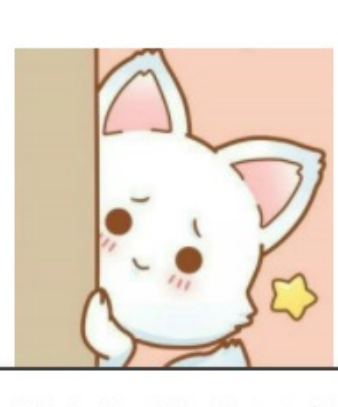
如何优雅地编码文本中的位置信息？三种positioanl encoding方法简述

原创 小鹿鹿lulu 夕小瑶的卖萌屋 2020-03-30 22:37:24 手机阅读

来自专辑
卖萌屋@自然语言处理



一只小狐狸带你解锁 **炼丹术&NLP** 秘籍

前言

相信熟悉BERT的小伙伴对**positional encoding**（位置表示） 肯定都不会陌生~ 虽然positional encoding只是BERT中比较小的一个组成部分，但是实际上却暗藏玄机。所以，今天呢我们就把positional encoding单独拎出来对其进行一个全面的剖析~~

- **Why？为什么需要positional encoding**
- **What？两种positional encoding方式：绝对位置编码与相对位置编码**
- **How？不同方法优缺点对比**

Why

众所周知，文本是时序型数据，词与词之间的**顺序关系**往往影响整个句子的含义。举个栗子：

小夕/是/一个/萌/妹子。
一个/妹子/是/萌/小夕？？
萌/小夕/是/一个/妹子？？

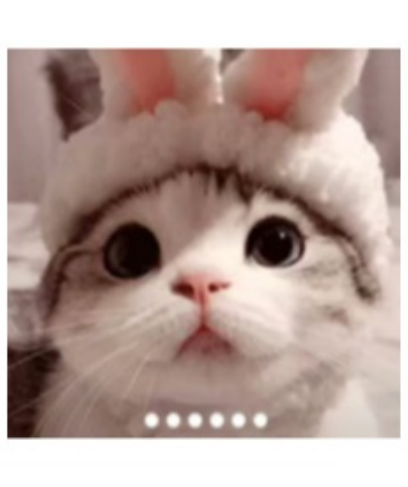
为了避免不必要的误会，所以我们在对文本数据进行建模的时候需要考虑词与词之间的顺序关系。

可是，要建模文本中的顺序关系必须要用positional encoding吗?

答案是No!

只有当我们使用对位置不敏感(position-insensitive)的模型对文本数据建模的时候，才需要额外使用positional encoding。

什么是**对位置敏感的模型**？？什么又是**对位置不敏感的模型**？？



如果模型的输出会随着输入文本数据顺序的变化而变化，那么这个模型就是关于位置敏感的，反之则是位置不敏感的。

用更清晰的数学语言来解释。设模型为函数$y=f(x)$,其中输入为一个词序列$x=\{x_1,x_2,...,x_n\}$，输出结果为向量y。对x的任意置换$x^{'}=\{x_{k_1},x_{k_2},...,x_{k_n}\}$,都有

$$f(x)=f(x^{'})$$

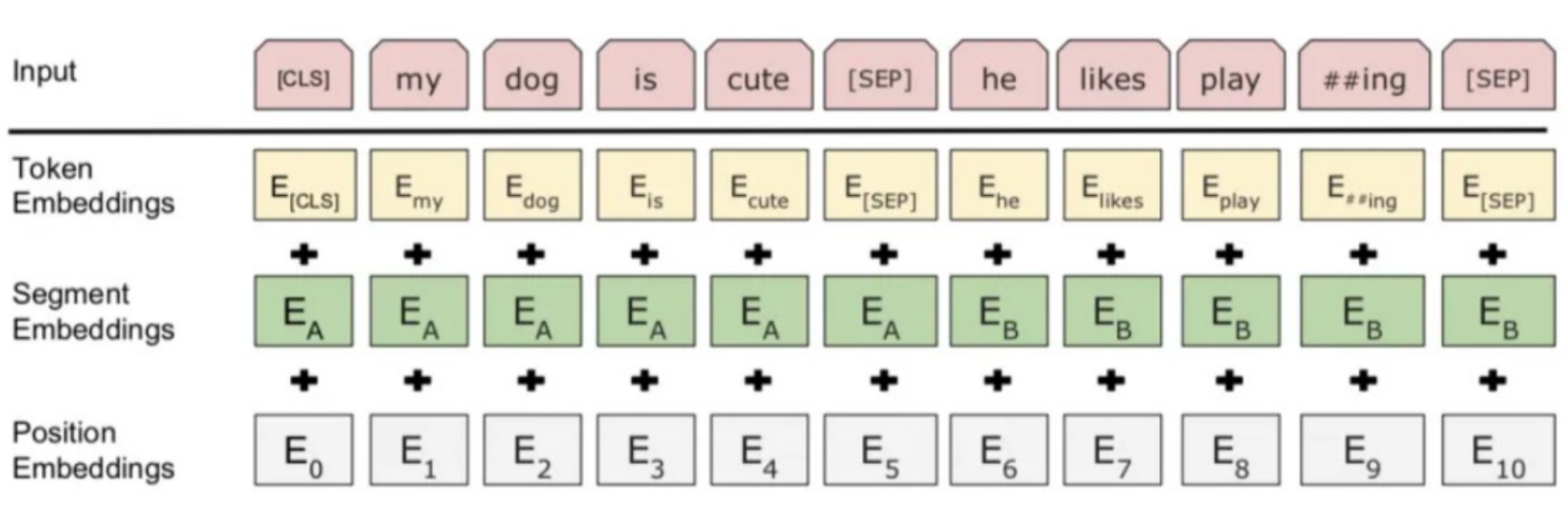
则模型f是关于位置不敏感的。

在我们常用的文本模型中，RNN和textCNN都是关于位置敏感的，使用它们对文本数据建模时，模型结构天然考虑了文本中词与词之间的顺序关系。而以attention为核心的transformer则是位置不敏感的，使用这一类位置不敏感的模型的时候需要额外加入positional encoding引入文本中词与词的顺序关系。

What

对于transformer模型的positional encoding有两种主流方式：

绝对位置编码



现在普遍使用的一种方法**Learned Positional Embedding编码绝对位置**，相对简单也很容易理解。直接对不同的位置随机初始化一个postion embedding，加到word embedding上输入模型，作为参数进行训练。

相对位置编码

使用绝对位置编码，不同位置对应的positional embedding固然不同，但是位置1和位置2的距离比位置3和位置10的距离更近，位置1和位置2与位置3和位置4都只相差1，这些关于位置的**相对含义**模型能够通过绝对位置编码get到吗？使用Learned Positional Embedding编码，位置之间没有约束关系，我们只能期待它隐式地学到，是否有更合理的方法能够显示的让模型理解位置的相对关系呢?

所以就有了另一种更直观地方法——相对位置编码。下面介绍两种编码相对位置的方法：Sinusoidal Position Encoding和Complex embedding。

Sinusoidal Position Encoding

使用正余弦函数表示绝对位置，通过两者乘积得到相对位置：

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

这样设计的好处是**位置$pos+k$的psotional encoding可以被位置pos线性表示，反应其相对位置关系。**

Sinusoidal Position Encoding虽然看起来很复杂，但是证明$pos+k$可以被pos线性表示，只需要用到高中的正弦余弦公式：**（注意：长公式可以左右滑动噢！）**

$$sin(\alpha+\beta)=sin\alpha\cdot cos\beta+cos\alpha\cdot sin\beta\quad（3）$$
$$cos(\alpha+\beta)=cos\alpha\cdot cos\beta-sin\alpha\cdot sin\beta\quad（4）$$

对于位置$pos+k$的positional encoding

$$PE_{(pos+k,2i)}=sin(w_i\cdot(pos+k))=sin(w_ipos)cos(w_ik)+cos(w_ipos)sin(w_ik)\quad（5$$
$$PE_{(pos+k,2i+1)}=cos(w_i\cdot(pos+k))=cos(w_ipos)cos(w_ik)-sin(w_ipos)sin(w_ik)\quad（$$

其中$w_i=\frac{1}{10000^{2i/d_{model}}}$

将公式（5）（6）稍作调整，就有

$$PE_{(pos+k,2i)}=cos(w_ik)PE_{(pos,2i)}+sin(w_ik)PE_{(pos,2i+1)}\quad（7）$$
$$PE_{(pos+k,2i+1)}=cos(w_ik)PE_{(pos,2i+1)}-sin(w_ik)PE_{(pos,2i)}\quad（8）$$

注意啦，pos和$pos+k$相对距离k是常数，所以有

$$\begin{bmatrix}PE_{(pos+k,2i)}\\PE_{(pos+k,2i+1)}\end{bmatrix}=\begin{bmatrix}u & v\\-v & u\end{bmatrix}\times\begin{bmatrix}PE_{(pos,2i)}\\PE_{(pos,2i+1)}\end{bmatrix}\quad（9）$$

其中$u=cos(w_i\cdot k),v=sin(w_i\cdot k)$为常数。

所以PE_{pos+k}可以被PE_{pos}线性表示。

计算PE_{pos+k}和PE_{pos}的内积，有

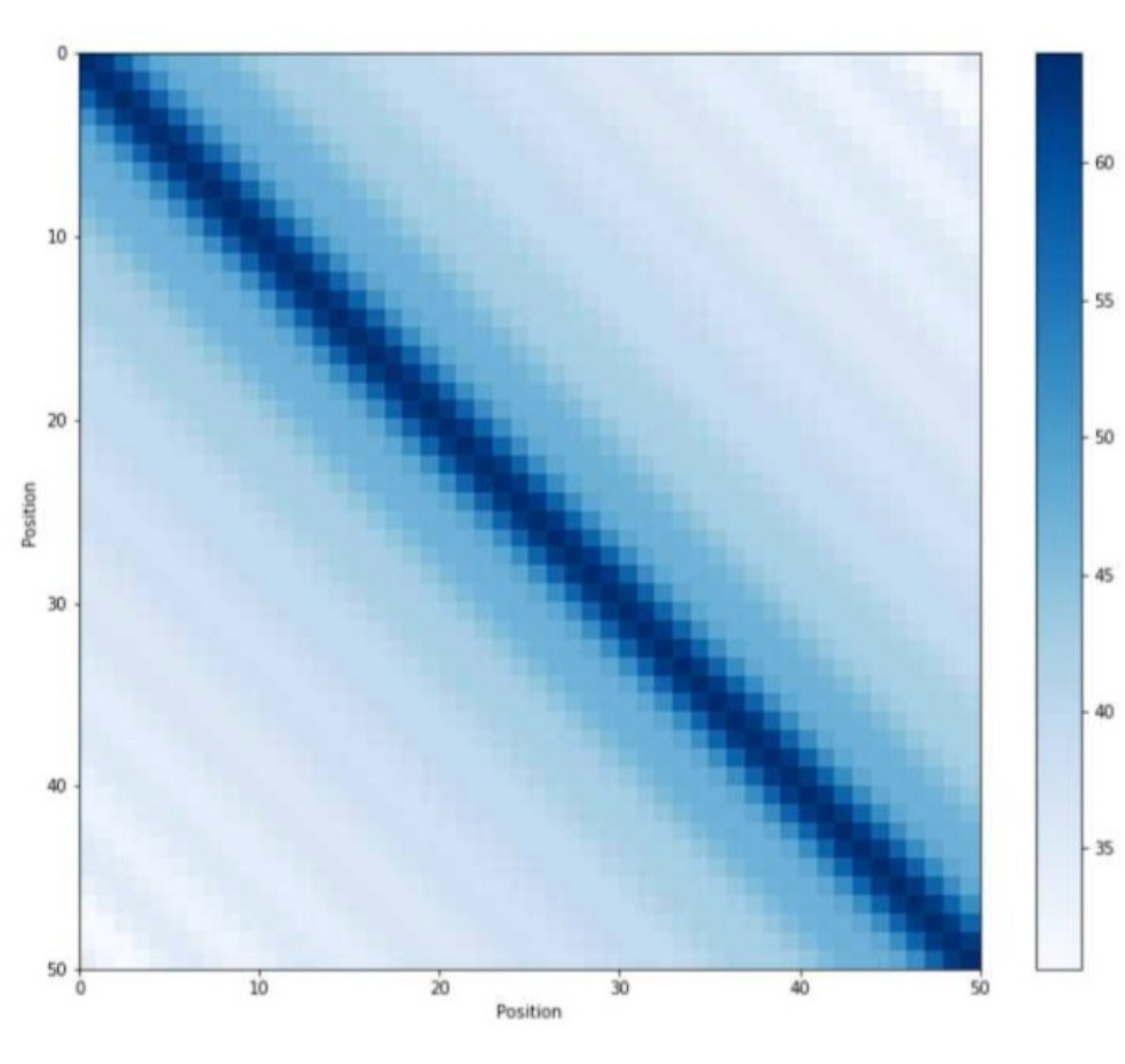
$$\begin{aligned}PE_{pos}\cdot PE_{pos+k}&=\sum_{i=0}^{\frac{d}{2}-1}sin(w_ipos)\cdot sin(w_i(pos+k))+cos(w_ipos)\cdot cos(w_i(pos+k))\\&=\sum_{i=0}^{\frac{d}{2}-1}cos(w_i(pos-(pos+k))\\&=\sum_{i=0}^{\frac{d}{2}-1}cos(w_ik)\quad（10）\end{aligned}$$

其中$w_i=\frac{1}{10000^{2i/d_{model}}}$.

PE_{pos+k}和PE_{pos}的内积会随着相对位置的递增而减小，从而表征位置的相对距离。但是不难发现，由于距离的对称性，Sinusoidal Position Encoding虽然能够反映相对位置的距离关系，但是无法区分方向/(ToT)/~~

$$PE_{pos+k}PE_{pos}=PE_{pos-k}PE_{pos}\quad（11）$$

更加直观的对其可视化[1]，可以看到图像关于$y=x$对称，无法区分前后关系。



Complex embedding

为了更好的让模型捕获更精确的相对位置关系，比如相邻，前序（precedence）等，ICLR 2020发表的文章《Encoding Word Oder In Complex Embeddings》使用了复数域的连续函数来编码词在不同位置的表示。

不管是Learned Postional Embdedding还是Sinusoidal Position Encoding，某个词w_j在pos位置上的表示为其word embedding加上对应位置的embedding，即：

$$f(j,pos)=f_{we}(j)+f_{pe}(pos)\quad（12）$$

$f_{pe}(\cdot)$同word embedding $f_{we}(\cdot)$都是从整数域N到实数域R^D的一个映射。

对于word embedding来说，这样的设计是合理的。因为不同词的index是独立的，仅和我们具体使用的词典怎么排序有关系，某个词是否在另外一个词前面或者相邻没有任何的信息。但是位置的index并不是满足独立的假设，其顺序关系对文本的正确理解有非常重要的影响。

所以，**为了解决pos index的依赖问题（position-insensitive problem），文章使用了关于位置的连续函数来表征词w_j在pos的表示**，即：

$$f(j,pos)=g_j(pos)\in R^D\quad（13）$$

把公式(13)展开有

$$[g_{j,1}(pos),g_{j,2}(pos),...,g_{j,D}(pos)]\in R^D\quad（14）$$

其中$g_{j,d}(\cdot)\in F:N\rightarrow R,d\in 1,2,..,D$是关于位置$pos$在复数域上的函数。

为了让上述函数更好的表征位置的相对信息，要求函数$g_{j,d}$满足以下两个性质：

1. Position-free offset transformation

存在一个函数$Transform_k(\cdot)=Transform(k,\cdot)$使得

$$g(pos+k)=Transform_k(g(pos))\quad（15）$$

其中在不影响理解的情况下，我们把$g_{j,d}$简写成$g(\cdot)$。也就是说，词w_j在pos或者pos+k的表示可以**由只和相对位置k有关的一个变换得到，而与具体这个词无关。**

2. Boundedness

要求函数g有界。非常合理的一个限制。

最后，论文证明了在复数域上满足这个两个条件的函数一定为下面这样的形式：

$$g(pos)=z_2z_1^{pos},z_1,z_2\in C,|z_1|\leq 1\quad（16）$$

将其改写成指数的形式，则为

$$g(pos)=r\cdot e^{i(w\cdot pos+\theta)}\quad（17）$$

其中，r为振幅，w为角频率，θ为初相，都是需要学习的参数~~

将式（17）代入（14）有

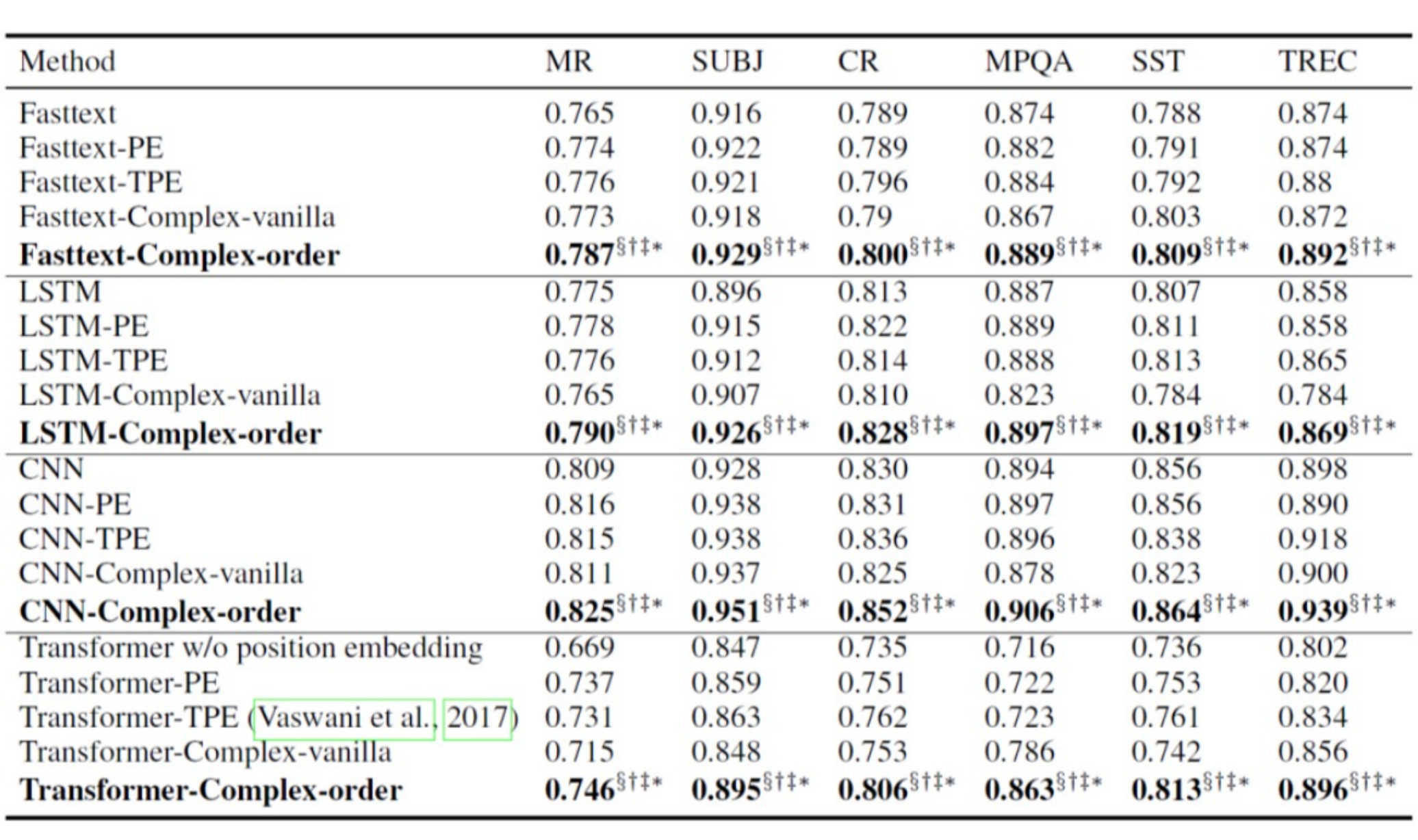
$$[r_{j,1}e^{i(w_{j,1}\cdot pos+\theta_{j,1})},r_{j,2}e^{i(w_{j,2}\cdot pos+\theta_{j,2})},...,r_{j,D}e^{i(w_{j,D}\cdot pos+\theta_{j,D})}]$$

要表征词w_j在pos上的embedding，需要学习的参数有$r=[r_{j,1},r_{j,2},...,r_{j,D}]$，$w=[w_{j,1},w_{j,2},...,w_{j,D}]$以及$\theta=[\theta_{j,1},\theta_{j,2},...,\theta_{j,D}]$。以此类推，要表示词表中所有的词，那么需要学习的参数量为$3\times vocab_size\times dimension$😮.由于参数量较大，论文后续还提出了一些减小参数量的方法，有兴趣的同学可以看具体查阅原文哦~~~

How

以上三种positional encoding都不同程度、各有侧重的编码了文本数据中的顺序关系，那么到底哪个更好？我们在平时使用的时候应该如何选择呢?

结果导向的话，肯定是哪种方法效果好选哪种啦~~在《Attention is all you need》[2]里面提到，Learned Positional Embedding和Sinusoidal Position Encoding两种方式的效果没有明显的差别。在论文[3]，实验结果表明使用Complex embedding相较前两种方法有较明显的提升。（不过介于这个方法还比较新，大家可以多多尝试对比）。



Method	MR	SUBJ	CR	MPQA	SST	TREC
Fasttext	0.765	0.916	0.789	0.874	0.788	0.874
Fasttext-PE	0.774	0.922	0.789	0.882	0.791	0.874
Fasttext-TPE	0.776	0.921	0.796	0.884	0.792	0.88
Fasttext-Complex-vanilla	0.773	0.918	0.79	0.867	0.803	0.872
Fasttext-Complex-order	**0.787**§†‡*	**0.929**§†‡*	**0.800**§†‡*	**0.889**§†‡*	**0.809**§†‡*	**0.892**§†‡*
LSTM	0.775	0.896	0.813	0.887	0.807	0.858
LSTM-PE	0.778	0.915	0.822	0.889	0.811	0.858
LSTM-TPE	0.776	0.912	0.814	0.888	0.813	0.865
LSTM-Complex-vanilla	0.765	0.907	0.810	0.823	0.784	0.784
LSTM-Complex-order	**0.790**§†‡*	**0.926**§†‡*	**0.828**§†‡*	**0.897**§†‡*	**0.819**§†‡*	**0.869**§†‡*
CNN	0.809	0.928	0.830	0.894	0.856	0.898
CNN-PE	0.816	0.938	0.831	0.897	0.856	0.890
CNN-TPE	0.815	0.938	0.836	0.896	0.838	0.918
CNN-Complex-vanilla	0.811	0.937	0.825	0.878	0.823	0.900
CNN-Complex-order	**0.825**§†‡*	**0.951**§†‡*	**0.852**§†‡*	**0.906**§†‡*	**0.864**§†‡*	**0.939**§†‡*
Transformer w/o position embedding	0.669	0.847	0.735	0.716	0.736	0.802
Transformer-PE	0.737	0.859	0.751	0.722	0.753	0.820
Transformer-TPE (Vaswani et al., 2017)	0.731	0.863	0.762	0.723	0.761	0.834
Transformer-Complex-vanilla	0.715	0.848	0.753	0.786	0.742	0.856
Transformer-Complex-order	**0.746**§†‡*	**0.895**§†‡*	**0.806**§†‡*	**0.863**§†‡*	**0.813**§†‡*	**0.896**§†‡*

从方法的可理解性上，相比相对位置编码的两种方法，Learned Positional Embedding更加的简单直接，易于理解。**从参数维度上**，使用Sinusoidal Position Encoding不会引入额外参数，Learned Positional Embedding增加的参数量会随max_seq_length线性增长，而Complex Embedding在不做优化的情况下，会增加三倍word embedding的参数量。**在可扩展性上**，Learned Positional Embedding可扩展性较差，只能表征在max_seq_length以内的位置，而另外两种方法没有这样的限制，可扩展性更强。

讲了这么多，相信大家对positional encoding已经有了充分的理解~~至于到底应该如何选择，还是需要基于大家对方法的理解实际问题实际分析哦🙃

参考文献

[1] 可视化: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/#what-is-positional-encoding-and-why-do-we-need-it-in-the-first-place
[2] Attention is all you need: https://arxiv.org/pdf/1706.03762.pdf
[3] Complex Embeddings: https://openreview.net/pdf?id=Hke-WTVtwr