[预训练语言模型专题] Transformer-XL 超长上下文注意力模型

原创 管扬 朴素人工智能

来自专辑

预训练语言模型

本文为预训练语言模型专题系列第十篇，同时增录之前的两篇为第十一和十二篇。

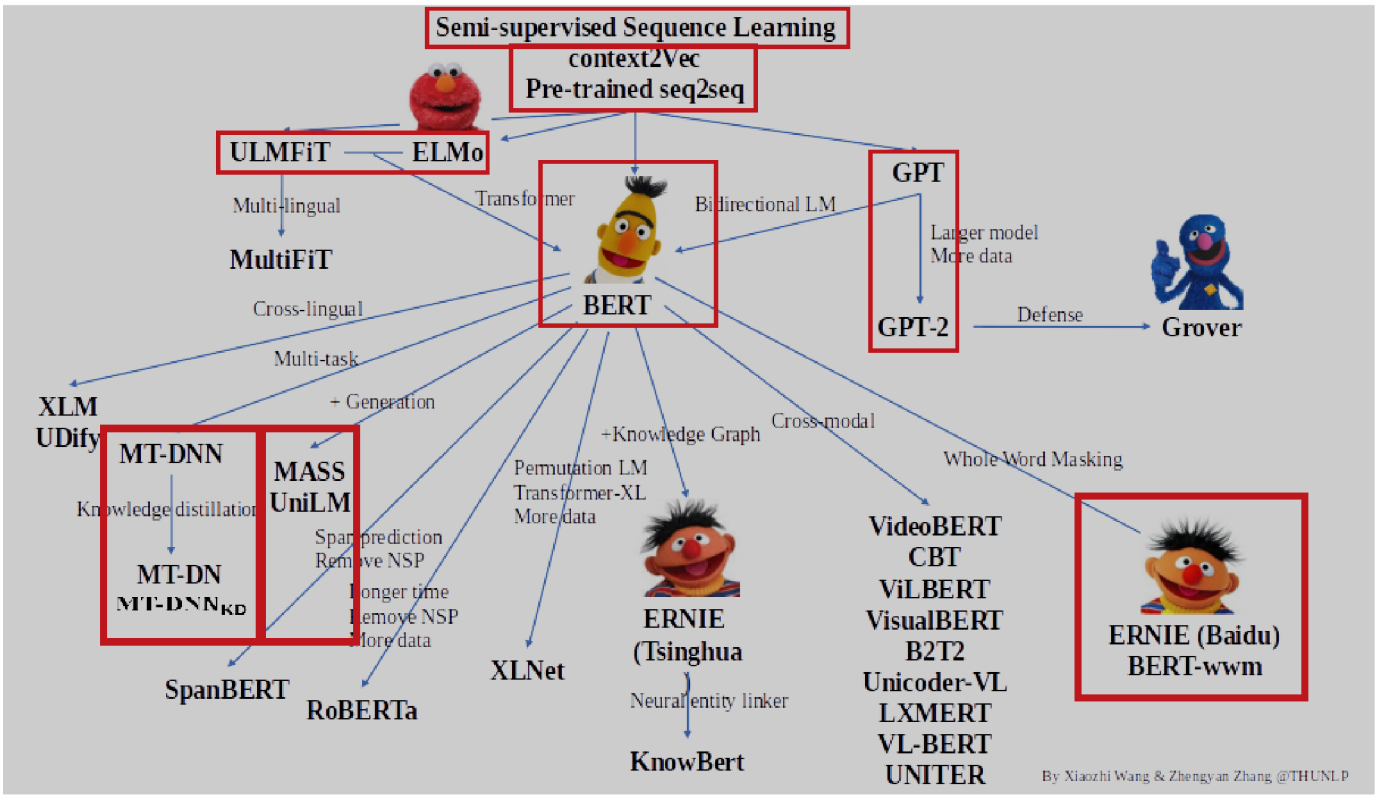
快速传送门

1-4:[萌芽时代]、[风起云涌]、[文本分类通用技巧] 、 [GPT家族]

5-8:[BERT来临]、[浅析BERT代码]、[ERNIE合集]、[MT-DNN(KD)]

9-12:[Transformer]、[Transformer-XL]、[UniLM]、[Mass-Bart]

感谢清华大学自然语言处理实验室对**预训练语言模型架构**的梳理，我们将沿此脉络前行，探索预训练语言模型的前沿技术，红框中为已介绍的文章，本期介绍的是Transformer-XL模型，欢迎大家留言讨论交流。



Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context（2019）

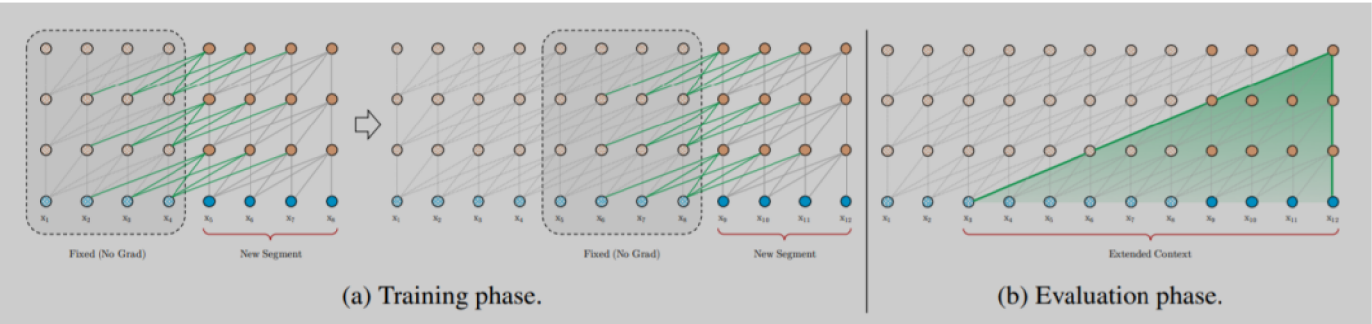
本期带来的是卡内基梅隆大学和Google Brain一同撰写的论文**Transformer-XL**，HuggingFace上也有代码的重现，大家有兴趣可以对照着看。

上期我们了解到Transformer是有能力学习到文本的长时依赖的，但是我们也不能不注意到，Transformer的复杂度是O（n^2)。所以随着文本的加长，Transformer的速度会下降得很快，所以大部分预语言模型的输入长度是有限制的，一般是512，当超过512时，长时文本的依赖Transformer是捕捉不到的。本文就提出了一种网络结构Transformer-XL，它不但可以捕捉文本更长时的依赖，同时可以解决文本被分成定长后产生的上下文碎片问题。据摘要中叙述，Transformer-XL能学习到的文本依赖比RNN长**80%**，比vanilla Transformer长**450%**。同时，它比vanilla Transformer在某些条件下evaluation时快了**1800倍**，而且短文本和长文本上都取得了不错的结果。

Vanilla Transformer Language Models

我们公众号之前也有跟大家分享过阅读理解竞赛的内容，在处理任意长的文本的时候，因为有限的算力和内存，通常的做法是把长文本**分割成短片段**比如512来进行处理。这样的缺点是，超越512长度的长时依赖就没有了，因为在片段之间，信息不会进行流动，会导致信息的碎片化。另外在模型evaluate的时候，为了利用之前511个token做context来解码，所以segment的区间每次都要滑动一位进行逐位解码，这相比train的时候是相当昂贵的。接下来，我们来介绍下Transformer-XL是如何解决这个问题。

Segment-Level Recurrence with State Reuse



为了解决短片段信息碎片等问题，文章对Transformer结构提出了一种**片段重用的循环机制**。

在训练的过程中，当处理新的Fragment的时候，之前计算的hidden_state已经被修补存储起来，会作为context信息来进行重用。虽然训练梯度依旧只在一个fragment之间流转，但过去的历史信息是可以实实在在传递到新的fragment训练中。

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

就像上面公式中描述的一样，第一个公式的o代表的是concat，SG代表stop gradient。首先将 τ 时刻 n-1层的hidden_state 和 τ+1时刻n-1层的hidden_state拼接形成新的隐层向量。然后经过计算得到当前的q, k, v向量再通过Transformer的层来获得 τ+1时刻第 n 层的 hidden_state。可以看到它和标准Transformer的关键区别是，τ+1时刻的k和v向量是包含有 **τ时刻 hidden_state**的信息的。如此一来，两个片段之前的上下文信息可以进行有效的传递。

另外，这种循环机制和循环神经网络不同的是，循环并非是建立在同一层上的，而是会将信息在层间以**三角形的形状**向上层传递，如上图右侧图(b)的深色部分所示。所以如果层数为N，片段长为L，那么最终最大的语义依赖距离大概为O(N * L)。除此以外，这种循环机制可以在验证时大大加快速度，因为在进行新的位置解码时，可以重用之前在循环机制中计算过的hidden_state。在作者的实验中，在enwiki8数据集上，Transformer-XL evaluate的解码速度比普通模型快了1800倍。进一步地，作者提出，在理论上不仅仅可以储存并重用之前一个片段的结果，只要GPU允许，完全可以重用前几个片段的结果，使上下文联系更远。

Relative Positional Encodings

在上述的循环机制中，有一点问题没有解决。就是在重用之前片段的信息时，我们如何保持原来的位置编码信息。之前Transformer中介绍过，在一个片段中，我们会根据token在片段中的位置，将这个位置对应的token的embedding和位置编码相加，因此位置编码是与token位置对应的绝对编码。这样就会遇到问题，当我们重用之前片段的信息时，前一个片段和本片段的相同位置使用的是同样的位置编码，没有办法区分。为了避免这种情况发生，本文提出了一种解决方案。

使用相对位置编码替代绝对位置编码。相比于原来在Embedding的绝对位置一起累加，作者提出在attention中当每两个位置进行attend时，根据他们的相对位置关系，加入对应的位置编码。这样的话，在重用前一段文本的时候，我们可以通过相对距离来进行区分，这样保持了文本的距离和相对位置信息。

首先我们看看标准的Transformer，Q和K的乘积可以分解成以下的四项。E为token的Embedding，U为绝对位置编码。

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

运用相对位置编码的思想，我们首先会把(b)项和(d)项中的U_j替换成相对的位置编码R_{i-j}。这样的改变可以让原本与j的绝对位置有关的编码部分转为了只与两者之间的相对位置有关，这个相对位置编码和原来的绝对位置编码一样也是不可学习的，只与i-j有关。另外将原来在(c)项和(d)项中一样的W_k，变成了与Embedding对应的$W_{k,E}$ 以及与位置编码对应的$W_{k,R}$。最后，作者引入了两个可学习的变量u和v，用以替代(c)项和(d)项中的绝对位置编码和query矩阵的乘积，因为在这里乘得的query相关向量应该与绝对位置无关。

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

经过了这样的替换，作者认为，每一项都有了一个具体含义，(a)项是content based addressing，也即主要是基于内容的寻址。(b)项是content dependent positional bias，和内容相关的位置编码偏置。(c)项是global content bias，全局的内容偏置。(d)项是global positional bias，全局的位置偏置。

最后，结合循环机制和相对位置编码，Transformer-XL一层的完整公式如下所示：

$$
\begin{aligned}
\widetilde{\mathbf{h}}_{\tau}^{n-1} &= \left[\mathrm{SG}(\mathbf{m}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau}^{n-1}\right] \\
\mathbf{q}_{\tau}^{n}, \mathbf{k}_{\tau}^{n}, \mathbf{v}_{\tau}^{n} &= \mathbf{h}_{\tau}^{n-1}\mathbf{W}_{q}^{n\top}, \widetilde{\mathbf{h}}_{\tau}^{n-1}\mathbf{W}_{k,E}^{n\top}, \widetilde{\mathbf{h}}_{\tau}^{n-1}\mathbf{W}_{v}^{n\top} \\
\mathbf{A}_{\tau,i,j}^{n} &= \mathbf{q}_{\tau,i}^{n\top}\mathbf{k}_{\tau,j}^{n} + \mathbf{q}_{\tau,i}^{n\top}\mathbf{W}_{k,R}^{n}\mathbf{R}_{i-j} \\
&\quad + u^{\top}\mathbf{k}_{\tau,j} + v^{\top}\mathbf{W}_{k,R}^{n}\mathbf{R}_{i-j} \\
\mathbf{a}_{\tau}^{n} &= \text{Masked-Softmax}(\mathbf{A}_{\tau}^{n})\mathbf{v}_{\tau}^{n} \\
\mathbf{o}_{\tau}^{n} &= \text{LayerNorm}(\text{Linear}(\mathbf{a}_{\tau}^{n}) + \mathbf{h}_{\tau}^{n-1}) \\
\mathbf{h}_{\tau}^{n} &= \text{Positionwise-Feed-Forward}(\mathbf{o}_{\tau}^{n})
\end{aligned}
$$

Experiment

作者在语言模型任务上将Transformer-XL和其他state-of-art的模型进行对比，效果拔群。其中One Billion Word数据集中不包含长时文本的依赖，可以看到效果也是相当好的。

Model	#Param	PPL
Grave et al. (2016b) - LSTM	-	48.7
Bai et al. (2018) - TCN	-	45.2
Dauphin et al. (2016) - GCNN-8	-	44.9
Grave et al. (2016b) - LSTM + Neural cache	-	40.8
Dauphin et al. (2016) - GCNN-14	-	37.2
Merity et al. (2018) - QRNN	151M	33.0
Rae et al. (2018) - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	**24.0**
Baevski and Auli (2018) - Adaptive Input◇	247M	20.5
Ours - Transformer-XL Large	257M	**18.3**

Table 1: Comparison with state-of-the-art results on WikiText-103. ◇ indicates contemporary work.

Model	#Param	bpc
Ha et al. (2016) - LN HyperNetworks	27M	1.34
Chung et al. (2016) - LN HM-LSTM	35M	1.32
Zilly et al. (2016) - RHN	46M	1.27
Mujika et al. (2017) - FS-LSTM-4	47M	1.25
Krause et al. (2016) - Large mLSTM	46M	1.24
Knol (2017) - cmix v13	-	1.23
Al-Rfou et al. (2018) - 12L Transformer	44M	1.11
Ours - 12L Transformer-XL	41M	**1.06**
Al-Rfou et al. (2018) - 64L Transformer	235M	1.06
Ours - 18L Transformer-XL	88M	1.03
Ours - 24L Transformer-XL	277M	**0.99**

Table 2: Comparison with state-of-the-art results on enwik8.

Model	#Param	bpc
Cooijmans et al. (2016) - BN-LSTM	-	1.36
Chung et al. (2016) - LN HM-LSTM	35M	1.29
Zilly et al. (2016) - RHN	45M	1.27
Krause et al. (2016) - Large mLSTM	45M	1.27
Al-Rfou et al. (2018) - 12L Transformer	44M	1.18
Al-Rfou et al. (2018) - 64L Transformer	235M	1.13
Ours - 24L Transformer-XL	277M	**1.08**

Table 3: Comparison with state-of-the-art results on text8.

Model	#Param	PPL
Shazeer et al. (2014) - Sparse Non-Negative	33B	52.9
Chelba et al. (2013) - RNN-1024 + 9 Gram	20B	51.3
Kuchaiev and Ginsburg (2017) - G-LSTM-2	-	36.0
Dauphin et al. (2016) - GCNN-14 bottleneck	-	31.9
Jozefowicz et al. (2016) - LSTM	1.8B	30.6
Jozefowicz et al. (2016) - LSTM + CNN Input	1.04B	30.0
Shazeer et al. (2017) - Low-Budget MoE	~5B	34.1
Shazeer et al. (2017) - High-Budget MoE	~5B	28.0
Shazeer et al. (2018) - Mesh Tensorflow	4.9B	24.0
Baevski and Auli (2018) - Adaptive Input◇	0.46B	24.1
Baevski and Auli (2018) - Adaptive Input◇	1.0B	23.7
Ours - Transformer-XL Base	0.46B	23.5
Ours - Transformer-XL Large	0.8B	**21.8**

Table 4: Comparison with state-of-the-art results on One Billion Word. ◇ indicates contemporary work.

另外，作者做了一系列对比实验，证实了循环机制和相对位置编码的重要性，不再赘述，比如下面这张图是表明在片段长度较长的时候，其和vanilla Transformer的速度的差距。

Attn Len	How much Al-Rfou et al. (2018) is slower
3,800	1,874x
2,800	1,409x
1,800	773x
800	363x

未完待续

本期的论文就给大家分享到这里，感谢大家的阅读和支持，下期我们会给大家带来其他预训练语言模型的介绍，敬请大家期待！

欢迎关注朴素人工智能，这里有很多最新最热的论文阅读分享，有问题或建议可以在公众号下留言。

往期推荐

- [预训练语言模型专题] BART & MASS 自然语言生成任务上的进步
- [预训练语言模型专题] 结合HuggingFace代码浅析Transformer
- [预训练语言模型专题] MT-DNN(KD)：预训练、多任务、知识蒸馏的结合
- [预训练语言模型专题] Huggingface简介及BERT代码浅析