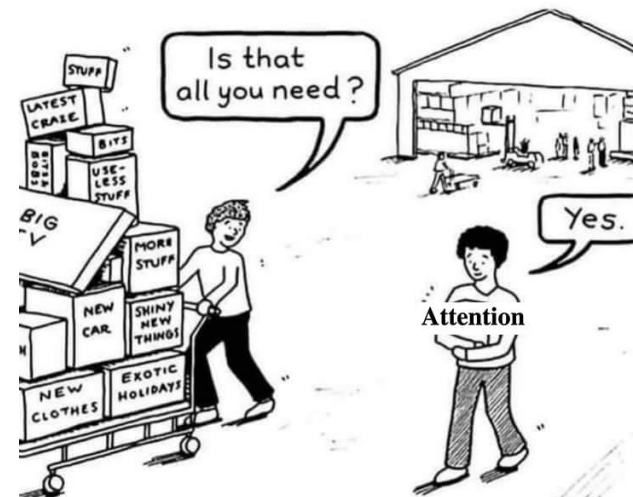


Transformer

Attention is All You Need



目录

- 前言
- 模型结构
- 数据处理
- 编码器
- 解码器

前言

相同的一句话，不同的人听的时候侧重点也可能不同。在自然语言处理中，根据任务内容的不同，句子中需要重点关注的部分也会不同。为此，我们引入注意力机制来判断在执行某个任务时，词在句子中的重要性，并通过注意力分数来表示词的重要程度。分数越高，说明该词对完成该任务的重要性越大。

女朋友:如果因为吵架，我离家出走，你会不会主动联系我？

A. 疯狂call你，你一个人在外面，我很担心的

B. 那要看是谁的错了

C. 吵架?不存在的，我绝对不会给你生气机会

D. 看你反省得怎么样吧，没反省好就不要回来

计算注意力分数时，我们主要参考三个因素：

query：任务内容

key：索引/标签（帮助定位到答案）

value：答案

《流浪地球》这部电影真的太赞了！

任务	结果
情感分类	positive

《流浪地球》这部电影真的太赞了！

任务	结果
电影名字？	《流浪地球》

《流浪地球》这部电影真的太赞了！

任务	结果
中译英	The Wandering Earth is awesome!

前言

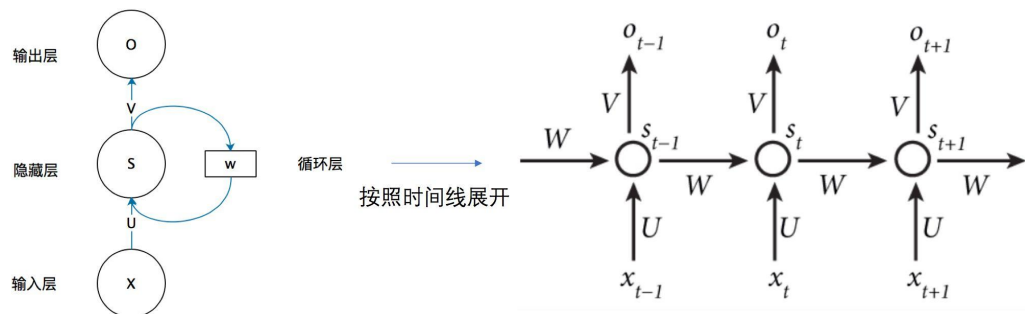
Transformer是一种基于注意力机制结构的神经网络，由Vaswani等人在2017年的论文“Attention Is All You Need”中提出，用于处理机器翻译、语言建模和文本生成等自然语言处理任务。

Transformer与传统NLP特征提取类模型的区别主要在以下两点：

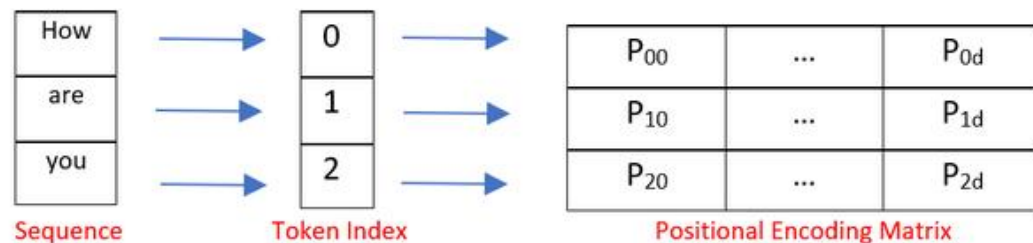
- Transformer将自注意力机制和多头注意力机制的概念运用到模型中；
- 由于缺少RNN模型的时序性，Transformer引入了位置编码，在数据上而非模型中添加位置信息；

以上的处理带来了2个优点：

- 更容易并行化，训练更加高效；
- 在处理长序列的任务中表现优秀，可以快速捕捉长距离中的关联信息。



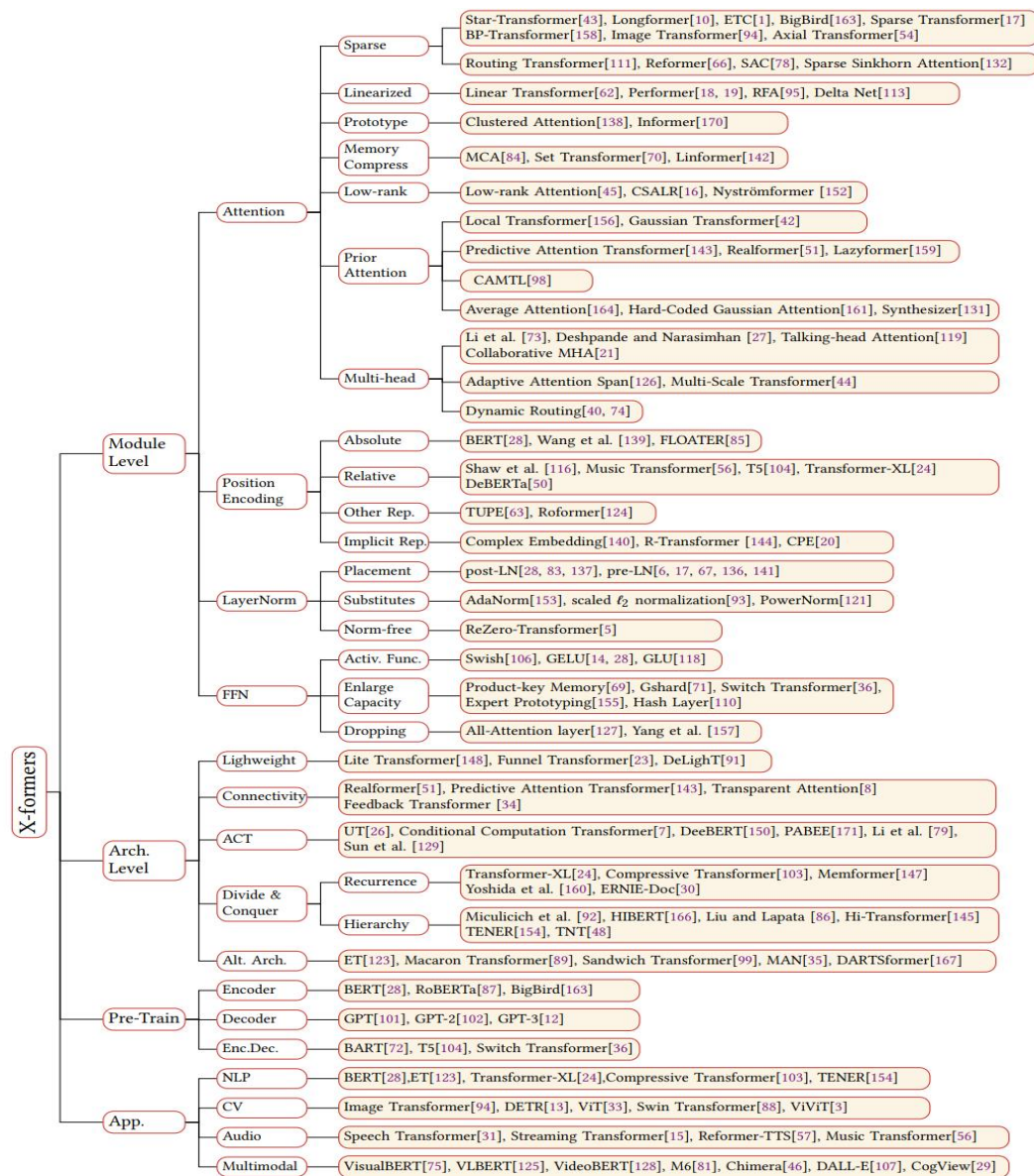
RNN时间线展开图



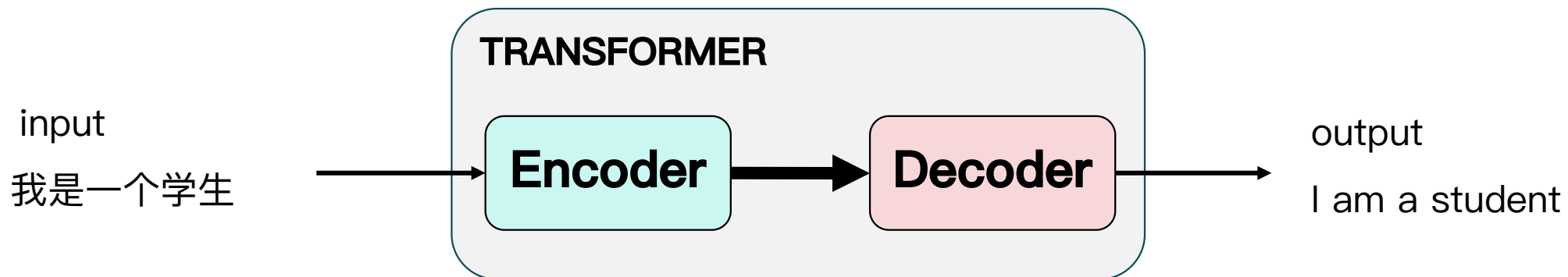
位置编码

前言

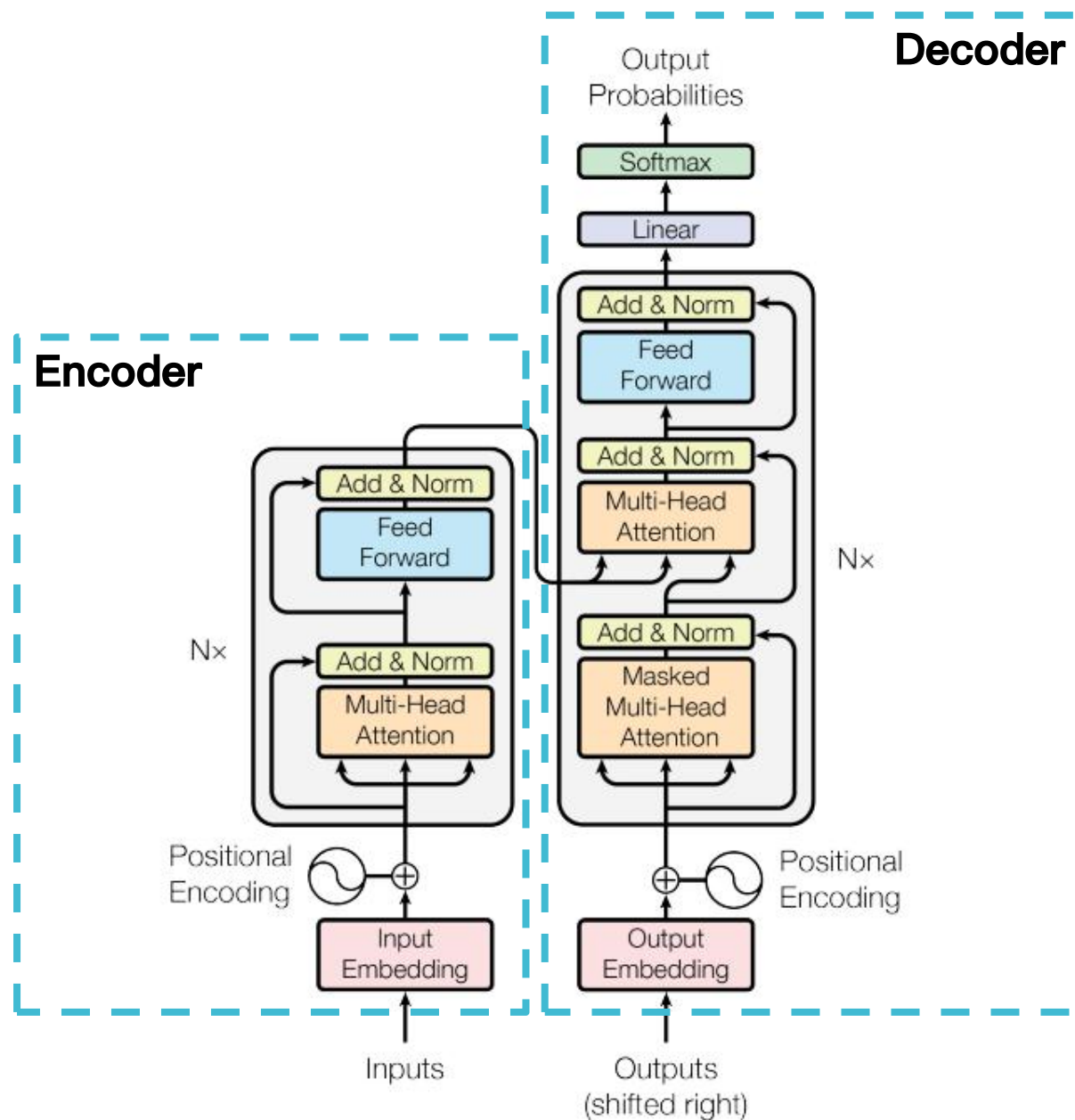
X-formers网络的总体分类



模型结构

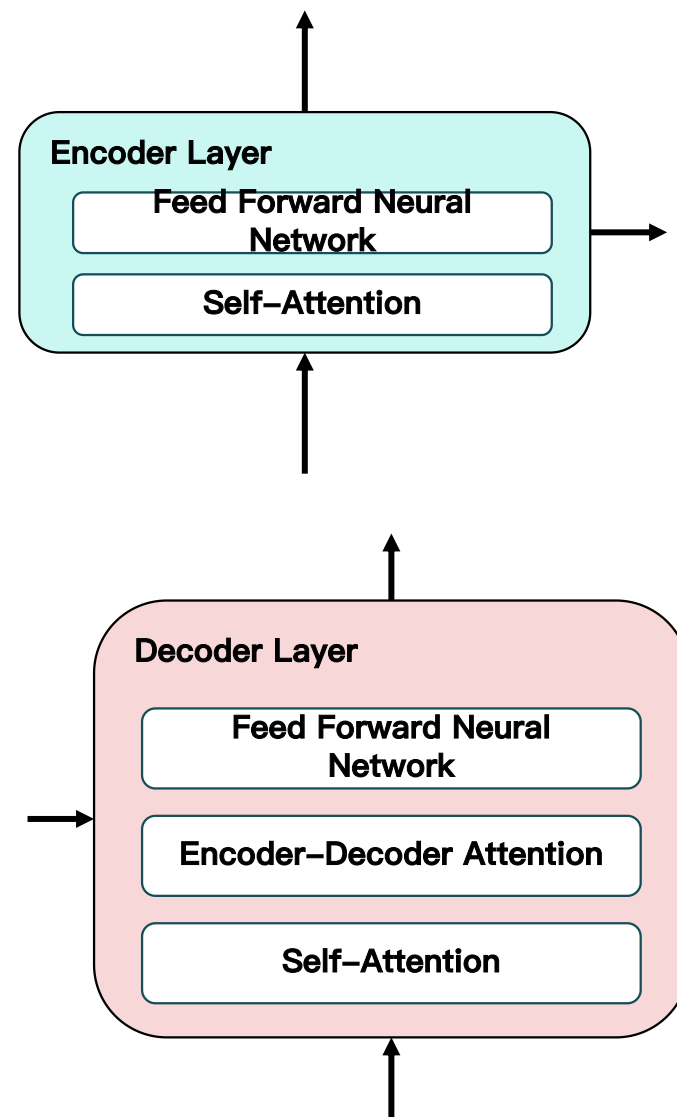
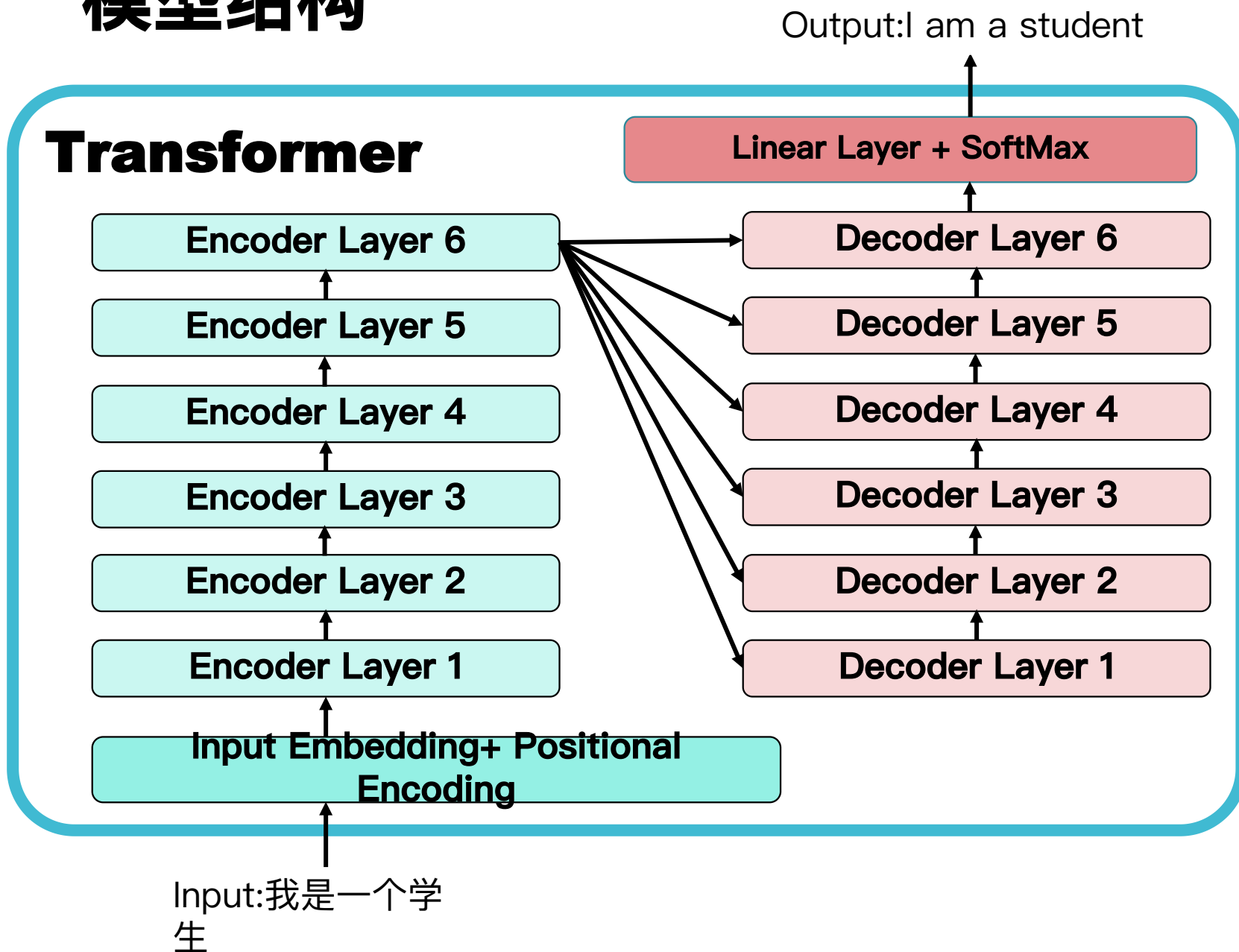


模型结构

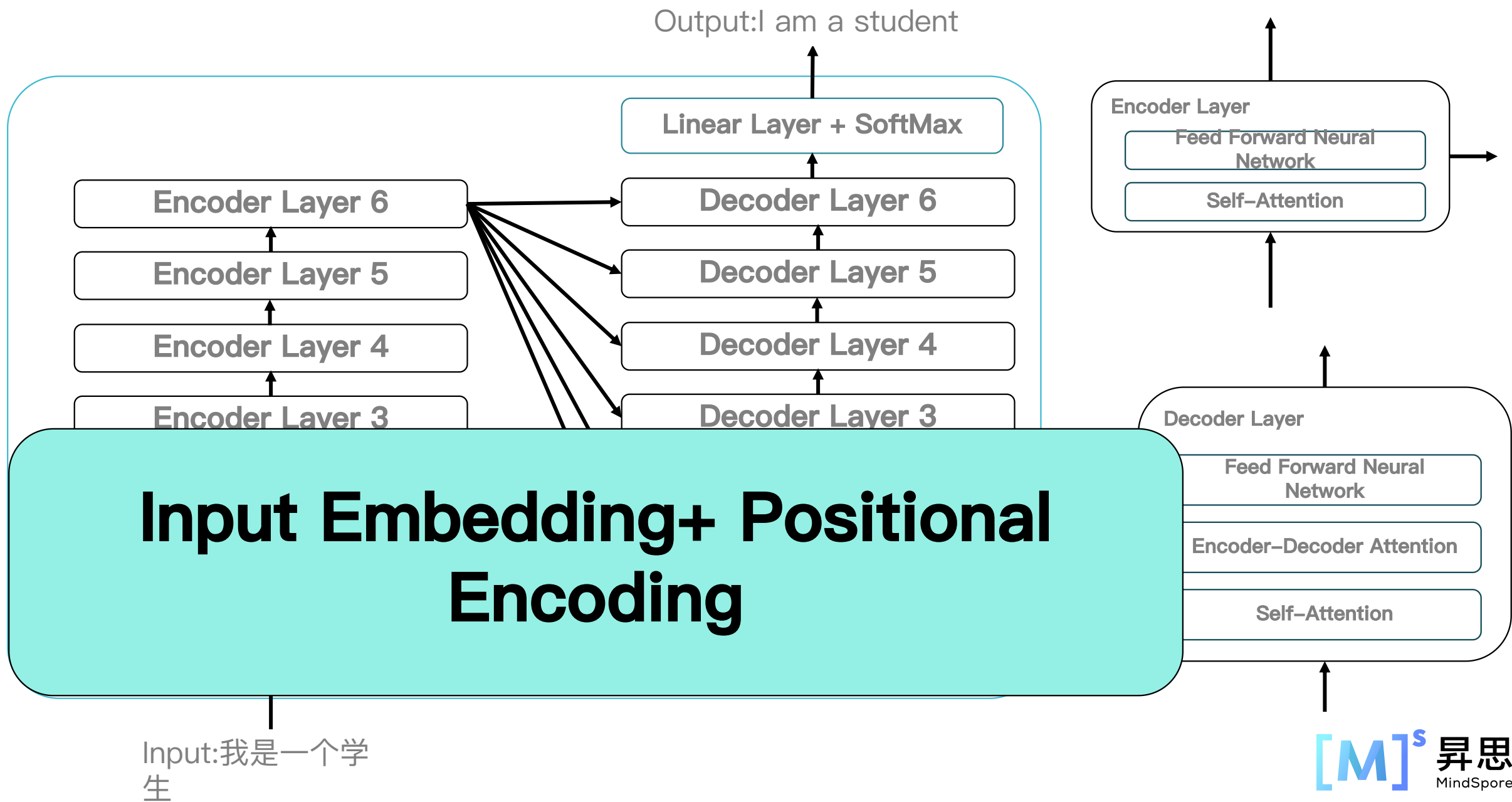


模型结构

Transformer



Output:I am a student



数据处理

Bag of Words (BoW)

BoW 是一种将句子拆解成每个词出现次数的方法，例如将以下两个句子拆解：

I like apple.

{"I": 1, "like": 1, "apple": 1}

I like mango.

{"I": 1, "like": 1, "mango": 1}

这方法乍看之下分析出了句子的组成，但是却忽略掉了很重要的次序关系，会使得以下两个意思相反句子得出的结果完全相同。

I like apple, but I don't like mango.

I like mango, but I don't like apple.

数据处理

one-hot 独热码

将句子中每个字转换为向量，最后结合起来变成矩阵。

["I", "like", "apple", "mango"]

I => [1, 0, 0, 0]

like => [0, 1, 0, 0]

apple => [0, 0, 1, 0]

mango => [0, 0, 0, 1]

"I like apple" => [[1, 0, 0, 0], [0, 1, 0, 0], **[0, 0, 1, 0]**]

"I like mango" => [[1, 0, 0, 0], [0, 1, 0, 0], **[0, 0, 0, 1]**]

如此一来不仅可以保留句子中字词的顺序，使 BoW 范例会中两个语意相反的句子有不同的呈现，且改变为适合电脑运算的矩阵形式，但产生了新的问题是“**当字词过多时每个字的向量会太长，句子转换的矩阵亦会更加庞大，且这矩阵中有大量的栏位皆为 0**”。这么一来使得电脑多了许多额外的运算及储存成本，而且整体的效率十分低。

数据处理

Tokenizer分词器 + Embedding词向量

词汇	I	like	apple	mango	and
index	1	2	3	4	5

词汇表

Tokenizer分词器的作用是把句子进行分割成字词token，然后给出token的索引index，token与index是一一对应的。

例如 “I like apple.”这句经过tokenizer后得到：

I => 1 like => 2 apple => 3

但这样仍然无法表示词与词之间的关系，比如apple和mango都是水果，这两个词的相关程度就比apple和like要强，因此我们需要使用词向量的方式来表示字词。这就涉及到index转词向量，即embedding操作。

embedding操作->

词汇	I	like	apple	mango	and
index	1	2	3	4	5
词向量	[x1,y1,z1...]	[x2,y2,z2...]	[x3,y3,z3...]	[x4,y4,z4...]	[x5,y5,z5...]

为什么不提前embedding，直接在词汇表里提供词向量呢？

- 1、刚开始训练时，词向量是随机初始化的，训练后期才趋于准确；
- 2、不同的任务对词向量的维度要求也不一样，这里的维度可以理解为词性分析；

数据处理

位置编码

经过**embedding**之后获取到的词向量虽然能表示词的含义，但是无法知道词在句子中的位置，因此我们需要进行位置编码**Positional Encoding**。

用整型值标记位置

一种自然而然的想法是，给第一个token标记1，给第二个token标记2...，以此类推。

这种方法产生了以下几个主要问题：

- (1) 模型可能遇见比训练时所用的序列更长的序列，不利于模型的泛化。
- (2) 模型的位置表示是无界的。随着序列长度的增加，位置值会越来越大。

用[0,1]范围标记位置

为了解决整型值带来的问题，可以考虑将位置值的范围限制在[0, 1]之内，其中，0表示第一个token，1表示最后一个token。比如有3个token，那么位置信息就表示成[0, 0.5, 1]；若有4个token，位置信息就表示成[0, 0.33, 0.69, 1]。

但这样产生的问题是，当序列长度不同时，token间的相对距离是不一样的。例如在序列长度为3时，token间的相对距离为0.5；在序列长度为4时，token间的相对距离就变为0.33。

因此，我们需要这样一种位置表示方式，满足于：

- (1) 它能用来表示一个token在序列中的绝对位置
- (2) 在序列长度不同的情况下，不同序列中token的相对位置/距离也要保持一致
- (3) 可以用来表示模型在训练过程中从来没有看到过的句子长度。

数据处理

位置编码

Transformer位置编码如下

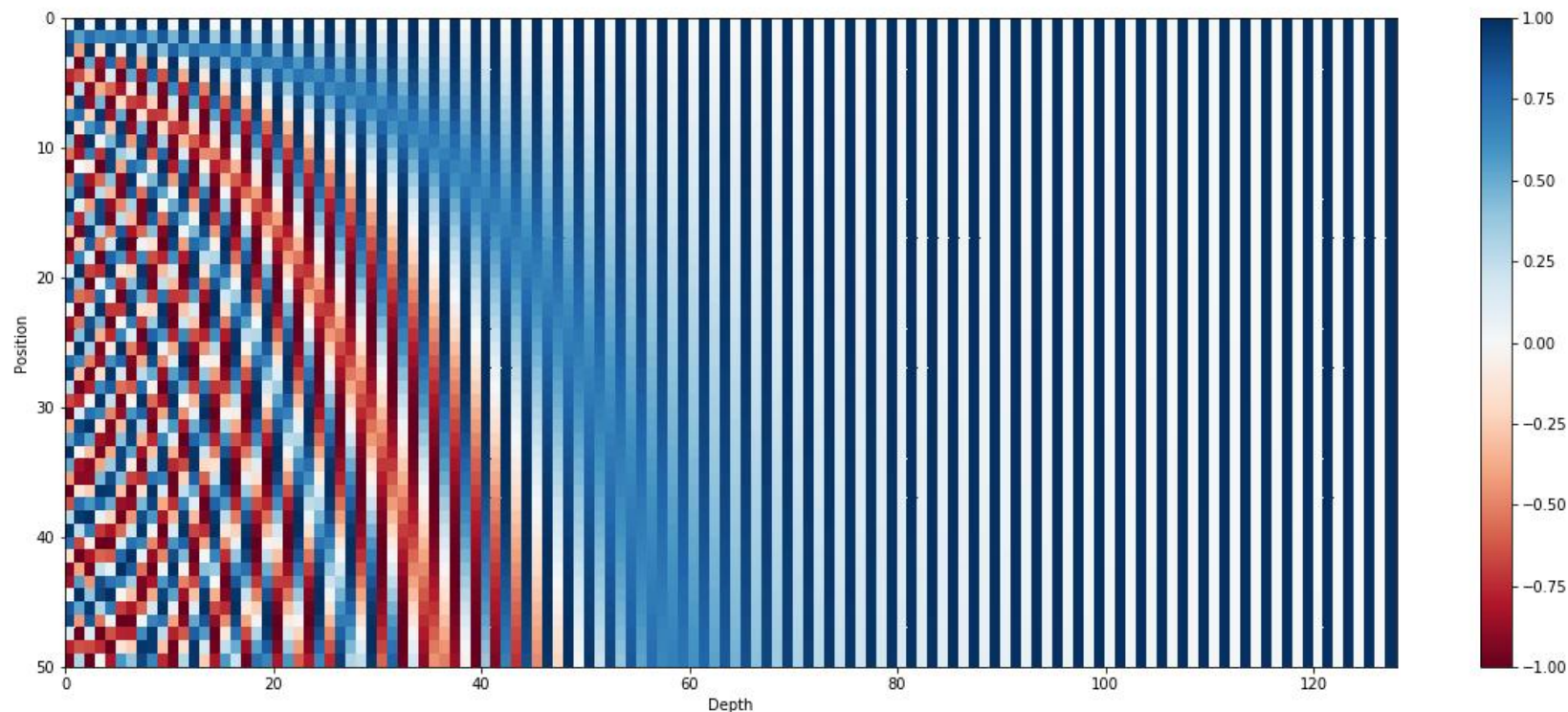
$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

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

pos: token位置

2i+1: 位置编码的第几维

d_{model} : 总维度数

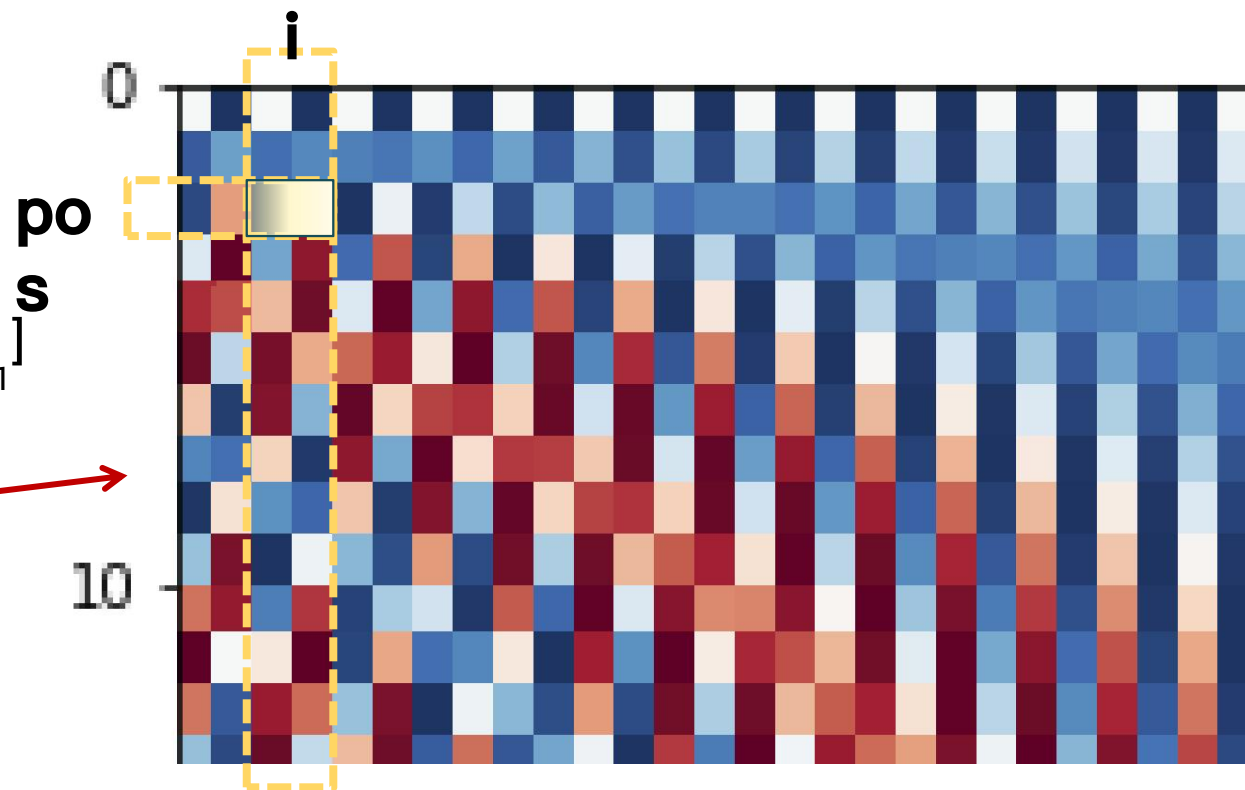
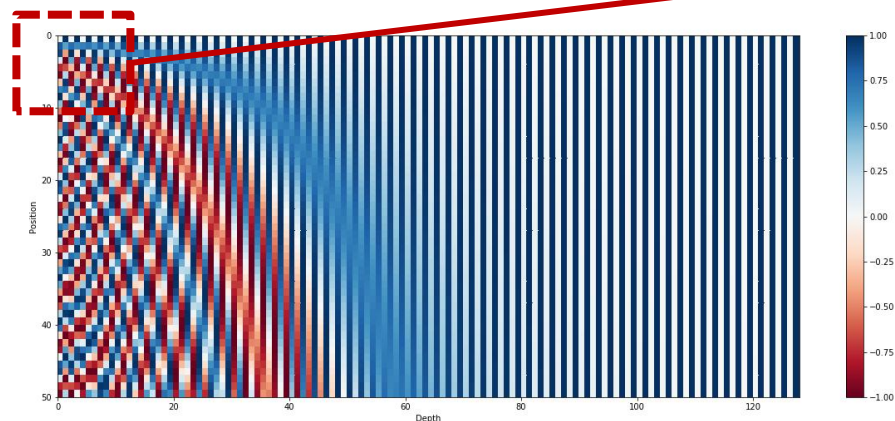


- (1) 它能用来表示一个token在序列中的绝对位置
- (2) 在序列长度不同的情况下，不同序列中token的相对位置/距离也要保持一致
- (3) 可以用来表示模型在训练过程中从来没有看到过的句子长度。

数据处理

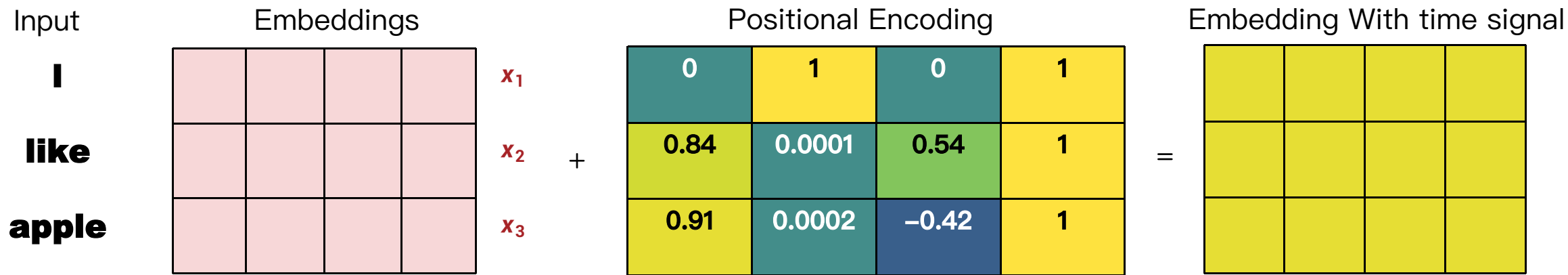
- 位于 $\text{pos}+\delta$ 出的位置编码可以线性投影到位于 i 处的位置编码
- 记 $\omega_i = 1/10000^{\frac{2i}{d_{\text{model}}}}$

$$\begin{bmatrix} \cos(\delta\omega_i) & \sin(\delta\omega_i) \\ -\sin(\delta\omega_i) & \cos(\delta\omega_i) \end{bmatrix} \begin{bmatrix} p_{\text{pos}, 2i} \\ p_{\text{pos}, 2i+1} \end{bmatrix} = \begin{bmatrix} p_{\text{pos} + \delta, 2i} \\ p_{\text{pos} + \delta, 2i+1} \end{bmatrix} \quad \mathbf{s}$$



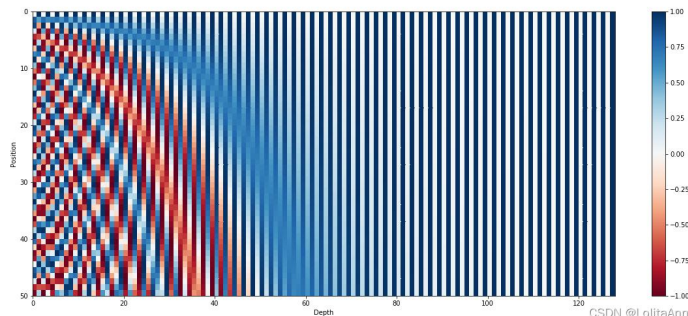
把 i 、 δ 固定住，比方说 $\delta=1$ ，只看 pos 变化

数据处理



为什么位置编码是和词嵌入相加而不是将二者拼接concat起来呢？

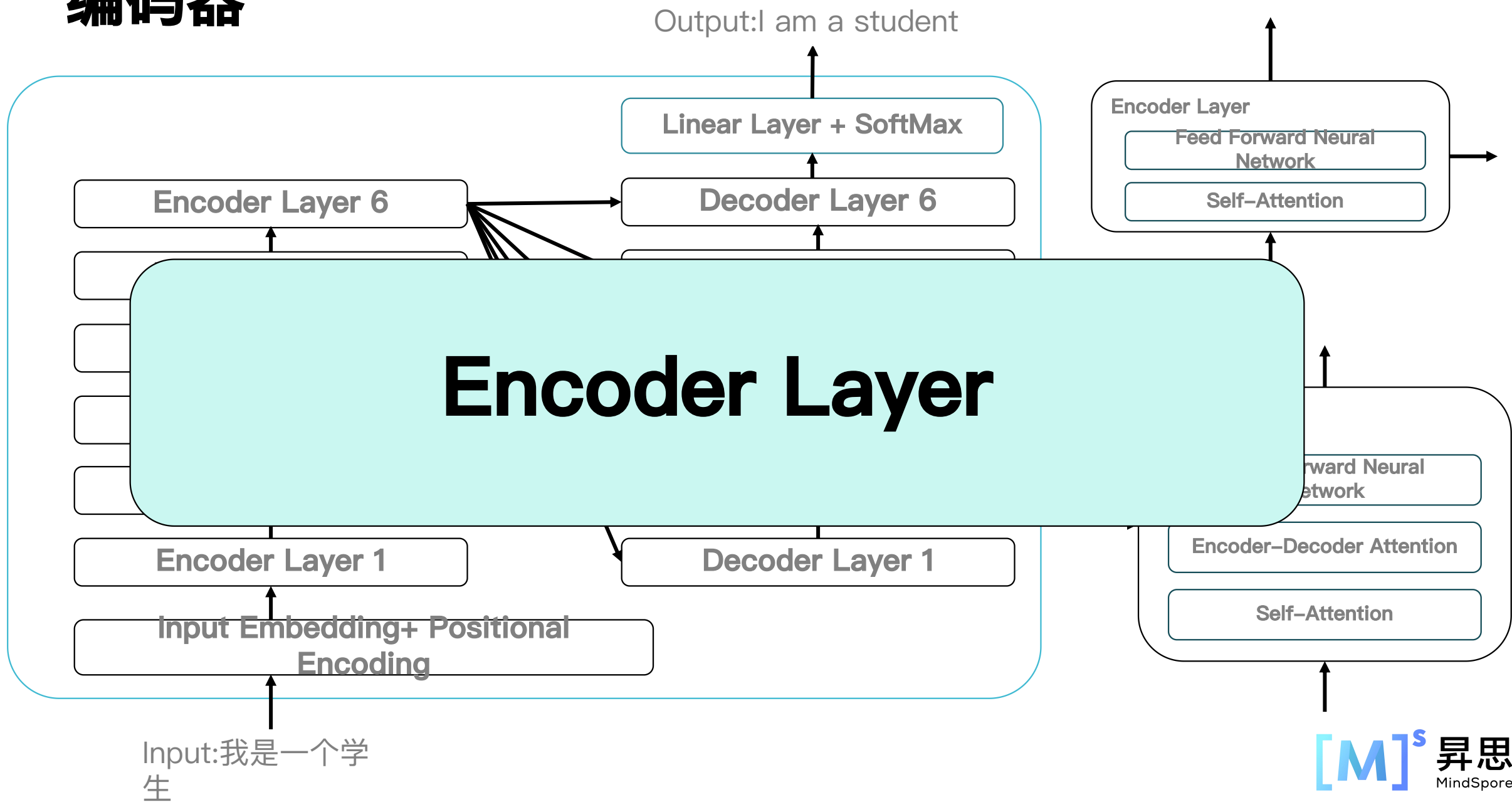
作者意思是，虽然没有进行直接concat，但是进行了隐式concat。位置编码前半段比较有用，所以在编码嵌入向量的时候，将其语义信息往后放。



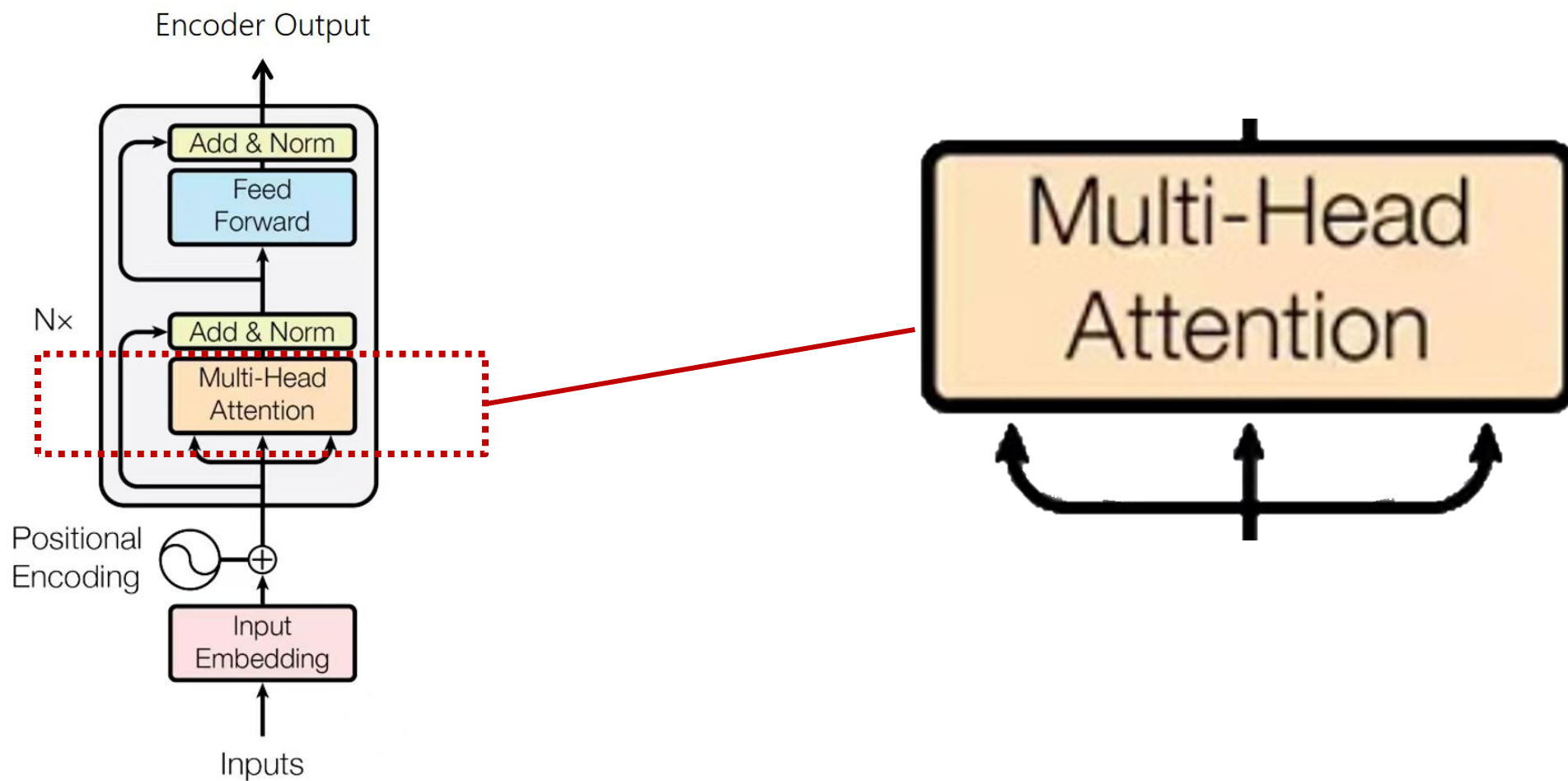
位置信息

词向量信息

编码器



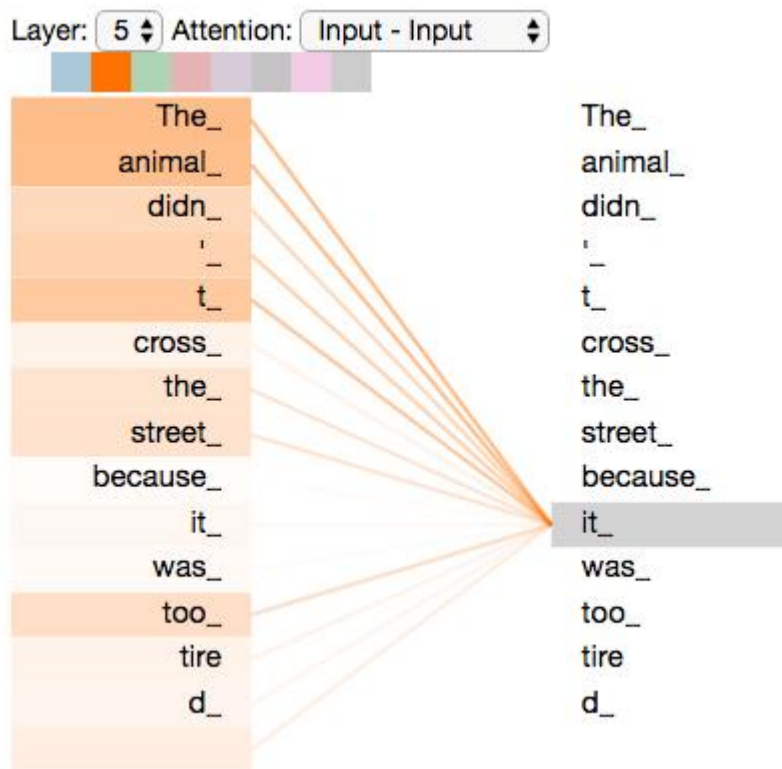
编码器



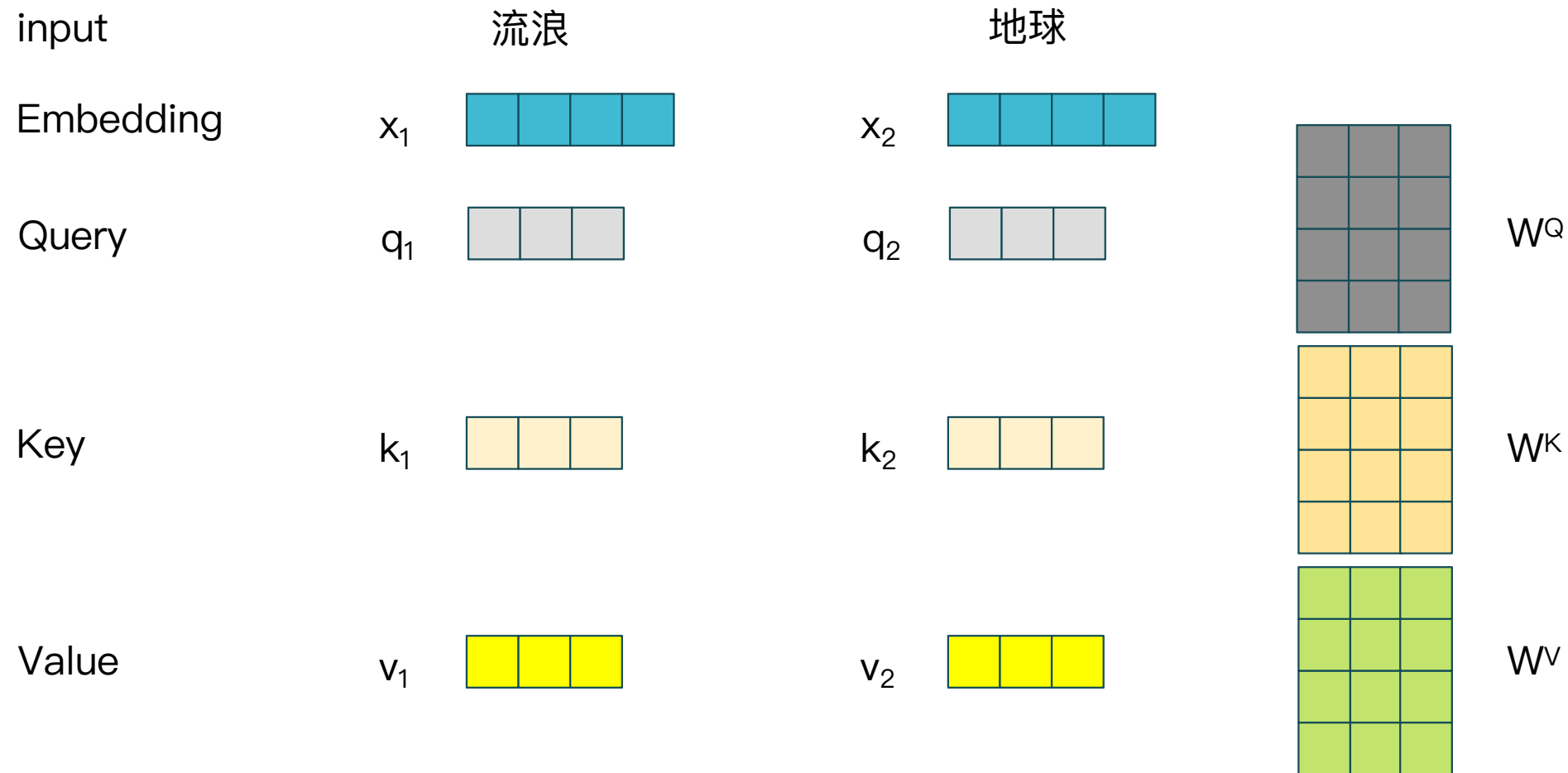
自注意力机制

The animal didn't cross the street because it was **too** tired.

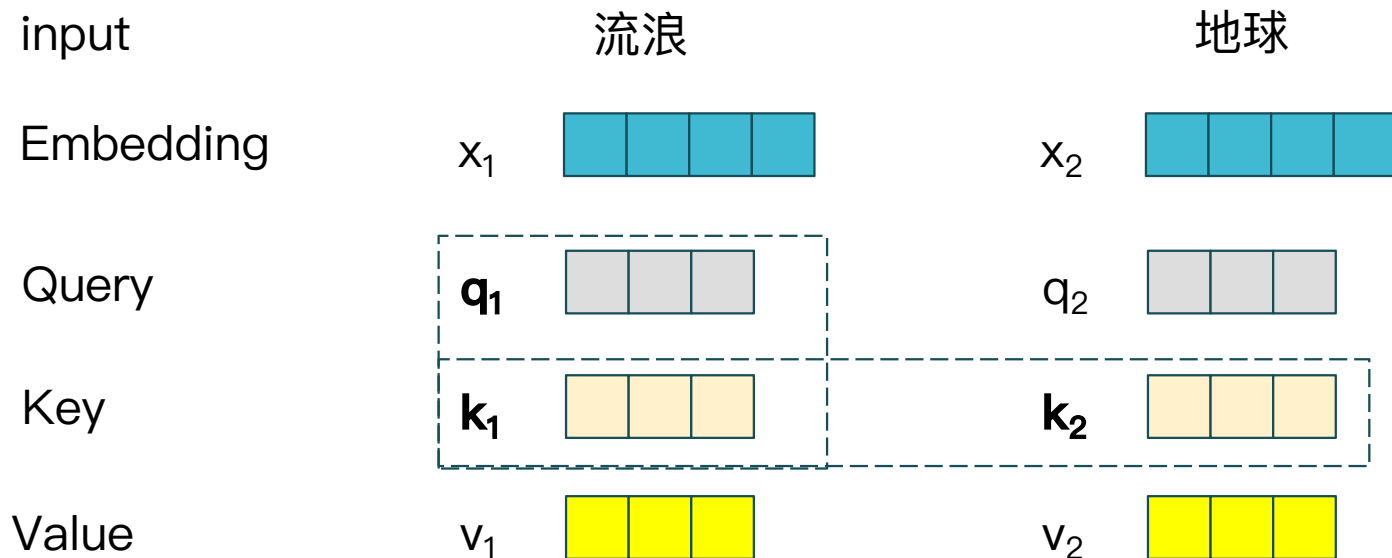
这句话中的“它”指的是什么？它指的是街道还是动物？



自注意力机制



自注意力机制

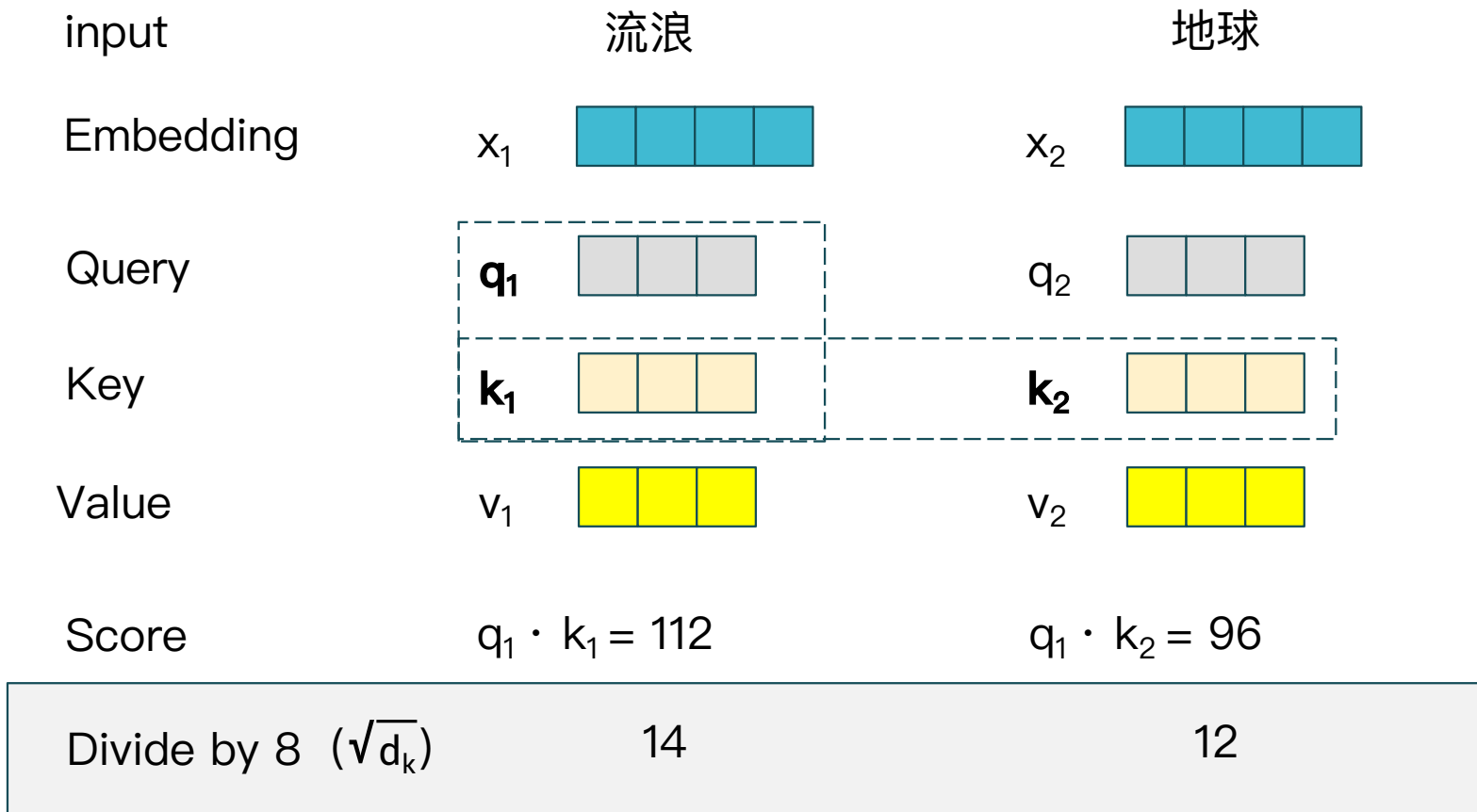


Score

$$q_1 \cdot k_1 = 112$$





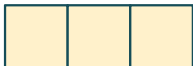
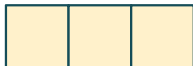


$$q_1 \cdot k_2 = 96$$

自注意力机制





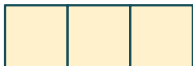
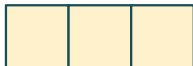



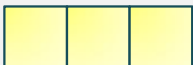


有些qk对在经过点积运算后的结果相对于其他点积对来说过大，这就导致在之后的softmax中占据了绝大多数注意力，而其他的点积对在经过softmax之后的分数则趋于0，结果就是会出现梯度消失的问题。因此这里除以一个常数的操作类似于归一化，使得各点积对的结果更加平滑。

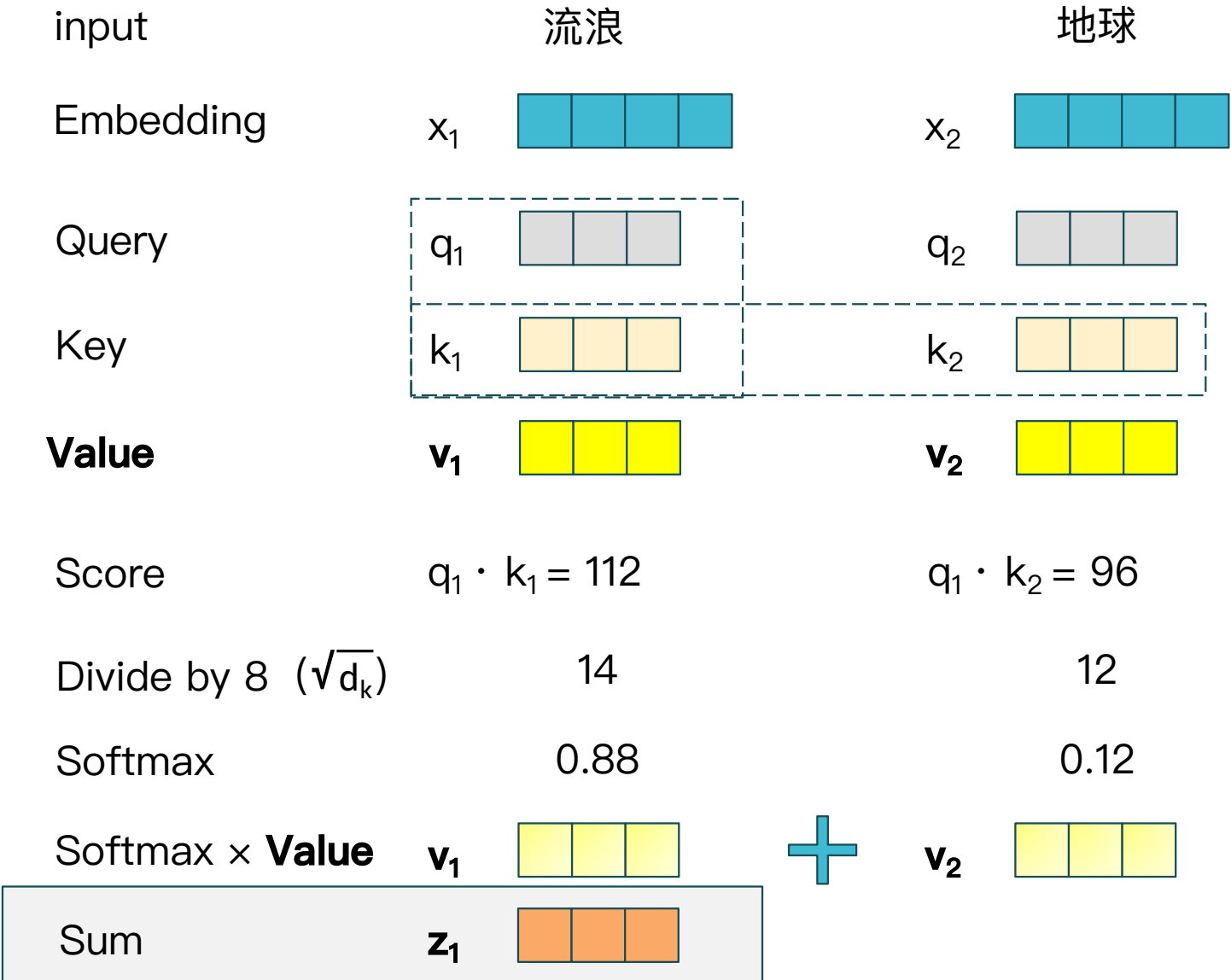
自注意力机制

input	流浪		地球	
Embedding	x_1		x_2	
Query	q_1		q_2	
Key	k_1		k_2	
Value	v_1		v_2	
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ($\sqrt{d_k}$)	14		12	
Softmax	0.88		0.12	

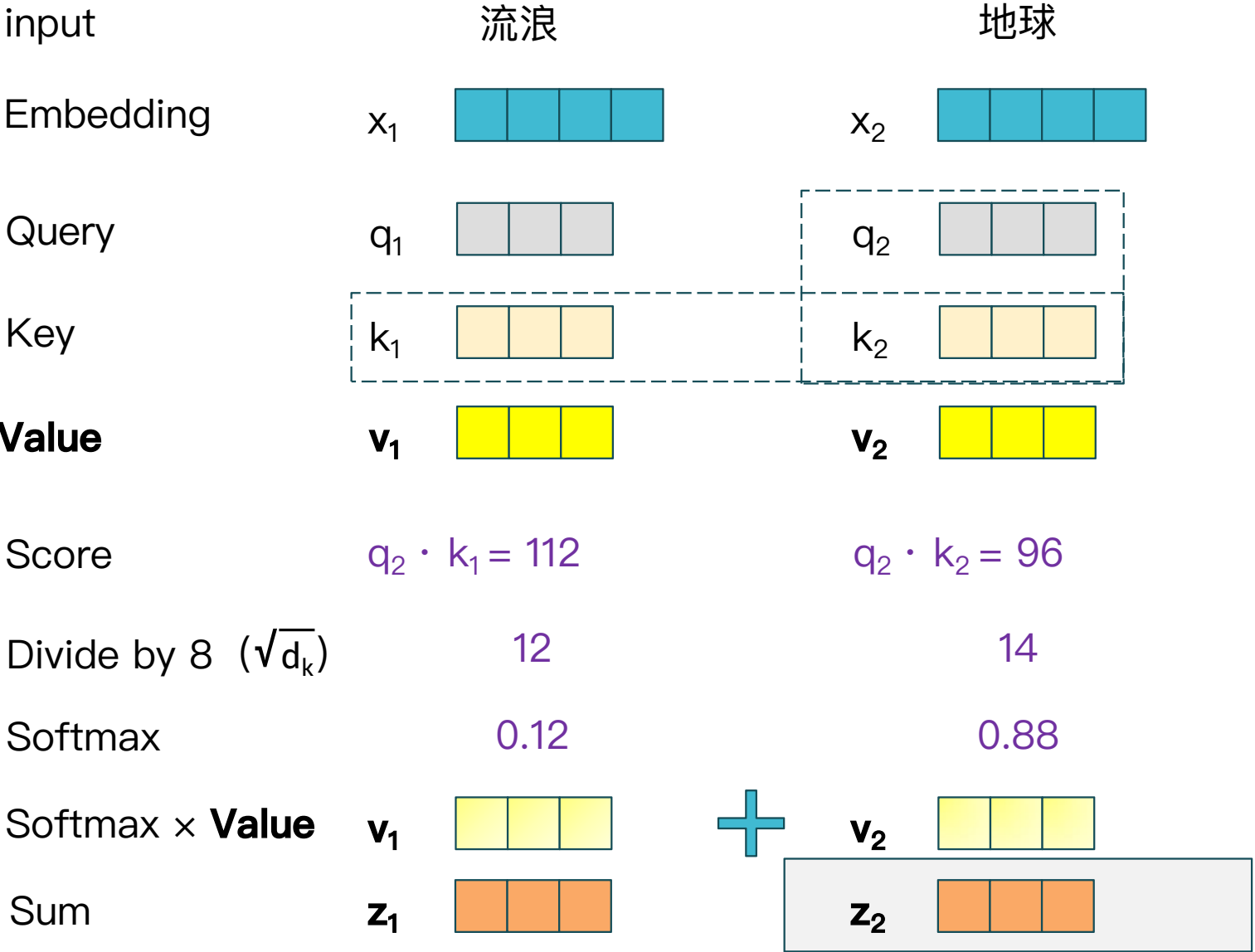
自注意力机制

input	流浪		地球	
Embedding	x_1		x_2	
Query	q_1		q_2	
Key	k_1		k_2	
Value	v_1		v_2	
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ($\sqrt{d_k}$)	14		12	
Softmax	0.88		0.12	
Softmax \times Value	v_1		v_2	

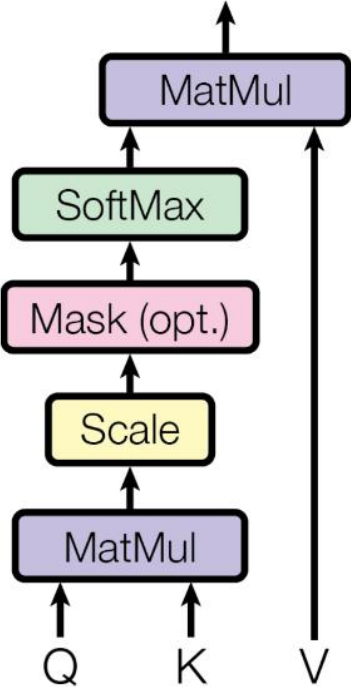
自注意力机制



自注意力机制



Scaled Dot-Product Attention



自注意力机制

$$\text{softmax} \left(\frac{q \times k^T}{\sqrt{d_k}} \right) v = \begin{matrix} z_1 \\ z_2 \end{matrix}$$

The diagram illustrates the self-attention mechanism. It shows a query matrix q (a 2x3 grid of light blue squares) multiplied by the transpose of a key matrix k^T (a 3x2 grid of light orange squares). The result is divided by the square root of the key dimension $\sqrt{d_k}$. This is followed by a softmax operation. The result is then multiplied by the value matrix v (a 2x3 grid of yellow squares). The final output is a 2x3 grid of orange squares, labeled z_1 and z_2 on the left.

多头注意力

input

流浪

attention1

attention2

Embedding

x_1



Query

q_{11}

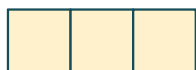


q_{12}

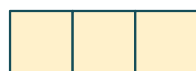


Key

k_{11}

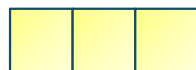


k_{12}

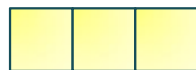


Value

v_{11}



v_{12}

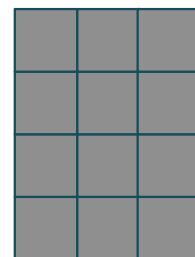


Sum

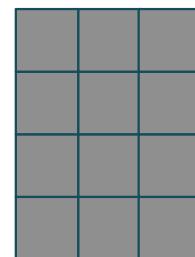
z_{11}



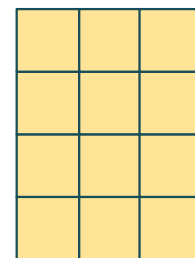
z_{12}



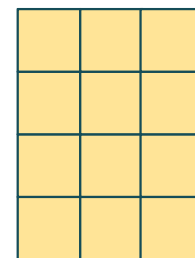
W_1^Q



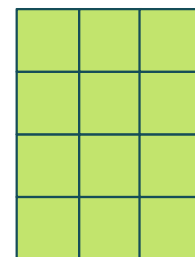
W_2^Q



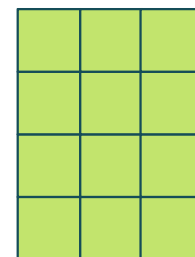
W_1^K



W_2^K

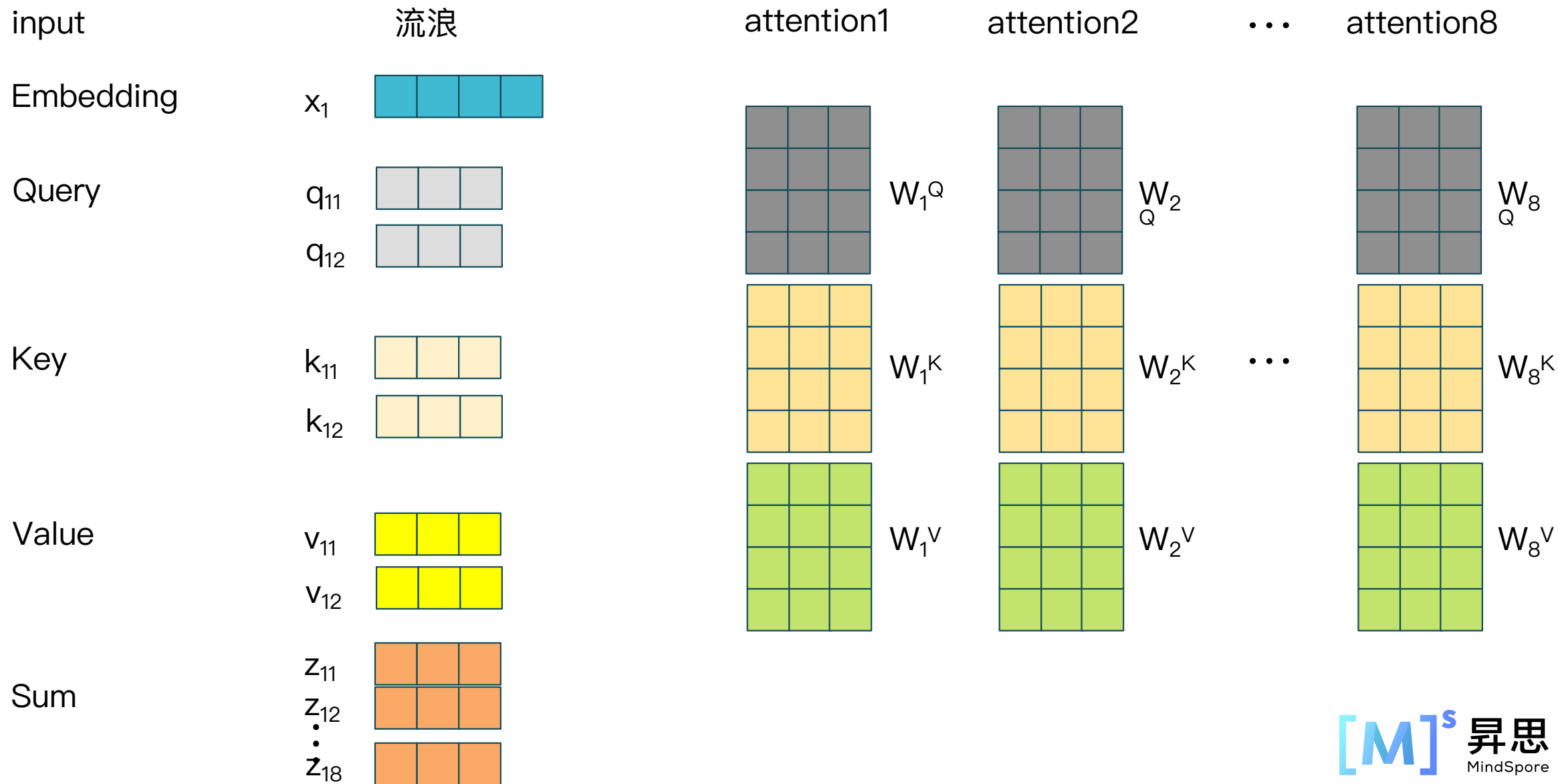


W_1^V



W_2^V

多头注意力



多头注意力

input

流浪

Embedding

x_1



Sum

z_{11}



z_{12}

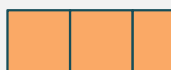


\vdots

z_{18}



Concat



z_{11}

z_{12}

\dots

z_{18}

多头注意力

input

流浪

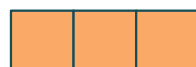
Embedding

x_1



Sum

z_{11}

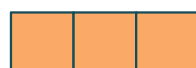


z_{12}

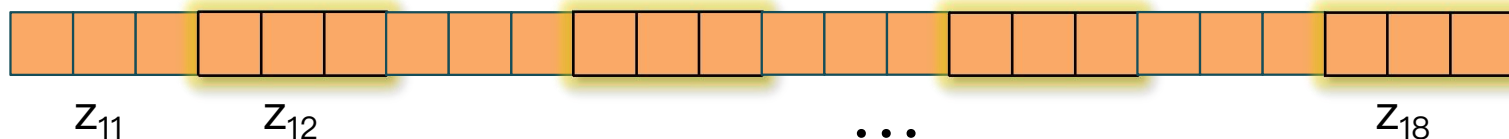


\vdots

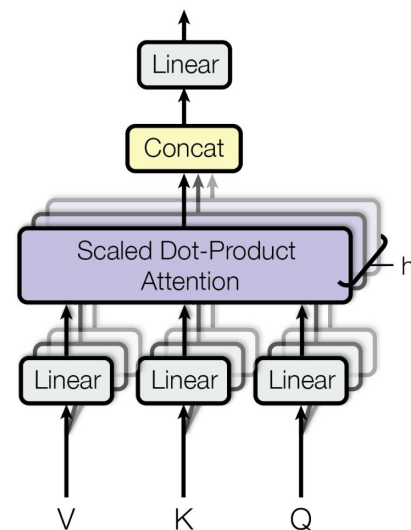
z_{18}



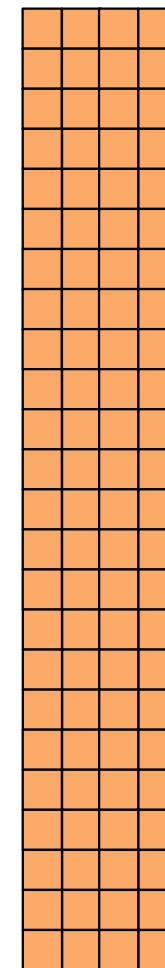
Concat



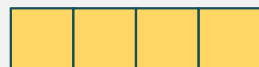
Multi-Head Attention



W^1

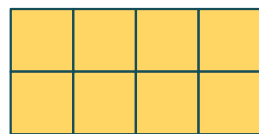


Concat $\times W^1$

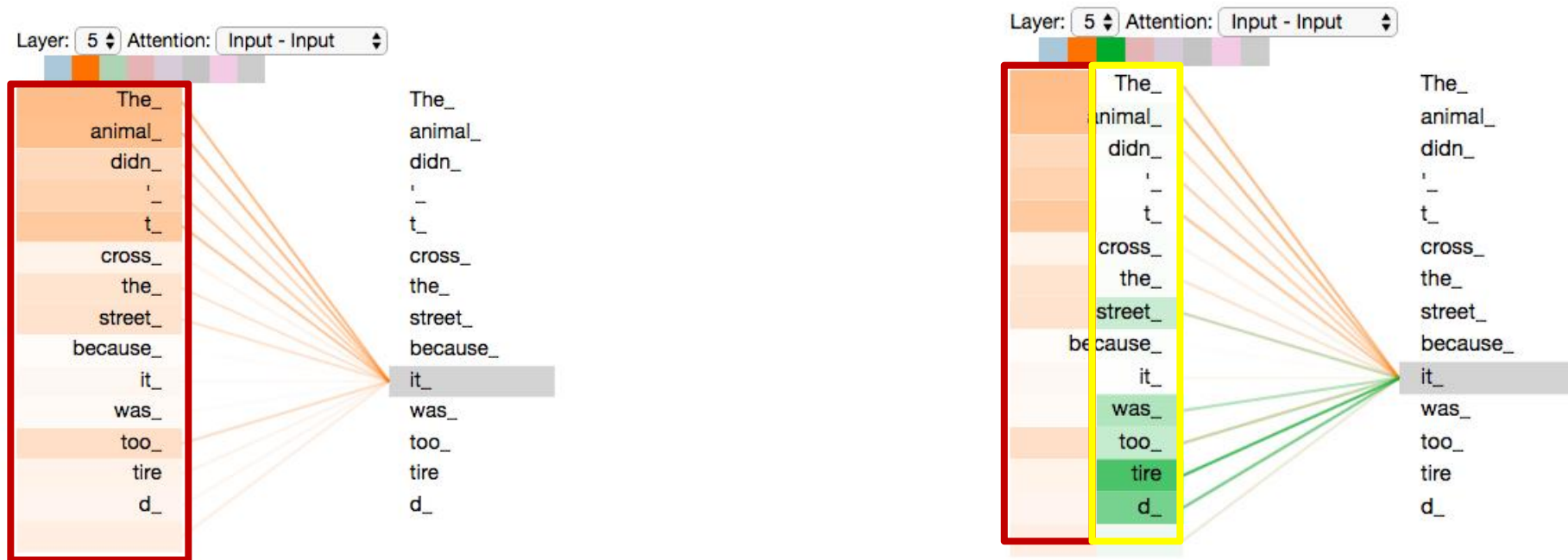


z

流浪
地球

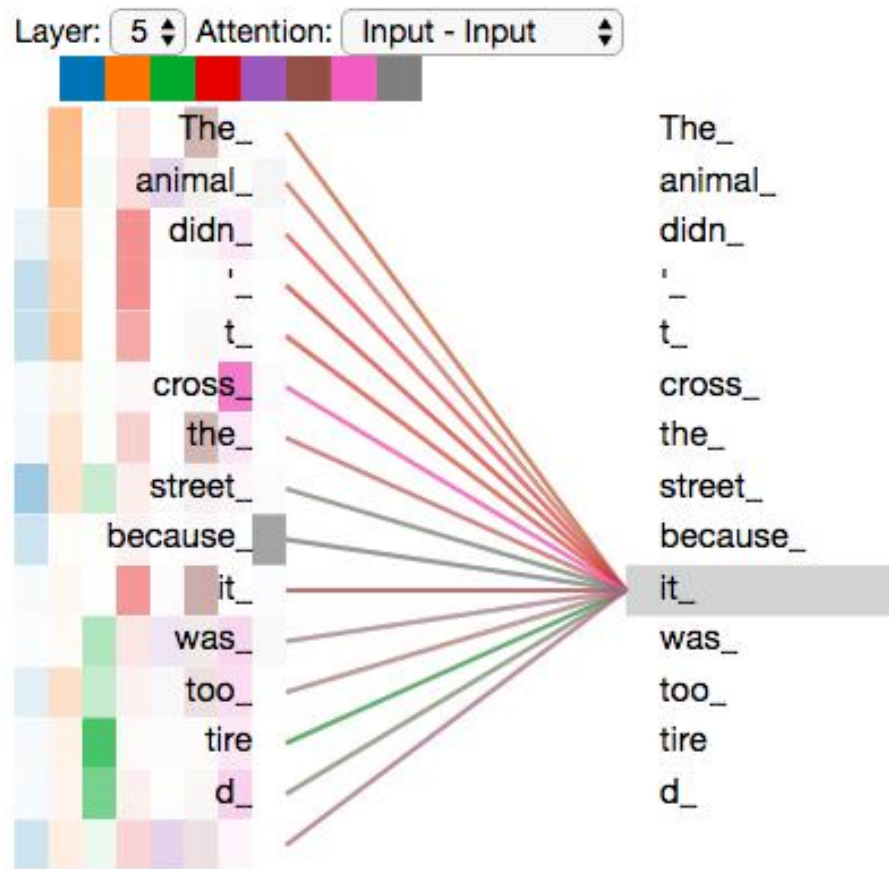


多头注意力

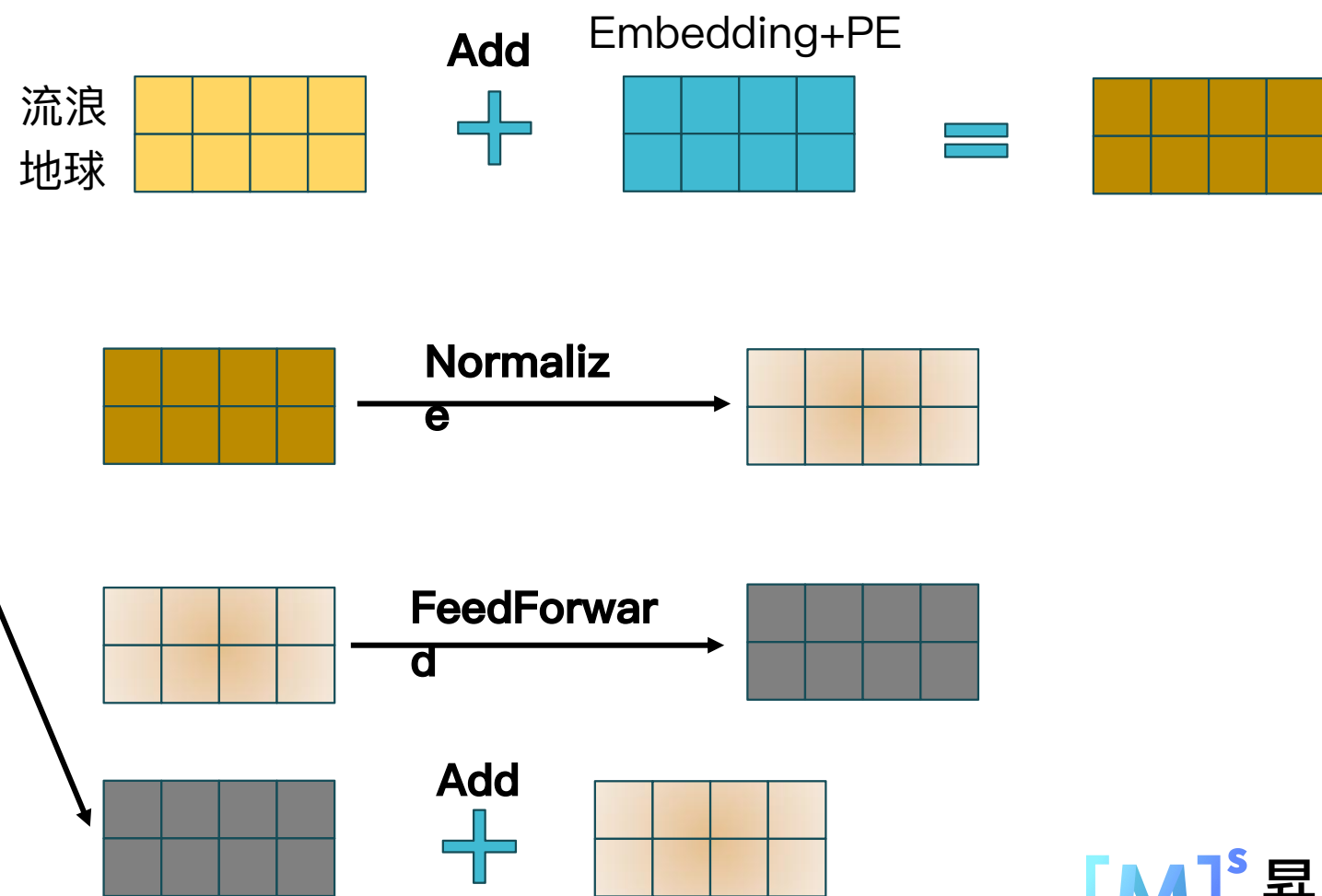
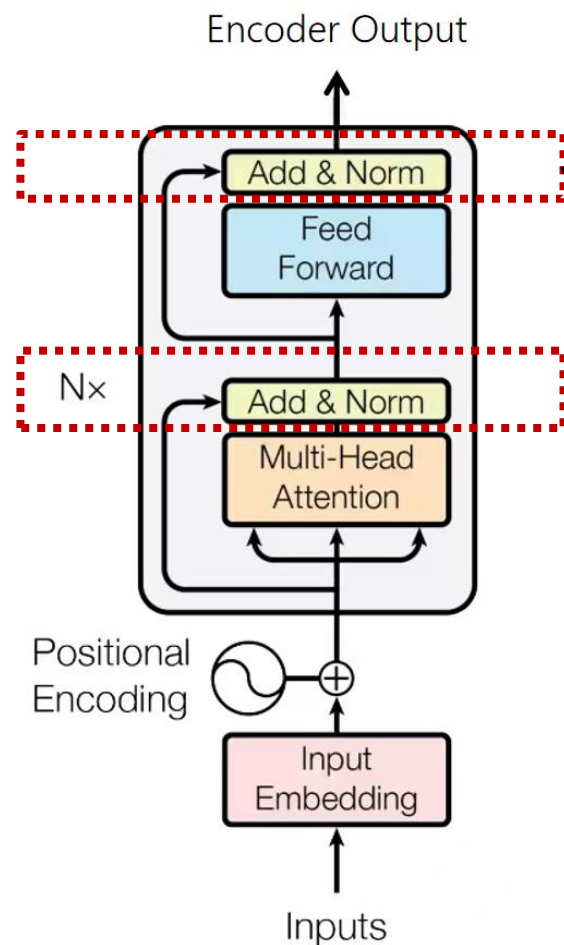


每个attention可以关注不同的部分

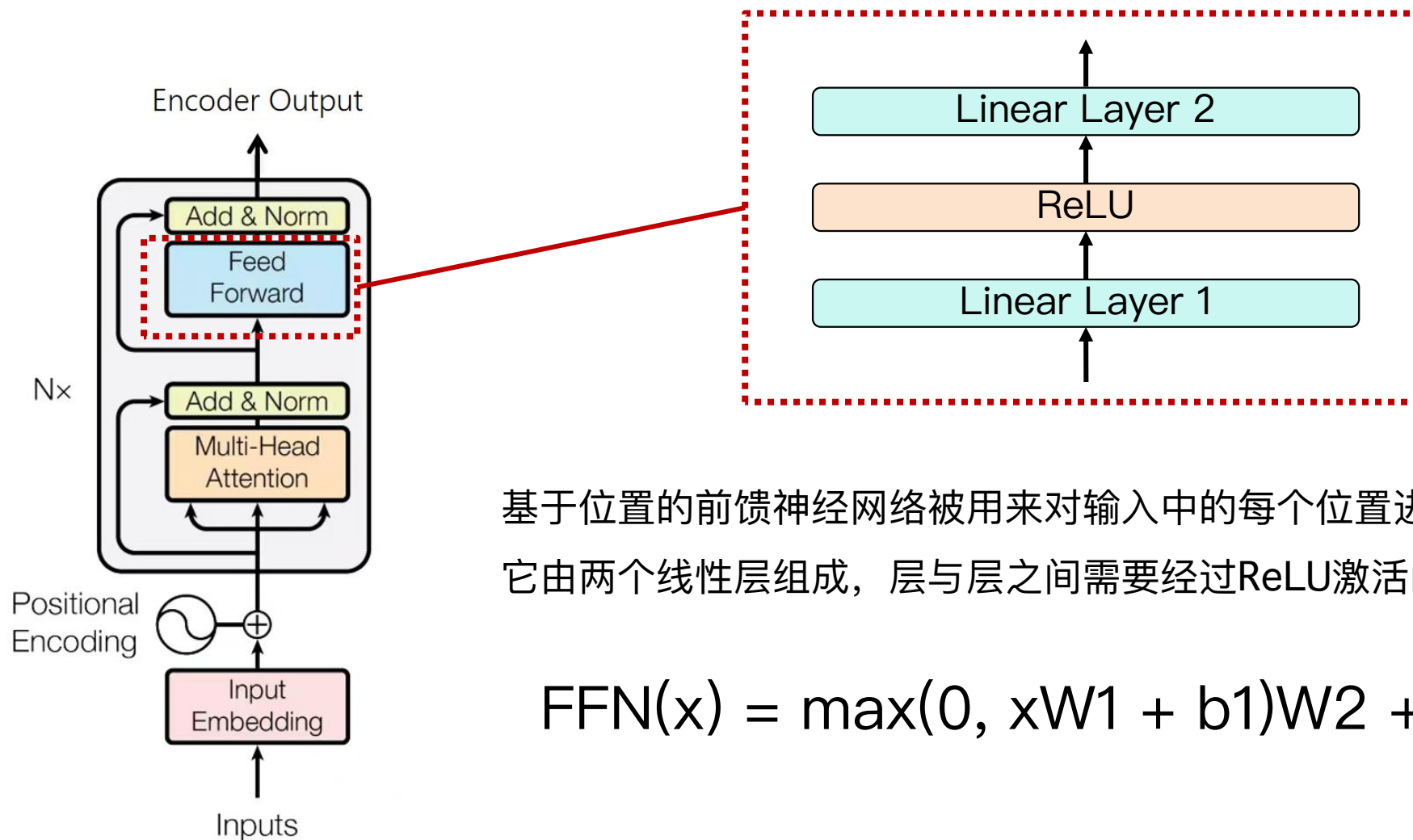
多头注意力



Add&Normalize



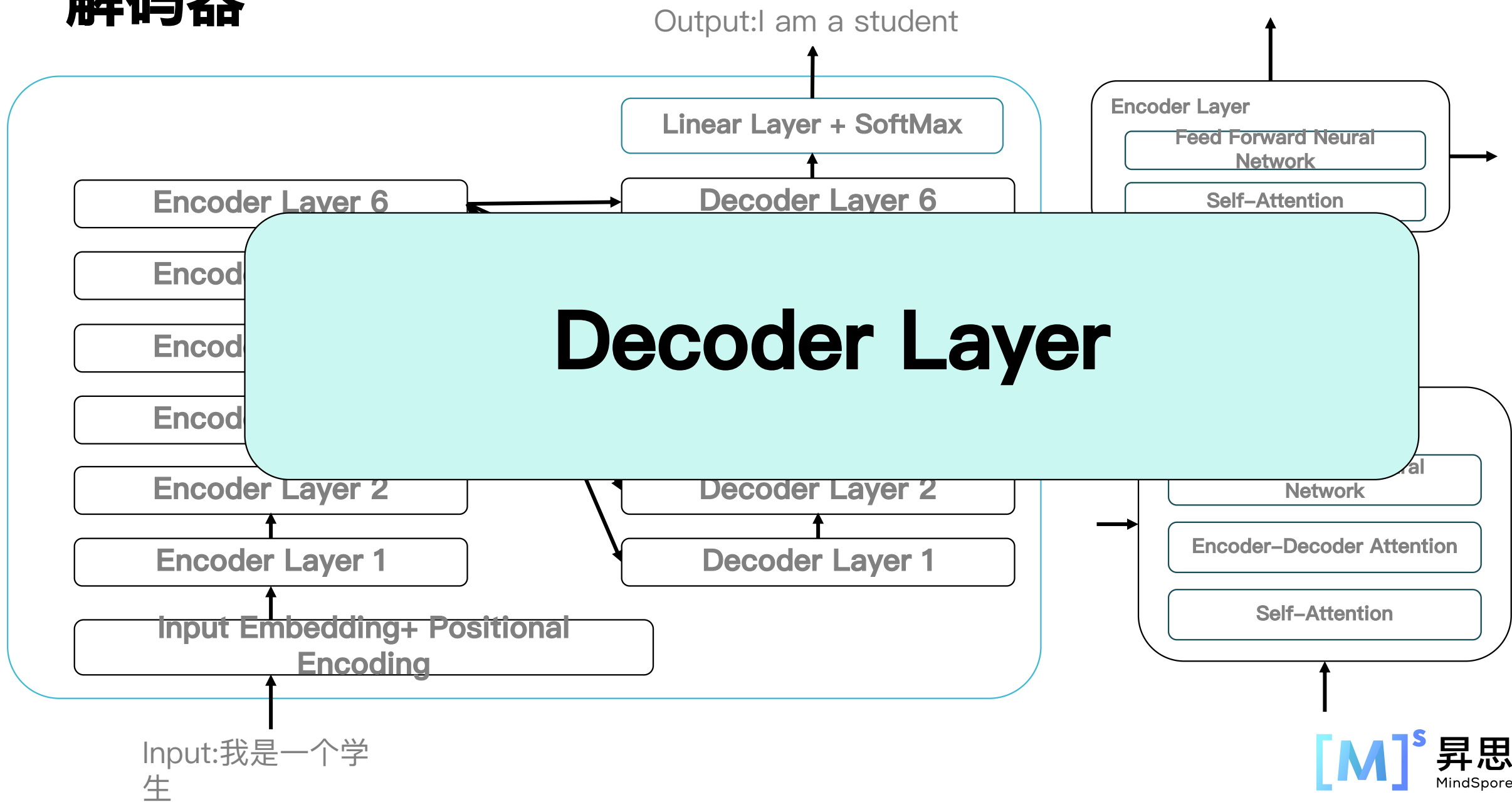
Feed Forward



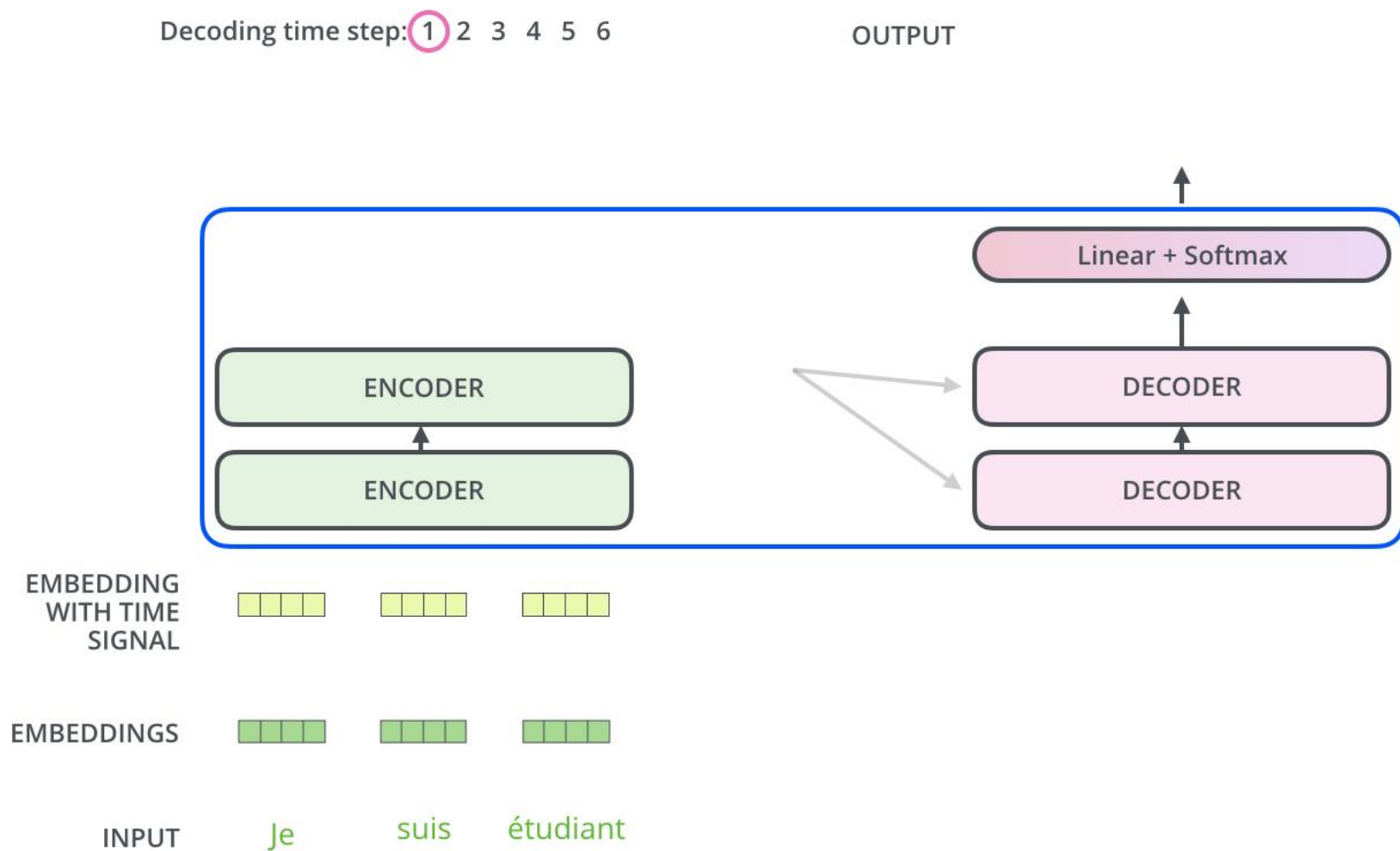
基于位置的前馈神经网络被用来对输入中的每个位置进行非线性变换。
它由两个线性层组成，层与层之间需要经过ReLU激活函数。

$$\text{FFN}(x) = \max(0, xW1 + b1)W2 + b2$$

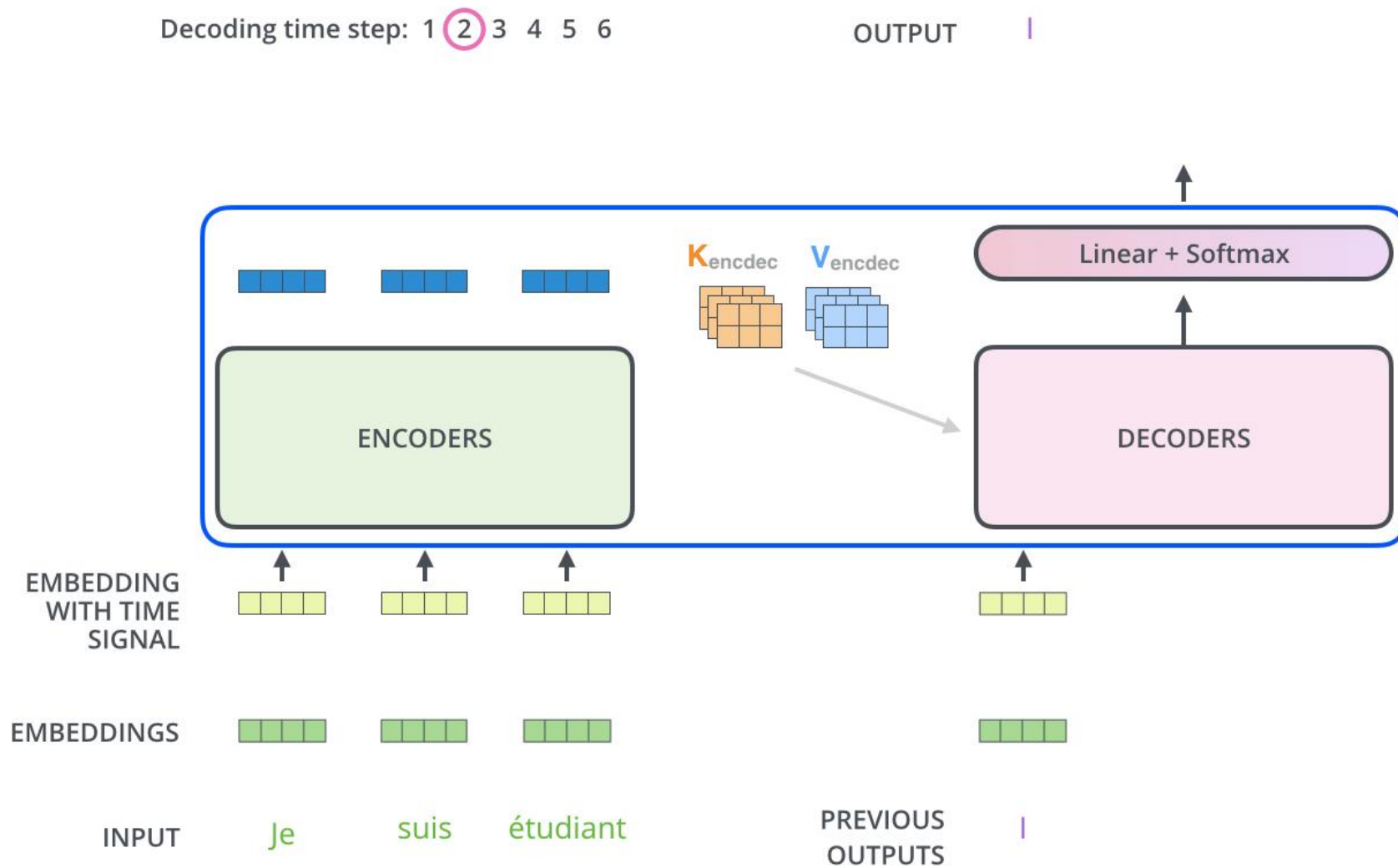
解码器



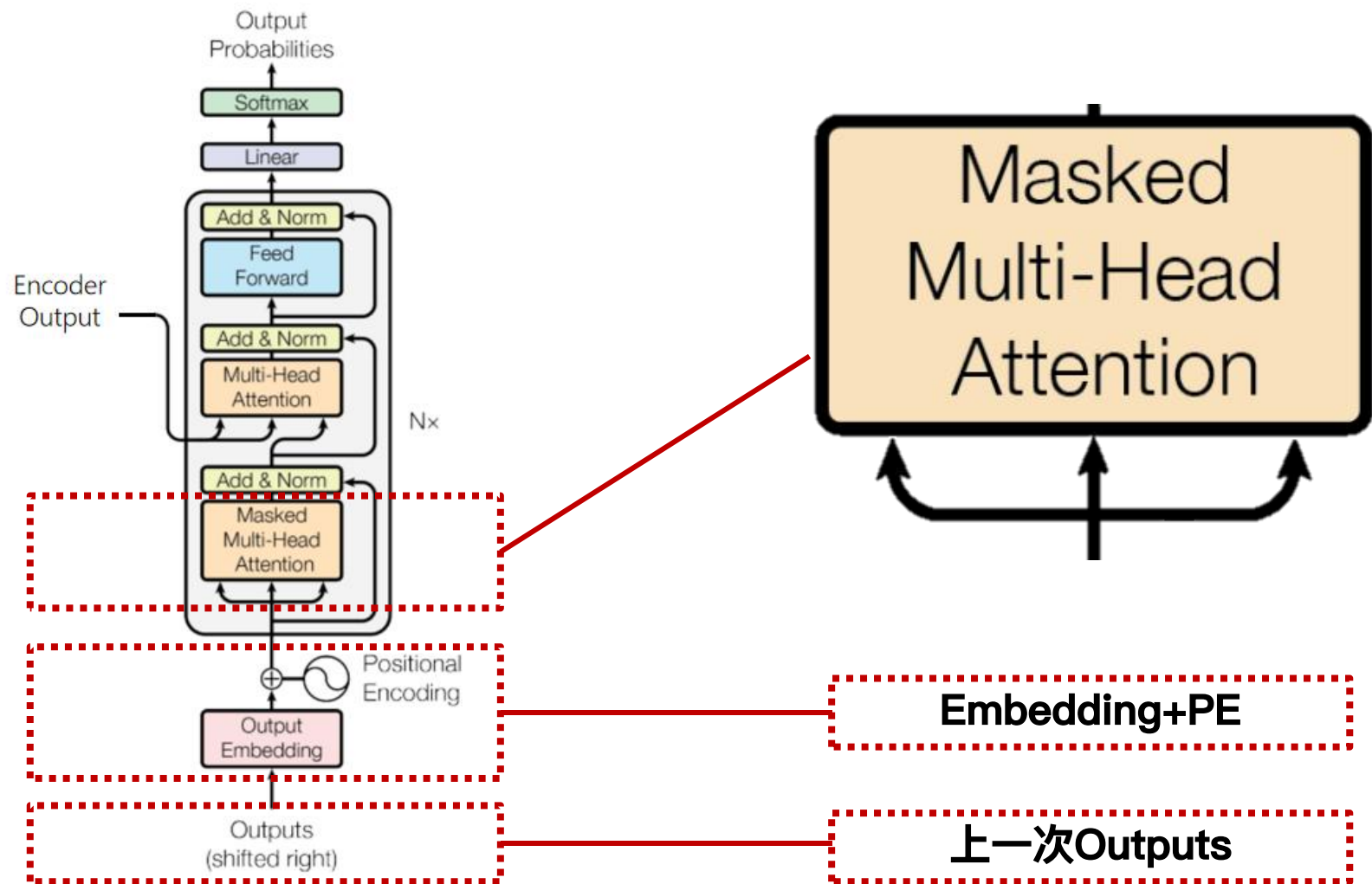
解码器



解码器



解码器

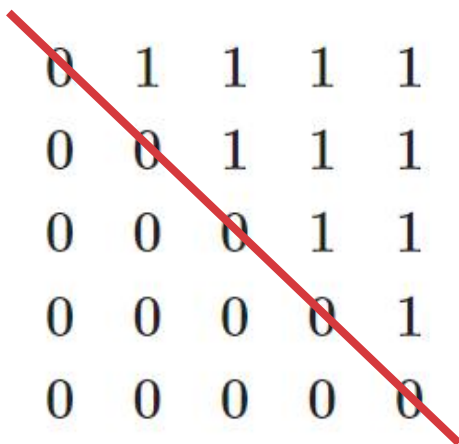


Shifted right右移一位：句子开始需要有一个“开始标识符”

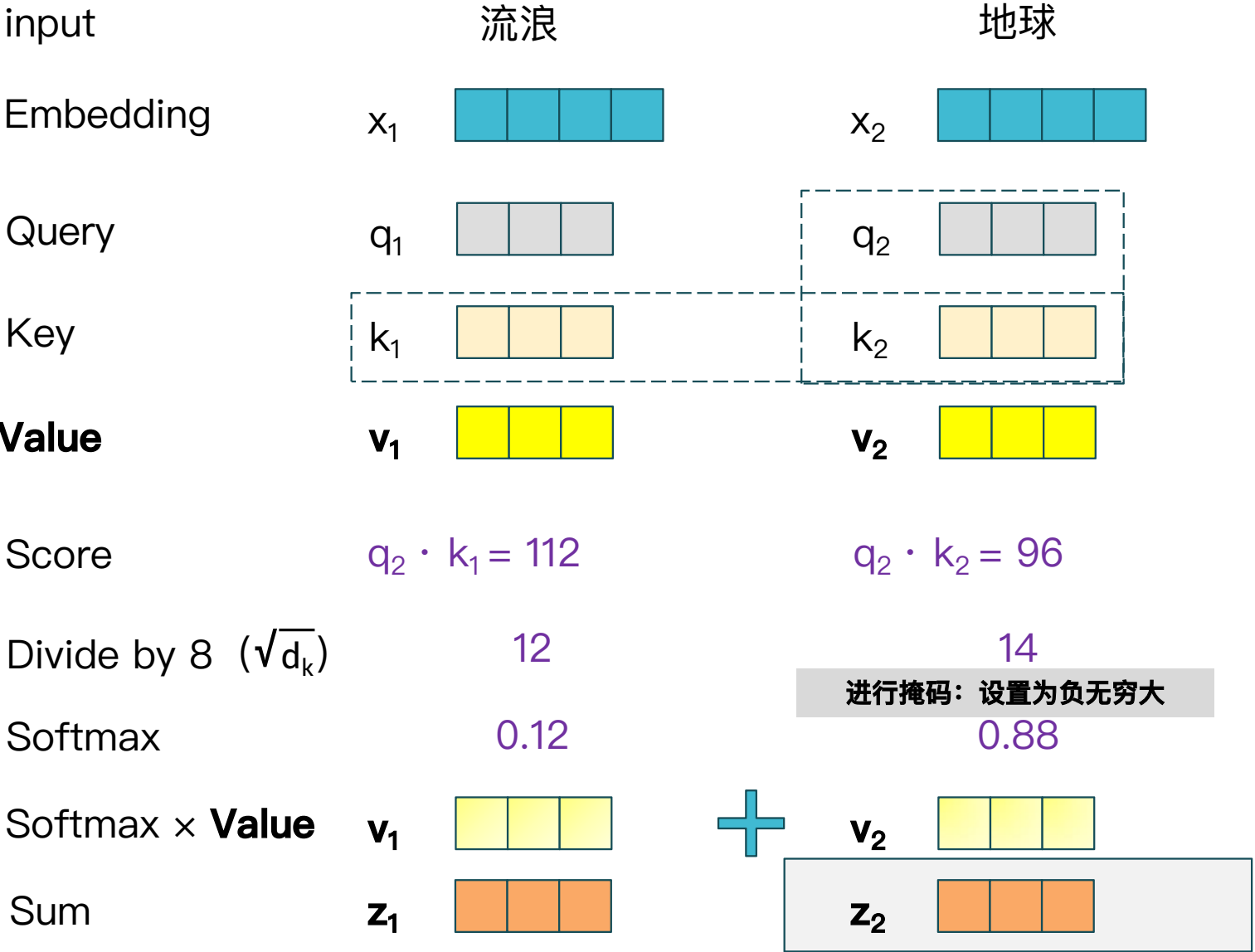
解码器

在处理目标序列的输入时， t 时刻的模型只能“观察”直到 $t-1$ 时刻的所有词元，后续的词元不应该一并输入Decoder中。

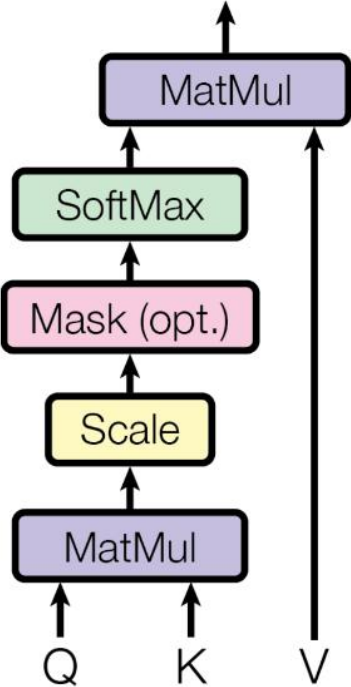
为了保证在 t 时刻，只有 $t-1$ 个词元作为输入参与多头注意力分数的计算，我们需要在第一个多头注意力中额外增加一个掩码，使目标序列中的词随时间发展逐个被暴露出来。该掩码一般被称作 subsequent mask，可通过三角矩阵实现，对角线以上的词元表示为不参与注意力计算的词元，标记为1。在计算注意力分数时，将标记为1的词元注意力分数设置成负无穷大，以便于让其在通过 softmax之后变成0



自注意力机制



Scaled Dot-Product Attention



解码器

最后，将subsequent mask和padding mask合并为一个整体的掩码，确保模型既不会注意到t时刻以后的词元，也不会关注为<pad>的词元。

输入序列

<bos>	Nice	to	meet	you	.	<eos>	<pad>	<pad>
-------	------	----	------	-----	---	-------	-------	-------

padding mask

<bos>	Nice	to	meet	you	.	<eos>	<pad>	<pad>
-------	------	----	------	-----	---	-------	-------	-------

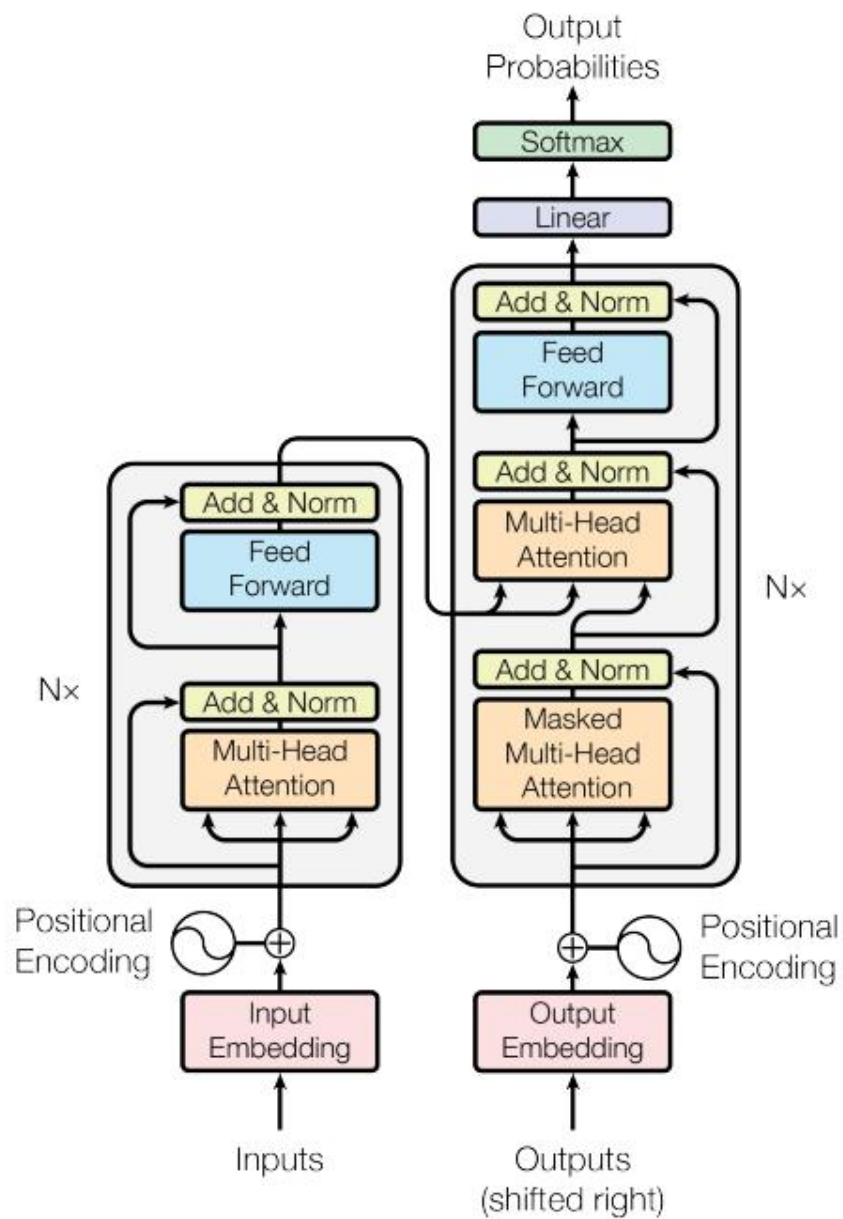
subsequent mask (t=1)

<bos>	Nice	to	meet	you	.	<eos>	<pad>	<pad>
-------	------	----	------	-----	---	-------	-------	-------

padding mask + subsequent mask

<bos>	Nice	to	meet	you	.	<eos>	<pad>	<pad>
-------	------	----	------	-----	---	-------	-------	-------

模型结构



参考

- ① <http://jalammar.github.io/illustrated-transformer/>
- ② https://blog.csdn.net/qq_36667170/article/details/124359818
- ③ <https://www.cnblogs.com/Last--Whisper/p/16934108.html>
- ④ <https://blog.csdn.net/u013177138/article/details/122361872>
- ⑤ https://blog.csdn.net/qq_36667170/article/details/124336971
- ⑥ <https://baijiahao.baidu.com/s?id=1736304808394163592&wfr=spider&for=pc>
- ⑦ https://blog.csdn.net/qq_38253797/article/details/127461558
- ⑧ <https://zhuanlan.zhihu.com/p/454482273>
- ⑨ https://www.cnblogs.com/Last--Whisper/p/16934108.html#nlp_1--%E4%BB%80%E4%B9%88%E6%98%AFembedding
- ⑩ <https://zhuanlan.zhihu.com/p/454482273>
- ⑪ <https://arxiv.org/pdf/2106.04554.pdf>
- ⑫ https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- ⑬ https://www.bilibili.com/video/BV19o4y1m7mo/?spm_id_from=333.337.search-card.all.click&vd_source=81f4e1ddb672b15776eb3b4731d7bc27
- ⑭ https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- ⑮ <https://www.mindspore.cn/>

Thanks