Transfromer-xl

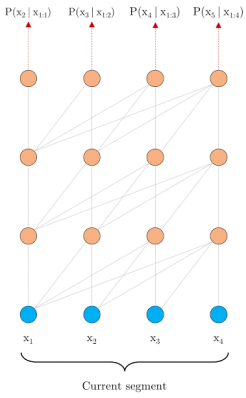
1.Introduction

Transformer的提出则进一步提升了获长期依赖的能力，但是Transformer的捕获长期依赖的能力是无限长的吗？如果有一个需要捕获几千个时间片的能力的模型才能完成的任务，Transformer能够胜任吗？答案从目前Transformer的设计来看，它还是做不到。

这篇文章介绍的Transformer-XL（extra long）则是为了进一步提升Transformer建模长期依赖的能力。它的核心算法包含两部分：片段递归机制（segment-level recurrence）和相对位置编码机制(relative positional encoding)。Transformer-XL带来的提升包括：1. 捕获长期依赖的能力；2. 解决了上下文碎片问题（context segmentation problem）；3. 提升模型的预测速度和准确率。

在最近流行的XLNet中就是使用Transformer-XL作为基础模块。在下文中，是将Trm-XL放在类似GPT这样的语言模型框架中来介绍，所以理解的时候要放在整个模型中去理解，而不是一个单独的Trm-XL。

NLP相关的任务都很难避免处理输入为变长数据的场景，这个问题的解决方案有两个，一是将数据输入到类似前馈神经网络这样的模型中得到长度固定的特征向量，这个方法往往因为计算资源的限制很难执行；另一个是通过数据切段或者padding的方式将数据填充到固定长度。Transfomer采取的便是第二个方案，这个值这里用 L来表示,L在原生transfromer为512.将数据分完段之后，接下来便是将分段的数据依次喂到网络中进行模型的训练.



这种分段式的提供数据的方式的一个很大的问题是数据并不会在段与段之间流通，因此模型能够捕获的长期依赖的上限便是段的长度。另外这种将数据分段，而不考虑段与段之间的关系无疑是非常粗暴的，对于模型的能力无疑是要打折的。这个问题便是我们所说的上下文碎片问题。

Vanilla Transformer在预测的时候，会对固定长度的segment做计算，一般取最后一个位置的隐向量作为输出。为了充分利用上下文关系，在每做完一次预测之后，就对整个序列向右移动一个位置，再做一次计算，这导致计算效率非常低。

为了解决上面提到的问题，在Vanilla Transformer的基础上，Trm-XL提出了一个改进，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的隐向量序列，而且上一个segment的所有隐向量序列只参与前向计算，不再进行反向传播，这就是所谓的segment-level Recurrence。

2.segment-level Recurrence

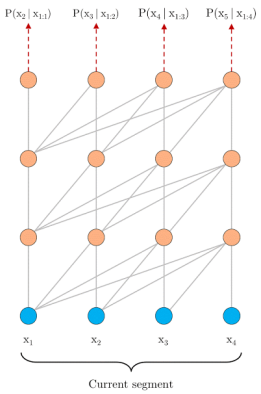
两连续的segments表示为 $s_\tau = [x_{\tau,1}, x_{\tau,2}, \ldots, x_{\tau,L}]$ 和 $s_{\tau+1} = [x_{\tau+1,1}, x_{\tau+1,2}, \ldots, x_{\tau+1,L}]$，L为序列长度。假设整个模型中，包含N层Trm，那么每个segment中就有N组长度为L的隐向量序列。d为隐向量的维度。

- $h_\tau^n \in R^{L\times d}$ ：第 τ 个segment的第n层隐向量序列

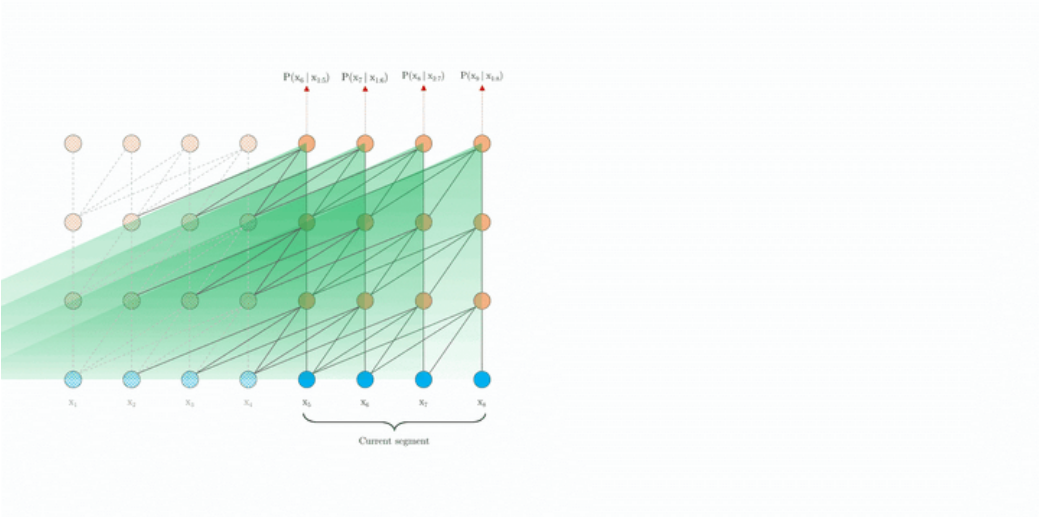
$h_\tau^n \in R^{L\times d}$，d是隐向量的维度。那么第 $\tau+1$ 个segment的第n层隐向量序列，可以由下面的一组公式计算得出。SG是stop-gradient，不再对 s_τ 的隐向量做反向传播。 $\tilde{h}_{\tau+1}^{n-1}$ 是对两个隐向量序列沿长度方向的拼接，[]内两个隐向量的维度都是 $L \times d$，拼接之后的向量维度是 $2L \times d$。3个W分别对应query，key和value的转化矩阵。注意q的计算方式不变，只使用当前segment中的隐向量，计算得到的q序列长度仍然是L。k和v采用拼接之后的 $\tilde{h}$ 来计算，计算出来的序列长度是2L。之后的计算就是标准的Transformer计算。计算出来的第n层隐向量序列长度仍然是L，而不是2L。Trm的输出隐向量序列长度取决于query的序列长度，而不是key和value。

$$\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{n-1} &= \left[\mathrm{SG}\left(\mathbf{h}_{\tau}^{n-1}\right) \circ \mathbf{h}_{\tau+1}^{n-1}\right] \text{(表示对两个向量的拼接, 拼接后为 } 2L \times d\text{)} \\
\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} &= \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top} \\
\mathbf{h}_{\tau+1}^{n} &= \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right)
\end{aligned}$$

$Q[L\times d]\cdot K^T[d\times 2L] = [L\times 2L]\cdot V[2L\times d] = [L\times d]$



Transformer-XL的上一个片段的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个时间片的隐层状态。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。片段递归的另一个好处是带来的推理速度的提升，对比Transformer的自回归架构每次只能前进一个时间片，Transfomer-XL的推理过程通过直接复用上一个片段的表示而不是从头计算，讲推理过程提升到以片段为单位进行推理，这种简化带来的速度提升是成百上千倍的。



3. embedding

在vanilla Trm中，为了表示序列中token的顺序关系，在模型的输入端，对每个token的输入embedding，加一个位置embedding。位置编码embedding或者采用正弦\余弦函数来生成，或者通过学习得到。在Trm-XL中，这种方法行不通，每个segment都添加相同的位置编码，多个segments之间无法区分位置关系。Trm-XL放弃使用绝对位置编码，而是采用相对位置编码，在计算当前位置隐向量的时候，考虑与之依赖token的相对位置关系。具体操作是，在算attention score的时候，只考虑query向量与key向量的相对位置关系，并且将这种相对位置关系，加入到每一层Trm的attention的计算中。

1.absolute position encoding

Tansformer的位置编码是以段为单位的，它使用的是无参数的sinusoid decoding matrix，表示为 $\mathbf{U} \in \mathbb{R}^{L_{\max} \times d}$ ，第 i 个元素 $\mathbf{U}_i$ 表示的是在这个分段中第 i 个元素的相对位置， $L_{\max}$ 表示的是能编码的最大长度。然后这个位置编码会通过单位加的形式和词嵌入（word Embedding）合并成一个矩阵，表示为：

$$\begin{aligned} \mathbf{h}_{\tau+1} &= f(\mathbf{h}_{\tau}, \mathbf{E}_{s_{\tau+1}} + U_{1:L}) \\ \mathbf{h}_{\tau} &= f(\mathbf{h}_{\tau-1}, \mathbf{E}_{s_{\tau}} + U_{1:L}) \end{aligned} \tag{4}$$

其中 $\mathbf{E}_{s_\tau} \in \mathbb{R}^{L \times d}$ 表示第 τ 个碎片 $\mathbf{s}_\tau$ 的词嵌入， f 表示转换方程。从(1)式中我们可以看出，对于第 τ 和第 $\tau+1$ 个片段来说，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。

在Transformer中，self-attention可以表示为:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V \tag{5}$$

考虑到词嵌入， $Q^T K$ 的完整表达式为：

$$\mathbf{A}^{\text{abs}}_{i,j} = (\mathbf{W}_q(\mathbf{E}_{x_i} + \mathbf{U}_i))^\top (\mathbf{W}_k(\mathbf{E}_{x_j} + \mathbf{U}_j)) \tag{6}$$

我们使用乘法分配律将其展开，展开式会在后面使用：

$$A_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)} \quad (7)$$

Transformer的问题是无论对于第几个片段，它们的位置编码 $\mathbf{U}_{1:L}$ 都是一样的，也就是说Transformer的位置编码是相对于片段的绝对位置编码（absulate position encoding），与当前内容在原始句子中的相对位置是没有关系的。

2.Relative position encoding

在vanilla Transfromer中，为了表示序列中token的顺序关系，在模型的输入端，对每个token的输入embedding，加一个位置embedding。位置编码embedding或者采用正弦\余弦函数来生成，或者通过学习得到。在Trm-XL中，这种方法行不通，**每个segment都添加相同的位置编码，多个segments之间无法区分位置关系。**Trm-XL放弃使用绝对位置编码，而是采用相对位置编码，在计算当前位置隐向量的时候，考虑与之依赖token的相对位置关系。

最先介绍相对位置编码的是论文《self-attention with relative positional representation》(后面简称RPR)。对比RNN系列的模型，Transformer的一个缺点是没有从网络结构上对位置信息进行处理，而只是把位置编码加入到了输入层。RPR的动机就是解决Transformer的这个天然缺陷，它的做法是把相对位置编码加入到了self-attention的内部。

Transformer-XL的相对位置编码参考了RPR中把相对位置编码加入到self-attention中的思想，Transfomer-XL在(7)式的基础上做了若干变化，得到了下面的计算方法：

$$A_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)} \quad (16)$$

- 第一个变化出现在了(a)，(b)，(c)，(d)中：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。
- 第二个变化是(b)和(d)中将绝对位置编码 $\mathbf{U}_j$*换成了相对位置编码*$\mathbf{R}_{i-j}$，其中 $\mathbf{R}$ 是Transformer中采用的不需要学习的sinsoid编码矩阵，原因正如第二借所介绍的，相对位置比绝对位置更为重要。
- 第三个变化是(c)，(d)中引入了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。**即无论query位置如何，对不同词的注意偏差都保持一致。

改进之后(16)中的四个部分也有了各自的含义：

- (a) 没有考虑位置编码的原始分数，只是基于内容的寻址；
- (b) 相对于当前内容的位置偏差；
- (c) 从内容层面衡量键的重要性，表示全局的内容偏置；
- (d) 从相对位置层面衡量键的重要性，表示全局的位置偏置。

式(16)使用乘法分配律得到的表达式为:

$$A_{i,j}^{rel} = (\mathbf{W}_q \mathbf{E}_{x_i} + u)^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j} + (\mathbf{W}_q \mathbf{E}_{x_i} + v)^\top W_{k,R} R_{i-j} \quad (17)$$

4.summary

最后来看一下Trm-XL的完整计算公式，如下所示，只有前3行与vanilla Trm不同，后3行是一样的。第3行公式中，计算A的时候直接采用query向量，而不再使用 表示。最后需要注意的是，每一层在计算attention的时候，都要包含相对位置编码。而在vanilla Trm中，只有在输入embedding中才包含绝对位置编码，在中间层计算的时候，是不包含位置编码的。

$$\tilde{h}_{\tau+1}^{n-1}=\left[SG\left(h_{\tau}^{n-1}\right), h_{\tau+1}^{n-1}\right]$$
$$q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n}=h_{\tau+1}^{n-1} W_{q}^{T}, \tilde{h}_{\tau+1}^{n-1} W_{k}^{T}, \tilde{h}_{\tau+1}^{n-1} W_{v}^{T}$$
$$h_{\tau+1}^{n-1}=\text{Transformer}\quad\text{Layer}\left(q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n}\right)$$
$$A_{i,j}^{rel}=E_{x_i}^{T} W_{q}^{T} W_{k,E} E_{x_j}+E_{x_i}^{T} W_{q}^{T} W_{k,R} R_{i-j}+U_{i}^{T} W_{q}^{T} W_{k,E} E_{x_j}+U_{j}^{T} W_{q}^{T} W_{k,R} R_{i-j}$$
$$\alpha_{\tau}^{n}=\text{Masked}\quad\text{Softmax}\left(A_{\tau}^{n}\right) V_{\tau}^{n}$$
$$o_{\tau}^{n}=\text{LayerNorm}\ \text{Linear}\left(\alpha_{\tau}^{n}\right)+h_{\tau+1}^{n-1}\Big)$$
$$h_{\tau}^{n}=\text{Positionwise Feed}\quad\text{Forward}\left(o_{\tau}^{n}\right)$$

ref

1. transfromer-XL论文详解
2.Transfromer xl介绍
3.https://nn.labml.ai/transformers/xl/index.html
4.详解Transformer-XL