



赛尔笔记 | Transformer及其变种详解

 **忆臻**   
哈尔滨工业大学 计算机科学与技术博士在读

关注他

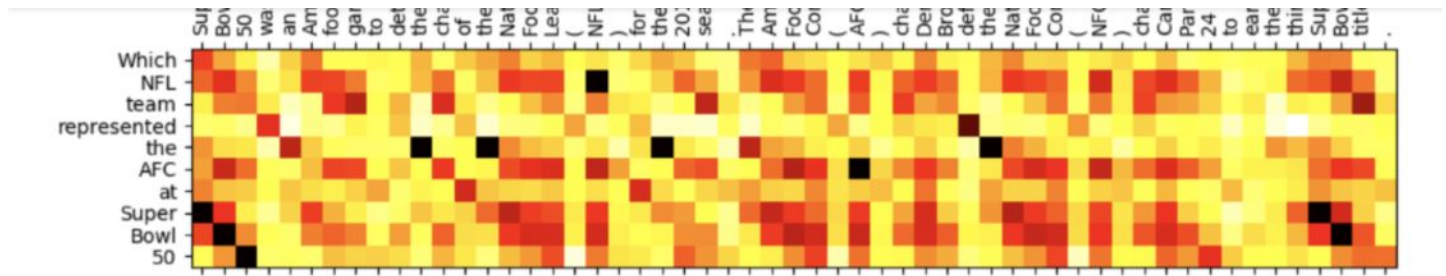
112 人赞同了该文章

来源|哈工大SCIR实验室  
作者：哈工大SCIR 蒋润宇

简介

近年来NLP领域最让人印象深刻的成果，无疑是以谷歌提出的Bert为代表的预训练模型了。它们不断地刷新记录（无论是任务指标上，还是算力需求上），在很多任务上已经能超越人类平均水平，还具有非常良好的可迁移性，以及一定程度的可解释性。

例如，当我们需要在论文里解释为什么算法或者改动能够work的时候，一张基于attention的热力图显然更容易说明我们的代码究竟做到了什么。



而目前主流的预训练模型，都是以17年谷歌提出的Transformer模型作为基础进行修改，作为自己的特征抽取器。可以说，Transformer自从出现以来就彻底改变了深度学习领域，特别是NLP领域。

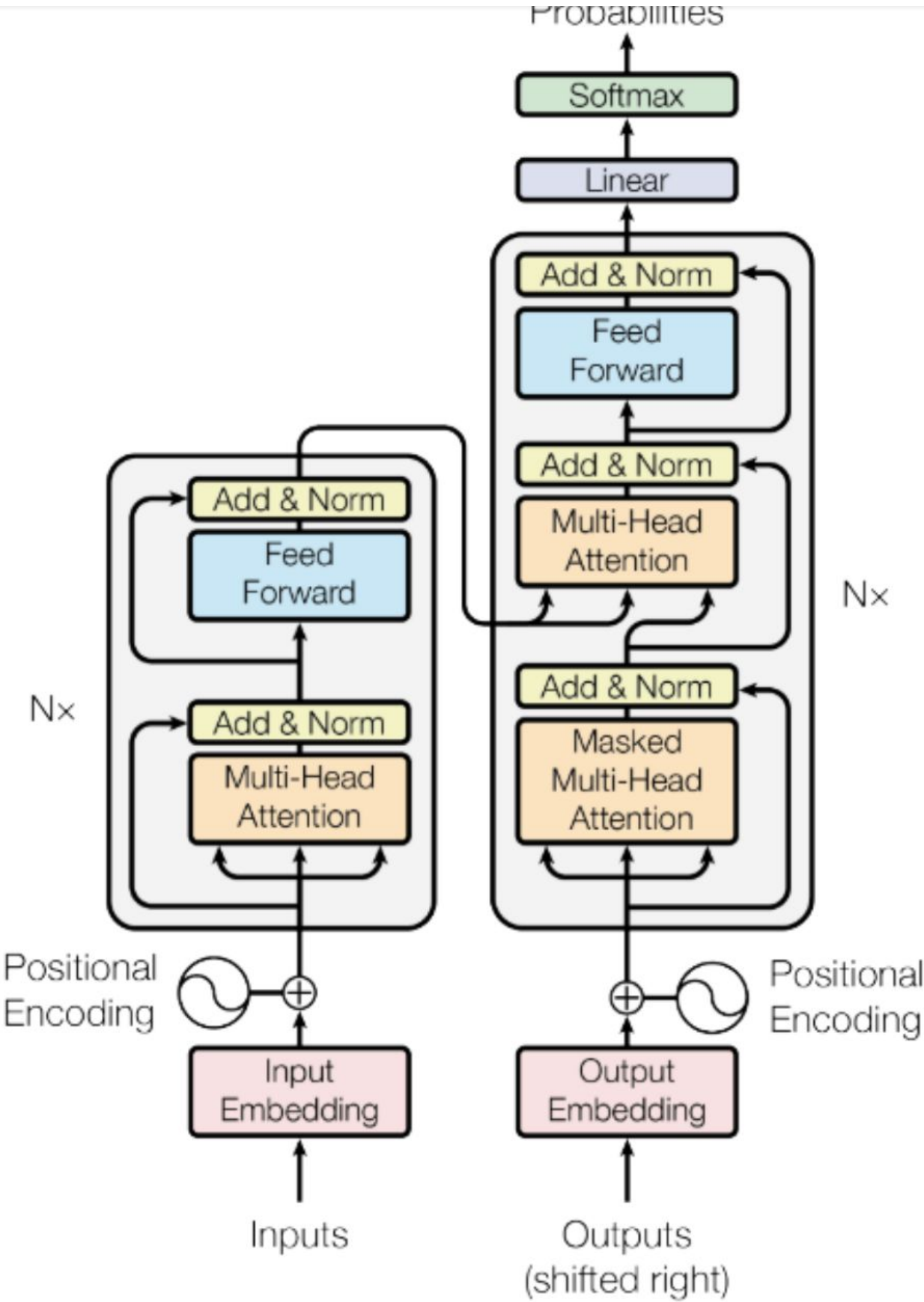
本文主要介绍了Transformer以及其在近年来的一些优化变种。

## Transformer

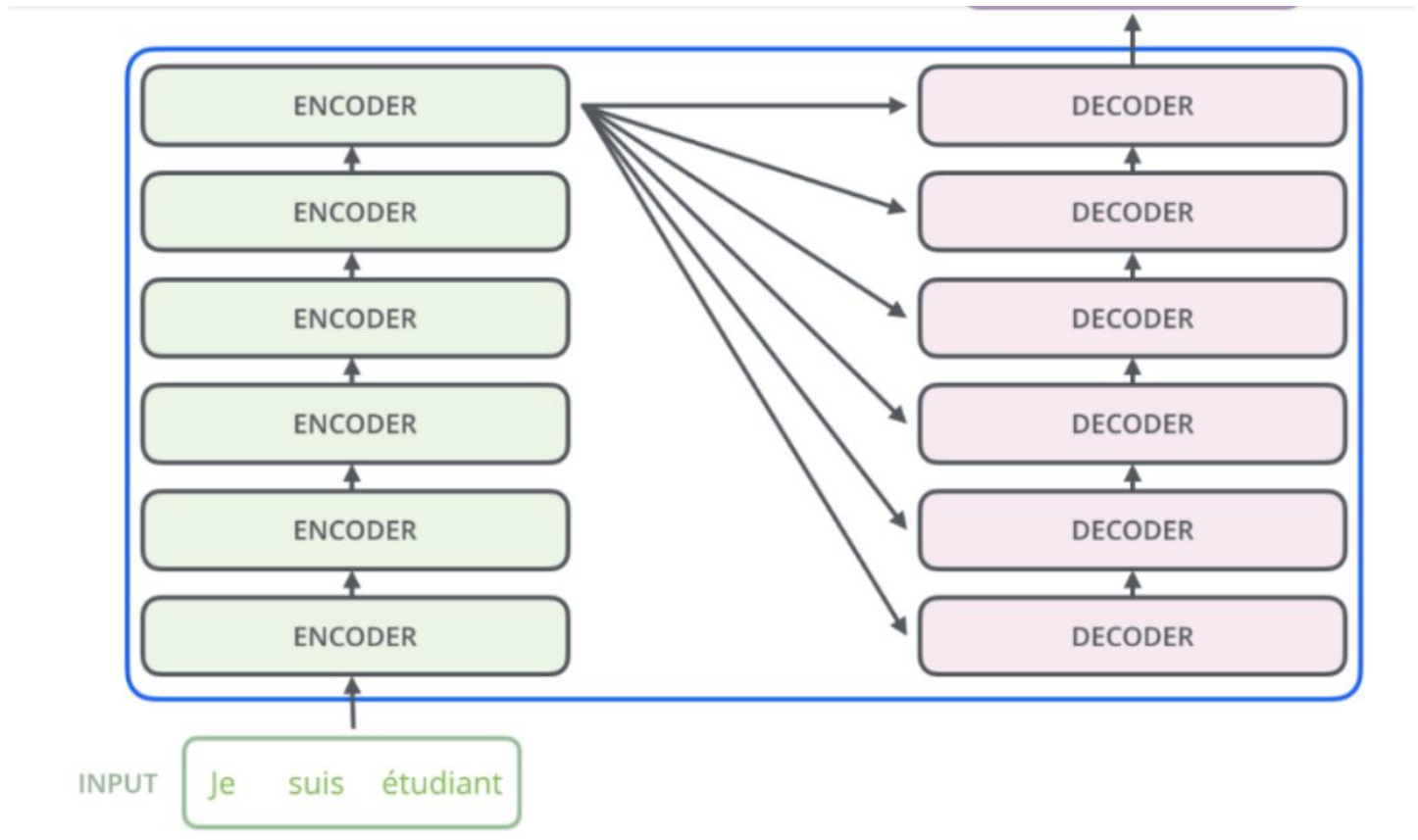
如果用一句话来介绍Transformer，那就是：“首个完全抛弃RNN的recurrence，CNN的convolution，仅用attention来做特征抽取的模型。” 也就是论文标题所写的，《Attention Is All You Need》。

Attention机制在NLP领域的应用最早可以追溯到2014年，Bengio团队将Attention引入NMT(神经机器翻译)任务。但那时Attention仅仅是作为一项外挂结构，模型的核心构架还是RNN。而到了Transformer则完全地以Attention机制作为模型的基础构架，抛弃了之前的CNN和RNN的网络。

Transformer的基本构架如下图所示，其中，左半边是Encoder部分，右半边是Decoder部分。Transformer有6层这样的结构。



以翻译模型为例给出Transformer的总的结构图：



以上是对Transformer的整体介绍，下面将对Transformer各个创新之处进行讲解。

Attention

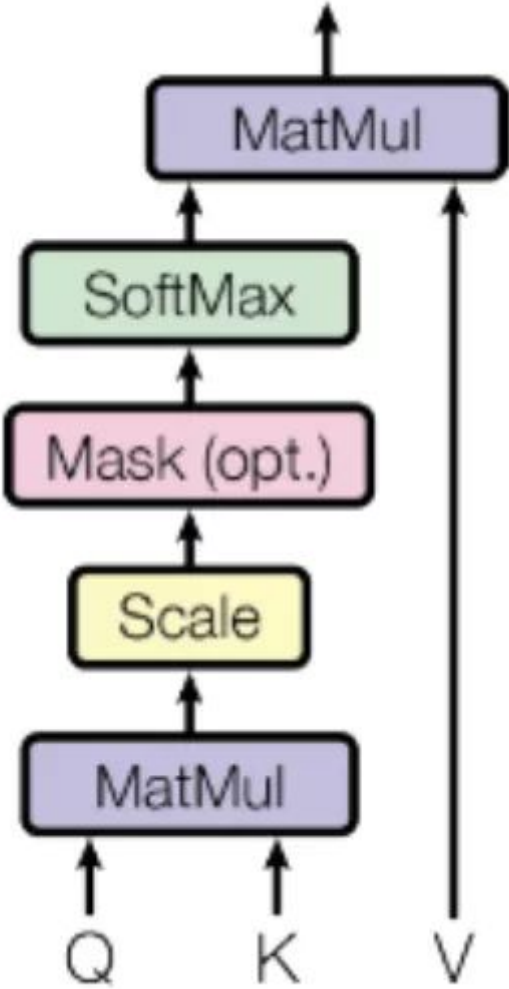
Transformer中一共使用了三次Attention。其中Decoder部分多一层比较特殊的Masked Attention。即在解码时，模型应当只知道当前中心词的上文，因此通过masking的方式，屏蔽中心词下文的内容，保持了自回归的特性。

Scaled Dot-Product Attention

Self-Attention本质上是通过对当前词引入其上下文的信息，以增强对当前词的表示，写入更多的信息。这点基本类似于2014年，Bengio团队将Attention引入NMT(神经机器翻译)。

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

在Transformer中，这部分是通过At value。通过Q和K的点积的结果来体



### Multi-Head Attention

Multi-Head Attention基本是完全创新之处。

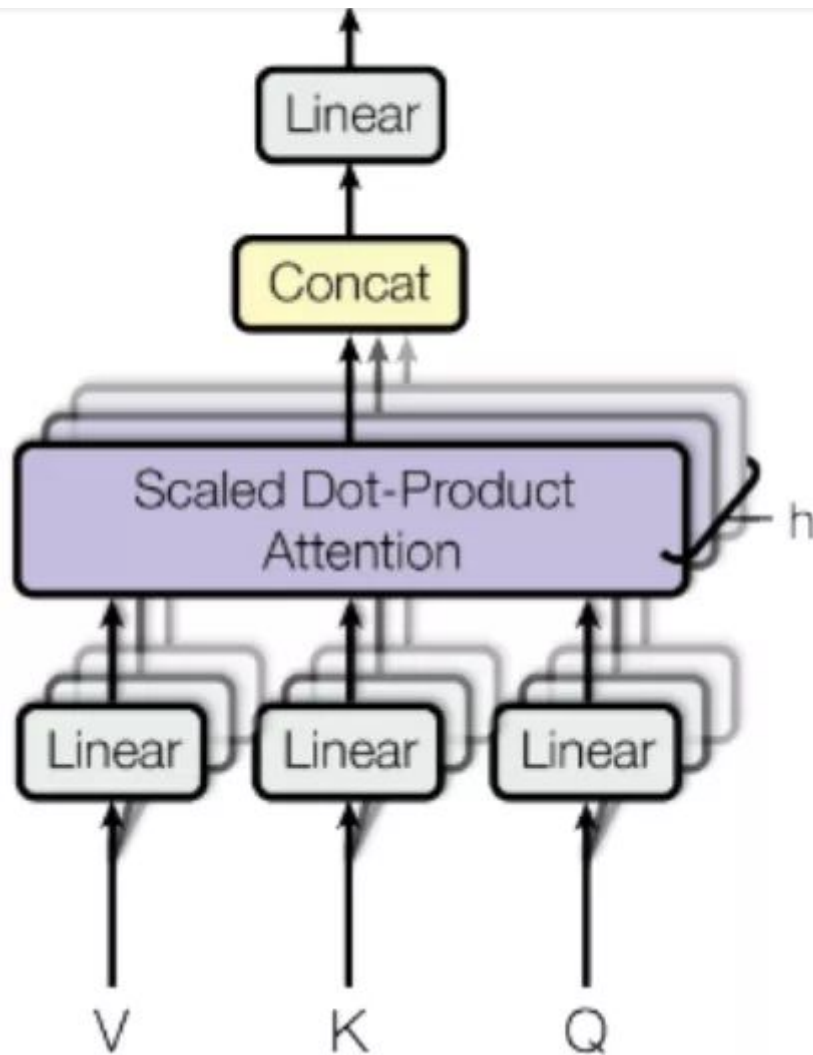


图5 Multi-Head Attention计算路径

其会将原本512维的Q、K、V，通过8次不同的线性投影，得到8组低维的 $Q_i$ 、 $K_i$ 、 $V_i$ ，其维度均为64维。公式如下：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

这样做由于每个注意力头的大小都相应地缩小了，实际上计算量并没有显著增加。

关于为什么使用多头注意力,而不是单头注意力,《Attention Is All You Need》作者认为: 平均注意力加权降低了有效的分辨率, 即它不能充分体现来自不同表示子空间的信息。而使用多头注意力机制有点类似于CNN中同一卷积层内使用多个卷积核的思想。可以增强模型对于文本在不同子空间中体现出的不同的特性, 避免了平均池化对这种特性的抑制。

但是关于多头注意力机制是不是有用

▲ 赞同 112 ▼

● 添加评论

➤ 分享

★ 收藏

...



知乎

首发于  
机器学习算法与自然语言处理

言，不同的头关注点应该也是一样的。这就和作者的解释有些矛盾。

实际上，在《A Multiscale Visualization of Attention in the Transformer Model》这篇文章中，作者分析了前几层BERT的部分注意力头，如下图所示，结果显示，同一层中，总有那么一两个头关注的点和其他的头不太一样，但是剩下的头也相对趋同【考虑到同一层的注意力头都是独立训练的，这点就很神奇】。

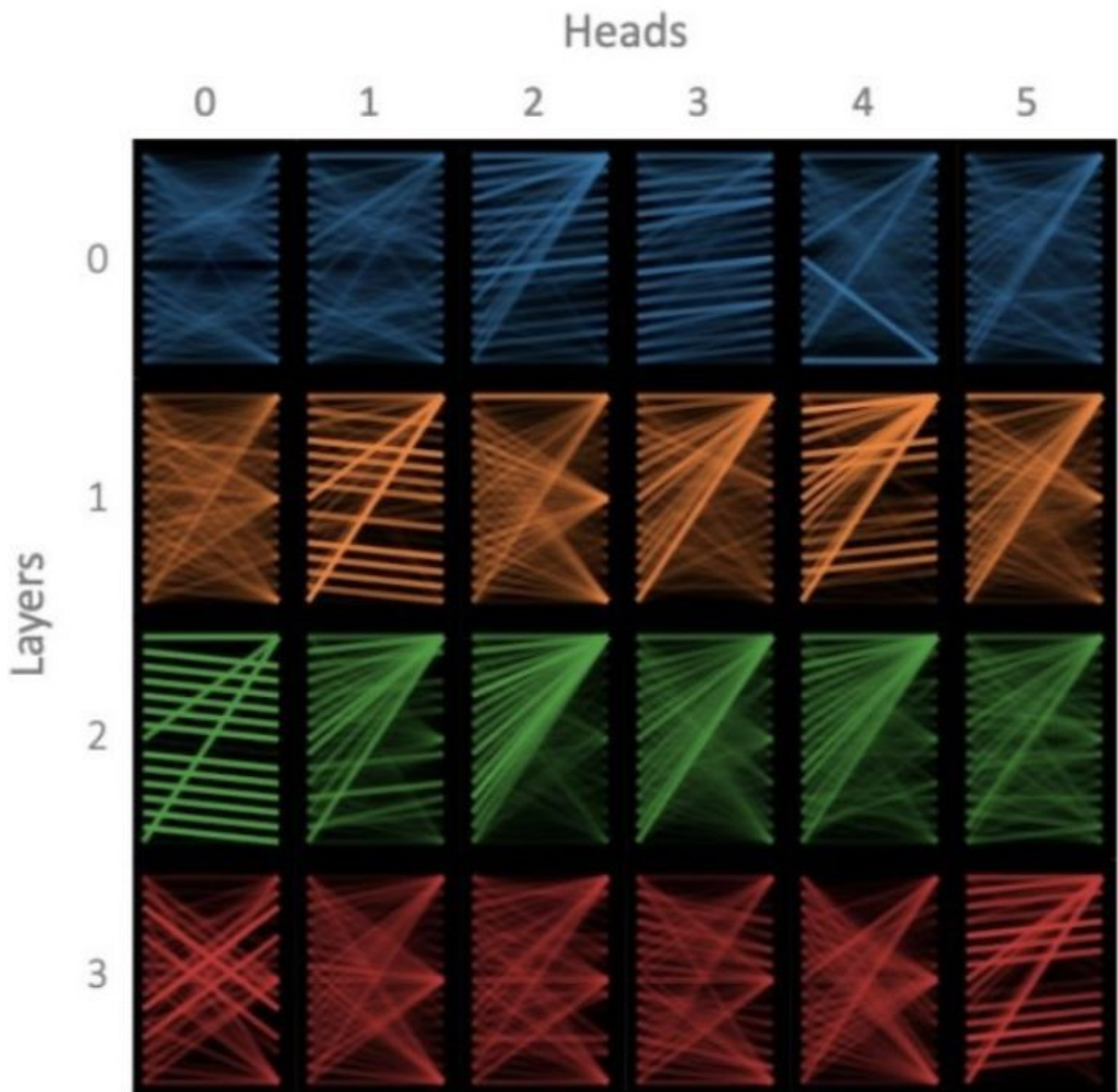


图6 Bert的0-3层中，第0-5的head对于同样的输入所关注的内容。

而在《What Does BERT Look At? An Analysis of BERT's Attention》一文中，作者分析了，同一层中，不同的头之间的差距，以

▲ 赞同 112 ▼

● 添加评论

➤ 分享

★ 收藏

...

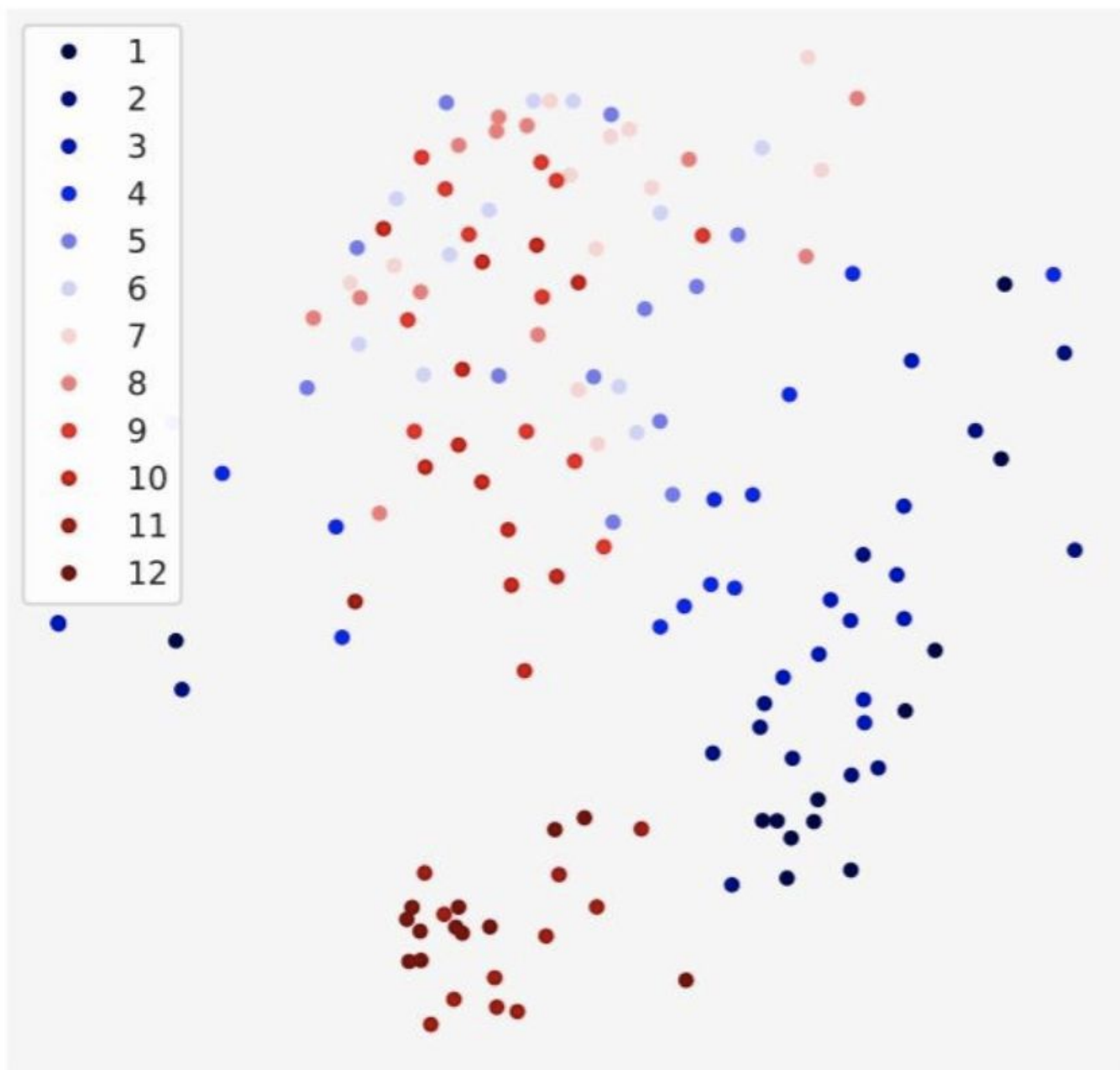


图7 对于Bert中，每一层的head之间的差异在二维平面上的投影

就我个人观点，多头注意力机制的作用可能是这样：注意力机制的冗余性很高（即使是独立计算的注意力头，大概率关注的点还是一致的），所以那些仅有很少部分的相对离群的注意力头，能够使得模型进一步优化。但是这些离群的头出现的概率并不高，因此需要通过提高头的基数，来保证这些离群头的出现频率。

## Positional Encoding

▲ 赞同 112 ▼

● 添加评论

➤ 分享

★ 收藏

...



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

使用上述位置编码的理由很简单，因为它能够很好的编码两个词之间的相对位置关系。三角函数有着非常方便的和差化积公式。

作者还提到了可以使用Learned embedding，但是实验表明两种方法效果上并没有差别，但使用公式方法更为简单，可以处理比训练时更长的序列。

## 缺点

Transformer从现在看来也存在着一些缺点和不足：

非图灵完备：证明略过不表。通俗来说，就是Transformer不能处理所有问题。例如，当我们需要输出直接复制输入时，Transformer并不能很好地学习到这个操作。

不适合处理超长序列：当针对文章处理时，序列的长度很容易就超过512。而如果选择不断增大模型的维度，训练时计算资源的需求会平方级增大，难以承受。因此一般选择将文本直接进行截断，而不考虑其自然文本的分割（例如标点符号等），使得文本的长距离依赖建模质量下降。

计算资源分配对于不同的单词都是相同的：在Encoder的过程中，所有的输入token都具有相同的计算量。但是在句子中，有些单词相对会更重要一些，而有些单词并没有太多意义。为这些单词都赋予相同的计算资源显然是一种浪费。

原始版的Transformer虽然并不成熟，层数固定不够灵活、算力需求过大导致的不适合处理超长序列等缺陷限制了其实际应用前景。但是其优秀的特征抽取能力吸引了很多学者的关注。很多人提出了不同的变种Transformer来改进或者规避它的缺陷。其中，Universal Transformer、Transformer-XL、Reformer就是典型的代表。

## Universal Transformer

从构架来看，Universal Transformer和Transformer没有本质的区别，这里就不详细讲解了，主要谈谈其最大的创新之处。

▲ 赞同 112 ▼

● 添加评论

➤ 分享

★ 收藏

...

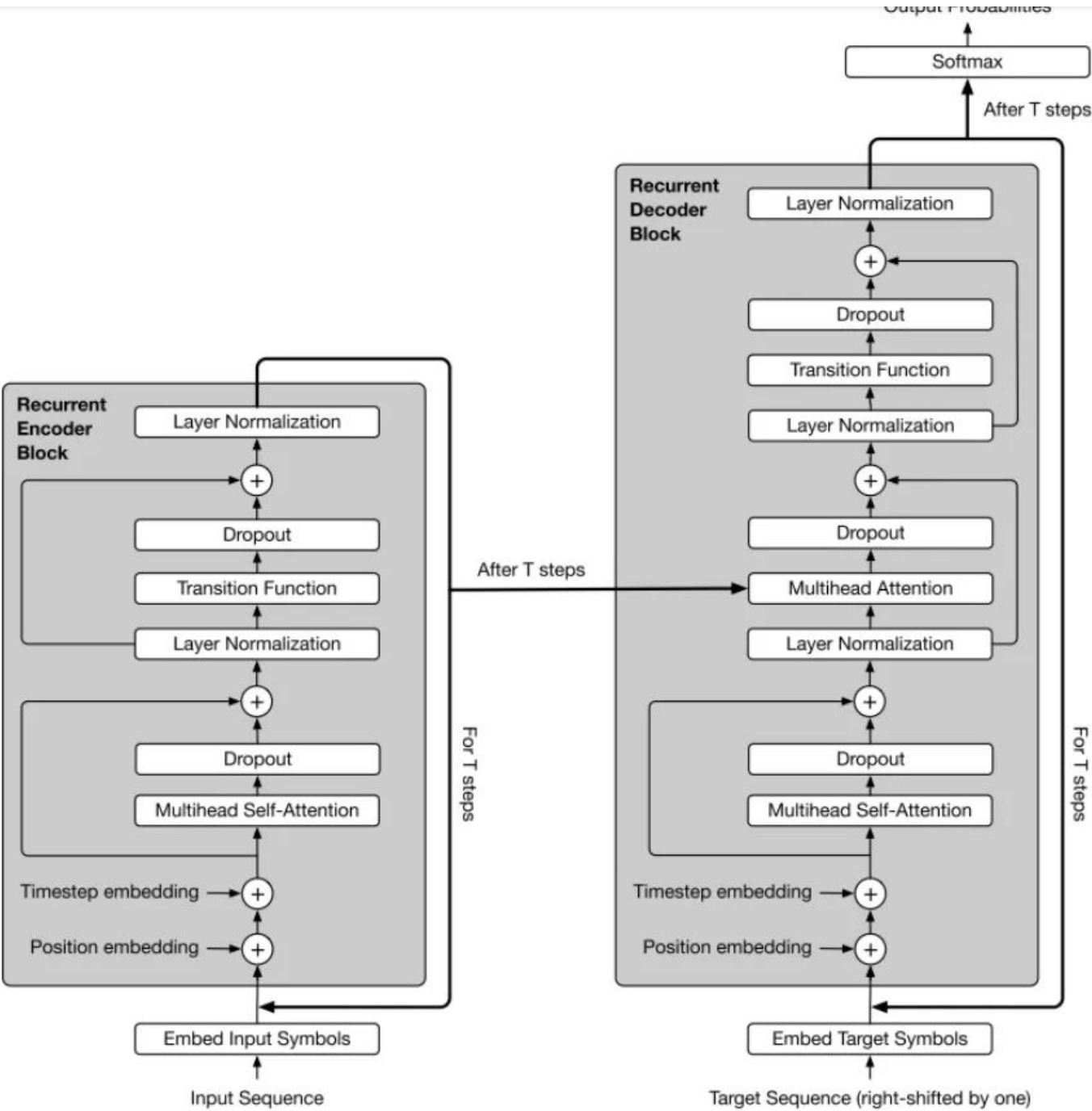


图8 Universal Transformer模型架构

在Transformer中，输入经过Attention后，会进入全连接层进行运算，而Universal Transformer模型则会进入一个共享权重的transition function继续循环计算

知乎

首发于  
机器学习算法与自然语言处理

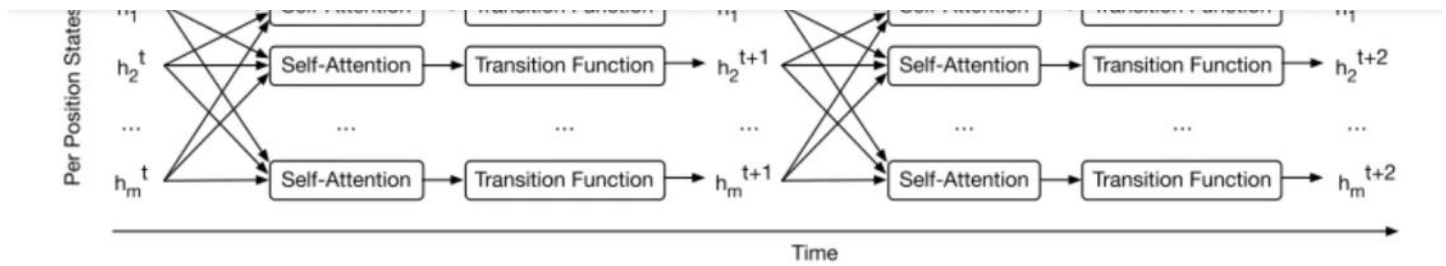


图9 Universal Transformer重新启用了循环机制

其中，纵向看是文本的序列顺序，横向看是时间步骤。其中每一步的计算公式如下：

$$H^t = \text{LAYERNORM}(A^t + \text{TRANSITION}(A^t))$$
$$A^t = \text{LAYERNORM}((H^{t-1} + P^t) + \text{MULTIHEADSELFATTENTION}(H^{t-1} + P^t))$$

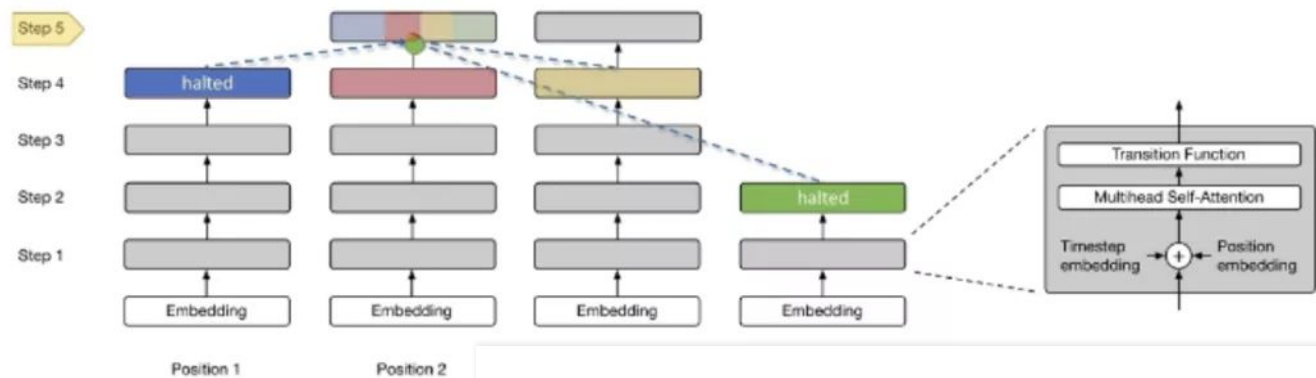
这里Transition function可以和之前一样是全连接层，也可以是其他函数层。

之前Transformer的位置编码因为因为层数是固定的，所以不需要将层数编码进去。大师Universal Transfore模型多了一个时间维度，因此每一次循环都需要进行一轮坐标编码，公式为：

$$P_{i,2j}^t = \sin(i/10000^{2j/d}) + \sin(t/10000^{2j/d})$$
$$P_{i,2j+1}^t = \cos(i/10000^{2j/d}) + \cos(t/10000^{2j/d})$$

为了控制循环的次数，模型引入了Adaptive Computation Time（ACT）机制。

ACT可以调整计算步数，加入ACT机制的Universal transformer被称为Adaptive universal transformer。以下图为例，可以看出，引入ACT机制后，模型对于文本中更重要的token会进行更多次数的循环，而对于相对不重要的单词会减少计算资源的投入。



Universal Transformer对transformer的缺点进行了改进，解决了Transformer非图灵完备的缺点，和计算资源投入平均的问题。

## Transformer-XL

理论上，attention机制可以让Transformer模型捕获任意距离之间的token之间的依赖关系，但是受限于算力问题（下一个介绍的模型就是解决这个问题），Transformer通常会将本文分割成长度小于等于  $d_{model}$ （默认是512）的segment，每个segment之间互相独立，互不干涉。

这也就意味着，segment之间的依赖关系，或者说距离超过512的token之间的依赖关系就完全无法建模抽取。同时，这还会带来一个context fragmentation的问题，因为segment的划分并不是根据语义边界，而是根据长度进行划分的，可能会将一个完整的句子拆成两个。那么在对这样被拆分的句子进行预测时，可能就会缺少必要的语义信息。

而Transformer-XL提出了Segment-level Recurrence来解决这个问题。

用一句话概括Segment-level Recurrence，那就是，在对当前segment进行处理的时候，缓存并利用上一个segment中所有layer的隐向量，而且上一个segment的所有隐向量只参与前向计算，不再进行反向传播。

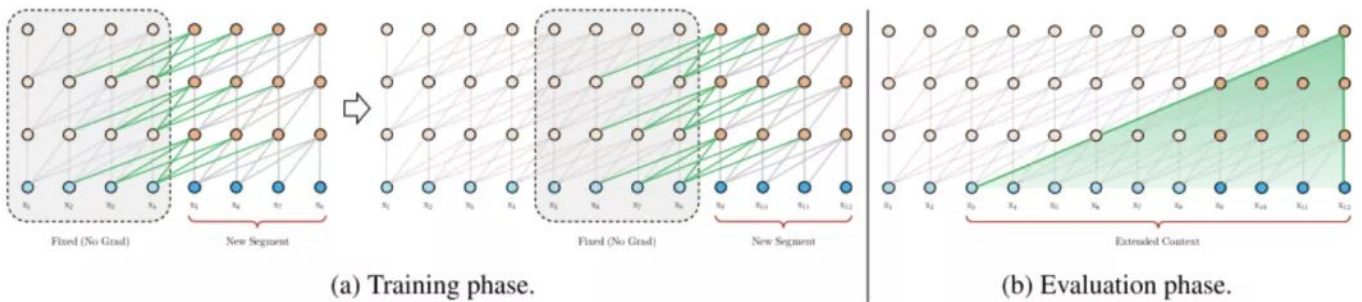


图11 Transformer-XL中，节点能够“看到”之前的segment中的内容

的第 $n$ 层隐变量向量表示为 $h_t^n$ ，其大小为 $L \times d$ ， $d$ 是隐变量向量长度，那么第 $t+1$ 个segment的第 $n$ 层隐变量向量可以由如下公式算出，其中SG指的是stop-gradient，即不对上一个segment的隐变量进行反向传播。

$$\begin{aligned}\tilde{h}_{\tau+1}^{n-1} &= [\text{SG}(h_{\tau}^{n-1}) \circ h_{\tau+1}^{n-1}], \\ \mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n &= h_{\tau+1}^{n-1} \mathbf{W}_q^{\top}, \tilde{h}_{\tau+1}^{n-1} \mathbf{W}_k^{\top}, \tilde{h}_{\tau+1}^{n-1} \mathbf{W}_v^{\top}, \\ h_{\tau+1}^n &= \text{Transformer-Layer}(\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n).\end{aligned}$$

从图中可以看出，在当前segment中，第 $n$ 层的每个隐向量的计算，都是利用下一层中包括当前位置在内的，连续前 $L$ 个长度的隐向量。这也就意味着，每一个位置的隐向量，除了自己的位置，都跟下一层中前 $(L-1)$ 个位置的token存在依赖关系，而且每往下走一层，依赖关系长度会增加 $(L-1)$ 。所以最长的依赖关系长度是 $N(L-1)$ ， $N$ 是模型中layer的数量。在对长文本进行计算的时候，可以缓存上一个segment的隐向量的结果，不必重复计算，大幅提高计算效率。

由于考虑了之前的segment，那么先前的位置编码就不足以区分不同segment之间的相同位置的token，因此作者提出了使用Relative Positional Encoding来替代之前的位置编码。具体来说，就是使用相对位置编码来替代绝对位置编码。这种做法在思想上是很容易理解的，因为在处理序列时，一个token在其中的绝对位置并不重要，我们需要的仅仅是在计算attention时两个token的相对位置。由于这部分工作起到的作用主要是补丁，这里不再展开说。

总结来看。Transformer-XL在没有大幅度提高算力需求的情况下，一定程度上解决了长距离依赖问题。

## Reformer

之所以Transformer的 $d_{model}$ 定为512，而不是更大的值，一个很重要的因素是，在计算attention的过程中，需要计算 $QK^T$ （Multi-head attention并不会减少计算量），这也是为什么Transformer处理长距离依赖问题并不太好的原因之一。另一方面，多层的Transformer对于内存的占用（从只有几层的GB字节到有数千层的模型的TB字节）也限制了Transformer的应用。

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



原始Transformer计算attention的过程中，需要计算 $QK^T$ ，其复杂度为 $O(L^2)$ ，其中L为序列长度。那么为什么需要计算这个 $QK^T$ 呢？是为了找到Q和K中相似的部分。那么应用局部敏感哈希的思想（类似于桶排序的思路），我们可以先将相近的向量先归为一类，然后只计算同类的向量之间的点乘。这样，通过LSH，我们就将计算复杂度降为 $O(L \times \log(L))$ 。

下图展示LSH attention的过程，首先用LSH来对每个segment进行分桶，将相似的部分放在同一个桶里面。然后将每一个桶并行化计算其中两两之间的点乘。

者还考虑到有一定的概率相似的向量会被分到不同的桶里，因此采用了多轮hashing来降低这个概率。

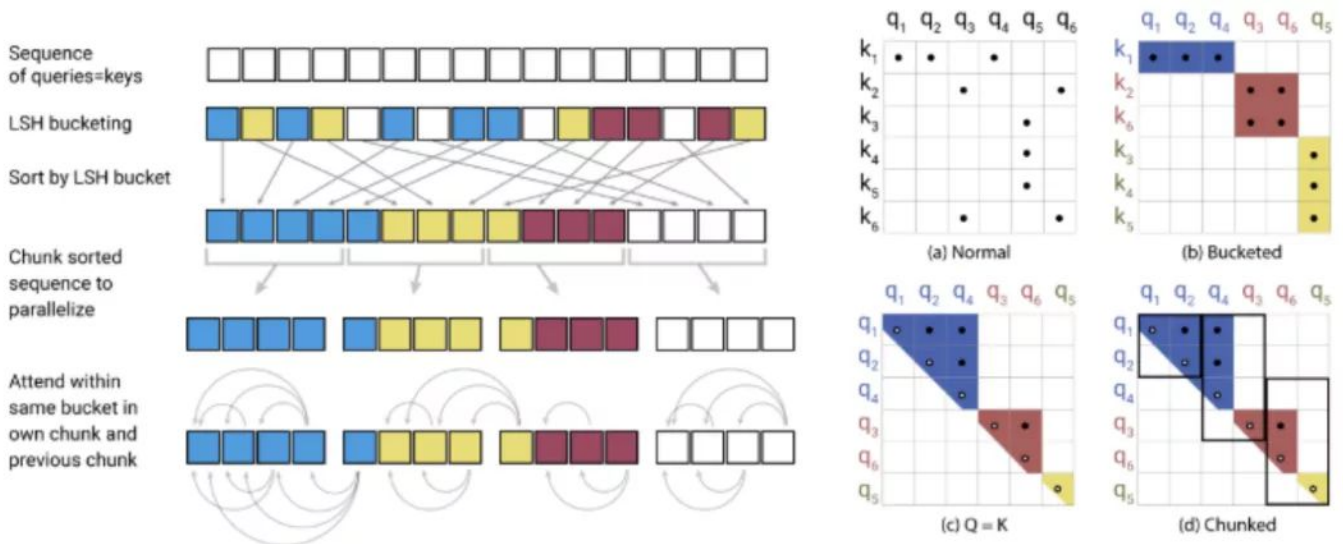


图12 Reformer模型预先使用了hashing筛选，类似桶排序，避免了对QK的计算

LSH解决了计算速度的问题，但仍有一个内存消耗的问题。一个单层网络通常需要占用GB级别的内存，但是当我们训练一个多层模型时，需要保存每一层的激活值和隐变量，以便在反向传播时使用。这极大地提高了内存的占用量。

这里作者借鉴了RevNet的思想，不保留中间残差连接部分的输入了，取而代之的，是应用一种“可逆层”的思路，就如同下图所示的那样，（a）为前向传播，（b）为反向传播。

知乎

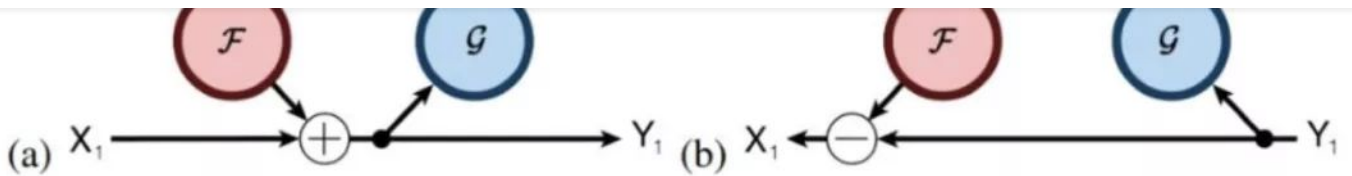
首发于  
机器学习算法与自然语言处理

图13 Reformer中的反向传播时，每一层的输入可以根据其输出计算出来

可逆层对每个层有两组激活。一个遵循正常的标准过程，并从一个层逐步更新到下一个层，但是另一个只捕获对第一个层的更改。因此，要反向运行网络，只需减去应用于每个层的激活。

这意味着不需要缓存任何激活来计算后向传播。类似于使用梯度检查点，虽然仍然需要做一些冗余计算，但由于每一层的输入都可以很容易地从它的输出中构造出来，内存使用不再随网络中层数的增加而增加。

总结来看，Reformer在减少了attention计算量的情况下，还减少了模型的内存占用，为未来大型预训练模型的落地奠定了基础。

## 总结

本文主要介绍了Transformer模型以及针对其缺点作出改进的一些变种模型，总结了它们的设计思路和优缺点。未来，以Transformer及其改进版为基础特征抽取器的预训练模型，一定能够在自然语言处理领域取得更大的突破。

## 参考文献

- [1] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.
- [2] Dehghani M, Gouws S, Vinyals O, et al. Universal transformers[J]. arXiv preprint arXiv:1807.03819, 2018.
- [3] Dai Z, Yang Z, Yang Y, et al. Transformer-xl: Attentive language models beyond a fixed-length context[J]. arXiv preprint arXiv:1901.02860, 2019.
- [4] Kitaev N, Kaiser Ł, Levskaya A. Reformer: The Efficient Transformer[J]. arXiv preprint arXiv:2001.04451, 2020.

▲ 赞同 112 ▼

● 添加评论

➤ 分享

★ 收藏

...

知乎

首发于  
机器学习算法与自然语言处理

[6] Clark K, Khandelwal U, Levy O, et al. What does bert look at? an analysis of bert's attention[J]. arXiv preprint arXiv:1906.04341, 2019.

本期责任编辑：丁 效

本期编辑：顾宇轩

编辑于 06-26

[自然语言处理](#)[深度学习 \(Deep Learning\)](#)[BERT](#)

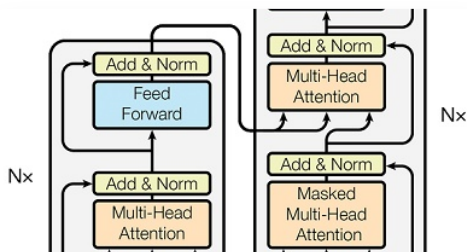
## 文章被以下专栏收录

**机器学习算法与自然语言处理**

公众号[自然语言处理与机器学习] 微信号yizhennotes

[关注专栏](#)

## 推荐阅读



史上最全Transformer面试题  
系列（一）：灵魂20问帮你...

DASOU

发表于NLP基础...



放弃幻想，全面拥抱  
Transformer：自然语言处...

张俊林

发表于深度学习前...



从BERT  
细看NL

Giant

[赞同 112](#)[添加评论](#)[分享](#)[收藏](#)

知乎

首发于  
机器学习算法与自然语言处理

评论由作者筛选后显示

