降低Transformer复杂度的方法

1. Sparse Transformer

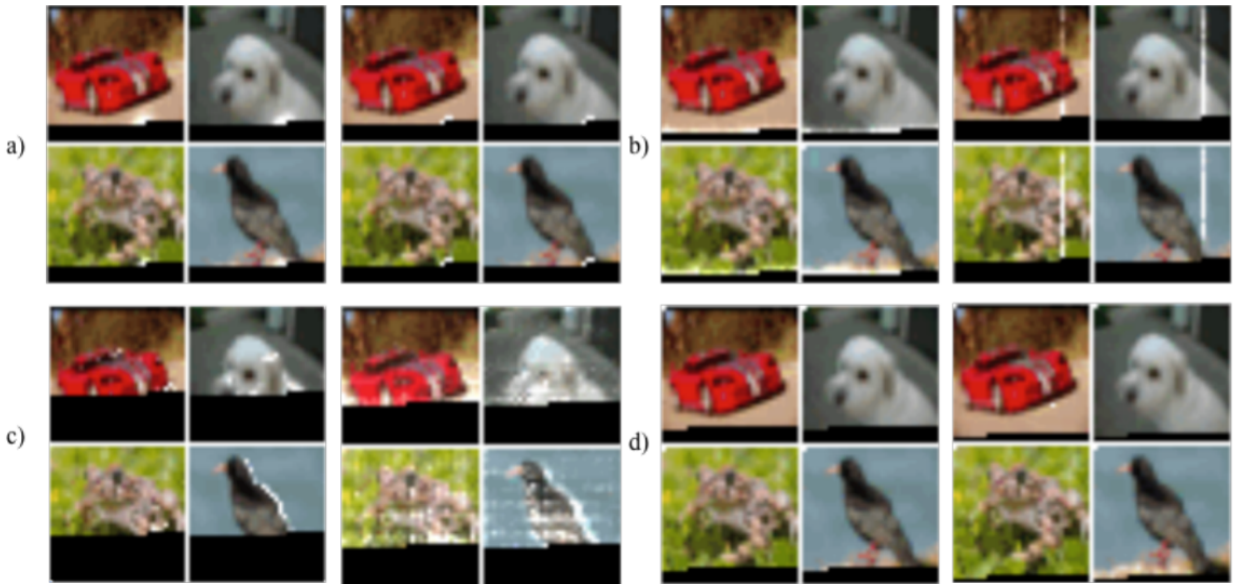
论文：https://arxiv.org/pdf/1904.10509.pdf?ref=https://githubhelp.com

Sparse Attention是为了解决Transformer模型随着长度n的增加，Attention部分所占用的内存和计算呈平方增加的问题。回忆Transformer的复杂度为$O(n^2d)$, 而sparse transformer试图把此复杂度降低为$O(n\sqrt{n}d)$.这样，就可以处理上千长度的输入，层数可以达到上百层。

1.1 Intuition

Transformer的Decoder部分是一个**自回归（AR）**模型。对于图像生成任务，可以把图像的像素点按照从上到下从左到右的方式当成一个序列，然后在序列上去做自回归。

论文中首先构造了一个128层的full-attention网络，并在Cifar10图像生成问题上进行了训练。如下图所示，底部的黑色部分表示尚未生成到的部分，白色凸显的部分则是当前步注意力权重高的地方。



(a)中是比较低层的layer的注意力，可以看到，低层次的时候主要关注的还是**局部区域**的部分。

(b)在第19层和20层，Attention学习到了横向和纵向的规律。

(c)还有可能学习到和数据本身相关的attention。比如下图，第二列第二张学习到了鸟的边缘。

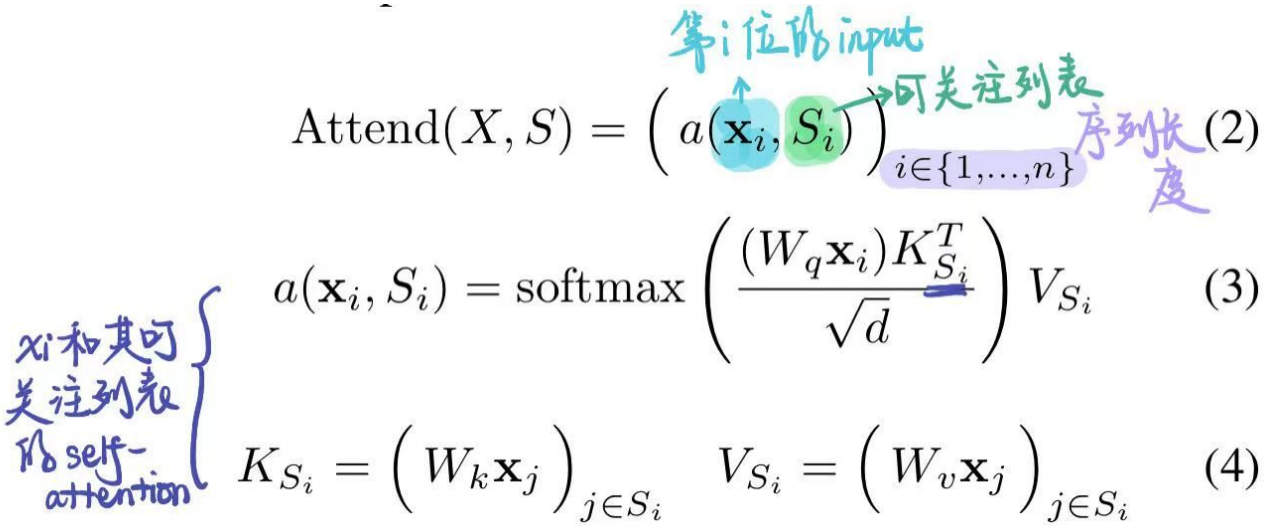
(d) 64-128层的注意力是高度稀疏的，只有极少的像素点有较高的注意力。

无论如何，注意力权重高的地方只占一小部分，这就为**稀疏注意力**提供了数据上的支持。作为解决注意力平方问题的早期论文，本文从图像生成的问题上揭示了attention的原罪，那就是其实不需要那么**密集**的注意力，Top-k的注意力已经足够可以保证效果了。

1.2 Factorized Self-attention

Sparse Transformer就是把full self-attention 分解成若干个小的、复杂度低的self-attention。这个过程叫做factorization。

定义集合$S = S_1, \dots S_n$， 这个集合中的每个元素还是集合，表示第i位input可以关注的位置。对于full-attention，S_i 显然就是 {j:j<i}. 每个位置的attention现在就变成了下图公式。其实没多大变化，只不过以前可以关注自己之前的所有位置，现在只能关注到一些特定的位置而已。



$$\text{Attend}(X, S) = \left(a(\mathbf{x}_i, S_i) \right)_{i \in \{1,\dots,n\}} \tag{2}$$

$$a(\mathbf{x}_i, S_i) = \text{softmax}\left(\frac{(W_q \mathbf{x}_i) K_{S_i}^T}{\sqrt{d}} \right) V_{S_i} \tag{3}$$

$$K_{S_i} = \left(W_k \mathbf{x}_j \right)_{j \in S_i} \quad V_{S_i} = \left(W_v \mathbf{x}_j \right)_{j \in S_i} \tag{4}$$

对于factorized self-attention，使用p个sparse注意力头，每个sparse注意力头有着不同的关注列表，记作$A_i^{(m)}$.

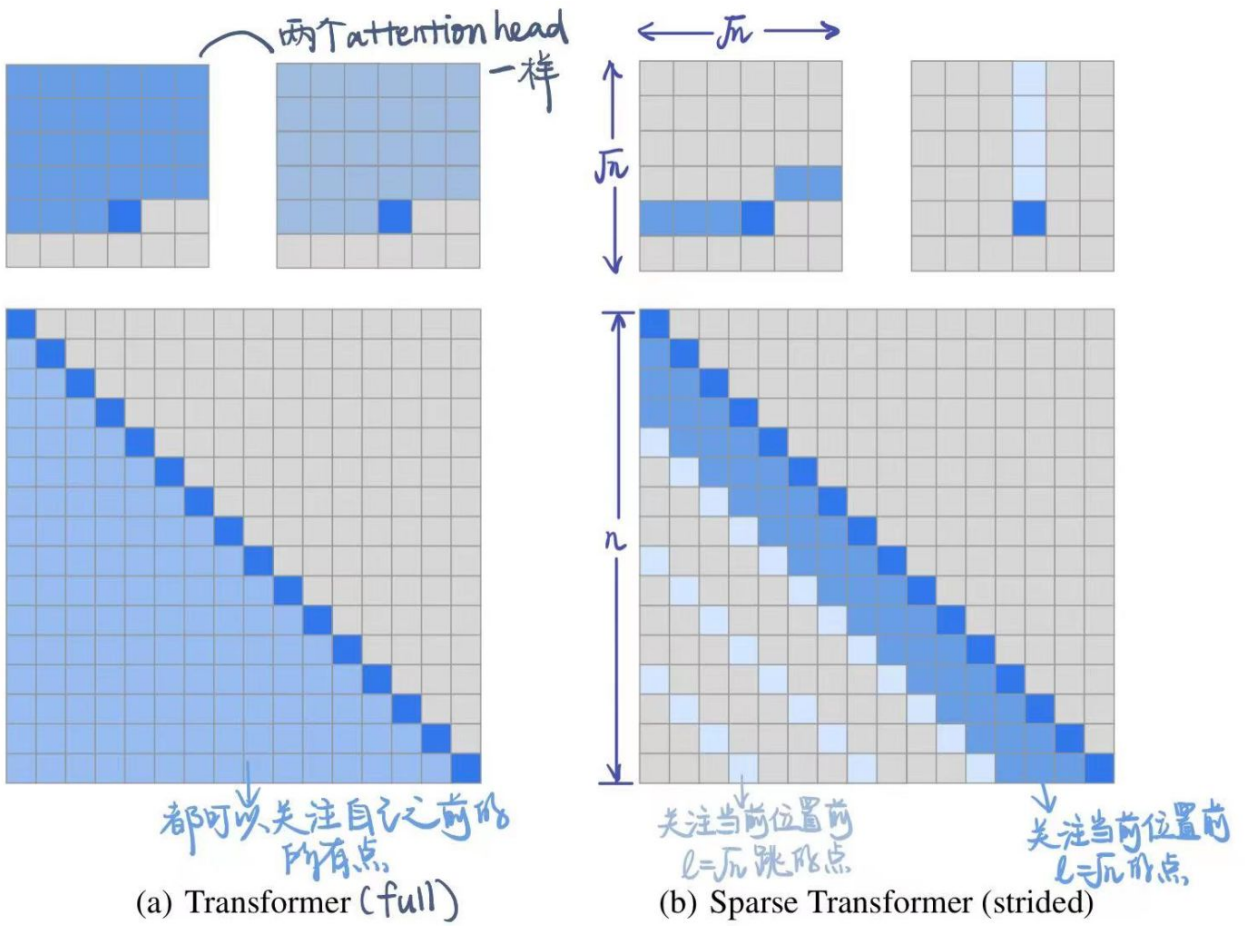
- 为了保证sparse注意力头的高效性(**efficiency**), 我们必须要保证 $|A_i^{(m)}|$是 $O(^p\sqrt{n})$复杂度的。
- 同时，为了保证sparse注意力头是有效(valid)的，我们需要保证每个位置都可以经过一些路径attend到**之前所有位置**（毕竟，这样才属于"factorize" full-attention）。同时这个路径长度不超过p+1，这样保证所有原本在全注意力上能够传递的信号在稀疏注意力的框架下仍然可以有效传递。

1.2.1 两种可能的sparse attention方法

当p = 2时，即两个注意力头的时候，文章给出了如下两种可以的sparse attention方法，能够满足上文所述的efficiency和valid条件。

（1）strided attention

- 一个注意力头只能关注当前位置前 $l = \sqrt{n}$ 个位置
- 另一个注意力头只能关注当前位置前面隔 $l = \sqrt{n}$ "跳"的位置

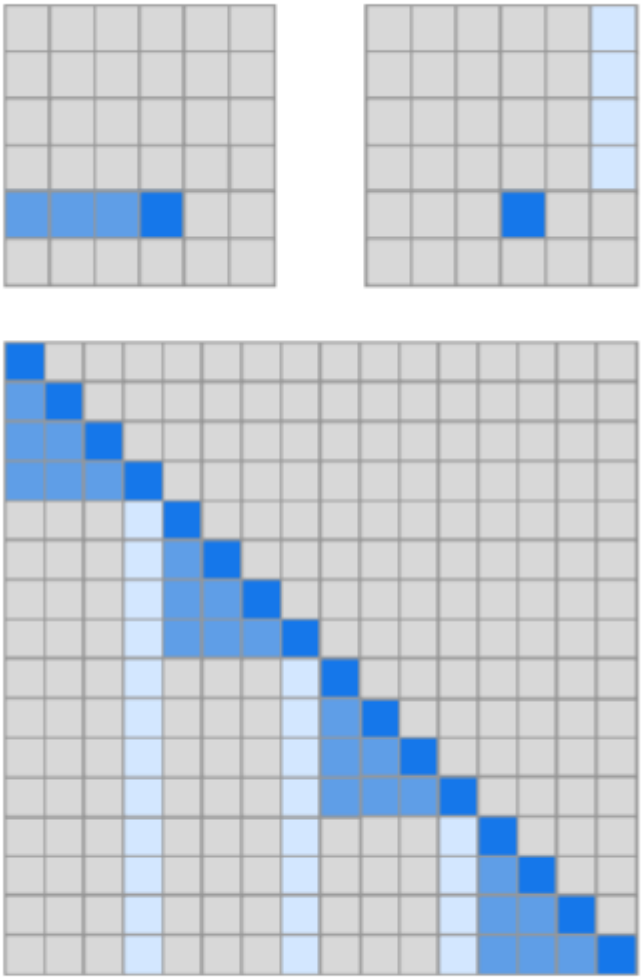


这样相当于关注当前行、当前列的信息，就如之前看的图像生成例子中的（b）一样。所以，这种注意力机制比较适用于图像。

（2）fixed attention

- Ai(1) = {j: floor(j/l) = floor(i/l)}
- Ai(2) = {j: j mod l ∈ {t, t+1, ..., l}}，其中t=l-c且c是超参数。

一般情况下，l取值为{128, 256}, c取值为{8, 16, 32}。这种模式非常适合于NLP问题，因为一般一句话的最后一个hidden state（下图浅蓝色）包含了整句话最多的意思，它也可以关注到这句话前面每个token。



(c) Sparse Transformer (fixed)

稀疏注意力的组合

一个直接的方法是在每个层使用同样的稀疏机制，在不同的块使用不同的。这样每个层的不同机制"交织(interleave)"在一起。

另一种方式则是在每个层使用组合的稀疏注意力，组合的方法则是把经过不同稀疏注意力机制的输出concat起来，就像普通的多头一样。

深度残差Transformer

深层次的Transformer训练起来十分困难，因为使用残差的方式会比较好。除了我们熟悉的transformer层内的layernorm之外，还增加了层间的残差连接，可以处理上百层的层。

$$H_0 = \text{embed}(X, W_e) \tag{9}$$

$$H_k = H_{k-1} + \text{resblock}(H_{k-1}) \tag{10}$$

$$y = \text{softmax}(\text{norm}(H_N)W_{out}) \tag{11}$$

$$a(H) = \text{dropout}(\text{attention}(\text{norm}(H))) \tag{12}$$

$$b(H) = \text{dropout}(\text{ff}(\text{norm}(H + a(H)))) \tag{13}$$

$$\text{resblock}(H) = a(H) + b(H) \tag{14}$$

2. Longformer

论文：https://arxiv.org/pdf/2004.05150.pdf

2.1 问题提出

BERT模型能处理的最大序列长度是512. 这是因为普通transformer的时间复杂度是随着序列长度n而平方增长的。如果我们想要处理更长的序列该怎么办呢?

- 最简单的方法就是直接截断成512长度的。这点普遍用于文本分类问题。
- 截成多个长度为512的序列段（这些序列段可以互相overlapping），每个都输入给Bert获得输出，然后将多段的输出拼接起来。
- 两个阶段去解决问题，就像搜索里面的召回 - 排序一样。一般用于Question-Answer问题，第一个阶段去选择相关文档，第二个阶段去找到对应的answer。

无论哪种方式，毫无疑问都会带来损失：截断会带来损失，两阶段会带来cascading error。如果能直接处理长序列就好了。

2.2 局部和全局attention的结合

Longformer将局部attention和全局attention结合起来，局部attention用来捕捉局部信息，一般用于**底层**（就像上文sparse attention中看到的，底层attention主要关注局部信息，是十分稀疏的）。全局attention则捕捉全局信息，用于**高层**，目的在于综合所有的信息，从而得到一个好的representation。

2.1 Sliding window

滑动窗口的大小为w，那么每个位置只attend前后w/2个位置。将模型多层叠加起来之后，**高层**的每个位置都可以关注到input的每个位置（就像卷积的感受野一样，这里可以有全局感受野）。一个*l*层的transformer，最上层的感受野是$l \times w$的。

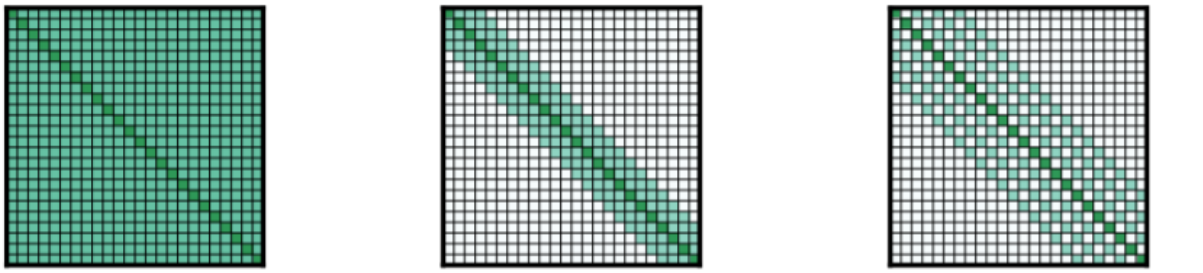
这样，每一层的计算复杂度就是$O(n * w)$而不是$O(n^2)$的了。

另外，每一层的w其实可以不同，鉴于越高层需要的全局信息越多，可以在层级较高的时候把w调大。来达到模型效率（efficiency）和模型表达能力（representation capacity）的平衡。

2.2 Dilated Sliding Window

引入dilated window的目的是为了再避免增加计算量的情况下继续增大感受野，类似空洞卷积。一个window有着大小为d的gap，那么最高层的感受野就是$l \times d \times w$的。

对于多头注意力，可以让有些头不用dilation，专注于关注局部信息；有些头用dilation，关注更远的信息。另外，底层不适合用dilated sliding window, 因为底层需要去学习局部的信息；高层可以适当的使用少量的dilated window，降低参数量。

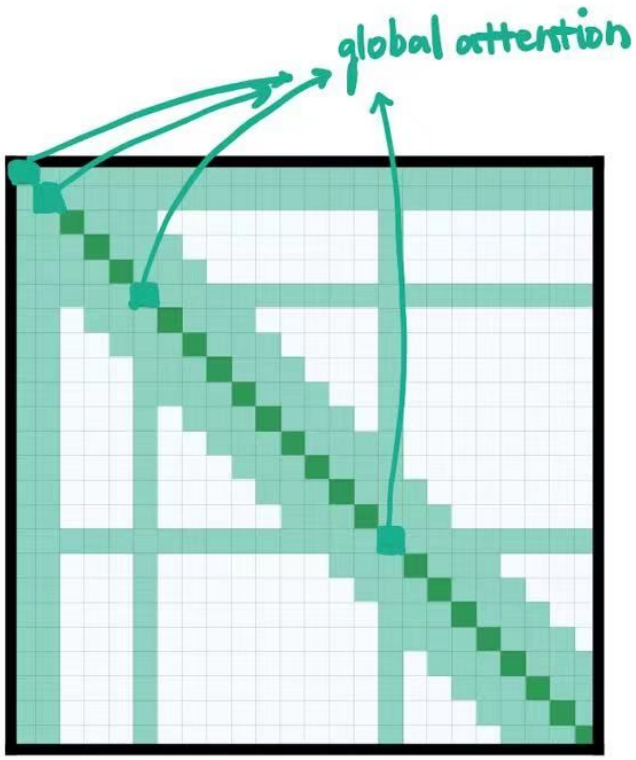


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window

2.3 Global Attention

究竟要选择哪种attention方式，其实是和任务有关的。对于MLM任务，或许只关注局部信息就足够了，所以使用滑窗是可以的；但是对于分类任务，BERT模型把整句话的信息都集中在了[CLS]中，所以**[CLS]应该能够关注到所有位置**。对于QA，我们将question和document拼接起来送入transformer中，由于每个位置都需要去比较看是否贴近question，所以理应所有位置都能关注到question的每个token，因此question的每个token需要具有全局注意力。

这里的"全局注意力"指的是，某个位置上的token可以关注所有其他位置，所有其他位置也都可以关注这个token。具体要选择那个位置赋予全局注意力，是和任务的性质有关的。



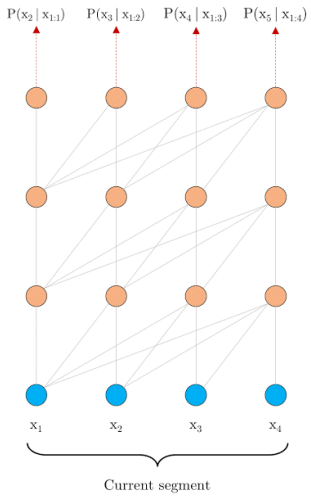
(d) Global+sliding window

3. Transformer-XL

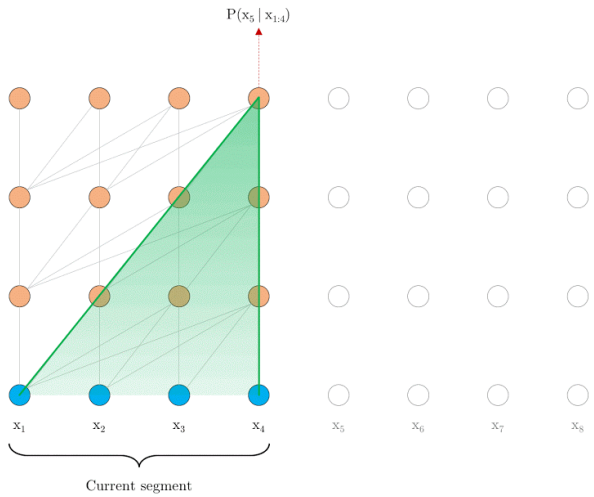
其实transformer-XL并不是解决transformer复杂度问题的，而是用来解决长文本的long-term dependency问题。但是transformer-XL在推理阶段可以达到比vanilla transformer快1800倍的加速，所以在这里也一并介绍了。

3.1 问题的提出

由于BERT等transformer模型的最长输入长度只有512，在处理长文本的时候只能像我们上文说的那样，截成若干个512长度的片段（segment），依次输入到BERT中训练，如下图所示。这样导致的问题就是，数据最多只能关注到自己所在片段的那512个token，段和段之间的信息丢失了。



在测试阶段，以文本生成这种自回归任务为例，需要依次取时间片为L = 512 的分段，然后将整个片段提供给模型后预测一个结果。在下个时间片时再将这个分段向**右移一个单位**，这个新的片段也将通过整个网络的计算后得到一个值。Transformer的这个特性导致其预测阶段的**计算量是非常大的。**



3.2 Transformer XL

Transformer-XL的核心包括两部分：片段循环(segment-level recurrence)和相对位置编码(relative positional encoding)

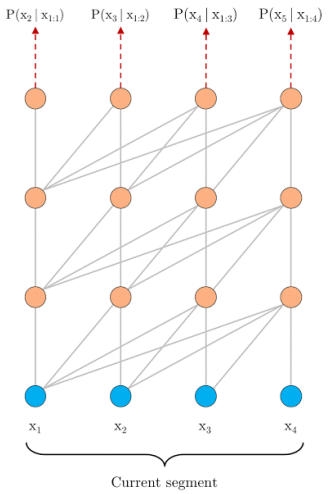
3.2.1 Segment-Level Recurrence with State Reuse

在训练阶段，上一个segment的隐藏状态会被**缓存下来**，然后在计算当前段的时候再重复使用上一个segment的隐层状态。因为上个片段的特征在当前片段进行了**重复使用**，这也就赋予了Transformer-XL建模更长期的依赖的能力。

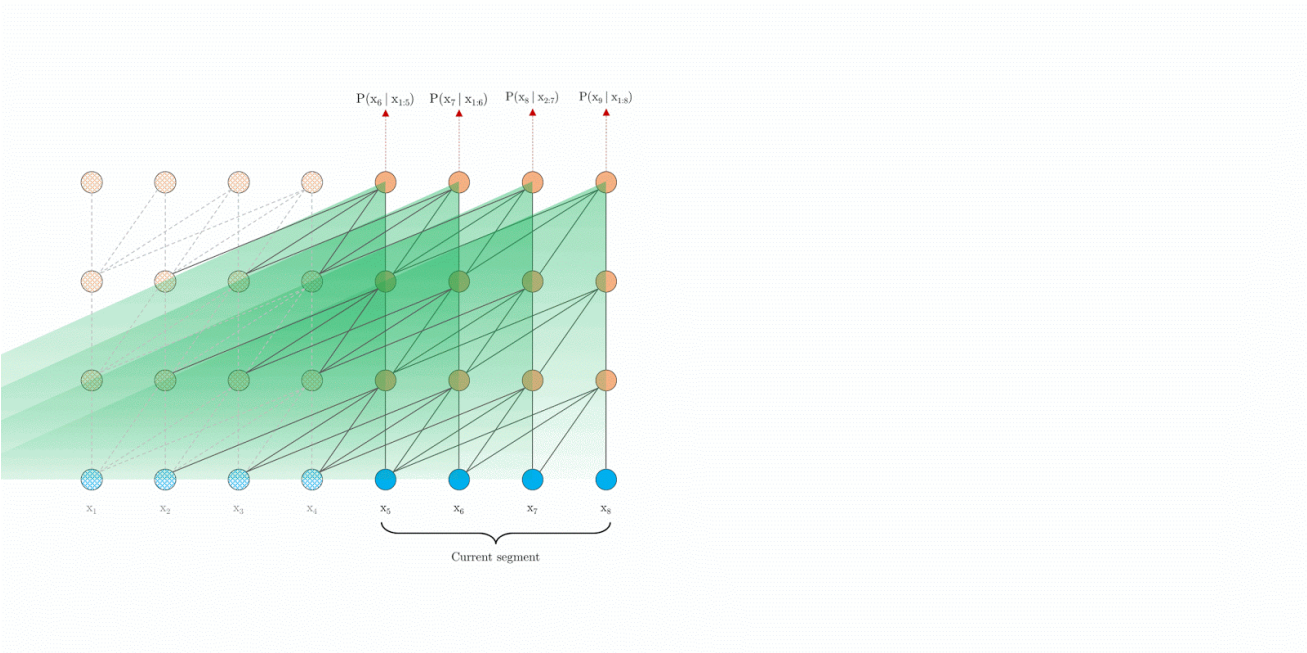
长度为 L 的连续两个片段表示为 $\mathbf{s}_\tau = [x_{\tau,1}, \cdots, x_{\tau,L}]$ 和 $\mathbf{s}_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}]$。$\mathbf{s}_\tau$ 的隐层节点的状态表示为 $\mathbf{h}_\tau^n \in \mathbb{R}^{L \times d}$，其中 d 是隐层节点的维度。$\mathbf{s}_{\tau+1}$ 的隐层节点的状态 $\mathbf{h}_{\tau+1}^n$ 的计算过程为：

$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\text{SG}(\mathbf{h}_\tau^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$
$$\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_q^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_k^\top, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_v^\top,$$
$$\mathbf{h}_{\tau+1}^n = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^n, \mathbf{k}_{\tau+1}^n, \mathbf{v}_{\tau+1}^n\right).$$

其中 $\text{SG}(\cdot)$ 表示stop-gradient，表示这一部分并不参与BP的计算，$[\mathbf{h}_u \circ \mathbf{h}_v]$ 表示两个隐层节点在长度维度进行拼接。不要被这个复杂的公式吓到！其实它想表达的意思很简单，就是每次在算一个segment的self-attention时，用当前这个segment的每个token当成Query向量，然后当前这个segment+上一个segment的每个token当成Key和Value向量，Query去关注Key和Value。这样，就把两个原本割裂的segment用attention给"粘合"了起来。记segment长度为N，那么一个L层的网络，最上面的层可以关注到的"感受野"就是O(N*L).



除了能够关注到更远的位置以外，另一个好处推理速度的提升。Transformer的自回归架构每次只能前进一个time step，而Transfomer-XL的推理过程直接复用上一个片段的表示而不是从头计算，每次可以前进一个segment长度。



3.2.2 相对位置编码

Transformer的位置编码是以segment为单位的，表示为 $\mathbf{U} \in \mathbb{R}^{L_{\max} \times d}$ ，第 i 个元素 $\mathbf{U}_i$ 表示的是在**这个segment中**第 i 个元素的相对位置， $L_{\max}$ 表示的是能编码的最大长度，即segment长度。对于不同的segment来说，它们的时间位置编码是完全相同的，我们完全没法确认它属于哪个片段或者它在分段之前的输入数据中的相对位置。

为了解决这个问题，可以采用相对位置编码的方式。一个位置在i的query向量去关注位置j的key向量，我们并不需要知道i和j是什么，真正重要的是i-j这个相对距离。所以，使用一个相对位置偏差的embedding矩阵$R \in \mathbb{R}^{L_{max} \times d}$来进行位置偏差的编码。之后，我们需要把这个描绘相对位置的embedding融入到传统transformer的attention计算中去。

位置i的向量x_i作为query，要去和位置j的向量x_j计算注意力权重（x_j作为key），使用绝对位置的计算公式如下。其中，E表示embedding，U代表绝对位置编码。

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(d)}.$$

将相对位置编码融入attention计算之后：

$$\mathbf{A}_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

可以发现做了如下的几处改进：

1） W_k 被拆分成立 $W_{k,E}$ 和 $w_{k,R}$ ，也就是说输入序列和位置编码不再共享权值。（蓝色和浅黄色部分）

2）绝对位置编码 $\mathbf{U}_j$ 换成了相对位置编码 $\mathbf{R}_{i-j}$ （棕色）

3）两个新的可学习的参数 $u \in \mathbb{R}^d$ 和 $v \in \mathbb{R}^d$ 来替换Transformer中的query向量 $\mathbf{U}_i^\top \mathbf{W}_q^\top$ 。表明**对于所有的query位置对应的query位置向量是相同的。**因为我们已经把相对位置编码融入了key端，那么query端就不再需要位置编码了。（红色和绿色部分）

参考：

https://zhuanlan.zhihu.com/p/260928791

https://zhuanlan.zhihu.com/p/271984518