Transformer

接前一讲

- 卷积神经网络（Convolution Neural Network，CNN）
- 循环神经网络（Recurrent Neural Network，RNN）
  - 长短期记忆网络（Long-short Term Memory，LSTM）
  - 门控循环单元（Gate Recurrent Unit，GRU）
- 注意力机制（Attention）
- **Transformer**

Transformer

什么是Transformer

Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution。

Transformer抛弃了传统的RNN和CNN，整个网络结构完全由Attention机制组成。更准确的讲，Transformer由且仅由self-attention和Feed Forward Neural Network组成。
Transformer 被广泛应用于NLP领域，并正在向其他领域推广

Transformer

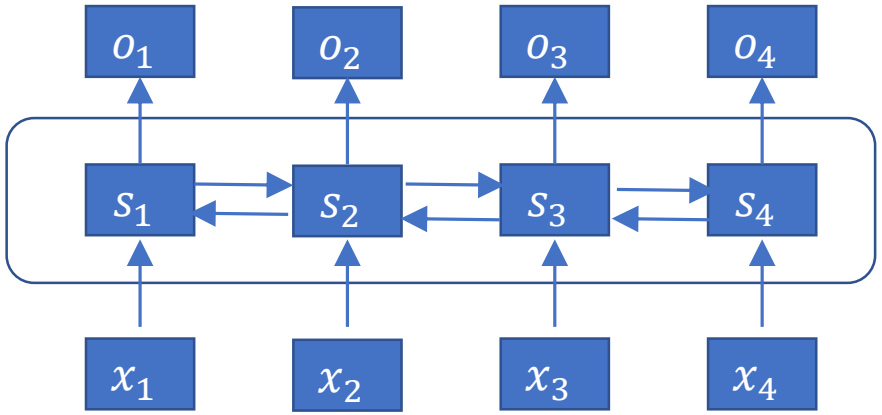
为什么采用Transformer

RNN(或者LSTM、GRU等)的计算限制为是顺序的，也就是RNN相关算法只能从左向右或从右向左依次计算，这种机制带来了两个问题：

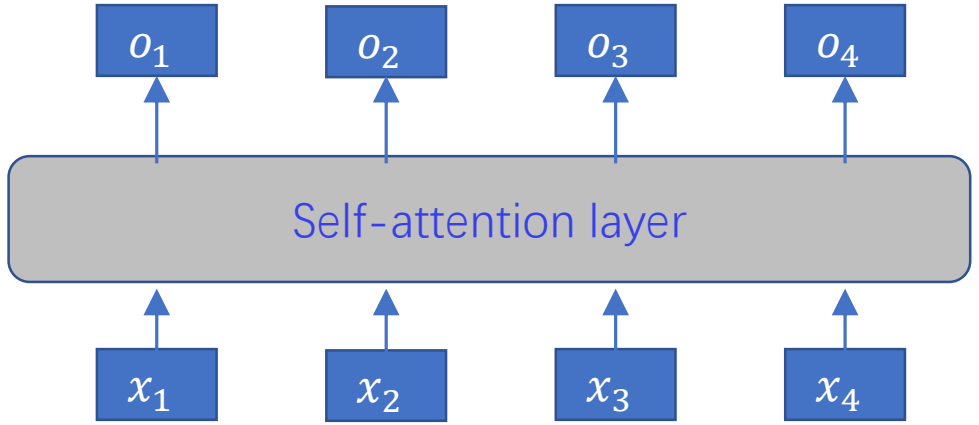
1. 时间片t的计算依赖于t-1时刻的计算结果，限制了模型的并行能力；
2. 顺序计算的过程中信息会丢失，尽管LSTM等门机制的结构在一定程度上缓解了长期依赖的问题，但对于特别长期的依赖现象，LSTM也无能为力。

Transformer的提出解决了上面两个问题，它使用了Attention机制，将序列中的任意两个位置之间的距离缩小为一个常量；其次他不是类似于RNN的顺序结构，因此具有更好的并行性，符合现有的GPU框架。

Transformer vs. RNN



RNN

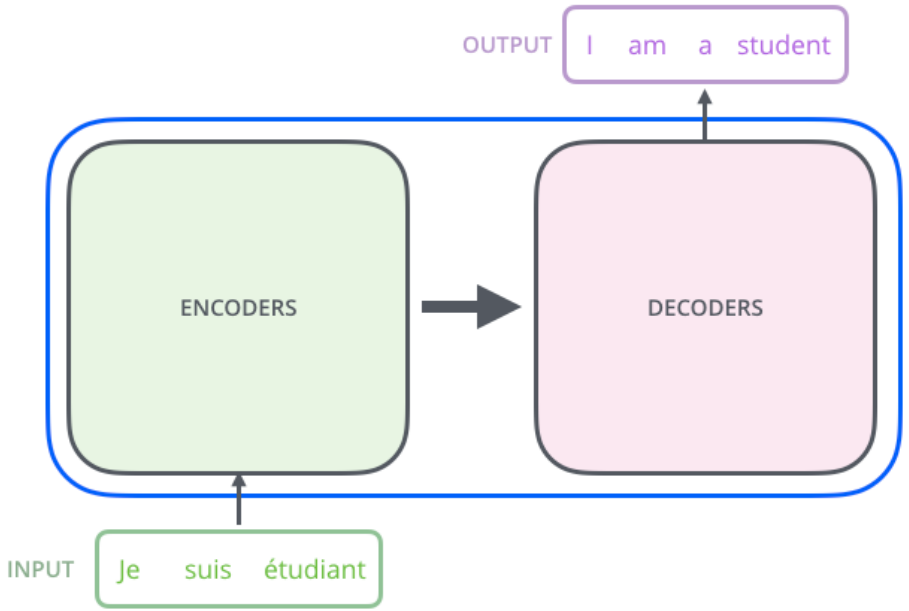


Transformer

- RNN(Recurrent Neural Network)存在时间序列依赖的问题，即是一种循环网络，其训练上是基于迭代的处理模式。因而上一个时间步（例如输入语句中的词）处理完毕以后，才能处理下一个时间步（例如输入语句中的下一个词）。
- **而Transformer的训练可以实现并行，即序列中的单词可以同时训练，因而效率较高。其引入位置编码来处理单词的顺序。**

Transformer

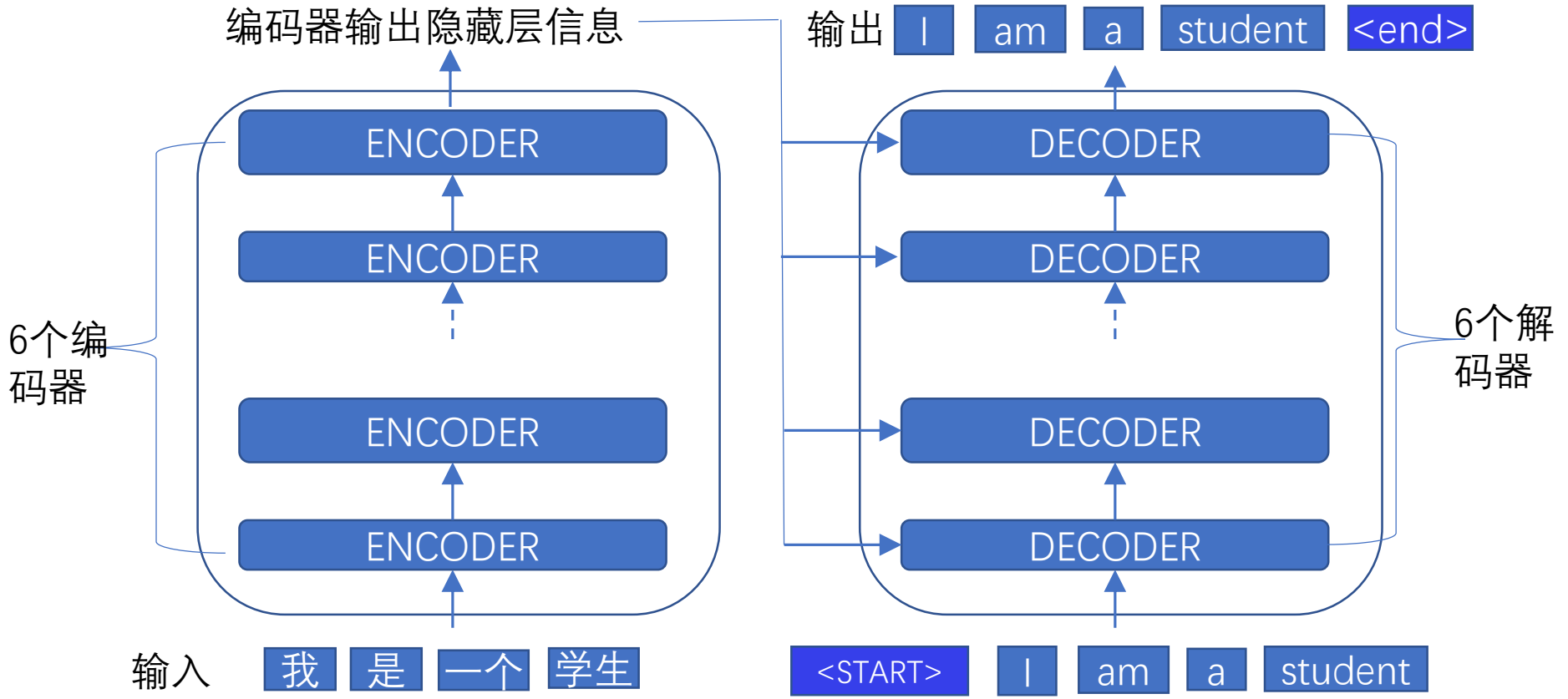
Transformer结构



Transformer本质上也是encoder-decoder结构，
可表示为上图所示

Transformer

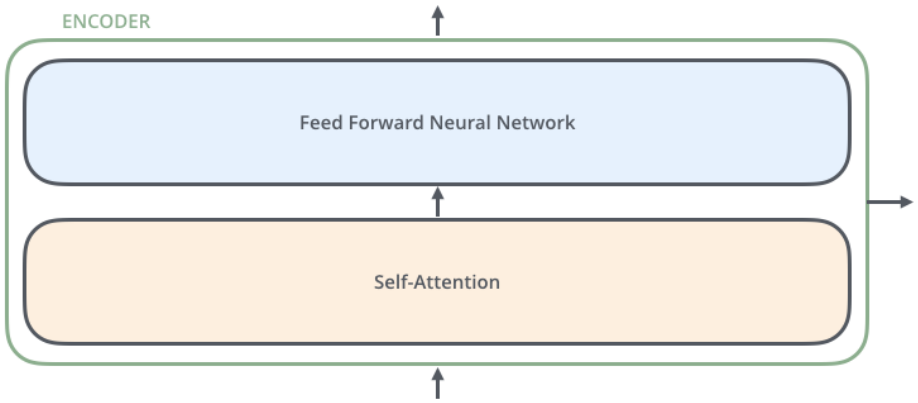
Transformer结构



编码器由6个编码block组成，解码器由6个解码block组成，编码器的输出会作为解码器的输入，如上图所示。

Transformer

➢Encoder结构及特点



Q、K、V分别是Q(Query)、K(Key)、V(Value)

数据首先经过self-attention模块得到一个加权之后的特征向量Z，即Attention(Q,K,V):

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

（公式含义会在后面介绍，先露个面）

得到Z之后，会被送到下一个模块Feed Forward Neural Network.这个全连接有两层，第一层激活函数是ReLU(线性整流函数)，第二层是一个线性激活函数，可以表示为:

$$FFZ(Z) = \max(0, ZW_1 + b_1)\,W_2 + b_2$$

Transformer微观结构

编码器结构

[sequence_length, embedding_size]

前馈神经网络

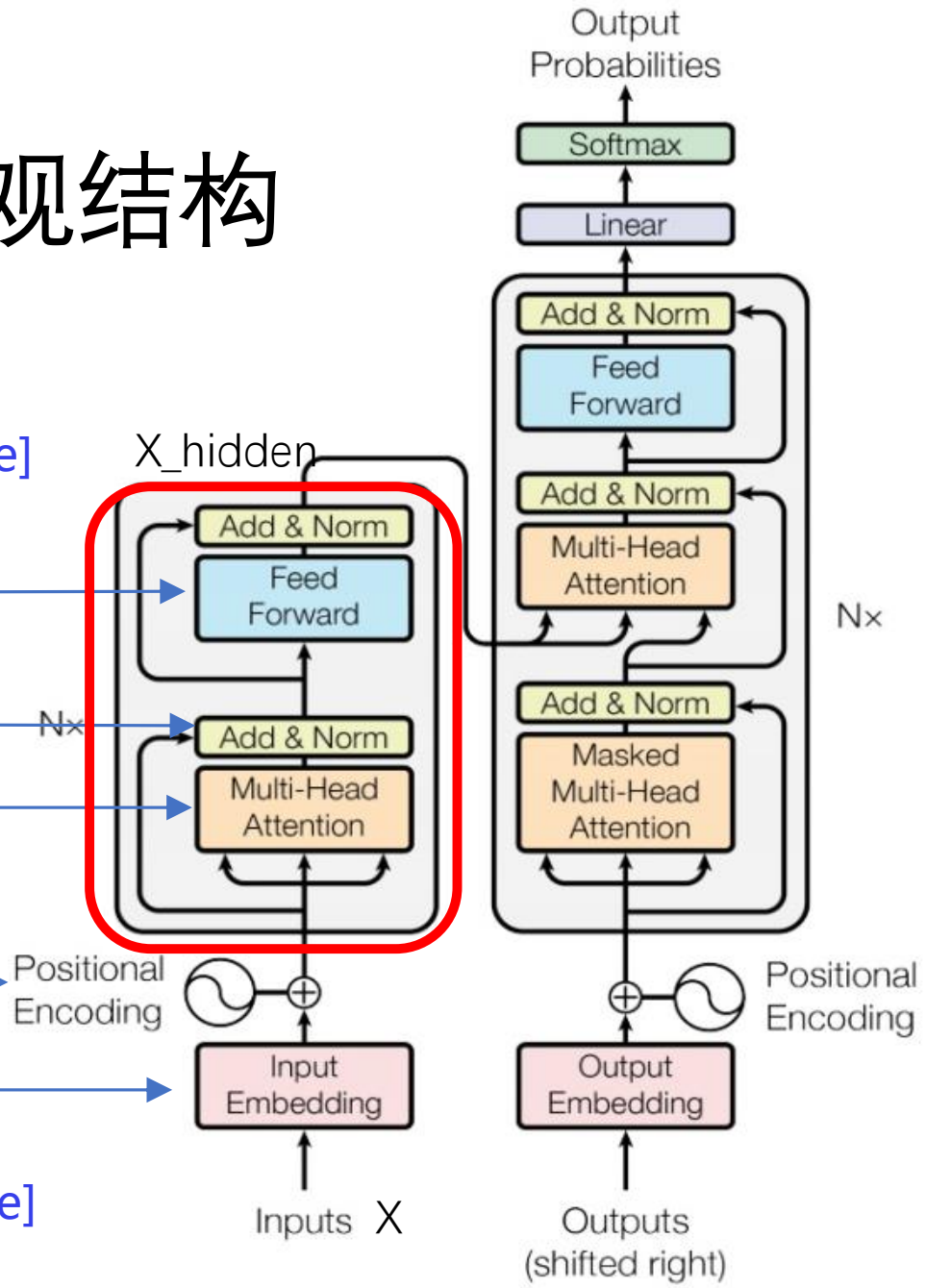
残差连接和层归一化

多头注意力机制

位置嵌入

词嵌入

[sequence_length, embedding_size]



[1] Vaswani A, Shazeer N, Parmar N, et al. ***Attention is all you need*** [C]//Advances in Neural Information Processing Systems. 2017: 5998-6008

Transformer微观结构

编码器结构

[sequence_length, embedding_size]

前馈神经网络

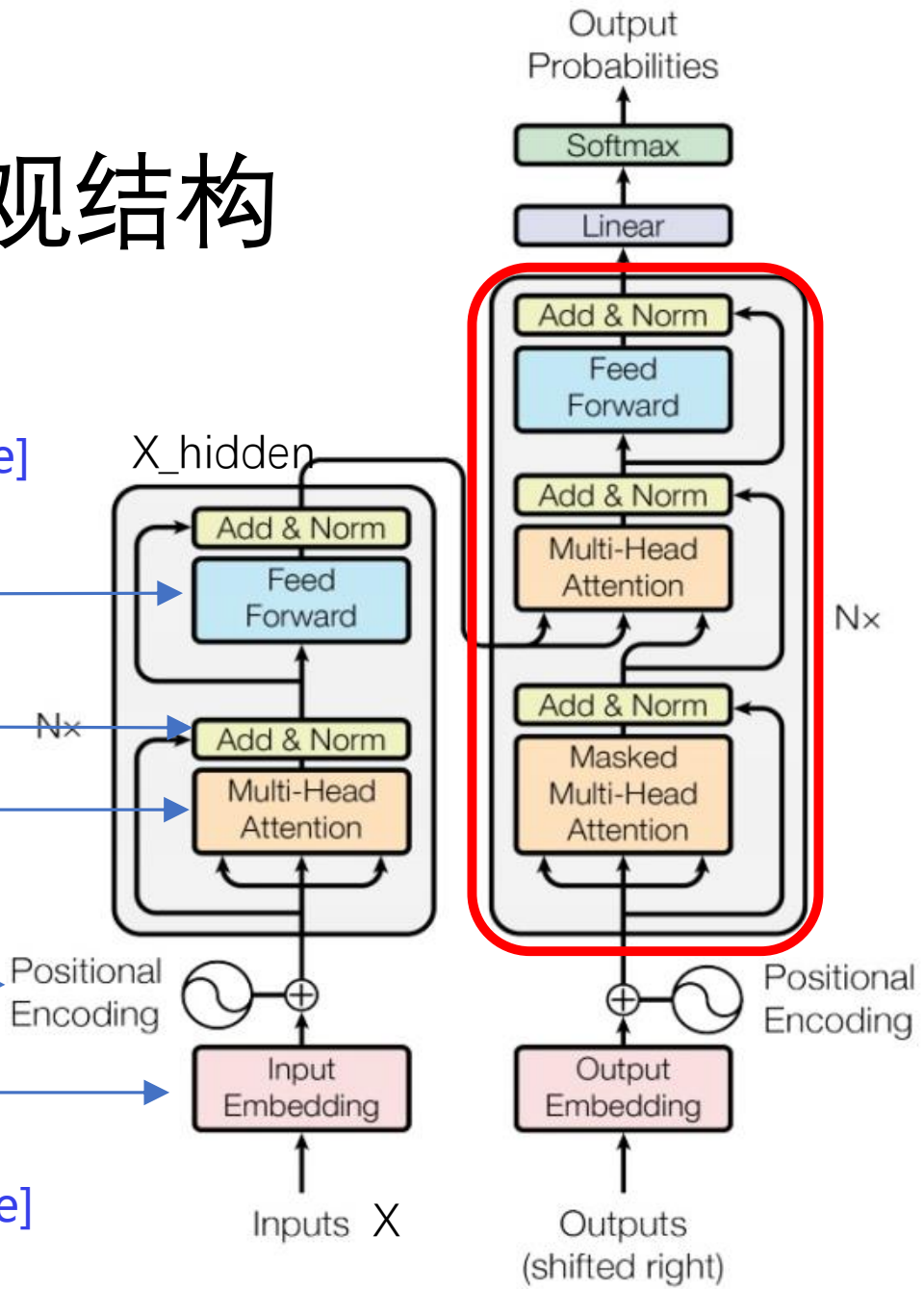
残差连接和层归一化

多头注意力机制

位置嵌入

词嵌入

[sequence_length, embedding_size]

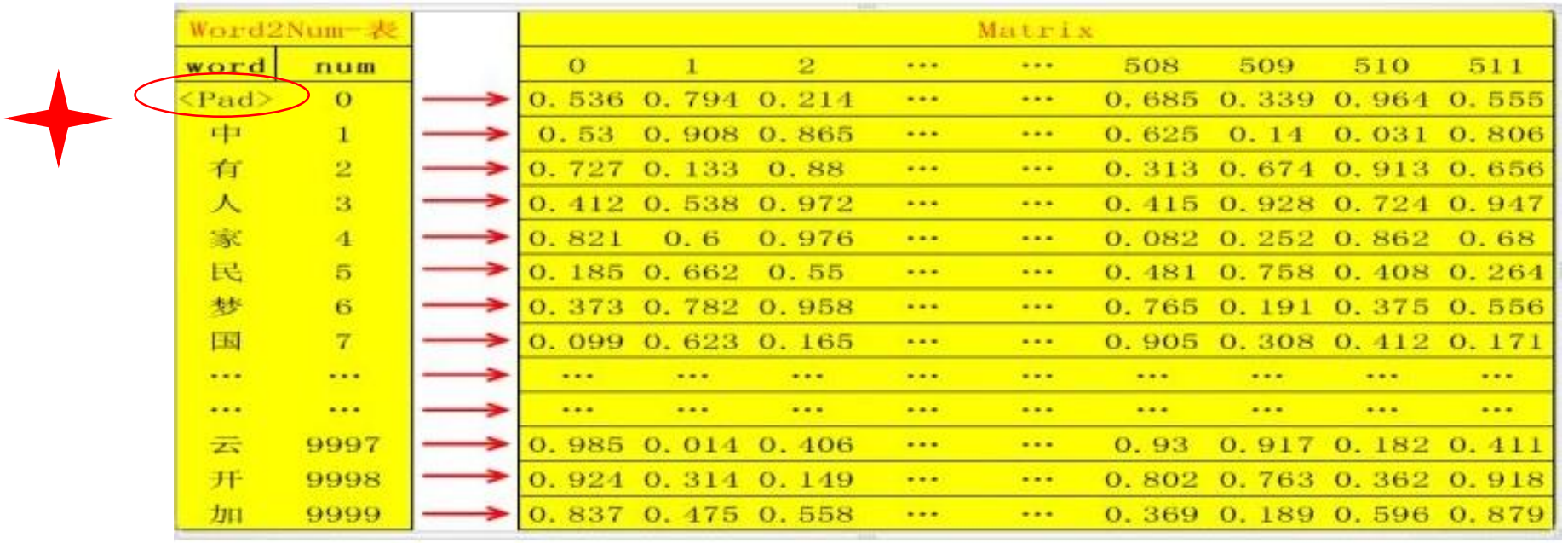


[1] Vaswani A, Shazeer N, Parmar N, et al. ***Attention is all you need*** *[C]//Advances in Neural Information Processing Systems. 2017: 5998-6008*

Transformer

Step1： Input Embedding

将原文所有单词汇总统计频率，删除低频词汇，假设总共选出了M个单词
利用Xaviers初始化方法随机生成矩阵Matrix$_{M*N}$，M对应选出的单词数，N则为固定值，通常取N=512，矩阵如图所示，图中取M=10000



Word2Num一表		Matrix								
word	num	0	1	2	…	…	508	509	510	511
<Pad>	0	0.536	0.794	0.214	…	…	0.685	0.339	0.964	0.555
中	1	0.53	0.908	0.865	…	…	0.625	0.14	0.031	0.806
有	2	0.727	0.133	0.88	…	…	0.313	0.674	0.913	0.656
人	3	0.412	0.538	0.972	…	…	0.415	0.928	0.724	0.947
家	4	0.821	0.6	0.976	…	…	0.082	0.252	0.862	0.68
民	5	0.185	0.662	0.55	…	…	0.481	0.758	0.408	0.264
梦	6	0.373	0.782	0.958	…	…	0.765	0.191	0.375	0.556
国	7	0.099	0.623	0.165	…	…	0.905	0.308	0.412	0.171
…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…
云	9997	0.985	0.014	0.406	…	…	0.93	0.917	0.182	0.411
开	9998	0.924	0.314	0.149	…	…	0.802	0.763	0.362	0.918
加	9999	0.837	0.475	0.558	…	…	0.369	0.189	0.596	0.879

Transformer

Step1：Input Embedding

随后，可以把每句话表示成矩阵形式，设句子长度为T，则矩阵维度=T*N
如“中国人有中国梦。”可以表示为下图所示,设为矩阵matX,
这里定义矩阵行数为10；第8行，即编号为100的一行可以理解为结束符，不足的补齐（padding）

	matX							
1	0.53	0.908	0.865	…	…	0.625	0.14	0.031	0.806
7	0.099	0.623	0.165	…	…	0.905	0.308	0.412	0.171
3	0.412	0.538	0.972	…	…	0.415	0.928	0.724	0.947
2	0.727	0.133	0.88	…	…	0.313	0.674	0.913	0.656
1	0.53	0.908	0.865	…	…	0.625	0.14	0.031	0.806
7	0.099	0.623	0.165	…	…	0.905	0.308	0.412	0.171
6	0.373	0.782	0.958	…	…	0.765	0.191	0.375	0.556
100	0.099	0.623	0.165	…	…	0.905	0.308	0.412	0.171
0	0.536	0.794	0.214	…	…	0.685	0.339	0.964	0.555
0	0.536	0.794	0.214	…	…	0.685	0.339	0.964	0.555

输入句表示矩阵matX

Word2Num-表			Matrix								
word	num		0	1	2	…	…	508	509	510	511
<Pad>	0	→	0.536	0.794	0.214	…	…	0.685	0.339	0.964	0.555
中	1	→	0.53	0.908	0.865	…	…	0.625	0.14	0.031	0.806
有	2	→	0.727	0.133	0.88	…	…	0.313	0.674	0.913	0.656
人	3	→	0.412	0.538	0.972	…	…	0.415	0.928	0.724	0.947
家	4	→	0.821	0.6	0.976	…	…	0.082	0.252	0.862	0.68
民	5	→	0.185	0.662	0.55	…	…	0.481	0.758	0.408	0.264
梦	6	→	0.373	0.782	0.958	…	…	0.765	0.191	0.375	0.556
国	7	→	0.099	0.623	0.165	…	…	0.905	0.308	0.412	0.171
…	…	→	…	…	…	…	…	…	…	…	…
…	…	→	…	…	…	…	…	…	…	…	…
云	9997	→	0.985	0.014	0.406	…	…	0.93	0.917	0.182	0.411
开	9998	→	0.924	0.314	0.149	…	…	0.802	0.763	0.362	0.918
加	9999	→	0.837	0.475	0.558	…	…	0.369	0.189	0.596	0.879

初始词表矩阵

Transformer

➢Step2：Position Encoding

Transformer模型并没有捕捉顺序序列的能力，也就是说无论句子结构怎么打乱，都会得到类似的结果。
为解决这个问题，模型在编码词向量时引入了位置编码的特征。具体来讲，位置编码会在词向量中加入了单词的位置信息，使Transformer能区分出不同位置的单词。
通常位置编码是一个长度为d_{model}的特征向量，以便于和词向量进行单位加的操作，编码公式如下：↑表示后面是指数

$$PE(pos, 2i) = \sin(\frac{pos}{10000 \uparrow \frac{2i}{d_{model}}})$$

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

上式中， pos表示单词的位置， i表示词向量第i个维度。得到的矩阵设为matP

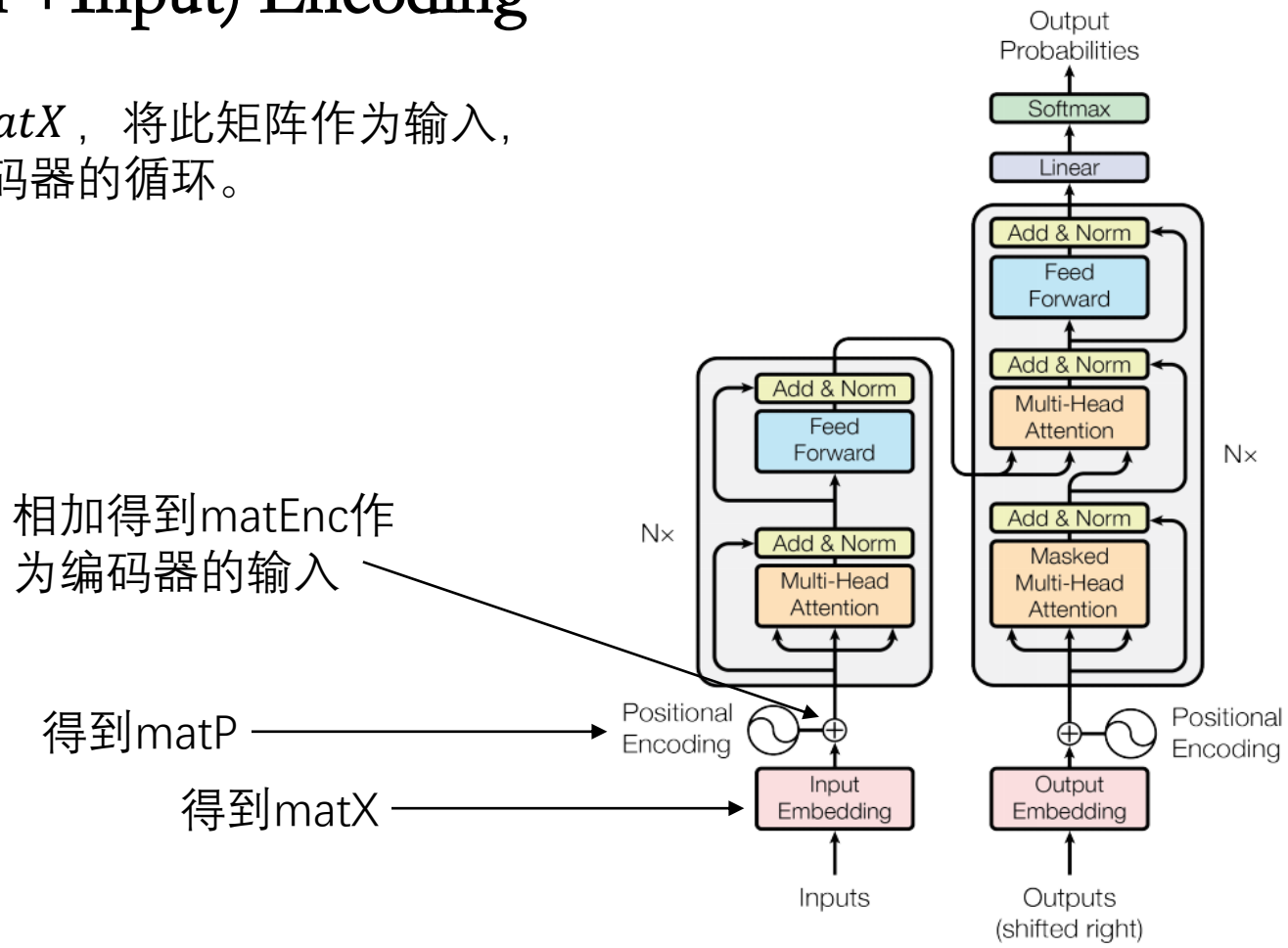
position	0	1	2	3	4	…	510	511
0	f(0,0)	f(0,1)	f(0,2)	f(0,3)	f(0,4)	…	f(0,510)	f(0,511)
1	f(1,0)	f(1,1)	f(1,2)	f(1,3)	f(1,4)	…	f(1,510)	f(1,511)
2	f(2,0)	f(2,1)	f(2,2)	f(2,3)	f(2,4)	…	f(2,510)	f(2,511)
3	f(3,0)	f(3,1)	f(3,2)	f(3,3)	f(3,4)	…	f(3,510)	f(3,511)
4	f(4,0)	f(4,1)	f(4,2)	f(4,3)	f(4,4)	…	f(4,510)	f(4,511)
5	f(5,0)	f(5,1)	f(5,2)	f(5,3)	f(5,4)	…	f(5,510)	f(5,511)
6	f(6,0)	f(6,1)	f(6,2)	f(6,3)	f(6,4)	…	f(6,510)	f(6,511)
7	f(7,0)	f(7,1)	f(7,2)	f(7,3)	f(7,4)	…	f(7,510)	f(7,511)
8	f(8,0)	f(8,1)	f(8,2)	f(8,3)	f(8,4)	…	f(8,510)	f(8,511)
9	f(9,0)	f(9,1)	f(9,2)	f(9,3)	f(9,4)	…	f(9,510)	f(9,511)

输入句位置编码矩阵matP

Transformer

➢Step2：(Position +Input) Encoding

令$matEnc = matP + matX$，将此矩阵作为输入，进入Transformer模型编码器的循环。



Transformer编码部分

self-attention

Self-attention(自注意力机制)：一种特殊的attention机制，其特点在于应用于计算句子中每个单词与其他单词之间的关联。

例如 “I arrived at the bank after crossing the river.”这句话的bank在翻译时有银行和河岸两种解释。在self-attention中，翻译bank一词时，会分配给river一个较高的attention score，利用该attention score可以得到一个加权的表示，再放到一个前馈神经网络中得到新的表示，这一表示可以很好的考虑到上下文信息，由此断定bank更大可能会翻译成河岸。

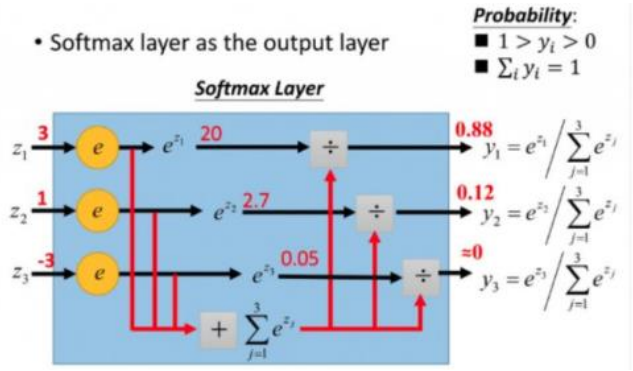
因此Self-attention在NLP中往往被认为扑捉了句子内部的某种关系。

Transformer编码部分

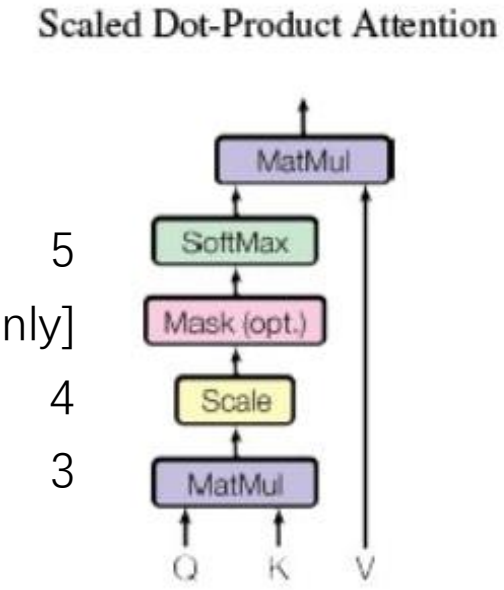
self-attention计算过程

1. 引入三个向量Q、K、V，初始值Q=K=V=matEnc（后面经过变化会不一样）
2. 首先分别对Q、K、V经过线性变换，即讲三者分别输入到三个单层神经网络，激活函数选择relu，输出新的Q、K、V（经过线性变换三者的维度不变）
3. 计算score= $\mathrm{Q} \cdot K^T$
4. 为了梯度的稳定，使score归一化，即score除以$\sqrt{d_k}$，若key的维数d_k特别大，可能点积变得很大导致softmax函数进入一个梯度很小的范围。
5. 对score施以$softmax$激活函数
6. 将归一化后的函数点乘V

公式表示：$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$



Softmax：归一化函数



5
[decoder-only]
4
3

Transformer编码部分

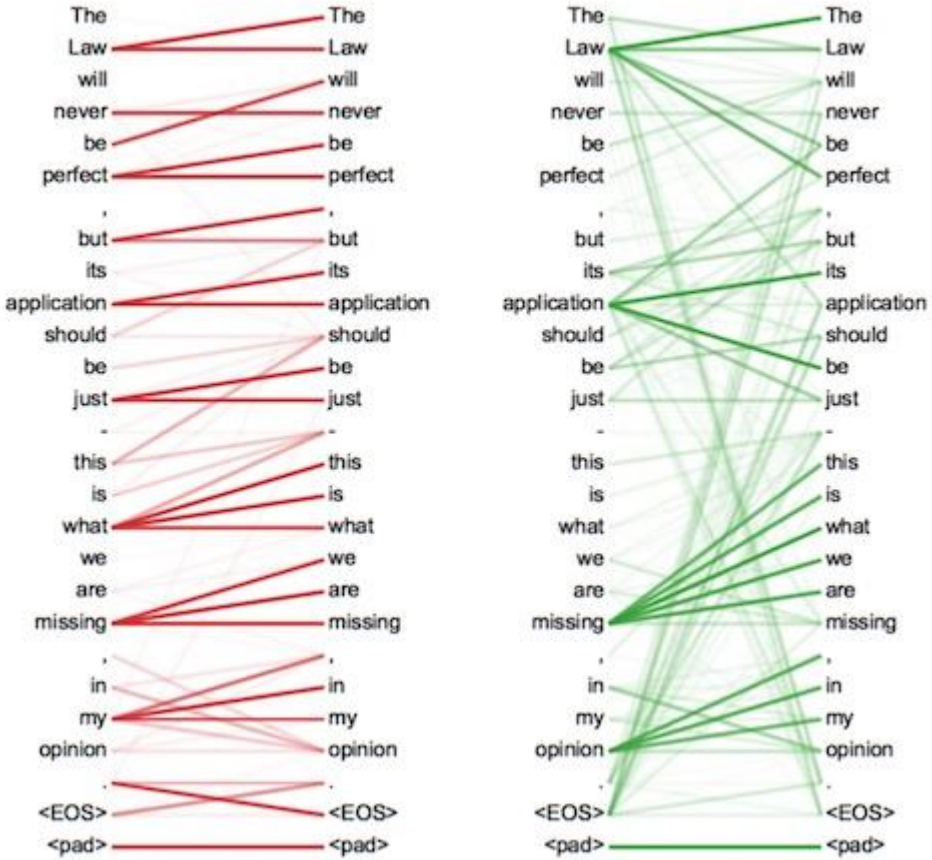
➢Step3：Multi head attention

- 多头注意力机制：多个self-attention的结合
  - 每个head学习到在不同表示空间中的特征

如图所示，两个head学习到的attention侧重点可能不同，这给了模型更大的容量。

- 计算过程与self-attention基本相同
- 在开始和结尾多出了切分和合成部分

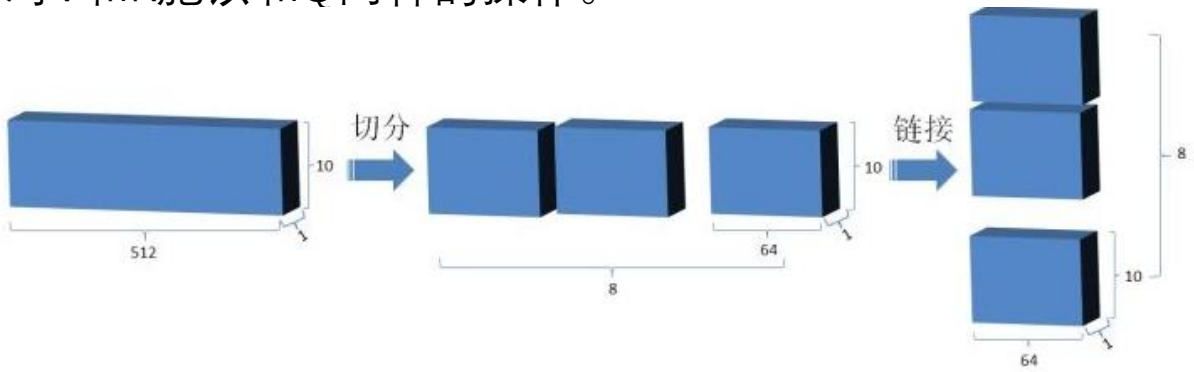
我们以head数=8为例说明。

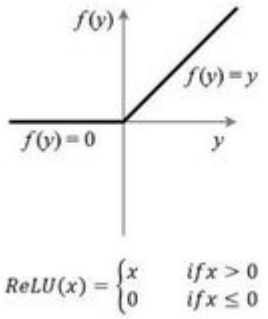


Transformer编码部分

Step3：Multi head attention

1. 引入三个向量Q、K、V，初始值Q=K=V=matEnc（后面经过变化会不一样）
2. 首先分别对Q、K、V经过线性变换，即将三者分别输入到三个单层神经网络，激活函数选择relu，输出新的Q、K、V（经过线性变换三者的维度不变）
3. 对Q的N维进行切分，整齐切分为8段(可以是任意段，但必须被N整除), 得到8个64*10的矩阵，如下图所示，multihead attention实际上可看作是由8个self-attention连接而成。
4. 对V和K施以和Q同样的操作。





ReLu：单侧抑制

切分后的Q如下图为例：

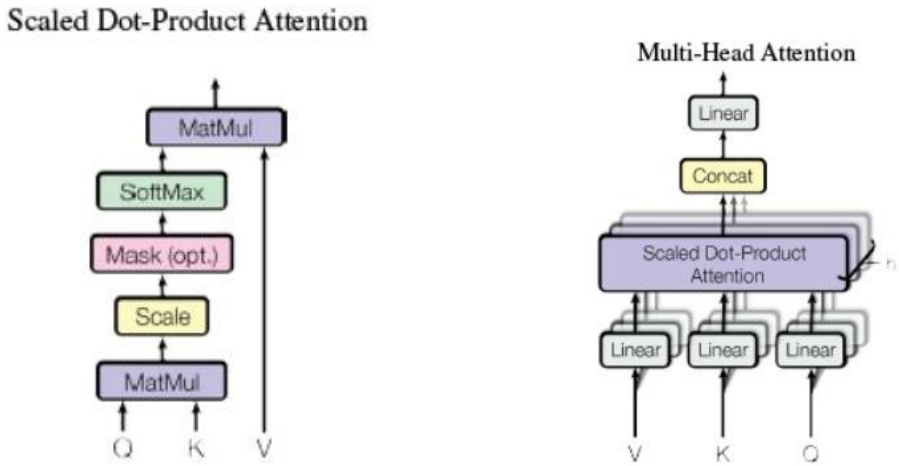
Q							
0	1	2	…	508	509	510	511
0.065	0.781	0.684	…	0.39	0.719	0.668	0.868
0.909	0.137	0.856	…	0.399	0.045	0.835	0.015
…	…	…	…	…	…	…	…
0.611	0.77	0.055	…	0.404	0.74	0.583	0.992

	Q							
	0	1	2	…	60	61	62	63
Head-1	0.065	0.781	0.684	…	0.474	0.551	0.304	0.604
	0.909	0.137	0.856	…	0.145	0.975	0.406	0.245
	…	…	…	…	…	…	…	…
	0.611	0.77	0.055	…	0.771	0.124	0.715	0.858
Head-2	0.895	0.279	0.318	…	0.565	0.934	0.922	0.78
	0.284	0.399	0.803	…	0.257	0.259	0.786	0.046
	…	…	…	…	…	…	…	…
	0.804	0.28	0.239	…	0.353	0.105	0.286	0.218
…	…	…	…	…	…	…	…	…
Head-7	0.575	0.283	0.308	…	0.315	0.85	0.029	0.345
	0.243	0.25	0.626	…	0.012	0.645	0.067	0.028
	…	…	…	…	…	…	…	…
	0.116	0.811	0.159	…	0.695	0.846	0.841	0.148
Head-8	0.595	0.998	0.764	…	0.39	0.719	0.668	0.868
	0.032	0.691	0.194	…	0.399	0.045	0.835	0.015
	…	…	…	…	…	…	…	…
	0.319	0.849	0.031	…	0.404	0.74	0.583	0.992

Transformer编码部分

Step3：Multi head attention

4.对得到的8个Q_{10*64}和K_{10*64}进行运算score= $\mathrm{Q}\cdot K^T$，得到新的10*10维矩阵
5.为了梯度的稳定，使score归一化，即score除以$\sqrt{d_k}$，若key的维数d_k特别大，可能点积变得很大导致softmax函数进入一个梯度很小的范围。
6.对score施以$softmax$激活函数
7.将归一化后的函数点乘V_{10*64}，得到10*64的矩阵，再将八个该矩阵做连接操作，得到维度为10*512的矩阵



整个过程即公式

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

Transformer编码部分

Step4：add&norm求和和归一化

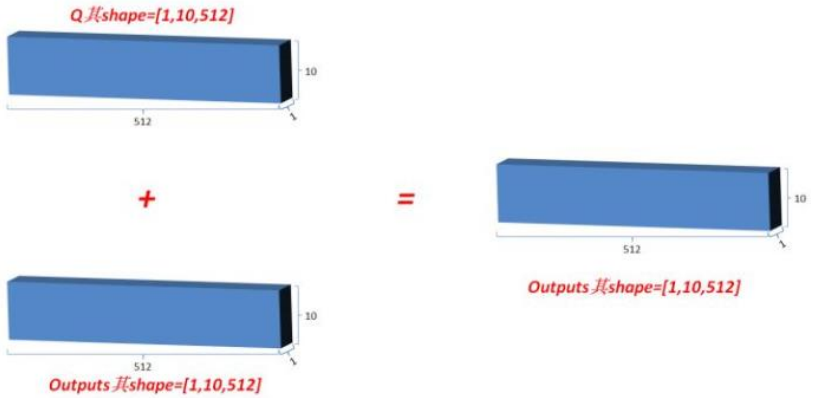
add实际上是为了使网络有效叠加，避免梯度消失，也就是残差网络的解决办法：

$Attention(Q,K,V)= Attention(Q,K,V)$+Q

Norm是标准化矫正一次，意图在不改变矩阵权重的情况下实现矩阵的独立同分布：

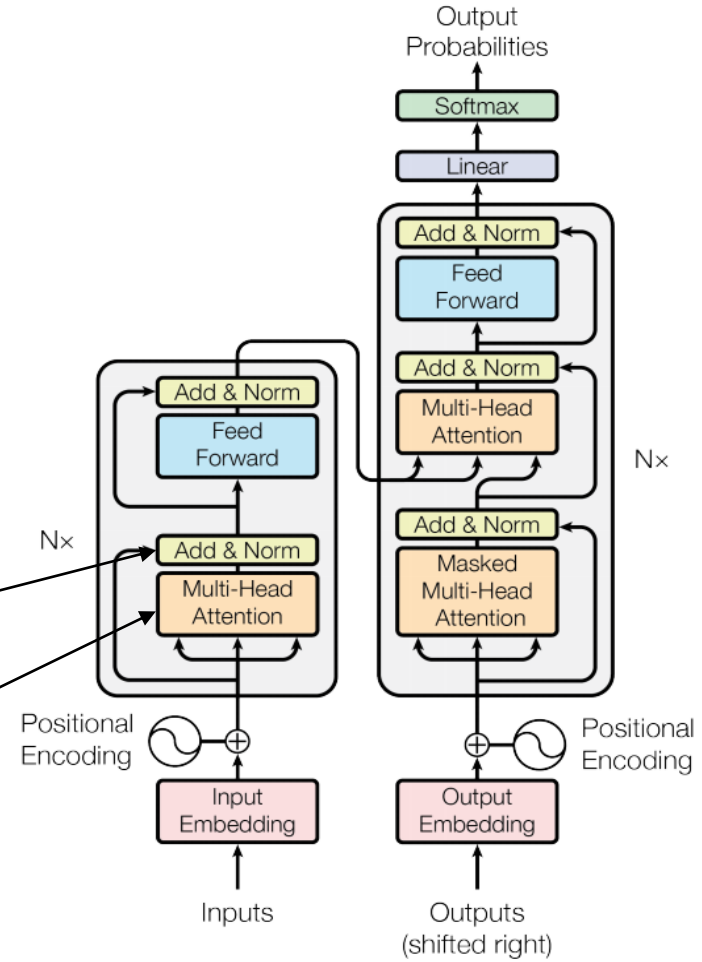
$$h = f(g \cdot \frac{x-\mu}{\sigma} + b)$$

其中μ、σ分别为$Attention(Q,K,V)$在512维的均值和方差



矩阵求和和归一化

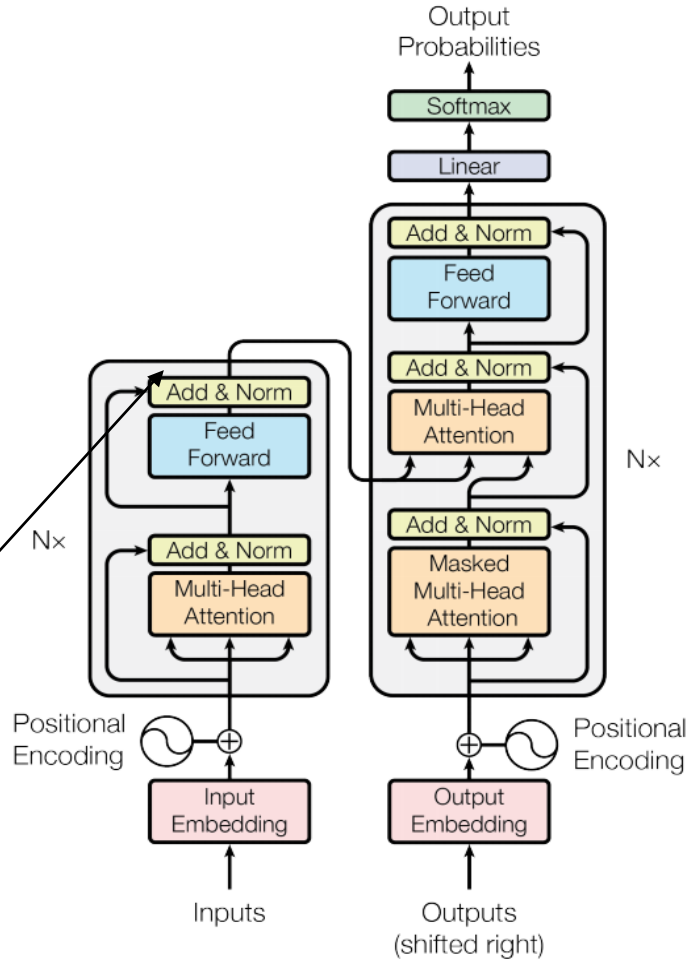
得到$Attention(Q,K,V)$



Transformer编码部分

Step5： feed forward

对上述步骤得到的归一化结果进行两次卷积
卷积后得到的output与matEnc的维度相同
至此一个Encoder完成，更新matEnc=output
重复N次上述步骤，可取N=6



重复N次得到最终matEnc

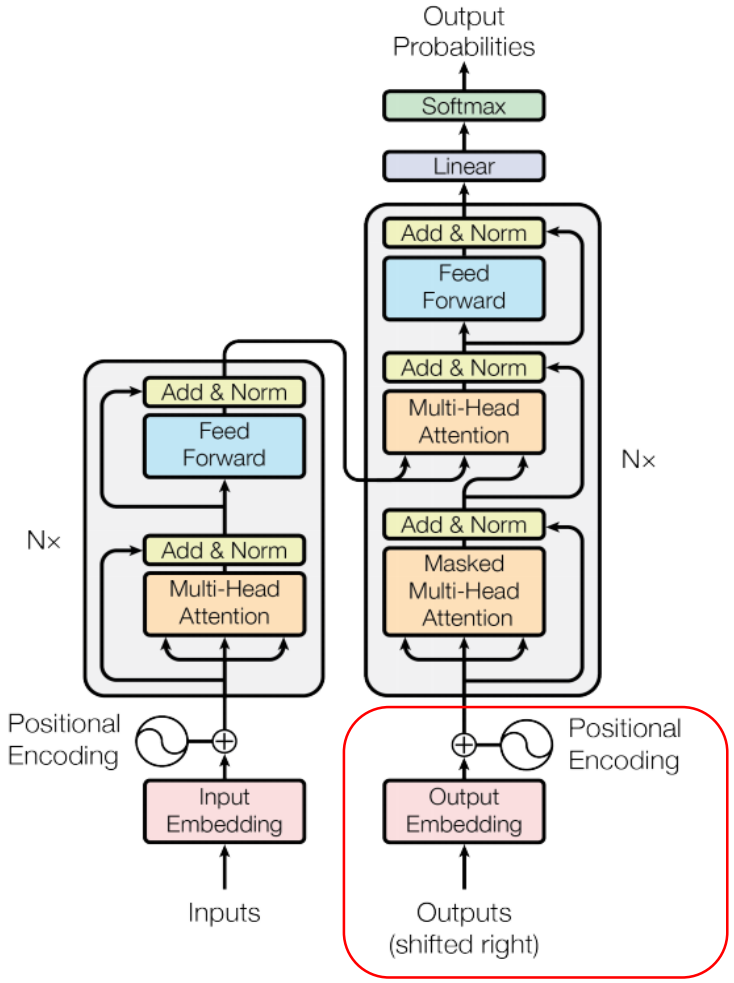
Transformer解码部分

Output Enbedding

将outputs右移一位，此操作是为了解码区最初初始化的第一次输入，可统一补齐为0

注意：此处的outputs不同于上文中output，上文最终输入解码器的结果为matEnc，此处的outputs在训练时是对应原文的译文，在预测时第一次输入为0

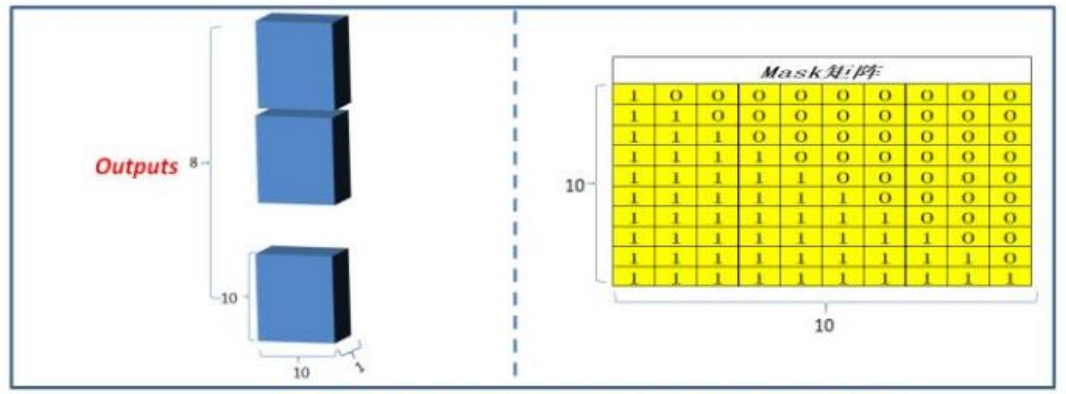
Output embedding和Positional Embedding同编码部分，此处不再赘述，经过这两步更新outputs

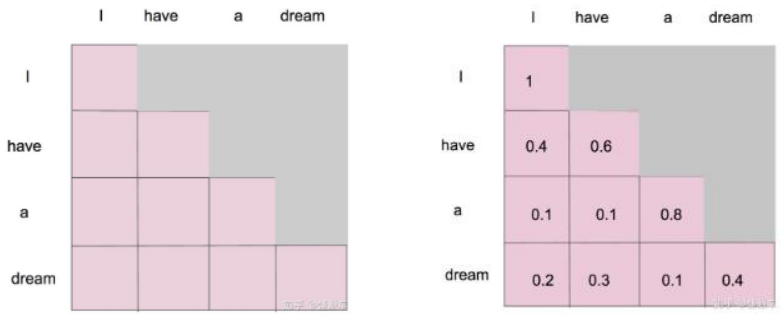


Transformer解码部分

Masked multi head attention

Masked multi head attention与编码器中的multi head attention类似，但多了一个mask矩阵，因为在解码部分，解码时是从左到右一次解码的。当解出第一个字时，第一个字只能与第一个字计算相关性；解码到第二个字时，第二个字只能与第一、二个字计算相关性。。。故需要乘一个mask矩阵

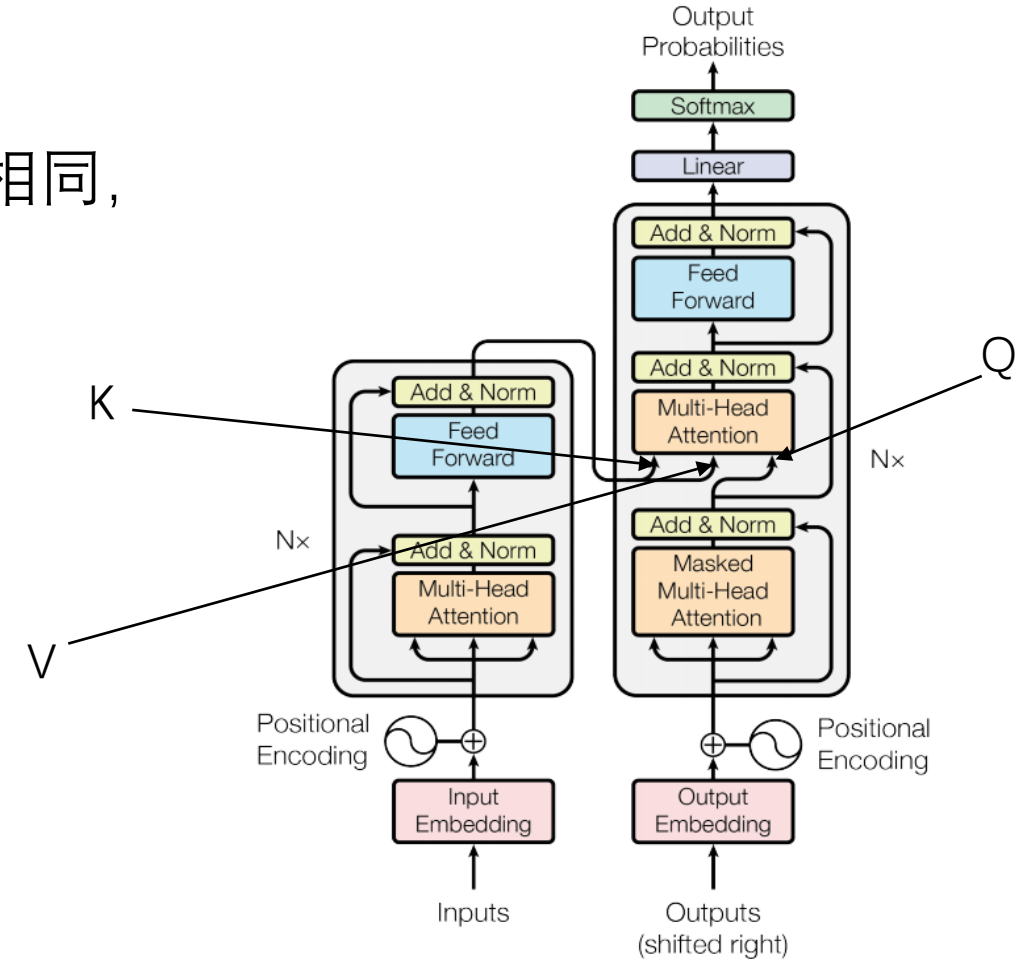




Transformer解码部分

- **multi head attention**

同编码部分
但Q和K、V不再相同，
Q=outputs，
K=V=macEnc

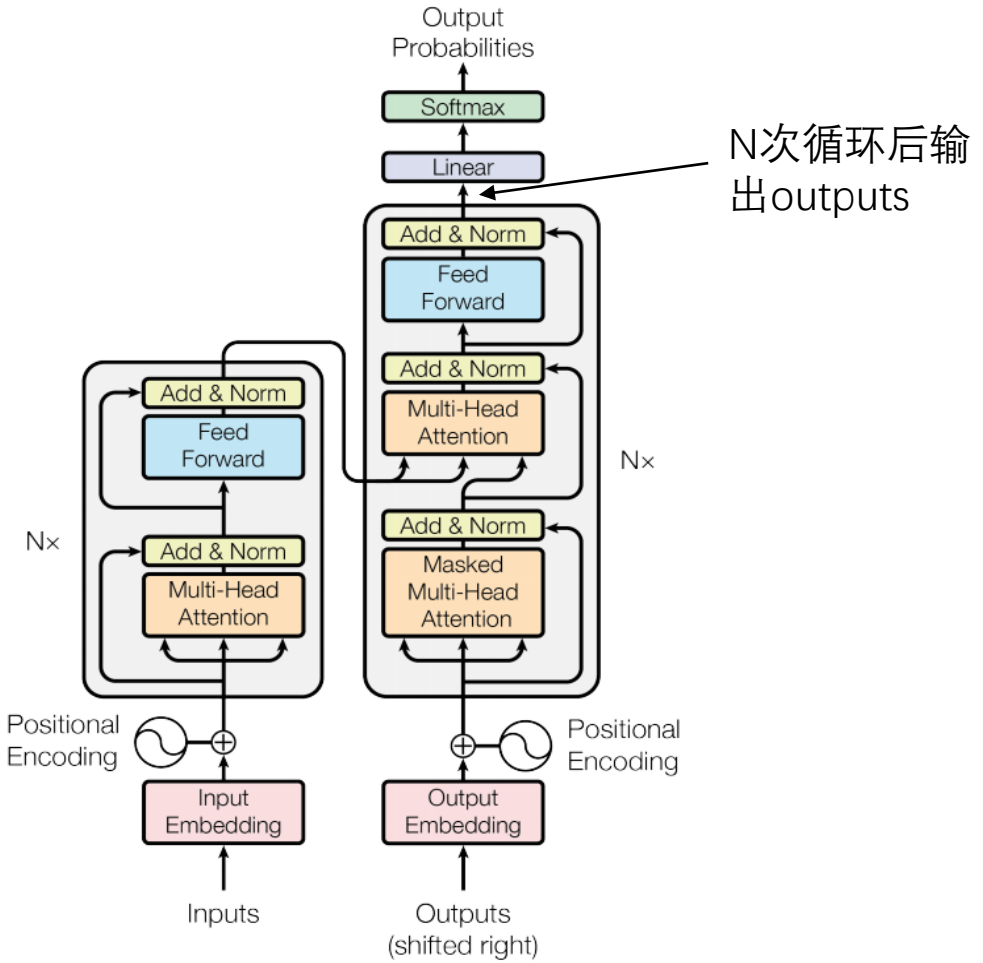


Transformer解码部分

➢其他部分

Add&norm部分、Feed-Forward部分和编码部分结构均相同

完成上述步骤的outputs同样作为新的输入送回解码部分循环N次(每一次循环结构均相同，但对应的参数不同，即是独立训练的)

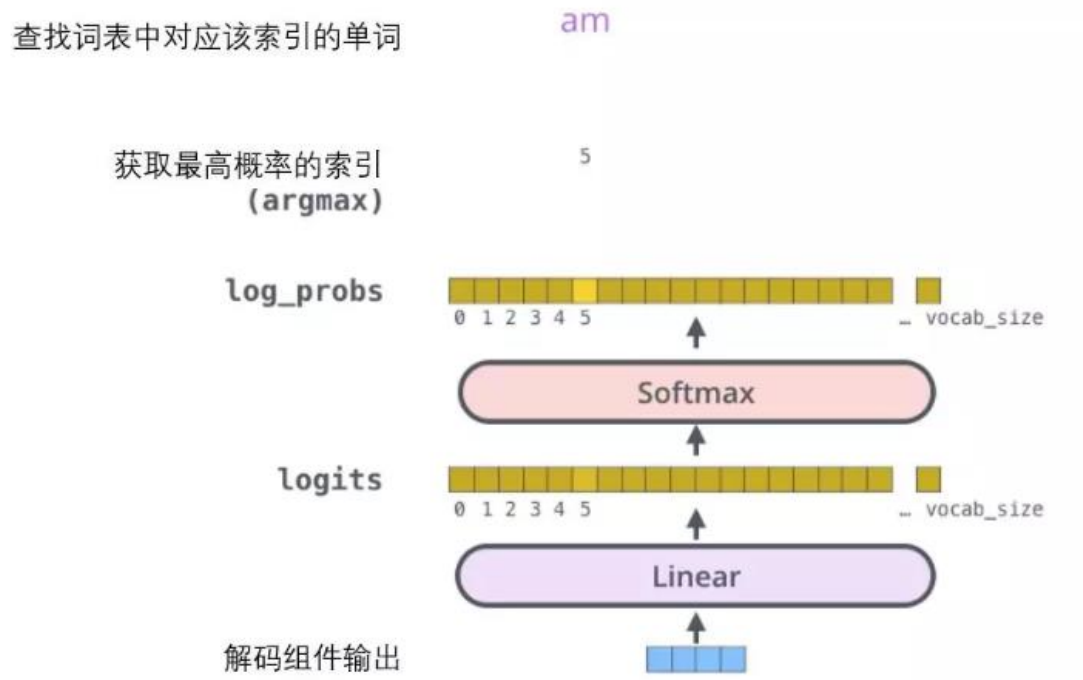


Transformer解码部分

➢其他部分

Linear：线性变换层是一个简单的全连接神经网络，它可以把outputs投射到一个比它大得多的、被称为对数几(logits)的向量里，假设一开始训练的单词是一万个，则维度为10000*512

对该向量进行softmax运算，将生成的分数变成概率，概率最高的单元格被选中，并把它对应的单词作为这个时间步的输出



Transformer

总结

1. 虽然Transformer最终也没有逃脱传统学习的套路，只是一个全连接(或者一维卷积)加Attention的结合体，但它抛弃了在NLP中最根本的RNN或CNN并取得了非常不错的效果。
2. Transformer的设计带来性能提升的关键是任意两个单词的距离是1，这对解决NLP中棘手的长期依赖问题是有效的。
3. Tran former算法的并行性非常好，符合目前的硬件(主要指GPU)环境。

缺点：

1. 抛弃CNN和RNN使模型的机制，缺乏捕捉局部特征的能力；
2. 位置信息在NLP中十分重要，位置编码的改进成为一个新问题；