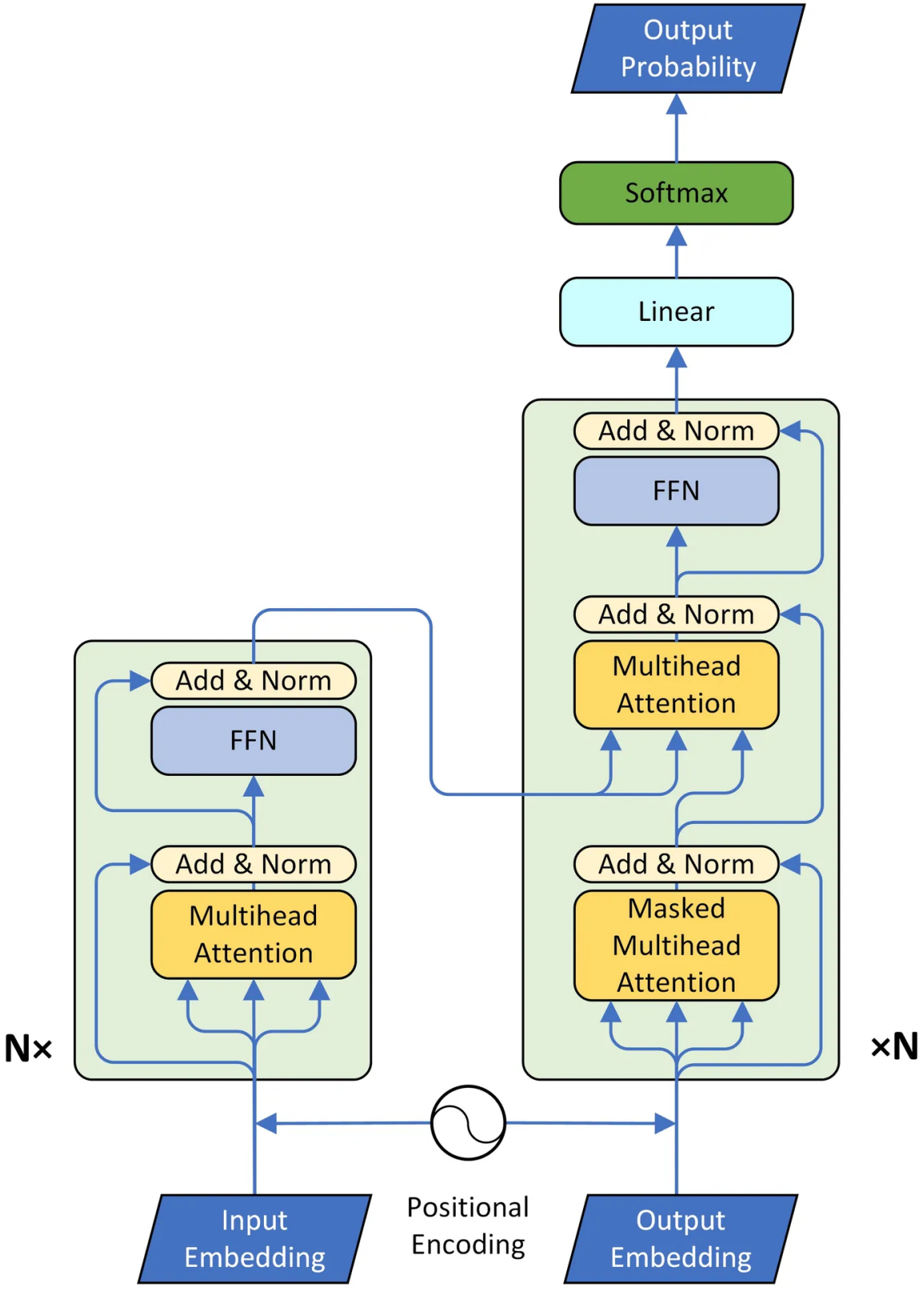


# Attention和Transformer

Elijha
忙于人工智能的人类智障，深度学习算法研究员暨神经网络算命工程师。
411 人赞同了该文章

Transformer是Attention is all you need这篇论文里提出的一个新框架。因为最近在MSRA做时序相关的研究工作，我决定写一篇总结。本文主要讲一下Transformer的网络结构，因为这个对RNN的革新工作确实和之前的模型结构相差很大，而且听我的mentor Shujie Liu老师说在MT（machine translation）领域，transformer已经几乎全面取代RNN了。包括前几天有一篇做SR（speech recognition）的工作也用了transformer，而且SAGAN也是……总之transformer确实是一个非常有效且应用广泛的结构，应该可以算是即seq2seq之后又一次“革命”。

## 写在前面

本文默认读者已经对seq2seq有一定了解，知道其结构，输入和输出的tensor shape，以及embedding的方法。如果不太了解，请先找一些相关资料，知乎和CSDN上面已经有很多啦。

## Seq2seq

下面这段是我论文里的一段简短的介绍, 我就不翻译了...

> A sequence-to-sequence model converts an input sequence $(x_1, x_2, \ldots, x_T)$ into an output sequence $(y_1, y_2, \ldots, y_{T'})$, and each predicted $y_t$ is conditioned on all previously predicted outputs $y_1, \ldots, y_{t-1}$. In most cases, these two sequences are of different lengths ( $T \neq T'$ ).
> In NMT, this conversion translates the input sentence in one language into the output sentence in another language, based on a conditional probability $p(y_1, \ldots, y'_T | x_1, \ldots, x_T)$ :
> $h_t = encoder(h_{t-1}, x_t)$
> $s_t = decoder(s_{t-1}, y_{t-1}, c_t)$
> where $c_t$ is the context vector calculated by an attention mechanism:
> $c_t = attention(s_{t-1}, \mathbf{h})$
> thus $p(y_1, \ldots, y'_T | x_1, \ldots, x_T)$ can be computed by
> $p(y_1, \ldots, y'_T | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | \mathbf{y_{<t}}, \mathbf{x})$
> and
> $p(y_t | \mathbf{y_{<t}}, \mathbf{x}) = softmax(f(s_t))$
> where $f(\cdot)$ is a fully connected layer. For translation tasks, this softmax function is among all dimensions of $f(s_t)$ and calculates the probability of each word in the vocabulary. However, in the TTS task, the softmax function is not required and the hidden states $\mathbf{s}$ calculated by decoder are consumed directly by a linear projection to obtain the desired spectrogram frames.

这是2014年Google提出的一个模型（论文链接），知乎和各大博客网站上有很多介绍，这里就不赘述了。简单来说，seq2seq又两个RNN组成，一个是encoder，一个是decoder。拿MT举例子，比如我们要把源语言“我爱中国”翻译

成目标语言“I love China”，那么定义输入序列：

$X=(x_0,x_1,x_2,x_3)$，其中 $x_0=$“我”，$x_1=$“爱”，$x_2=$“中”，$x_2=$“国”；

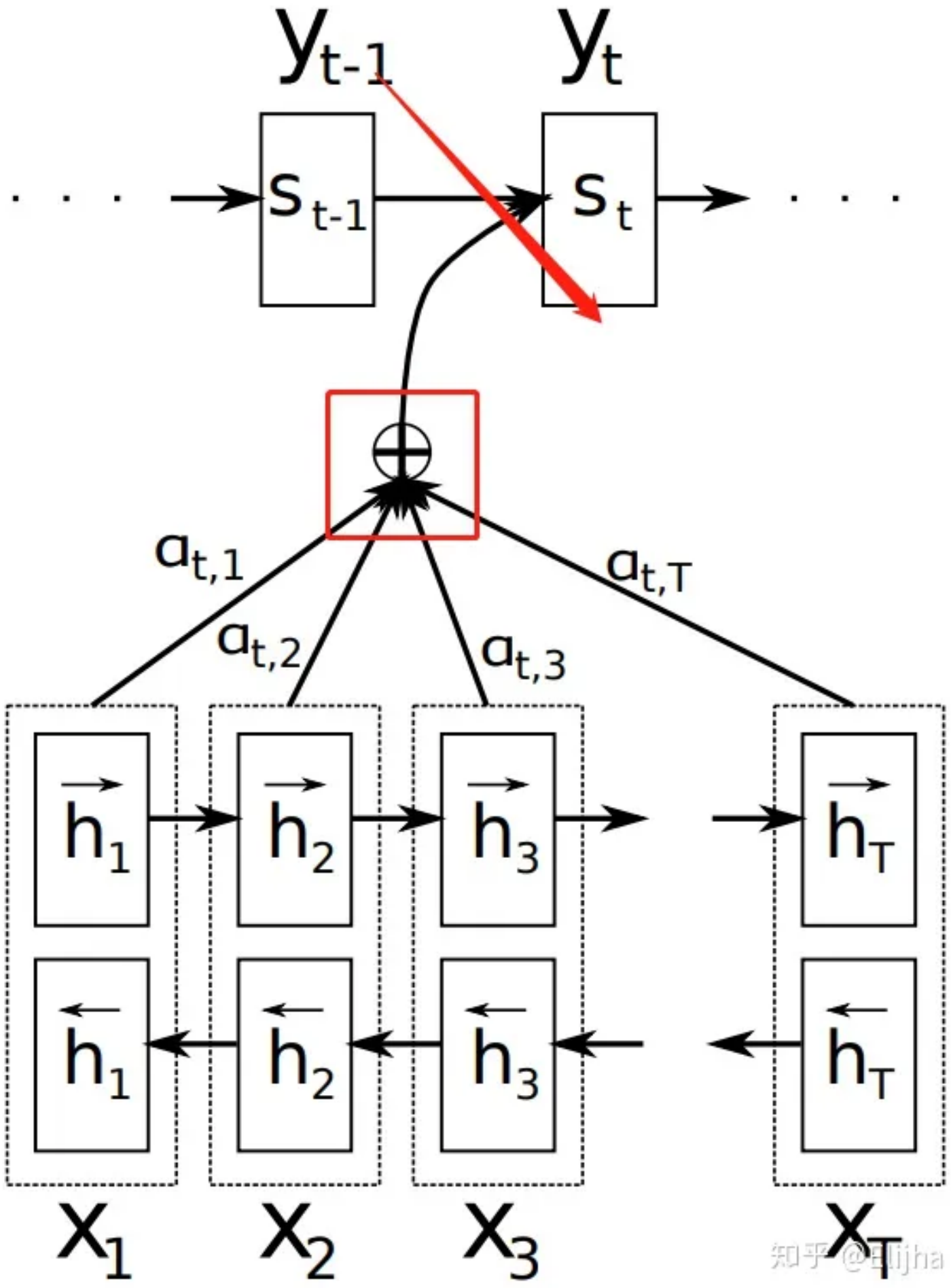
另外定义目标序列：

$T=(t_0,t_1,t_2)=$” $I\ love\ China$ ”。

通过encoder，把 $X=(x_0,x_1,x_2,x_3)$ 映射为一个隐层状态 $h$ ，再经由decoder将 $h$ 映射为 $Y=(y_0,y_1,y_2)$（注意这里向量长度可以发生变化，即与输入长度不同）；最后将Y与T做loss（通常为交叉熵），训练网络。

## Attention

同样是在MT问题中，在seq2seq的基础上提出了attention机制（Bahadanau attention）（论文地址）。现在由于性能相对没有attention的原始模型太优越，现在提到的seq2seq一般都指加入了attention机制的模型。同样上面的问题，通过encoder，把 $X=(x_0,x_1,x_2,x_3)$ 映射为一个隐层状态 $H=(h_0,h_1,h_2,h_3)$ ，再经由decoder将 $H=(h_0,h_1,h_2,h_3)$ 映射为 $Y=(y_0,y_1,y_2)$。这里精妙的地方就在于，Y中的每一个元素都与H中的所有元素相连，而**每个元素通过不同的权值给与Y不同的贡献。**


$y_{t-1}$
$y_t$
$s_{t-1}$
$s_t$
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$
$\overrightarrow{h}_2$
$\overrightarrow{h}_3$
$\overrightarrow{h}_T$
$\overleftarrow{h}_1$
$\overleftarrow{h}_2$
$\overleftarrow{h}_3$
$\overleftarrow{h}_T$
$x_1$
$x_2$
$x_3$
$x_T$

上图是Bahadanau attention（论文地址）的示意图。

1. 关键在于红框里面的部分，即attention，后面再讲。
2. 红框下面是encoder，输入 $\boldsymbol{X}=(x_0,x_1,x_2,\ldots,x_{T_x})$ ，通过一个双向LSTM得到两组$h^{\leftarrow}$和 $h^{\rightarrow}$ （向左和向右），然后concat起来得到最终的 $\boldsymbol{H}=(h_0,h_1,h_2,\ldots,h_{T_x})$ 。
3. 红框上面是decoder。以 $t$ 时刻为例，输入共有三个： $s_{t-1}$ ， $y_{t-1}$ ， $c_t$ 。其中$s_{t-1}$是上一个时刻的hidden state（一般用 $h$ 表示encoder的hidden state，用 $s$ 表示decoder的hidden state）； $y_{t-1}$ 是上一个时刻的输出，用来当做这个时刻的输入； $c_t$ 在图中没有，就是红框里得到的加权和，叫做context，即**由所有的encoder output（即 $h$ ，不定长）得到一个定长的向量，代表输入序列的全局信息**，作为当前decoder step的context（上下文）。计算方法为： $c_i=\sum_{j=1}^{T_x}\alpha_{ij}h_j$ ，其中$\alpha_{ij}$是权重，又称为**alignment**； $h$ 就是encoder所有step的hidden state，又叫做**value**或者**memory**； $i$ 代表decoder step， $j$ 代表encoder step。

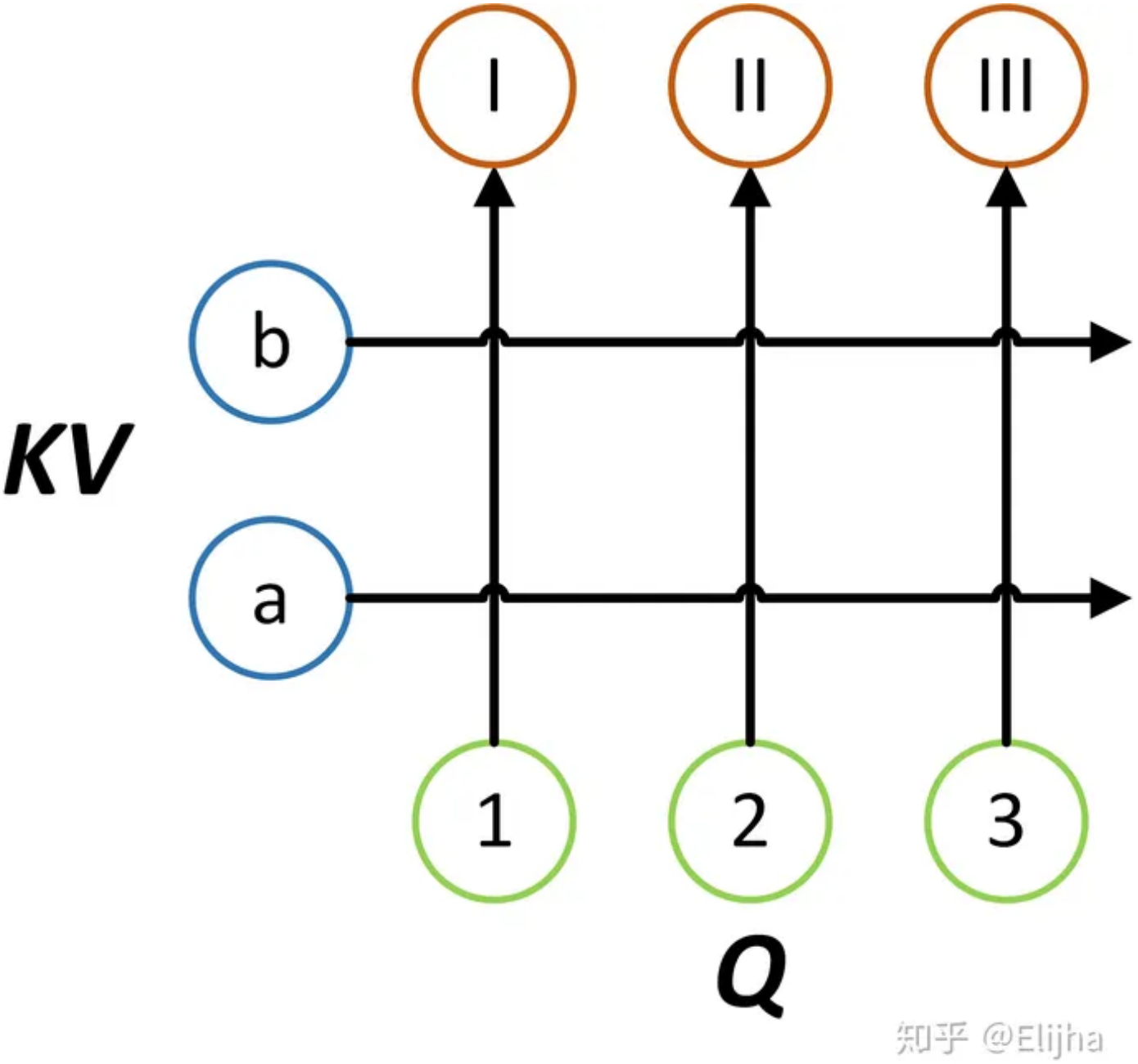
那么，这个权重 $\alpha_{ij}$ 如何计算呢？直接上公式：

$\alpha_{ij}=\frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}\exp(e_{ik})}$ ，其中 $e_{ij}=a(s_{i-1},h_j)$ 。看上去有点复杂，先说第一个公式，其实就是一个softmax，因为我们要算一组权重，这组权重的和为1。那这个 $e_{ij}$ 又是什么呢？是通过某种度量 $a(\bullet)$ 计算出来的$s_{i-1}$ 和 $h_j$ 的**相关程度**。即对于某个既定的decoder step，计算上个时刻的hidden state和所有encoder step的输出的相关程度，并且用softmax归一化；这样，相关度大的 $h$ 权重就大，在整个context里占得比重就多，decoder在当前step做解码的时候就越重视它（这就是attention的思想）。

那最后一个问题，这个度量 $a(\bullet)$ 怎么算呢？很神奇的是这部分论文里没说……可能是我看的论文少，大家其实都已经心知肚明了吧……根据tf里的代码，一般来说分为这么几步：

1. 对 $s_{i-1}$ 做一个线性映射，得到的向量叫做**query**，记做 $q_i$ ；
2. 对 $h_j$ 做一个线性映射，得到的向量叫做**key**，记做 $k_j$ ；
3. $e_{ij}=v^T\cdot(q_i+k_j)$ 。$k_j$和$q_i$的维度必须相同，记为 $d$ ； $v$ 是一个 $d\times 1$ 的向量。

上面三步中，**前两步的线性映射**和**第三步的 $v$ 都是可以训练**的。所以用下面这张图大概表示一下**query与key的计算相关性继而得到权重，并且用这个权重对value计算加权和**的过程:



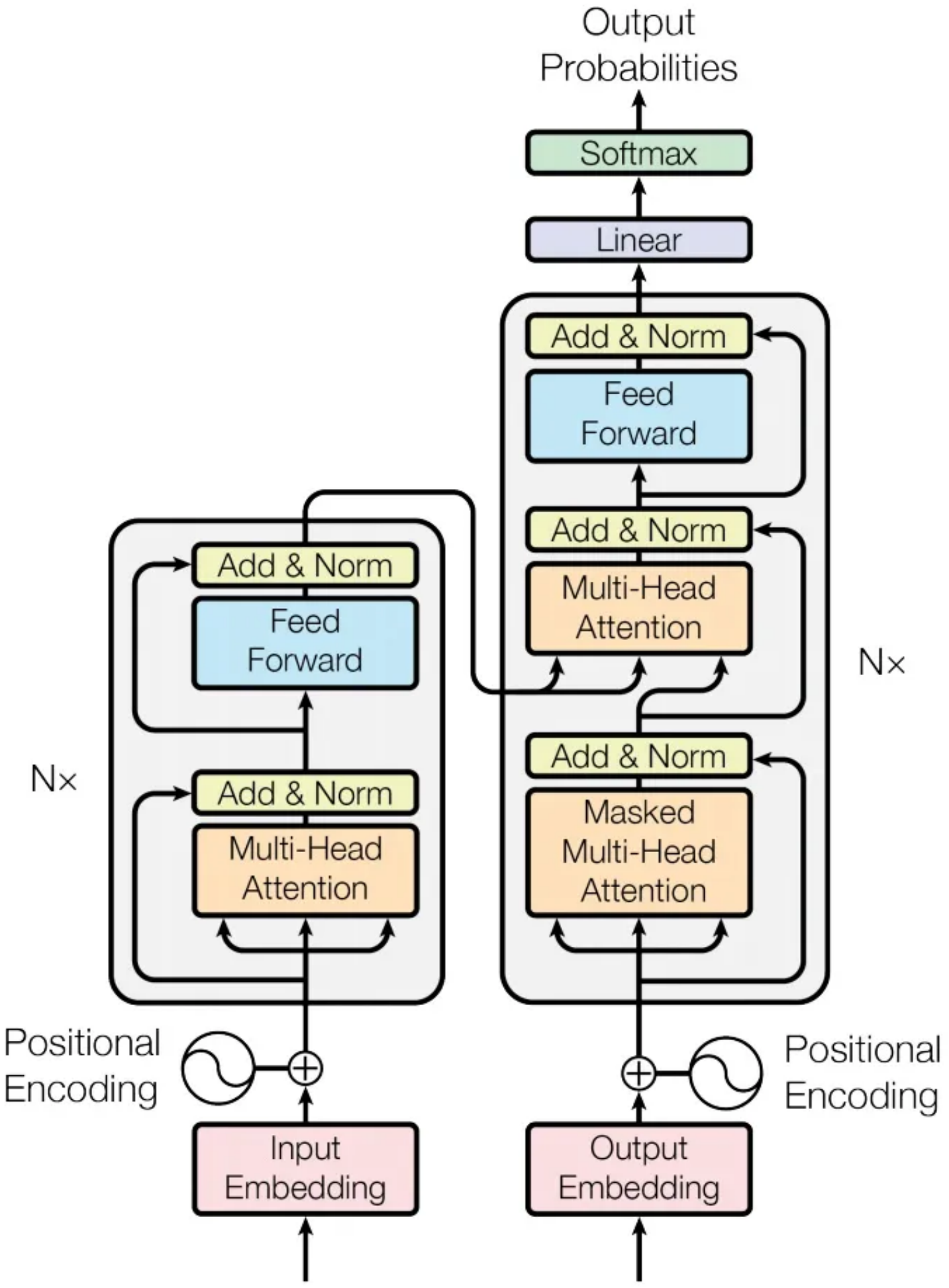

多说几点：

1. 上面第三步中，query和key是做的加法，然后通过一个权重矩阵变为一个标量，被称为“加性attention”；相对的，可以直接通过向量乘法得到一个标量，叫做“乘性attention”；
2. 后来出现了一种升级版，叫做local sensitive attention，思想是相邻 $\alpha_{ij}$ 之间的关系会相对较大，因此为了捕捉这种关系对alignment进行卷积。
3. query由很多种，这里是上个decode step的hidden state，也有论文里用的是当前的hidden state（ $s_i$ ），还有把 $s_{i-1}$ 和当前时刻的输入 $y_{i-1}$ concat起来一起用的。我做的tts，如果加入 $y_{i-1}$ 的话就很难work，原因可能是二者不在同一个空间里，强行concat会干扰训练，所以还是具体情况具体分析啦。

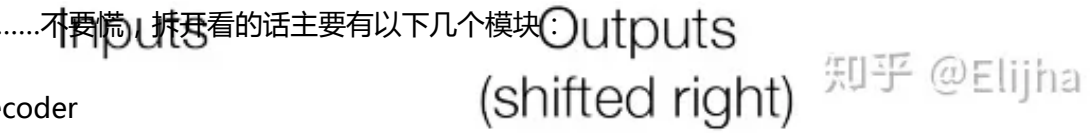
总结一下，attention就是算一个encoder output的加权和，叫做context；计算方法为，query和key计算相关程度，然后归一化得到alignment，即权重；然后用alignment和memory算加权和，得到的向量就是context。

## Transformer

讲了那么多，终于说完了attention，可以进入到transformer部分了。Transformer就是一个升级版的seq2seq，也是由一个encoder和一个decoder组成的；encoder对输入序列进行编码，即$\boldsymbol{X}=(\boldsymbol{x}_0,\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_{T_x})$变成$\boldsymbol{H}=(\boldsymbol{h}_0,\boldsymbol{h}_1,\boldsymbol{h}_2,\ldots,\boldsymbol{h}_{T_x})$；decoder对$\boldsymbol{H}=(\boldsymbol{h}_0,\boldsymbol{h}_1,\boldsymbol{h}_2,\ldots,\boldsymbol{h}_{T_x})$进行解码，得到$\boldsymbol{Y}=(\boldsymbol{y}_0,\boldsymbol{y}_1,\boldsymbol{y}_2,\ldots,\boldsymbol{y}_{T_y})$。但是神奇的是，encoder和decoder都不用RNN，而且换成了多个attention。先看图：


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding

乍一看好懵逼，结构好复杂……不要慌，拆开看的话主要有以下几个模块：




1. 左右分别是encoder和decoder
2. enc和dec的底部是embedding；而embedding又分为两部分：**input embedding**和**positional embedding**；其中**input embedding就是seq2seq中的embedding。**而为什么要positional embedding呢？因为transformer中只有attention；回想一下attention，任意一对(query, key)的计算都是完全一样的，不像CNN和RNN，有一个位置或者时序的差异：CNN框住一块区域，随着卷积核移动，边缘的少量点会跟着有序变化；RNN更明显了，不同时序的 $h_t$ 和 $s_t$ 不同，而且是随着输入顺序不同（正序，倒序）而不同。因此为了体现出时序或者在序列中的位置差异，要对input加入一定的位置信息，即positional embedding。而这个positional embedding的公式看了也是超级懵逼。我后面再讲。
3. enc和dec的中部分别是两个block，分别输入一个序列、输出一个序列；这两个block分别重复 $N$ 次。enc的每个block里有两个子网，分别是multihead attention和feedforward network（ffn），先别管都在干啥；dec的block里有三个子网，分别是两个multihead attention和一个ffn。这些子网后面都跟了一个add&norm，即像resnet一样加一个跳边，然后做一个layer norm。
4. dec最后还有一个linear和softmax。

好了，网络拆完了，其实懵逼的地方就三个：multihead attention和ffn，以及那个公式非常奇怪而且看不出来道理的positional embedding。先说positional embedding，我重新写一下公式：

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

我们重新回忆一下enc和dec输入序列的shape： $[T, d_{model}]$ 。即每个时刻的 $x_i$ 都是$d_{model}$ 维的，这里 $d_{model} = 512$ 。因此， $pos \in [0, T)$ ， $i \in [0, d_{model})$ 。即对于输入的这个$[T, d_{model}]$维的tensor，其中的每个标量都有一个独特的编码结果。其实我不太懂为什么要这么编码，希望有懂的大神指点我一下。另外还有一种编码方式，类似于input embedding，只不过id-->embedding变成了No-->embedding，即一个序列中的每个位置（No.1, No.2, ... ,No.T）都对应一个编码。而且听我的老师说，他们做过实验，发现这两种编码对于最终的训练结果是差不多的。**但是我最近的实验发现，第二种方法在处理长序列的时候会在序列末尾出现错误，比如合成的音频末尾丢失音节。而三角函数的PE可以很好地克服这一点。**

第二点ffn。其实ffn很简单，就是对一个输入序列 $(x_0, x_1, \dots, x_T)$ ，对每个 $x_i$ 都进行一次channel的重组： $512 \rightarrow 2048 \rightarrow 512$ ，可以理解为对每个$x_i$进行两次线性映射，也可以理解为对整个序列做一个1*1的卷积。

第三点，multihead attention。这是这篇文章的一个创新，下面详细讲一讲。

原始的attention, 就是一个query (以下简称Q) 和一组key (以下简称K) 算相似度, 然后对一组value (以下简称V) 做加权和; 假如每个Q和K都是512维的向量, 那么这就相当于在512维的空间里比较了两个向量的相似度. 而multihead就相当于把这个512维的空间人为地拆成了多个子空间, 比如head number=8, 就是把一个高维空间分成了8个子空间, 相应地V也要分成8个head; 然后在这8个子空间里分别计算Q和K的相似度, 再分别组合V. 这样可以让attention能从多个不同的角度进行结合, 这对于NMT是很有帮助的, 因为我们在翻译的时候源语言和目标语言的词之间并不是一一对应的, 而是受很多词共同影响的. 每个子空间都会从自己在意的角度或者因素去组合源语言, 从而得到最终的翻译结果.

## Transformer带来的其他优点

- 并行计算, 提高训练速度

Transformer用attention代替了原本的RNN; 而RNN在训练的时候, 当前step的计算要依赖于上一个step的hidden state的, 也就是说这是一个sequential procedure, 我每次计算都要等之前的计算完成才能展开. 而Transformer不用RNN, 所有的计算都可以并行进行, 从而提高的训练的速度.

- 建立直接的长距离依赖

原本的RNN里, 如果第一帧要和第十帧建立依赖, 那么第一帧的数据要依次经过第二三四五...九帧传给第十帧, 进而产生二者的计算. 而在这个传递的过程中, 可能第一帧的数据已经产生了biased, 因此这个交互的速度和准确性都没有保障. 而在Transformer中, 由于有self attention的存在, 任意两帧之间都有直接的交互, 从而建立了直接的依赖, 无论二者距离多远.

## Transformer没有解决的问题

上面说到, Transformer可以使模型进行并行训练, 但是仍然是一个autoregressive的模型; 也就是说, 任意一帧的输出都是要依赖于它之前的所有输出的. 这就使得inference的过程仍然是一个sequential procedure. 这也是当前绝大多数seq2seq模型的问题, 不论是ConvS2S等等. 因此如何打破这个特点, 建立Non autoregressive model, 并且取得和autoregressive model接近的性能是一个重要的问题.

编辑于 2018-11-12 02:04

神经机器翻译(NMT) 自然语言处理



63 条评论

默认

最新

夕小瑶

建议哈，最后一段补充一下自回归有什么不好的地方。毕竟既然说transformer的一点不足是decoding依然是个自回归过程，那么当然首先要说明自回归相比其他方法的缺陷这样逻辑才完整哦

2018-10-27

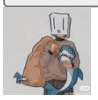
叮叮当当

我认为：在翻译这样的生成任务，自回归倒不是说不好吧；只是在做inference的时候，是一步一步往前走，效率不太行。

但是现在应该又找不到其他不是一步一步走的方式，还效果好。

2019-07-31

回复 1



Elijha
作者

好的哦, 抱歉刚看到, 下次我注意~

2018-11-12

回复 喜欢

鲜橙

这是第一篇我觉得解释清楚了key，query的文章

2020-04-06

回复 2

青青子衿
先赞为敬
2018-06-29

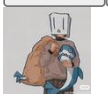
回复 1

弓长君

一开始yT要不要改成yT ‘，因为长度一个是T，一个是T ‘

2020-04-15

●回复 ♥喜欢



Elijha
作者

你是说那段英文里的y'T? 这个好像是写错了, 应该是yT'

2020-04-16

●回复 ♥喜欢

张小白

写得很好，谢谢！

2019-12-01

●回复 ♥喜欢

咖啡里安眠

想问一下关于QKV的概念是源于哪一篇论文，我想去细看一下。

2019-01-13

●回复 ♥喜欢

弓长君

这个梗可以参考一下attention的原文。。。
[1706.03762] Attention Is All You Need - http://arXivarxiv.org › cs

2020-04-16

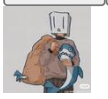
回复 喜欢

哼哼哈哈

而multihead就相当于把这个512维的空间人为地拆成了多个子空间 这句话是不是有问题？

2018-12-31

回复 喜欢



Elijha

作者

▸

哼哼哈哈

呃 我觉得是这样

2019-01-01

回复 喜欢

哼哼哈哈

这里不太懂，看论文感觉作者表达的不是这个意思

2018-12-31

回复 喜欢

Quokka

用三角函数是为了能够推断相对位置（对任意位置 k，它的位置向量都可以表示成相邻位置的位置向量的线性变换，变换矩阵与 k 无关，是个常量）。奇偶位置分开是为了错开正弦函数的波峰和波谷，避免相邻位置的位置向量太像，被神经网络搞混。

2018-12-20

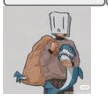
回复 喜欢

弓长君

感觉回到了n年前上信号处理课了。

2020-04-16

回复 喜欢



Elijha
作者
抱歉木有看到评论提示 刚看到
2019-01-01

回复 喜欢

展开其他 1 条回复 ❯

CeCiL-cHaRlIe
加个微信吧dongrixinyu89
2018-11-12

回复 喜欢

神特喵皮皮猴
非常感谢🙏，看完对它又多了些理解
2018-11-09

回复 喜欢