[预训练语言模型专题] 结合HuggingFace代码浅析Transformer

原创 管扬 朴素人工智能

来自专辑

预训练语言模型

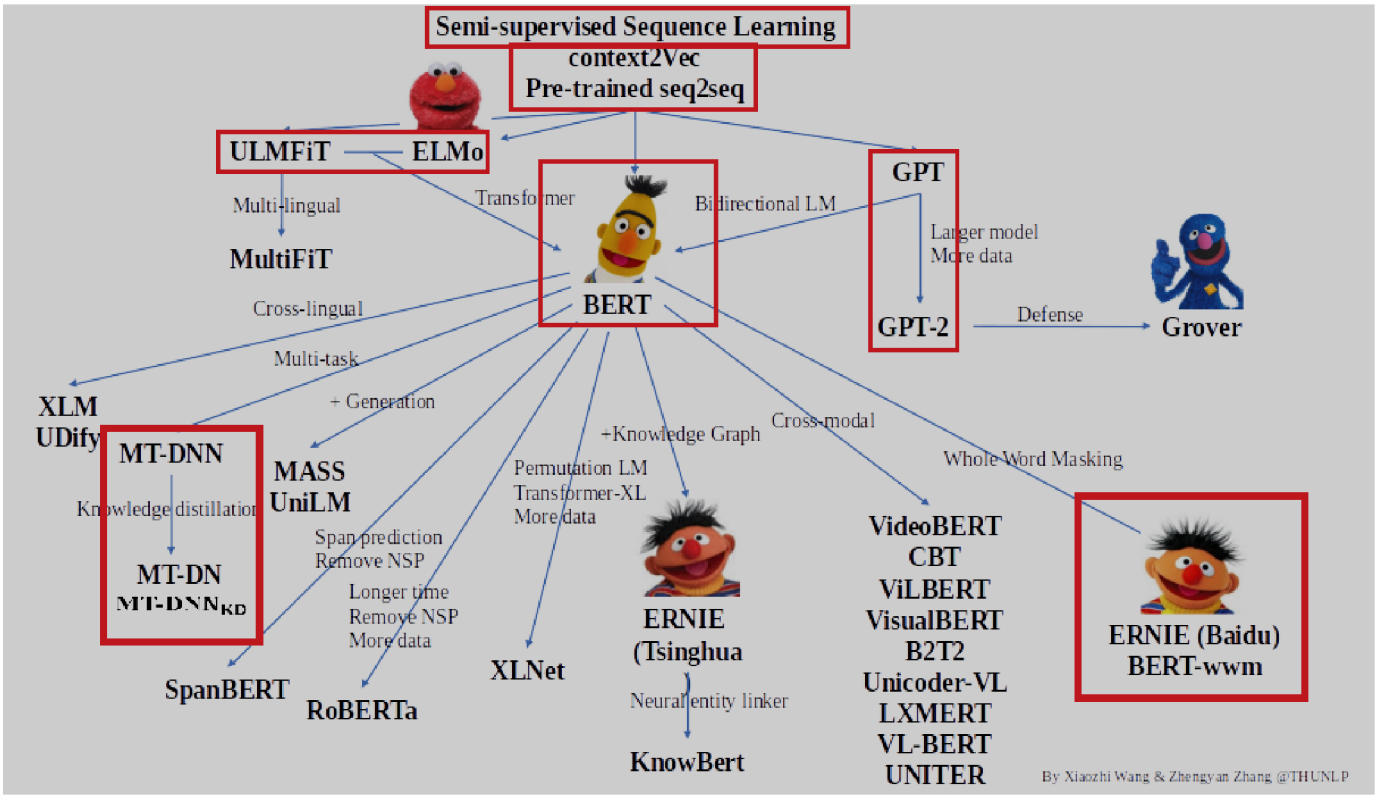
本文为预训练语言模型专题系列第九篇

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)]

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，本期将结合HuggingFace代码浅析Transformer代码，欢迎大家留言讨论交流。



1 前言

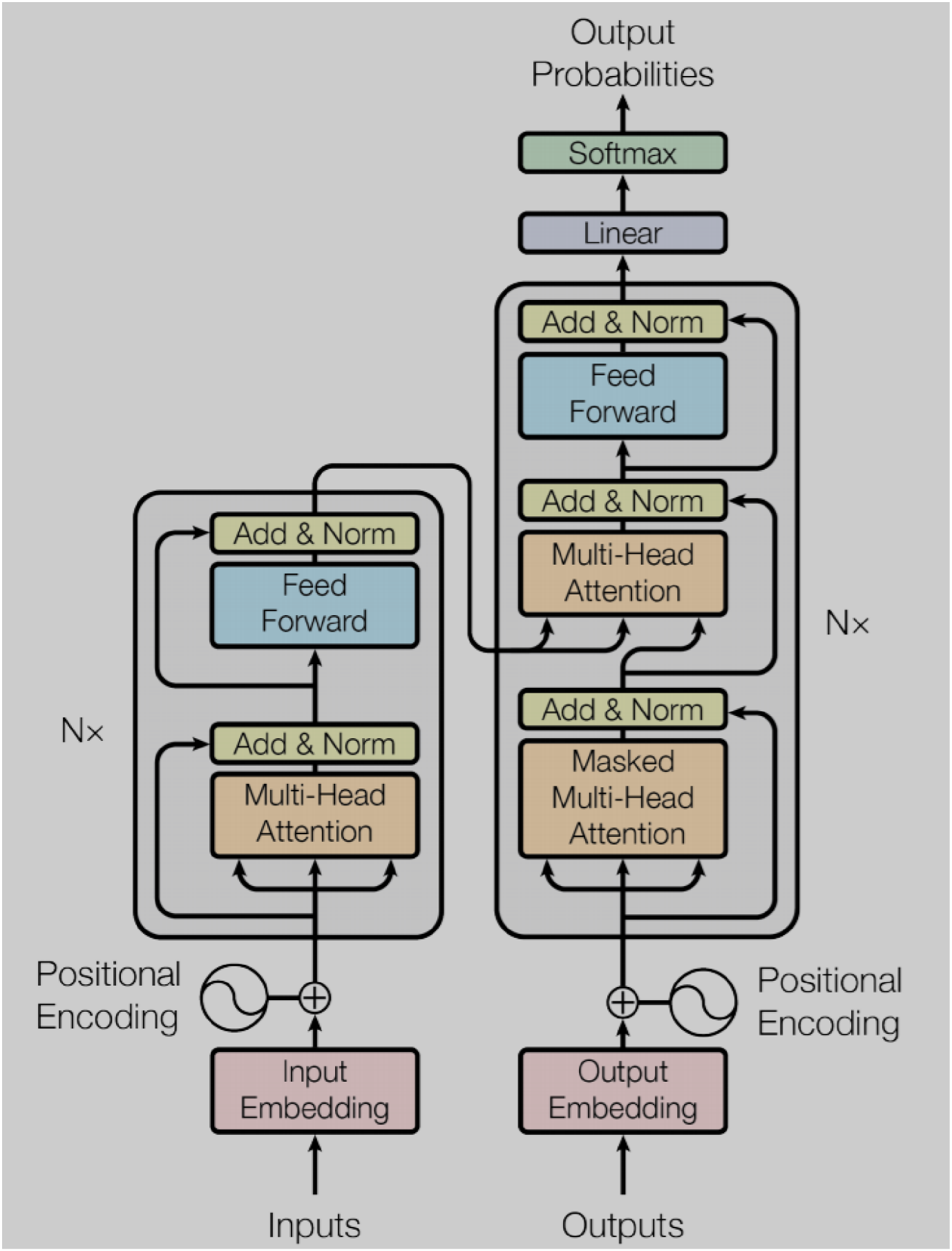
前面几期一起分享了这么多基于**Transformer**的预训练语言模型，本期想和大家一起来结合代码复习一下Transformer。它是目前state-of art 语言模型中最核心的模块，替代RNN成为NLP的柱石。

我在分享中会引用HuggingFace Transformers包中的代码，主要是BertAttention的相关代码，希望大家也能有所收获。

2 Attention Is All You Need（2017）

以前，处理NLP时序序列的关键模块是循环神经网络RNN（LSTM）或者卷积神经网络CNN。但是它们都有各自的问题。比如RNN无法进行并行计算，训练速度较慢，而且梯度传递有困难，容易梯度爆炸或消失。而卷积神经网络难以捕捉长距离的语义。所以，这篇文章提出了一种新的简单网络结构，称为Transformer，单纯基于attention的机制，既能并行计算提高训练速度，还能够捕捉句中的长序文本内部的联系。

直接上结构图：



Encoder-decoder

首先它是一个encoder-decoder的结构。从设计上看，左边的inputs会被encode成向量表示z，输入给右边decoder(encoder上方流出）。在解码的时候，decoder会结合z和outputs中某

token之前的token来生成当前的token，是比较典型的自回归模型。我们分别说说它的encoder以及decoder。

- Encoder：

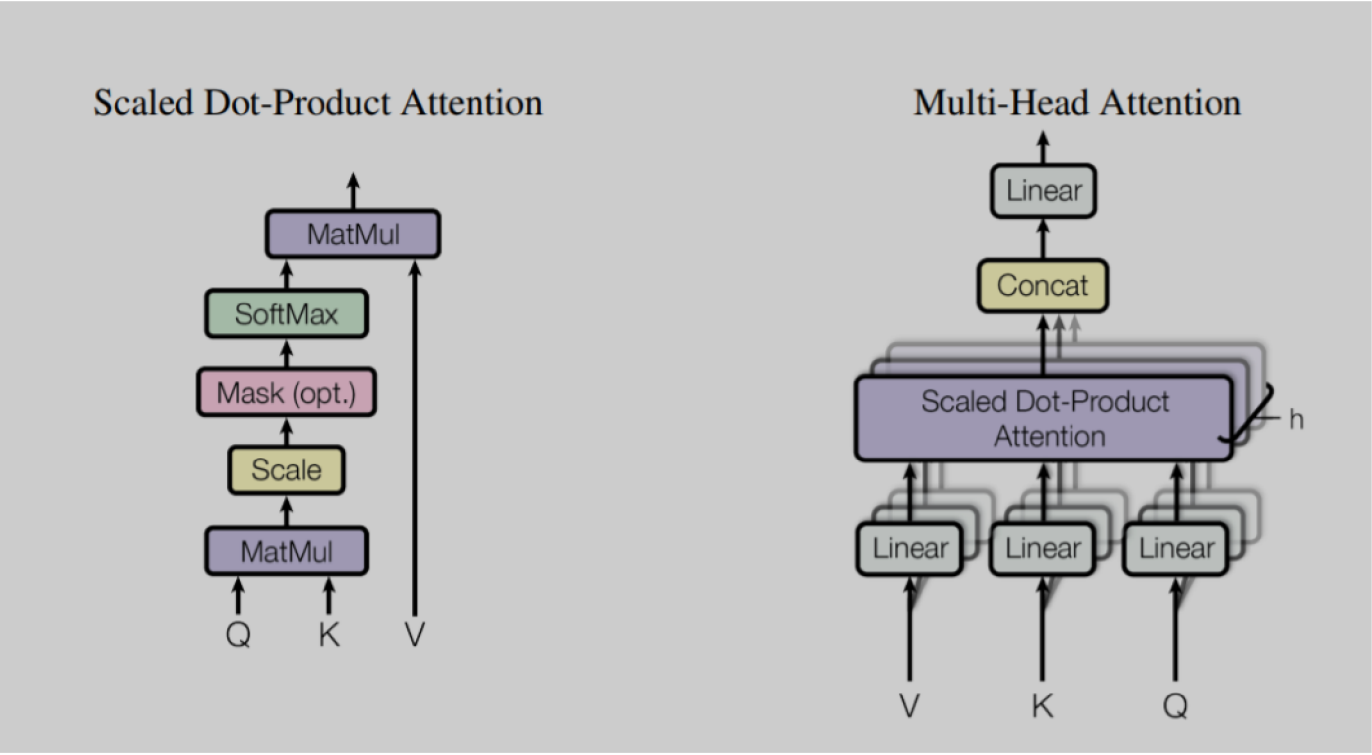
  首先在Transformer的encoder里有六层，每一层都是图中这样的两个sublayer。第一个sublayer是一个Multi-Head Attention，第二个sublayer是feed forward layer。在这两个sublayer之间都有**残差连接和层归一化**。

- Decoder:

  decoder也是六层，比起encoder有两个变化，一是第一个sublayer的multi-head attention需要进行mask，因为作为一个自回归模型在decode的时候，每个词的生成时，后面的词还没有生成出来，所以attention只能看到前面的词，后面的需要被mask掉。二是中间插了一个新层来**计算encoder传来的向量与output向量的attention**。

Attention

Transformers的精华就是Attention，接下来会结合论文和代码来介绍attention的基本概念和用法。



上图左侧的叫做**Scaled Dot-Product Attention**。计算的公式可以表示为下图，Q和K两矩阵相乘后进行scale，scale的因子dk为单个头的维度，也即是上述代码中的attention_head_size，矩阵乘V得到Attention的向量表达。

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

这里我们结合Transformers这个包BertSelfAttention类的代码来具体讨论。首先定义三个矩阵query, key, value。参数量和hidden_size和 head的数量有关，关于head我们后面再提。L为文本长度。

```
# 头的数量，以及每个头的size
self.num_attention_heads = config.num_attention_heads
self.attention_head_size = int(config.hidden_size / config.num_attention_heads)
self.all_head_size = self.num_attention_heads * self.attention_head_size
# 三个变换矩阵
self.query = nn.Linear(config.hidden_size, self.all_head_size)
self.key = nn.Linear(config.hidden_size, self.all_head_size)
self.value = nn.Linear(config.hidden_size, self.all_head_size)
```

这里定义的三个矩阵，实际上是要将某句文本的hidden_states变换成Q, K, V的表示，下面的代码中是具体的变换和计算。

```
# 将x从（batch_size, L, all_head_size)变为(batch_size, num_heads, L, attention_
def transpose_for_scores(self.x):
    new_x_shape = x.size()[:-1] + (self.num_attention_heads, self.attention_he
    x = x.view(*new_x_shape)
    return x.permute(0, 2, 1, 3)
# 对hidden_states进行三种变换，形成Q, K, V
mixed_query_layer = self.query(hidden_states)
mixed_key_layer = self.key(hidden_states)
mixed_value_layer = self.value(hidden_states)

query_layer = self.transpose_for_scores(mixed_query_layer)
key_layer = self.transpose_for_scores(mixed_key_layer)
value_layer = self.transpose_for_scores(mixed_value_layer)
# Q和K相乘得到本文自注意力的评分
attention_scores = torch.matmul(query_layer, key_layer.transpose(-1, -2))
attention_scores = attention_scores / math.sqrt(self.attention_head_size)
if attention_mask is not None:
    attention_scores = attention_scores + attention_mask
```

```
19
20  # Normalize the attention scores to probabilities.
21  attention_probs = nn.Softmax(dim=-1)(attention_scores)
22
23  # This is actually dropping out entire tokens to attend to, which might
24  # seem a bit unusual, but is taken from the original Transformer paper.
25  attention_probs = self.dropout(attention_probs)
26
27  # Mask heads if we want to
28  if head_mask is not None:
29      attention_probs = attention_probs * head_mask
30  # softmax过的attention_prob乘V得到Attention的表示向量
31  context_layer = torch.matmul(attention_probs, value_layer)
32
```

结合上面代码中，我们可以观察到

1. hidden_states的shape是(batch_size, L, hidden_size) 。为文本产生的向量表示如Embedding。
2. 我们通过query矩阵乘hidden_state得到mixed_query_layer (batch_size, L, all_head_size)
3. 经过 transpose变化成为query_layer(batch_size, num_heads, L, attention_head_size) 得到上图中的 Q
4. 同样的 key_layer（上图中K）的shape为(batch_size, num_heads, L, attention_head_size)，最后两维包括文本长度的维度进行了矩阵相乘，所以attention_scores维度变为(batch_size, num_heads, L, L)
5. attention_score 经过softmax ，长度方向归一化了，乘于value变为context_layer(batch_size, num_heads, L, attention_head_size)

所以Q(query) ，K(key)，V(value) 都是由原来文本向量乘于对应矩阵得到。这三种变换的矩阵参数都是我们需要通过训练学习。Q矩阵乘K以后，得到(batch_size, num_heads, L, L)的矩阵，我理解为文本的每个位置和其他位置都相乘得出一个数值，这个数值我们可以看作文本的每个token和其他token的相关度，即为self attention的**score**，越大一般这两个token关系就越密切，softmax以后变成一个0-1的数，这时候score再矩阵乘value我们就可以得到一个上下文相关向量的attention表示了。

前面还提到了**Multi-Head Attention**，多头注意力。相比于进行一次attention function，进行h次效果会更好。也就是说，我们会初始化h个不同的query, key, value矩阵，每个大小为（hidden_state, attention_head_size)， 在上面代码中，我们实际上初始了一个（hidden_state,

attention_head_size * num_heads) 大小的矩阵，与num_heads(h)个单头的attention矩阵是一致的，矩阵中其实参数是单头的num_heads份。然后每份参数去进行上面的attention运算，最后把多份的attention 拼接起来成为了最终的multi-head attention。

Positional Encoding

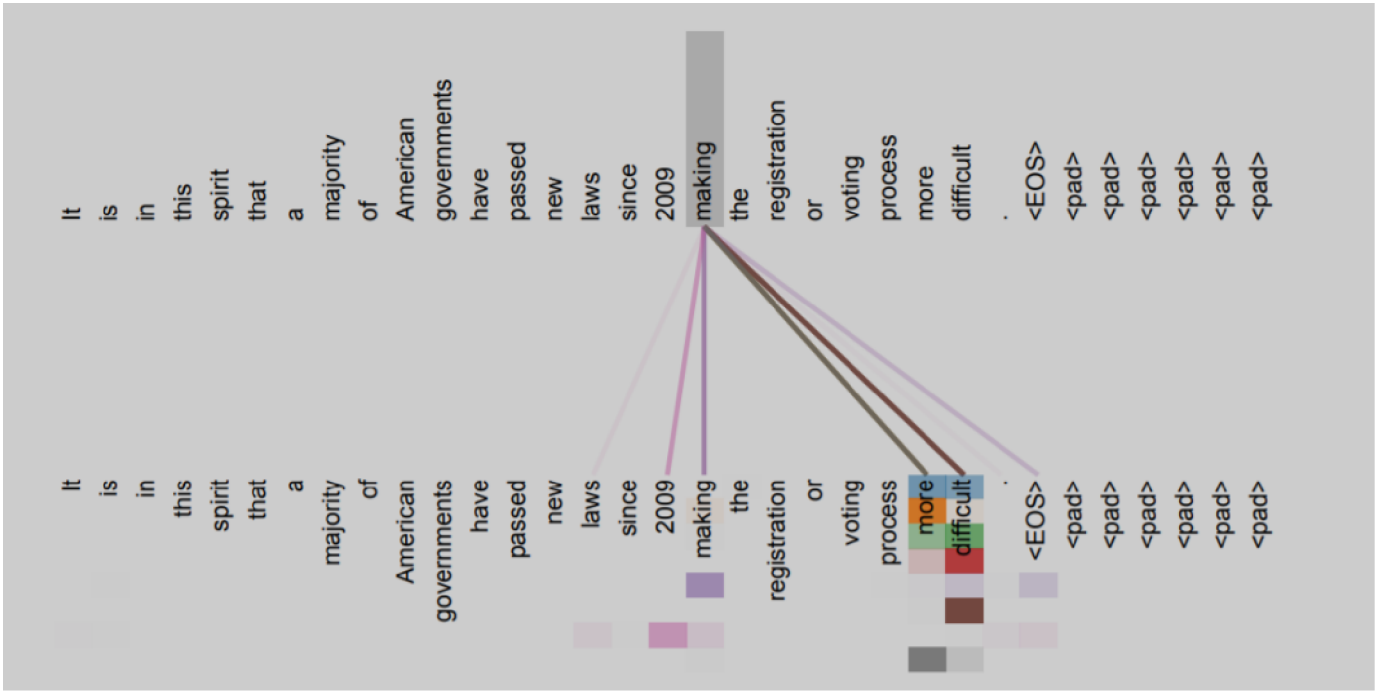
Attention机制让Transformer 得以能够建立长距离的语义关联，但是我们可以注意到，在encoder和decoder中，我们用的都是fully connected layer，所以每个位置的token都是独立的，你会失去语序的信息。所以有必要告诉网络，token之间的相对或绝对信息，所以文章引入了**Positional Encoding**。

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

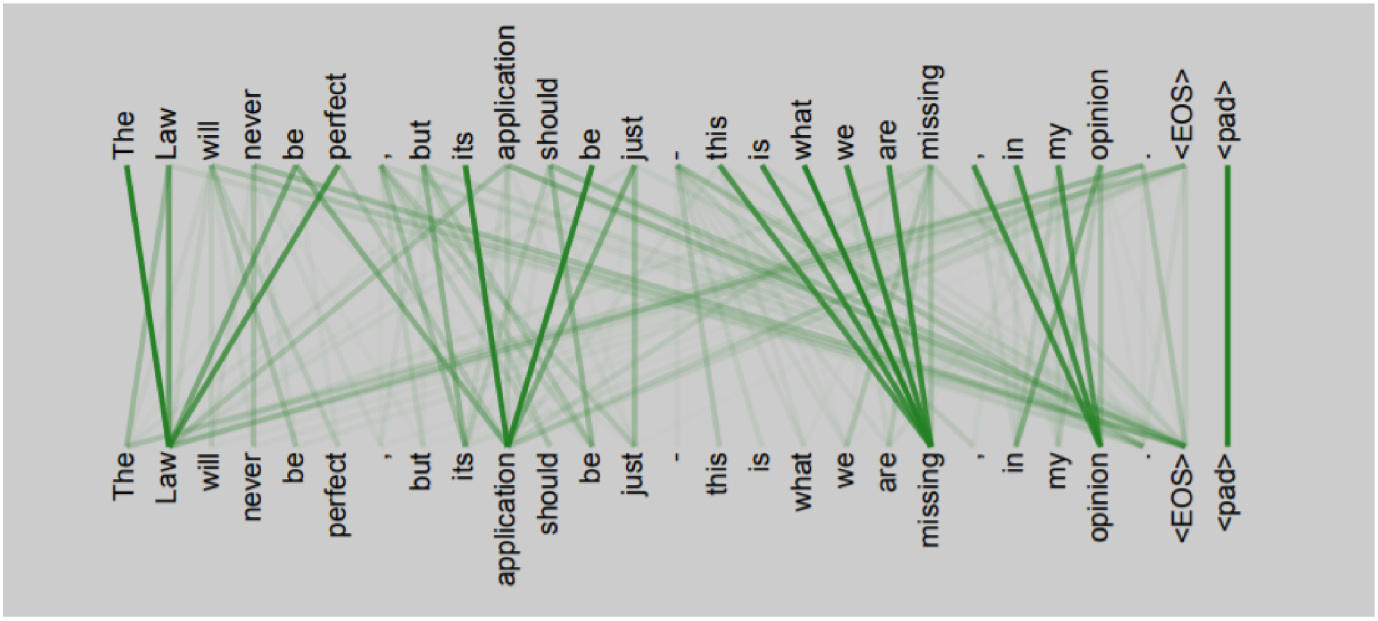
pos是位置，i是Embedding的某个维度, 2i指的偶数维度的Embedding, 2i+1指的是奇数维度的Embedding。i 是从 0到 dmodel / 2 ，所以对不同层的Embedding，sin和cos函数的波长是从 2pi 到 20000 pi。对于固定的i, pos变化就会引起Embedding以正弦或余弦变化。这样encode了以后，模型就能去对相对的位置进行建模。

Visualizations

Attention的另外一个好处是，**可视化和可解释性**加强了。我们从下图可以看到masking这个词，它主要和周围的词产生比较强的联系，尤其是和more difficult关系较大，这个是比较合理。我们还可以看到的是不同颜色代表不同的是不同的头，不同头的结果其实差别挺大的，这也是为什么多头注意力能带来收益的原因。



从下面这张图，我们可以看到attention基本可以捕捉句子的结构。



最后，作者比较了不同任务上Transformer的效果，我就贴出在机器翻译上的结果，显示Transformer确实在效果和效率上都有所提升。

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

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

- 表格问答完结篇：落地应用
- 大规模跨领域中文任务导向多轮对话数据集及模型CrossWOZ
- [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