Attention:** https://arxiv.org/abs/1908.11365

[10]**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:** https://zhuanlan.zhihu.com/p/54743941

[11]**Neural Machine Translation by Jointly Learning to Align and Translate:** https://arxiv.org/abs/1409.0473

[12]**How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures:** http://aclweb.org/anthology/P18-1167

[13]**transformer 中的 attention 为什么 scaled?:** https://www.zhihu.com/question/339723385

[14]**The Illustrated Transformer:** https://jalammar.github.io/illustrated-transformer/