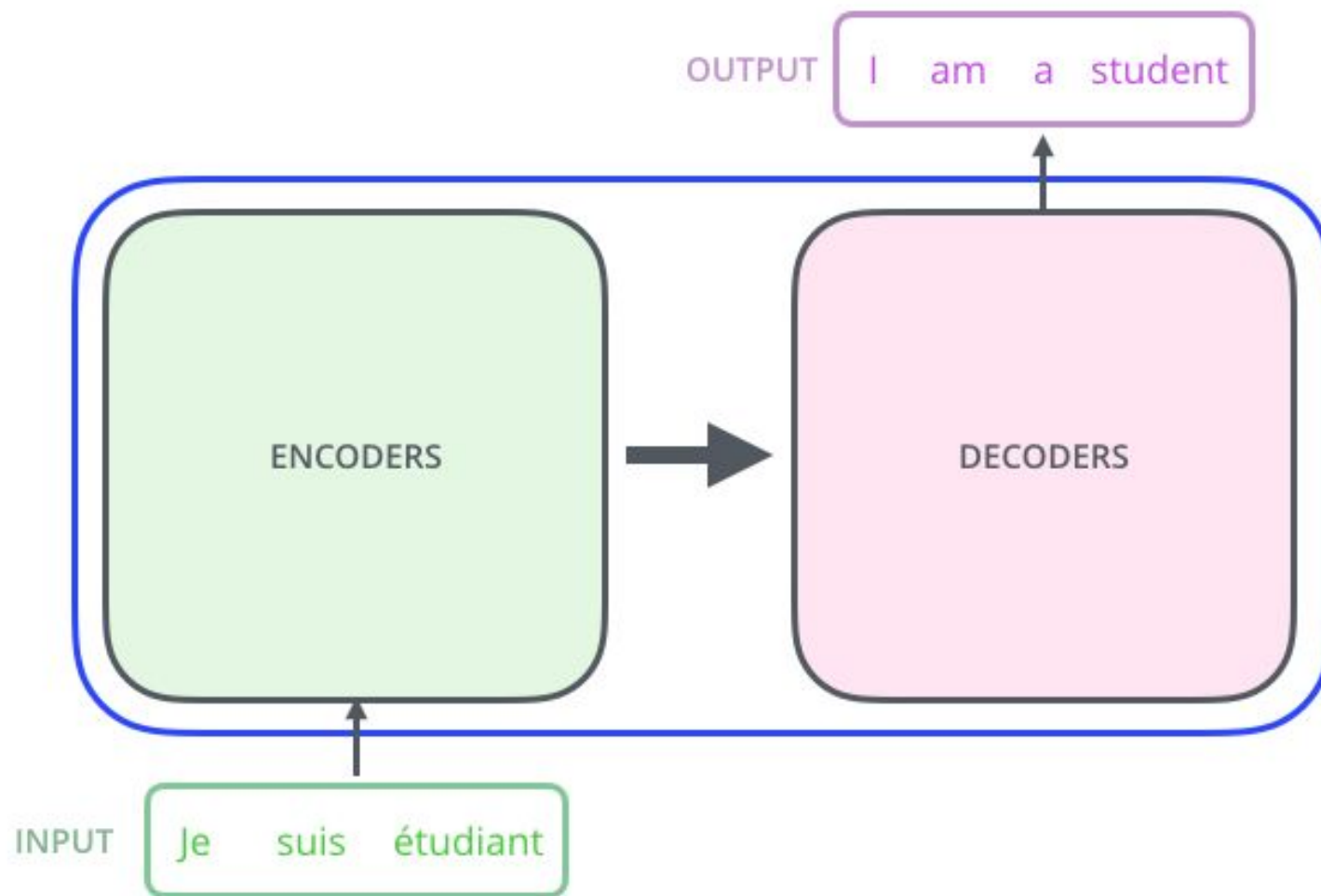


Transformer

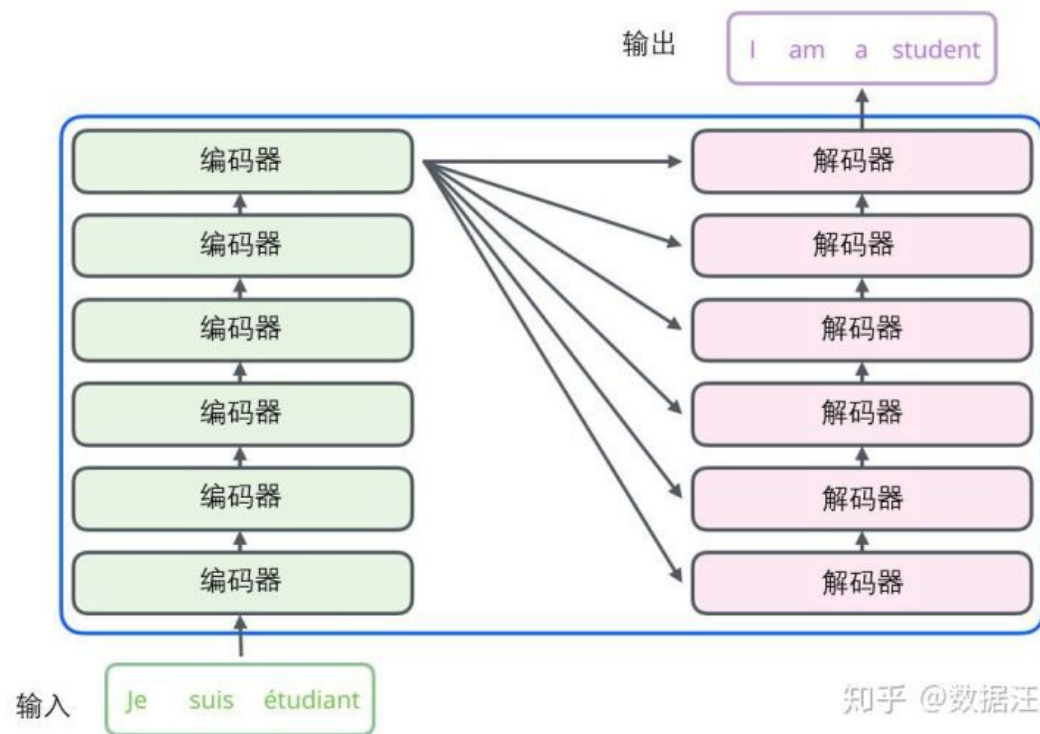
从宏观的视角开始



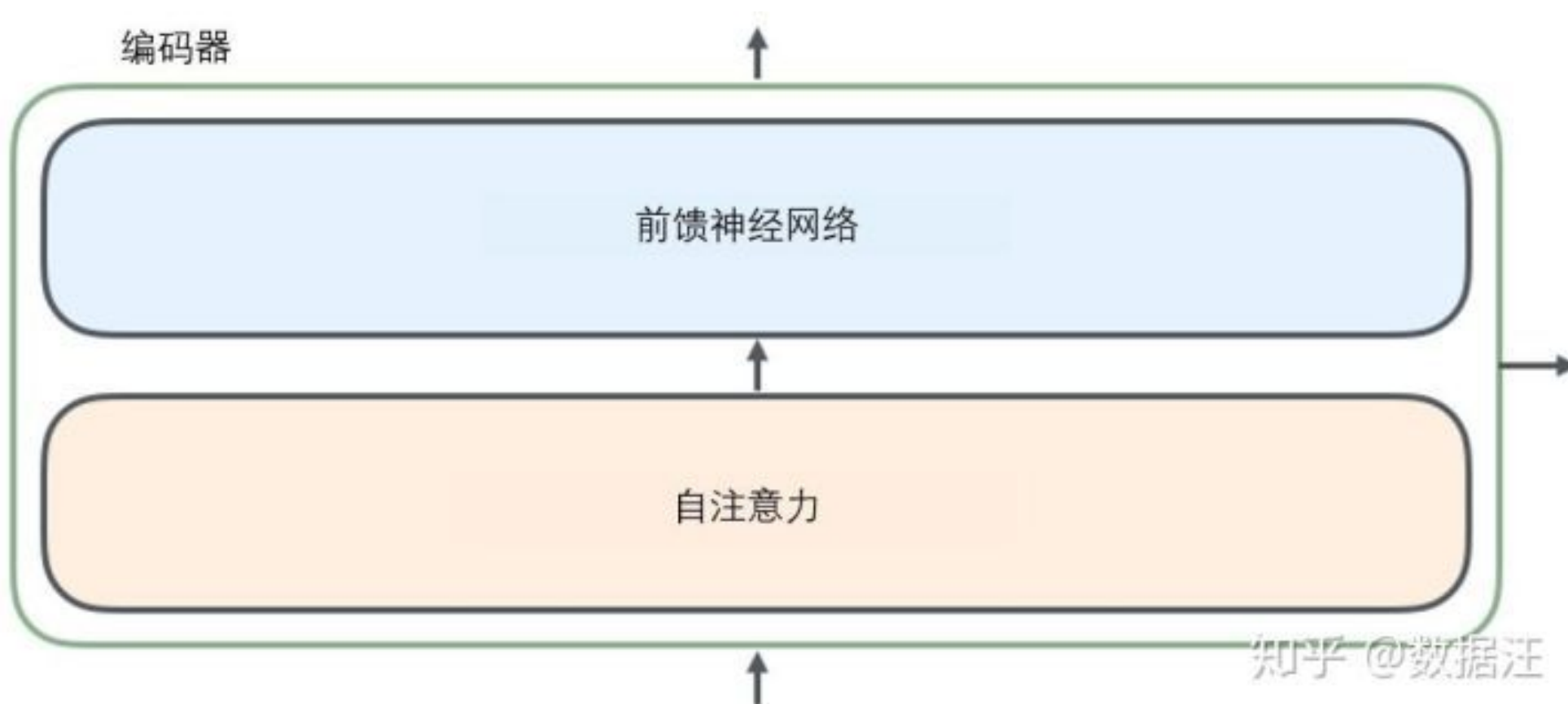
那么拆开这个黑箱，我们可以看到它是由编码组件、解码组件和它们之间的连接组成。



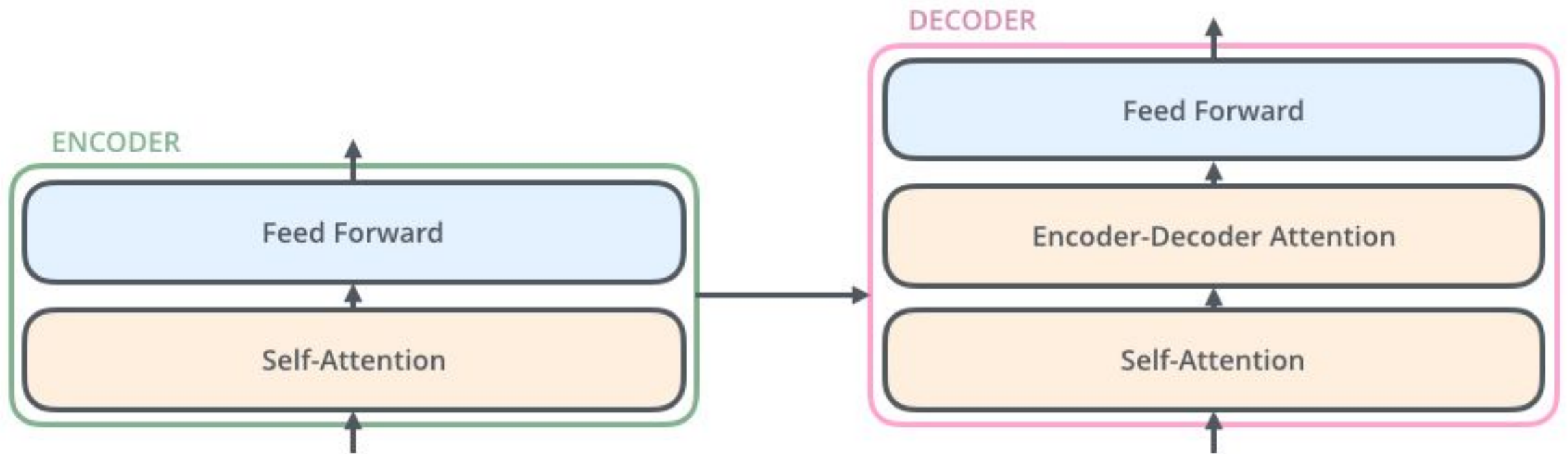
编码组件部分由一堆编码器（**encoder**）构成（论文中是将6个编码器叠在一起——数字6没有什么神奇之处，你也可以尝试其他数字）。解码组件部分也是由相同数量（与编码器对应）的解码器（**decoder**）组成的。



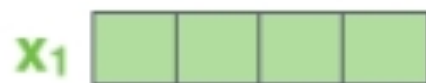
所有的编码器在结构上都是相同的，但它们没有共享参数。
每个解码器都可以分解成两个子层。



解码器中也有编码器的自注意力（*self-attention*）层和前馈（*feed-forward*）层。除此之外，这两个层之间还有一个注意力层，用来关注输入句子的相关部分（和*seq2seq*模型的注意力作用相似）。



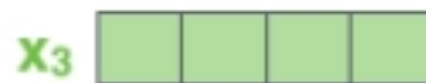
我们首先将每个输入单词通过词嵌入算法转换为词向量。



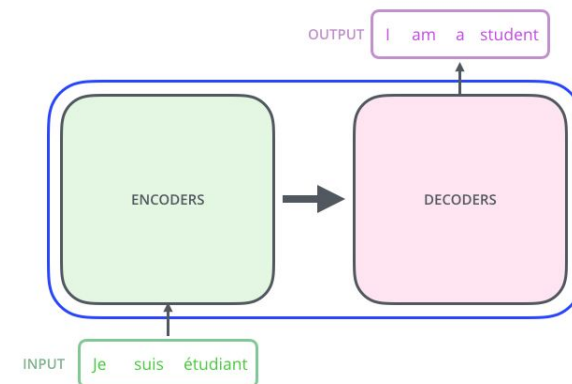
Je



suis

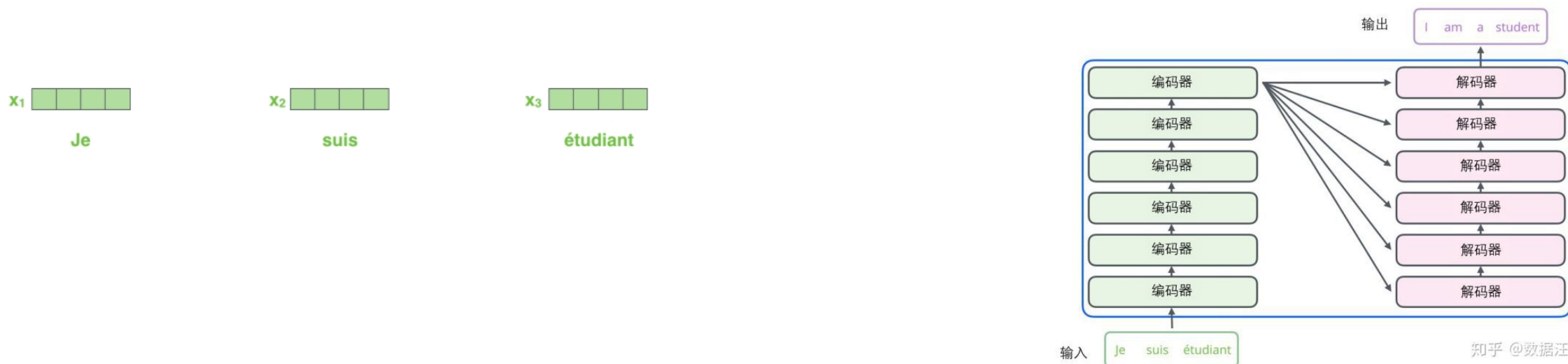


étudiant

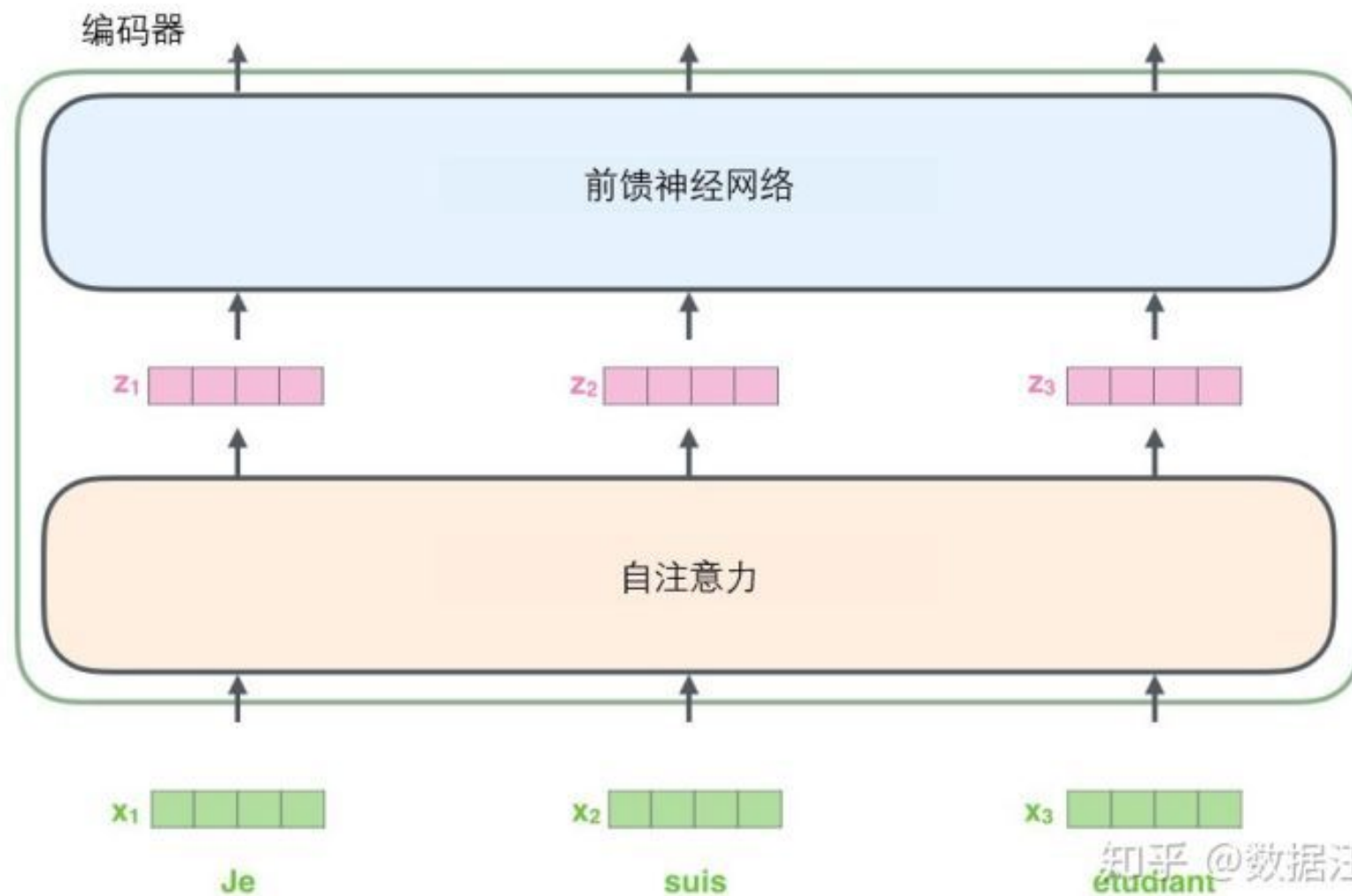


每个单词都被嵌入为**512**维的向量，我们用这些简单的方框来表示这些向量。

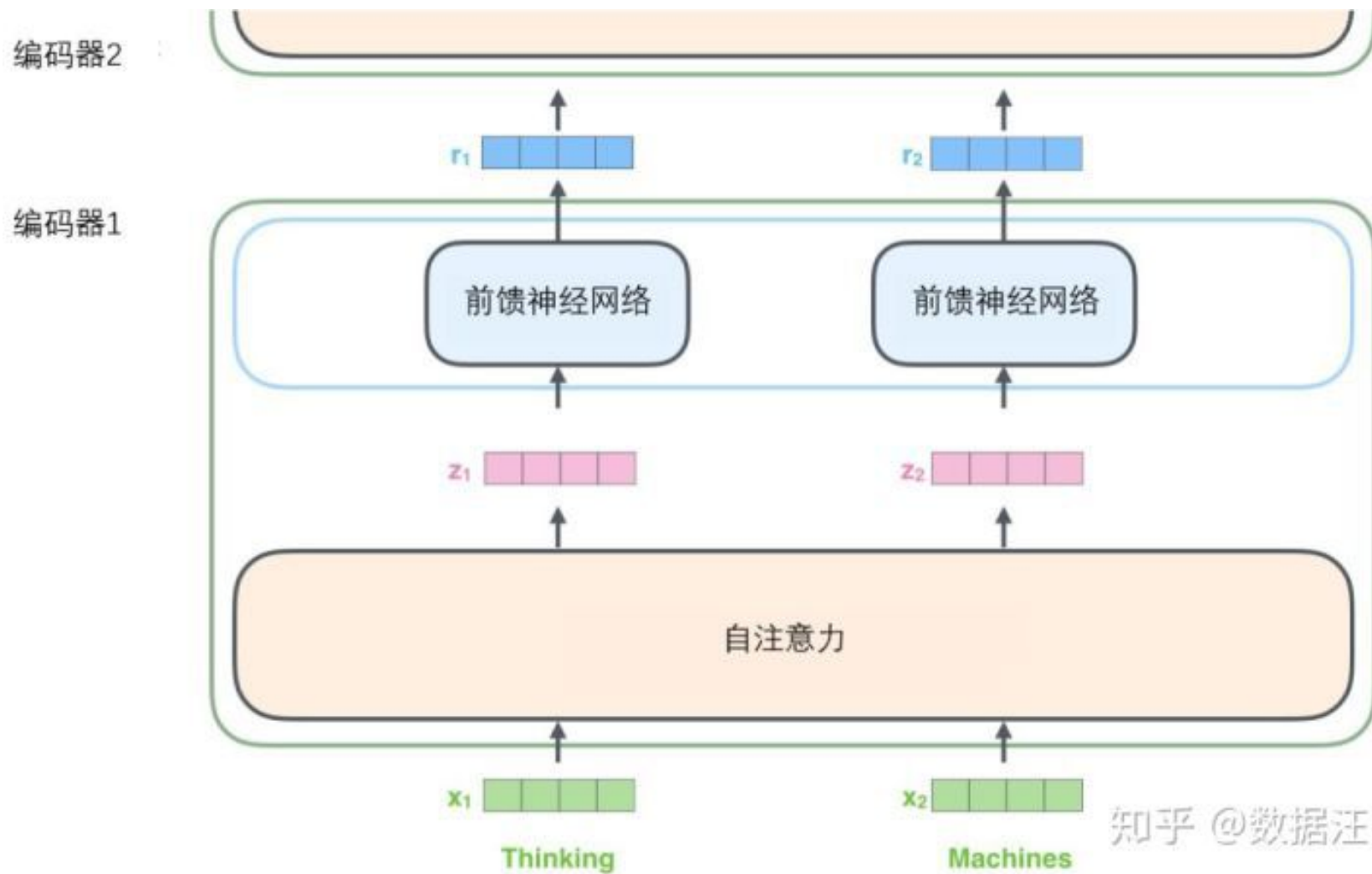
词嵌入过程只发生在最底层的编码器中。所有的编码器都有一个相同的特点，即它们接收一个向量列表，列表中的每个向量大小为**512**维。在底层（最开始）编码器中它就是词向量，但是在其他编码器中，它就是下一层编码器的输出（也是一个向量列表）。向量列表大小是我们可以设置的超参数——一般是我们训练集中最长句子的长度。



将输入序列进行词嵌入之后，每个单词都会流经编码器中的两个子层。



现在我们开始“编码”



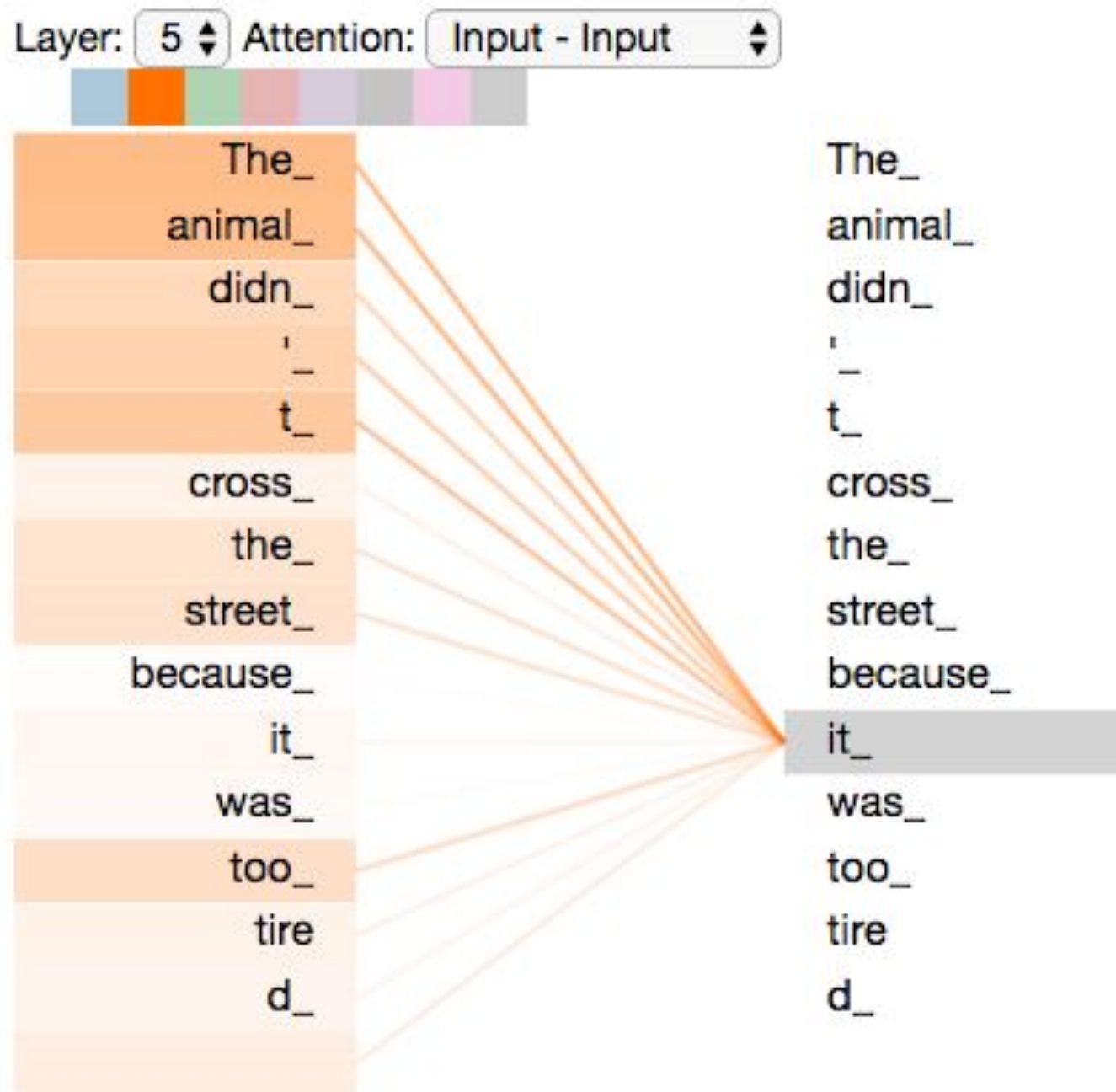
例如，下列句子是我们想要翻译的输入句子：

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

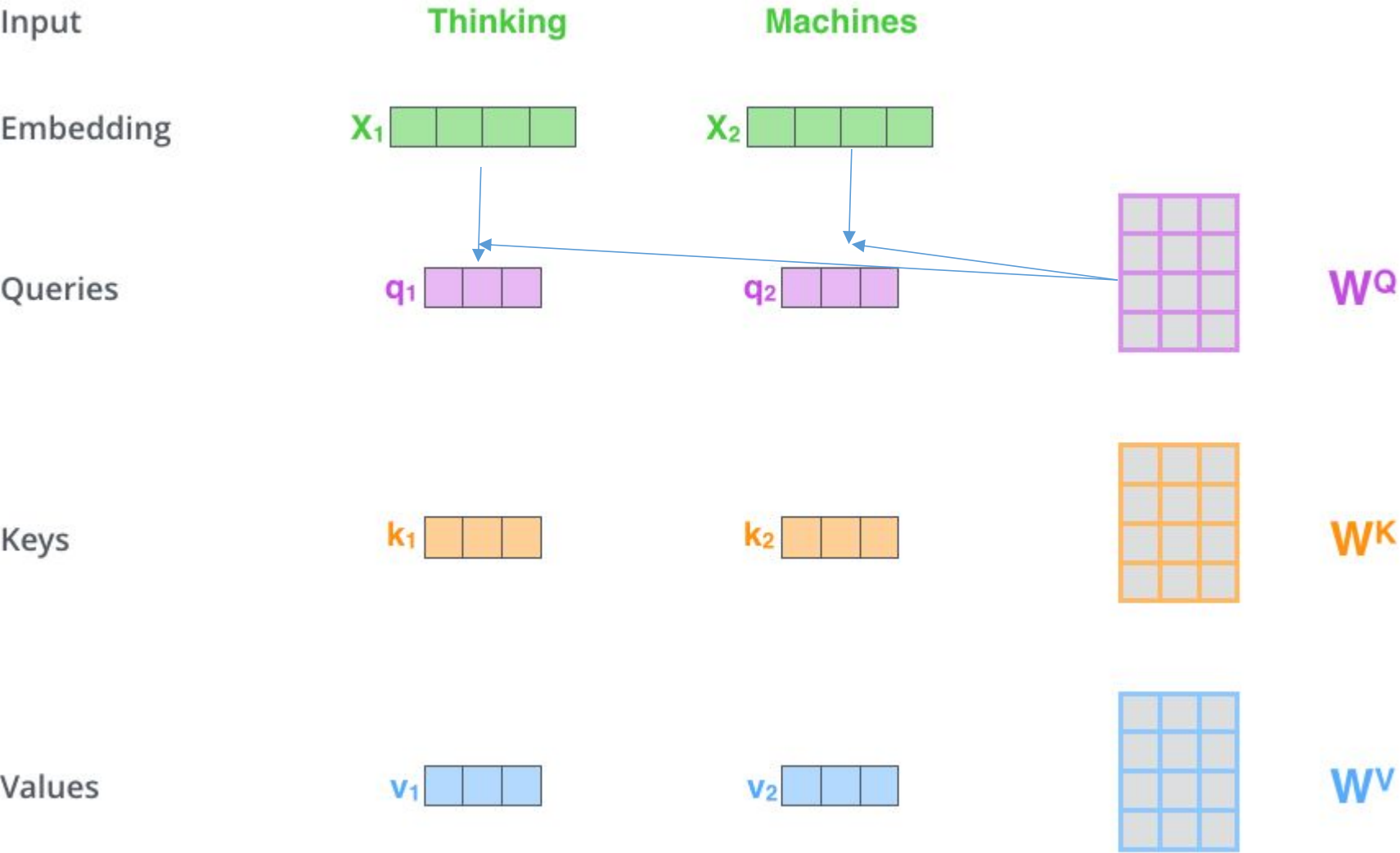
这个“it”在这个句子是指什么呢？它指的是**street**还是这个**animal**呢？这对于人类来说是一个简单的问题，但是对于算法则不是。

当模型处理这个单词“it”的时候，自注意力机制会允许“it”与“**animal**”建立联系。

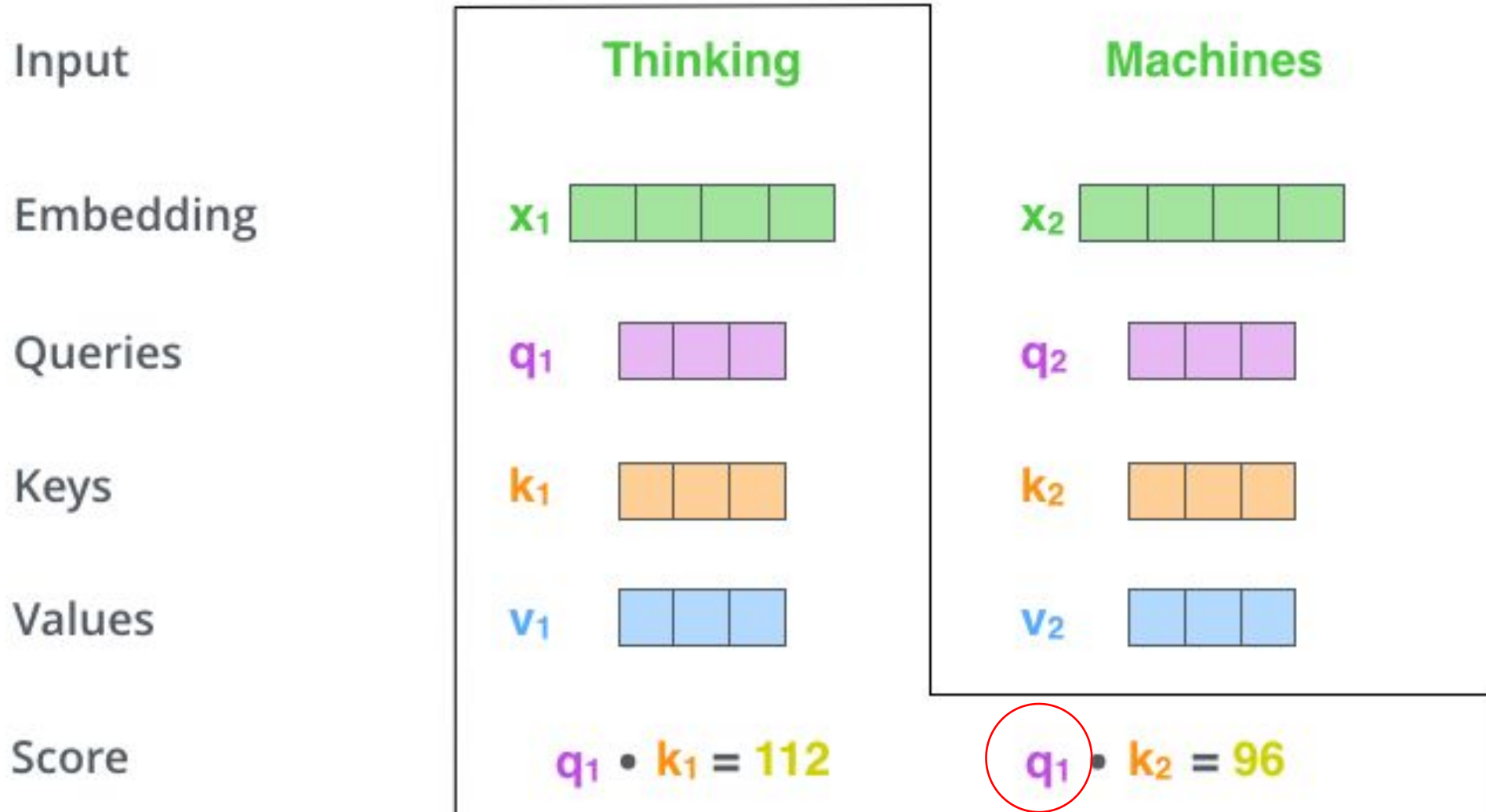
当我们在编码器中编码“*it*”这个单词的时候，注意力机制的部分会去关注“*The Animal*”，将它的表示的一部分编入“*it*”的编码中。



Step 1



Step 2



Step 3

Input

Embedding

Queries

Keys

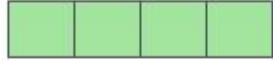
Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax

Thinking

x_1 

q_1 

k_1 

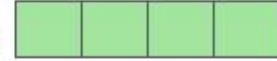
v_1 

$$q_1 \cdot k_1 = 112$$

14

0.88

Machines

x_2 

q_2 

k_2 

v_2 

$$q_1 \cdot k_2 = 96$$

12

0.12

Step 4

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum

Thinking

x_1

q_1

k_1

v_1

$q_1 \cdot k_1 = 112$

14

0.88

v_1

z_1

Machines

x_2

q_2

k_2

v_2

$q_2 \cdot k_2 = 96$

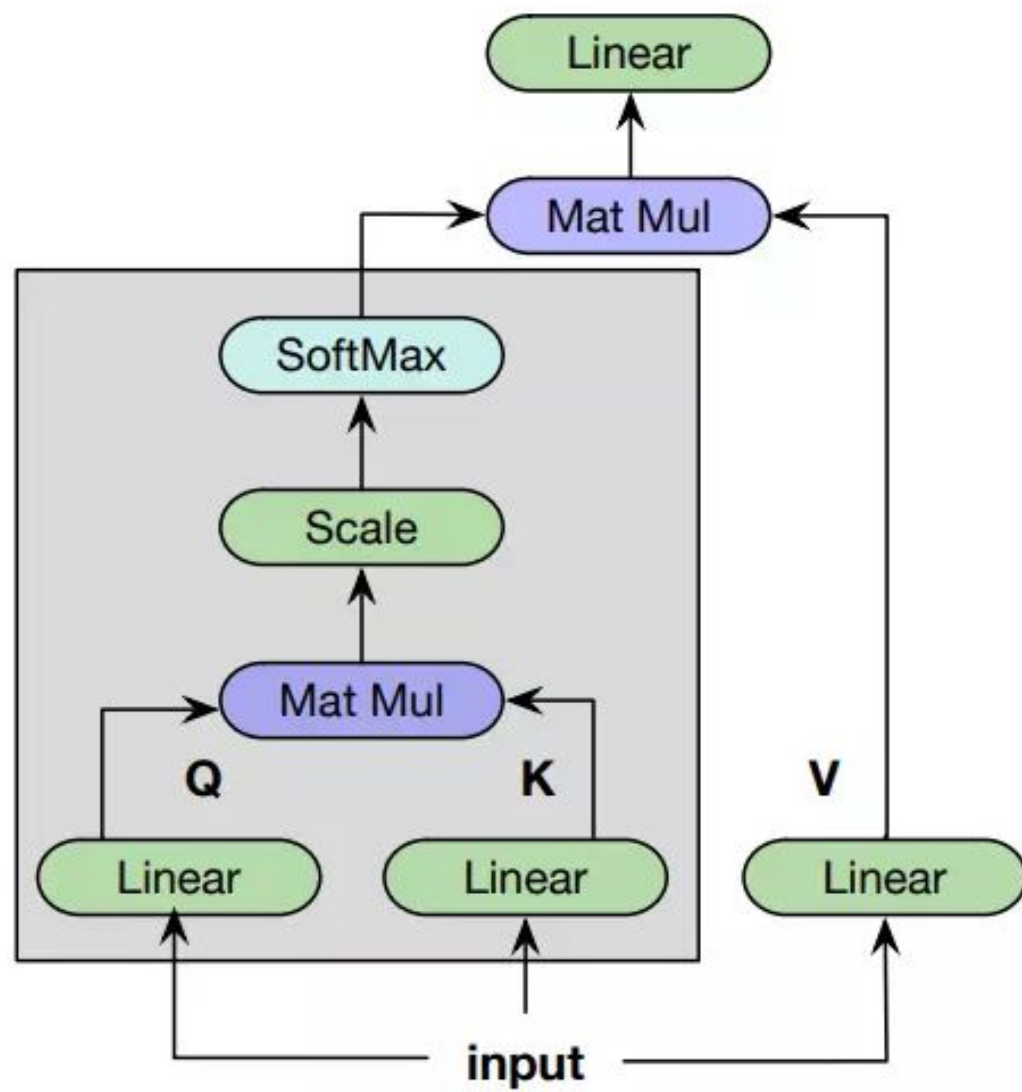
12

0.12

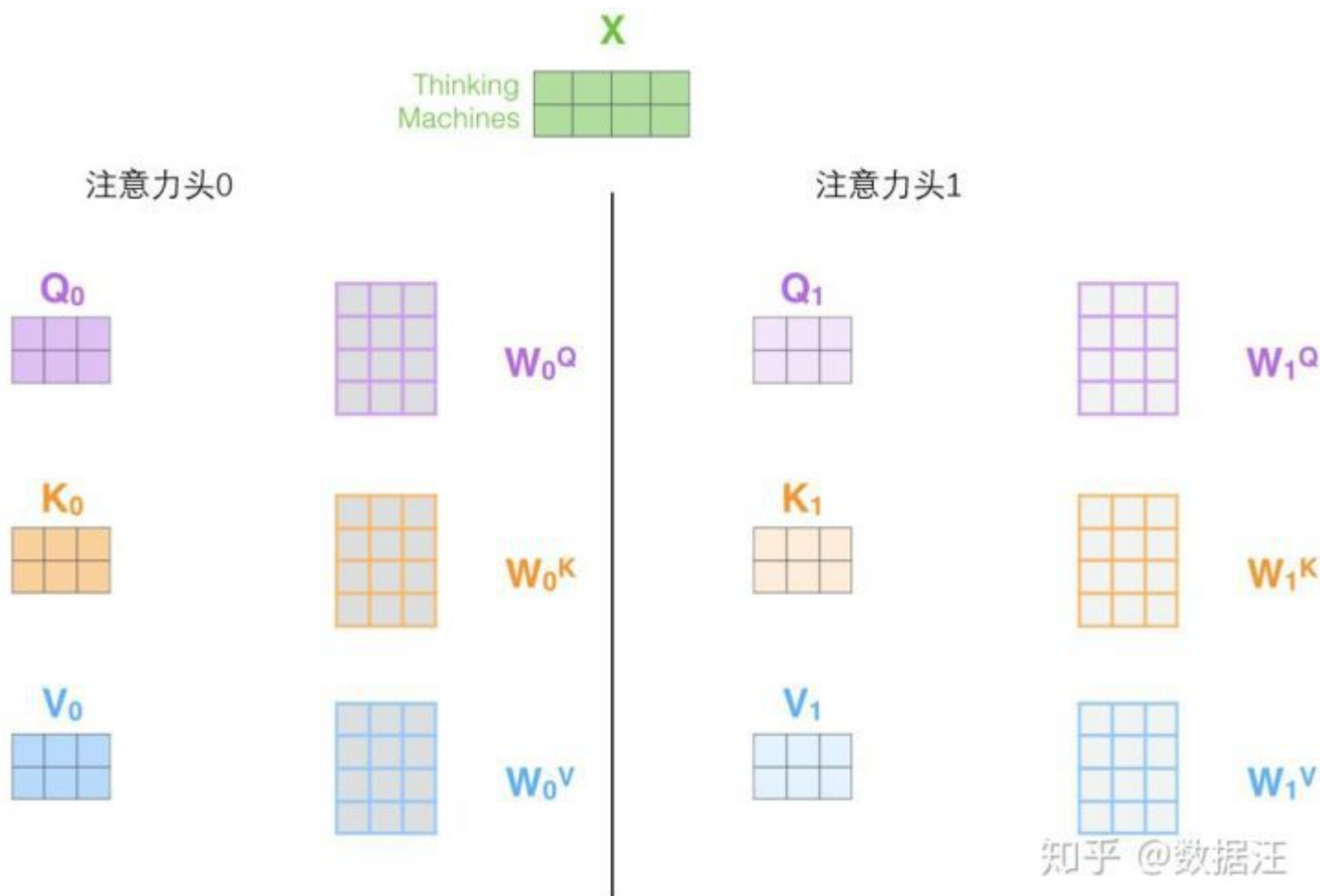
v_2

z_2

这个softmax分数决定了每个单词对编码当下位置
(“Thinking”)的贡献。显然，已经在这个位置上的单词将获得最高的softmax分数，但有时关注另一个与当前单词相关的单词也会有帮助。

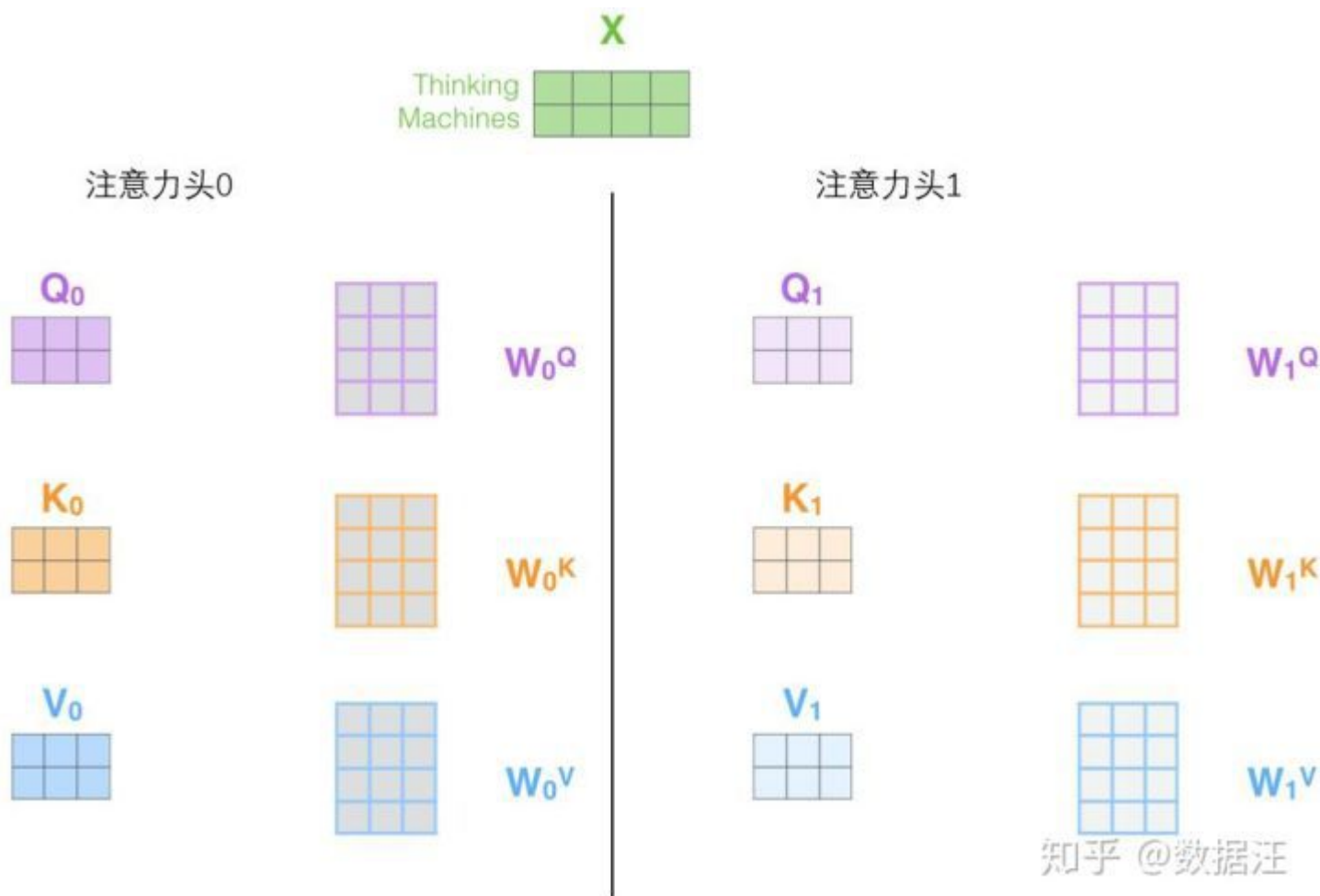


multi-head attention



它扩展了模型专注于不同位置的能力。在上面的例子中，虽然每个编码都在 z_1 中有或多或少的体现，但是它可能被实际的单词本身所支配。如果我们翻译一个句子，比如 “The animal didn’t cross the street because it was too tired”，我们会想知道 “it”指的是哪个词，这时模型的“多头”注意机制会起到作用。

multi-head attention

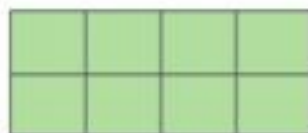


它给出了注意力层的多个“表示子空间”

(representation subspaces)。接下来我们将看到，对于“多头”注意机制，我们有多组查询/键/值权重矩阵集 (Transformer使用八个注意力头，因此我们对于每个编码器/解码器有八个矩阵集合)。这些集合中的每一个都是随机初始化的，在训练之后，每个集合都被用来将输入词嵌入(或来自较低编码器/解码器的向量)投影到不同的表示子空间中。

X

Thinking
Machines



↓
独立计算这8个注意力头

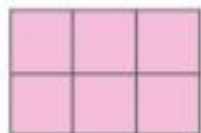
注意力头0

注意力头1

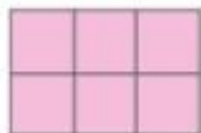
...

注意力头7

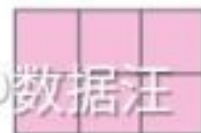
Z₀



Z₁



Z₇



知乎 @数据汪

1) 将所有注意力头拼接起来



2) 乘以矩阵 W^O , 它在模型中是联合训练的

x

3) 结果是一个融合所有注意力头信息的矩阵 Z , 我们可以将其送到前馈神经网络



知乎 @数据汪

1) 这是我们的输入句子*

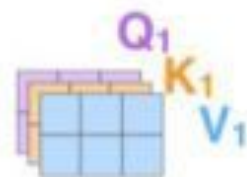
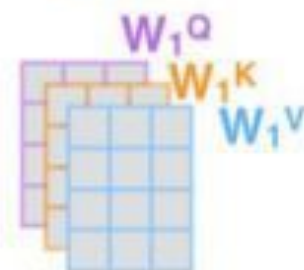
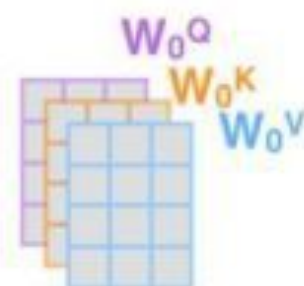
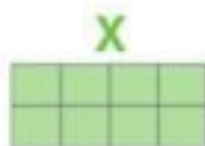
2) 编码每一个单词

3) 将其分为8个头，将矩阵X或R乘以各个权重矩阵

4) 通过输出的查询/键/值 (Q/K/V) 矩阵计算注意力

5) 将所有注意力头拼接起来，乘以权重矩阵 W^O

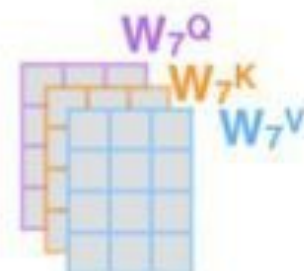
Thinking
Machines



...

...

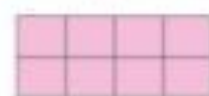
...



W^O



Z

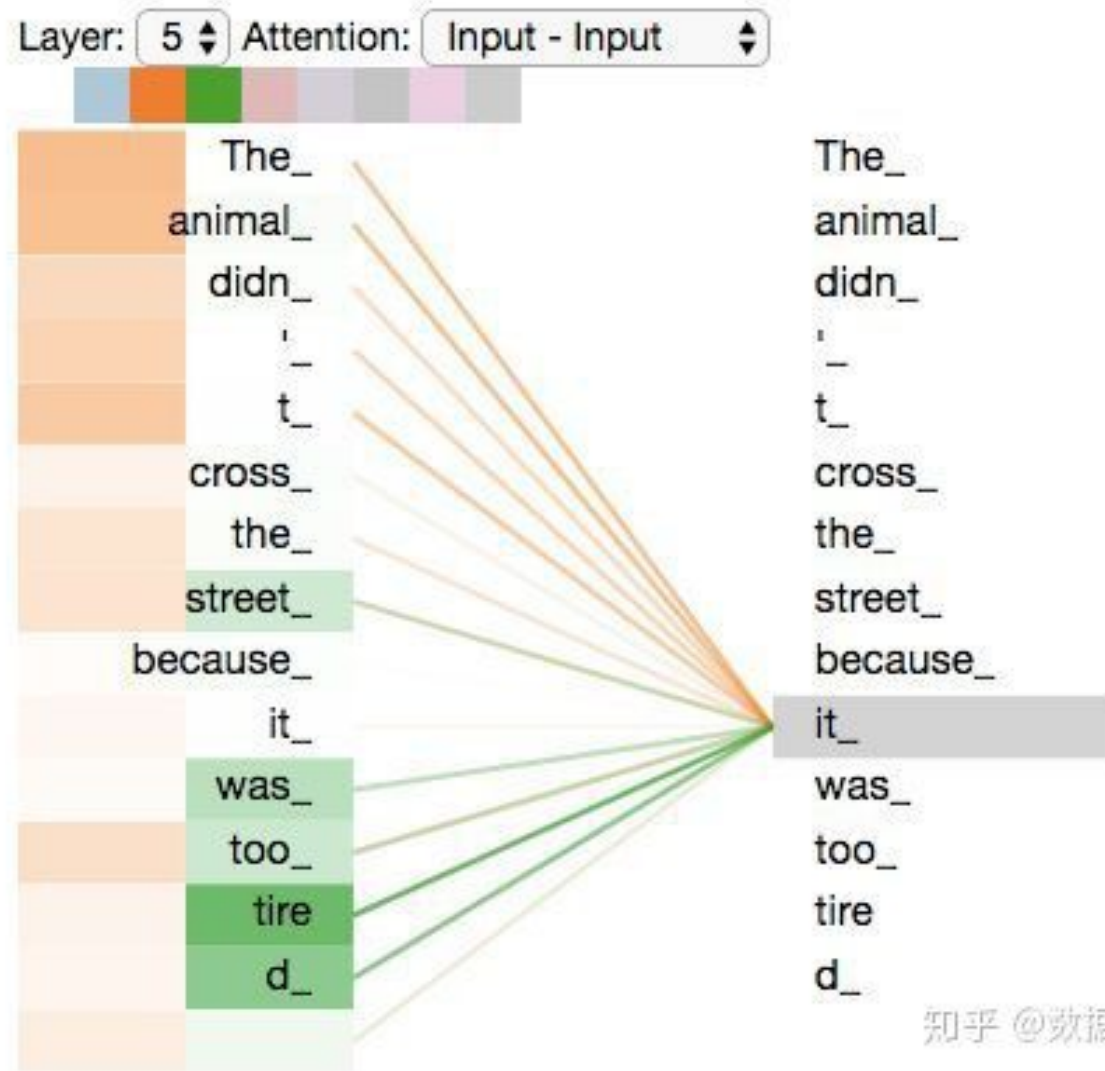


*注：除了第0个编码器，其他编码器都不需要进行词嵌入。它可以直接讲前面一层编码器的输出作为输入（矩阵R）。

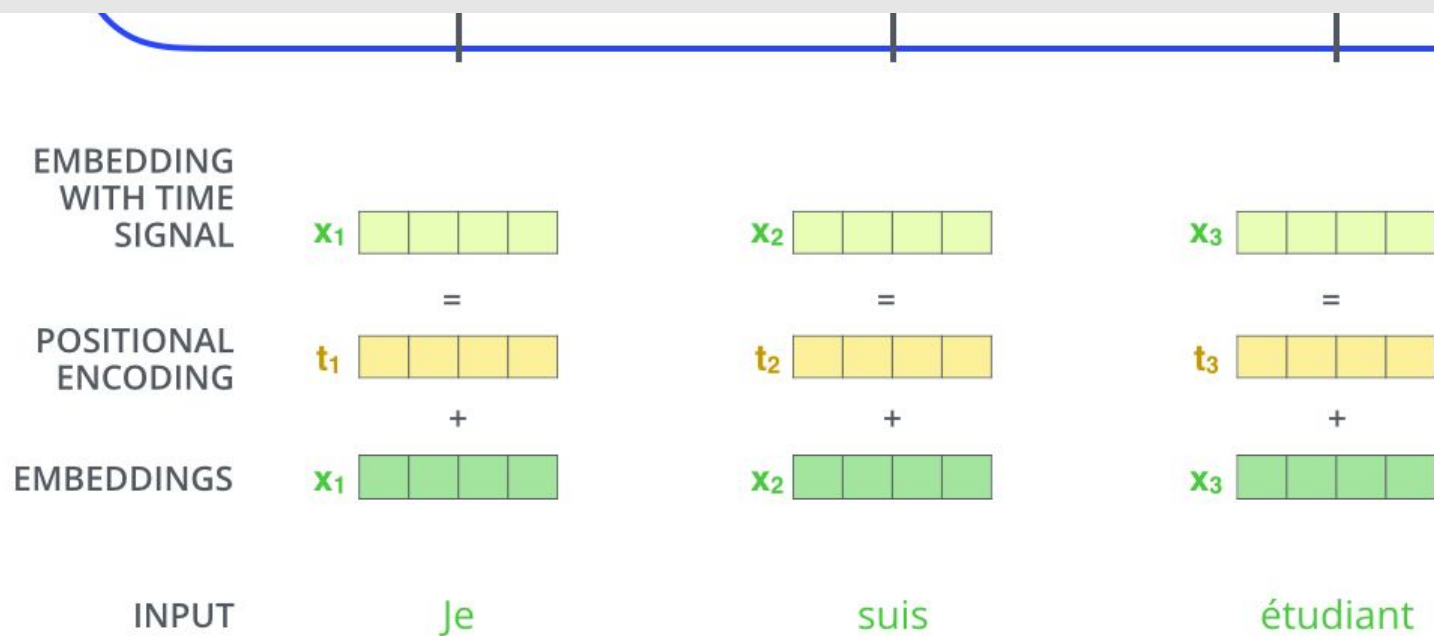


知乎 @数据汪

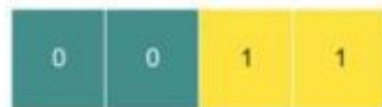
当我们编码 “*it*”一词时，一个注意力头集中在 “*animal*”上，而另一个则集中在 “*tired*”上，从某种意义上说，模型对 “*it*”一词的表达在某种程度上是 “*animal*”和 “*tired*”的代表。



Transformer为每个输入的词嵌入添加了一个向量。这些向量遵循模型学习到的特定模式，这有助于确定每个单词的位置，或序列中不同单词之间的距离。这里的直觉是，将位置向量添加到词嵌入中使得它们在接下来的运算中，能够更好地表达的词与词之间的距离。

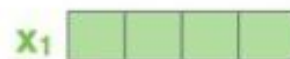


位置编码



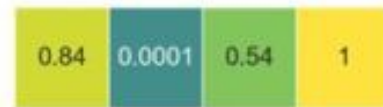
+

词嵌入

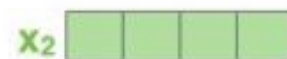


输入

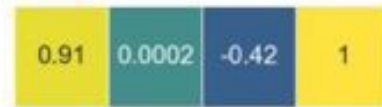
Je



+



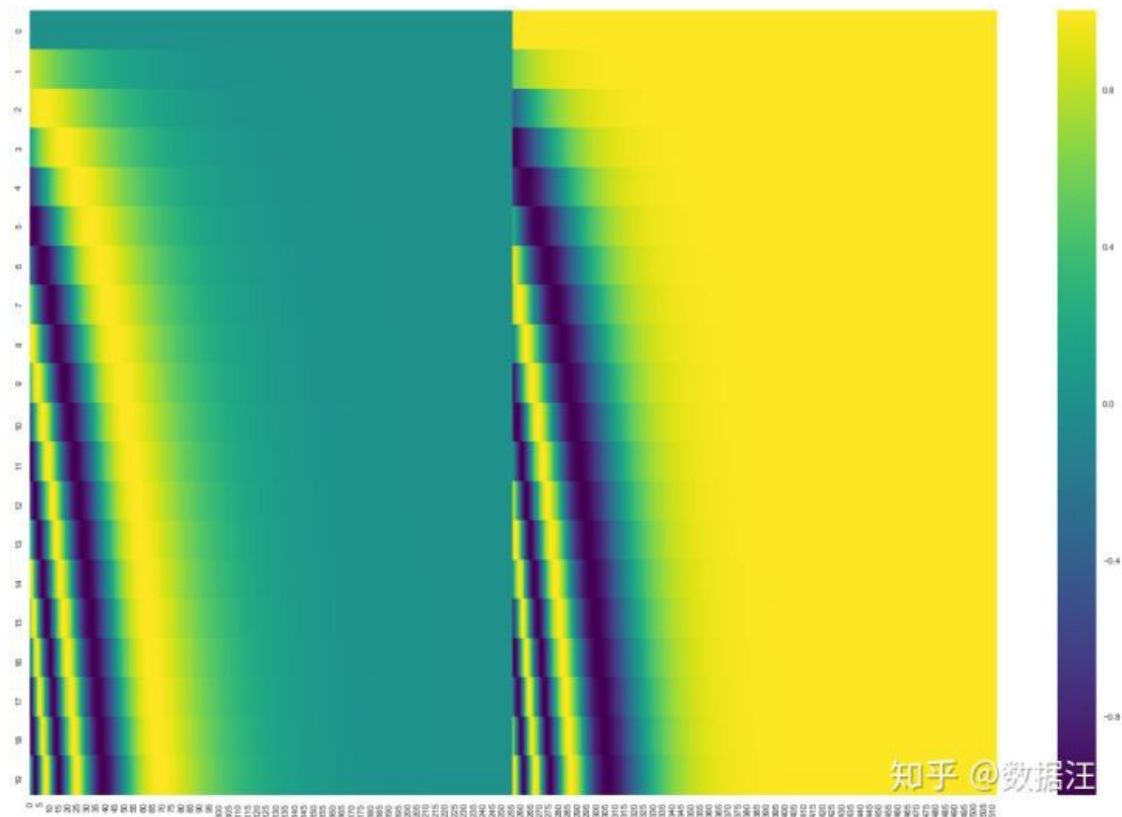
suis



+

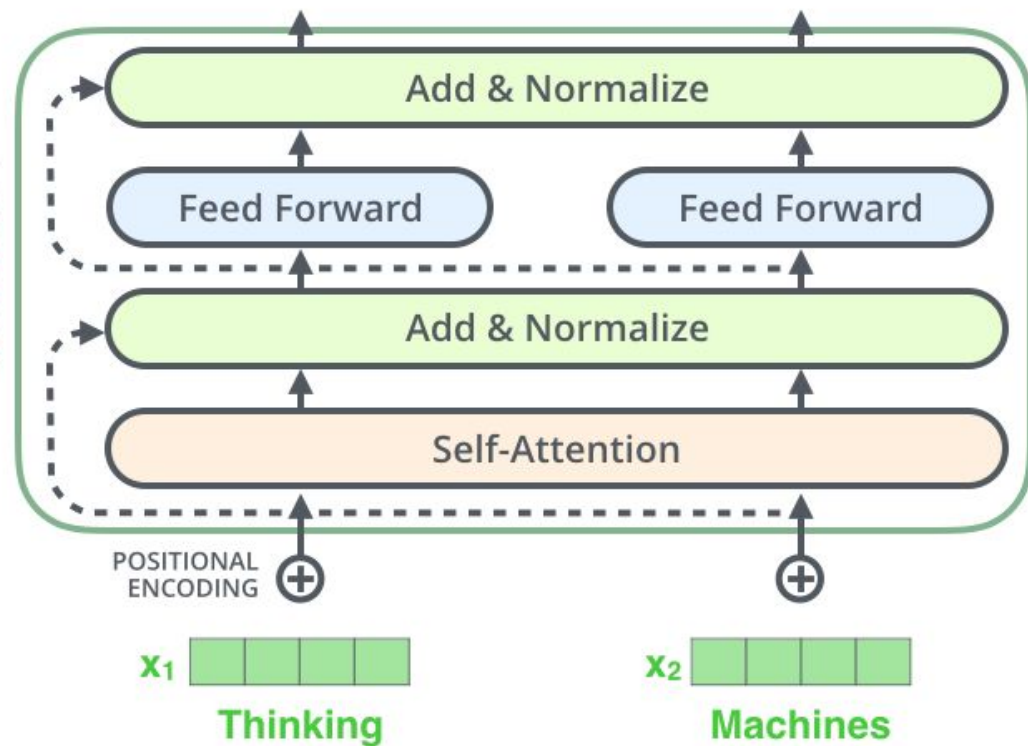


知乎@数据汪

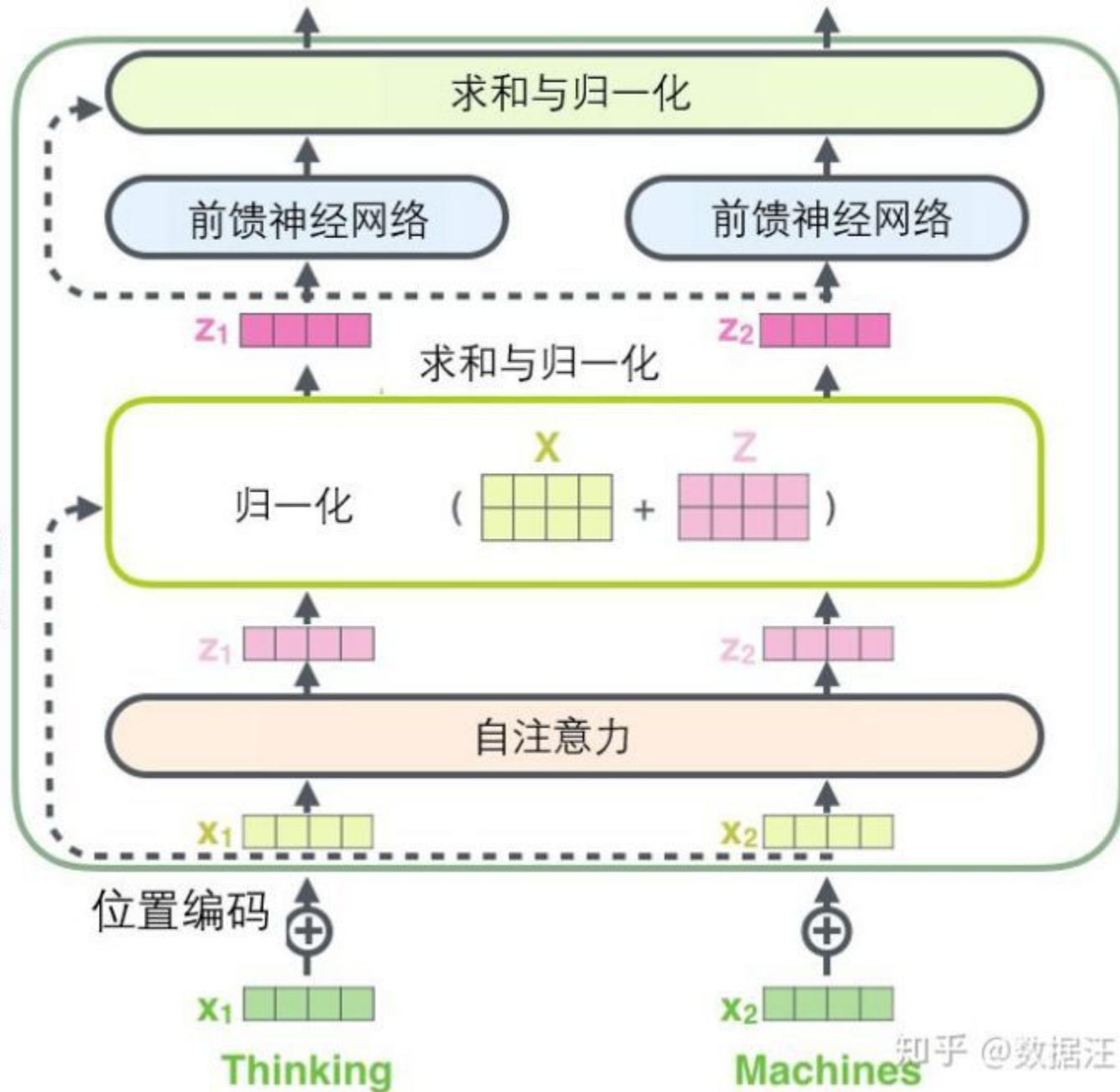


残差模块

ENCODER #1

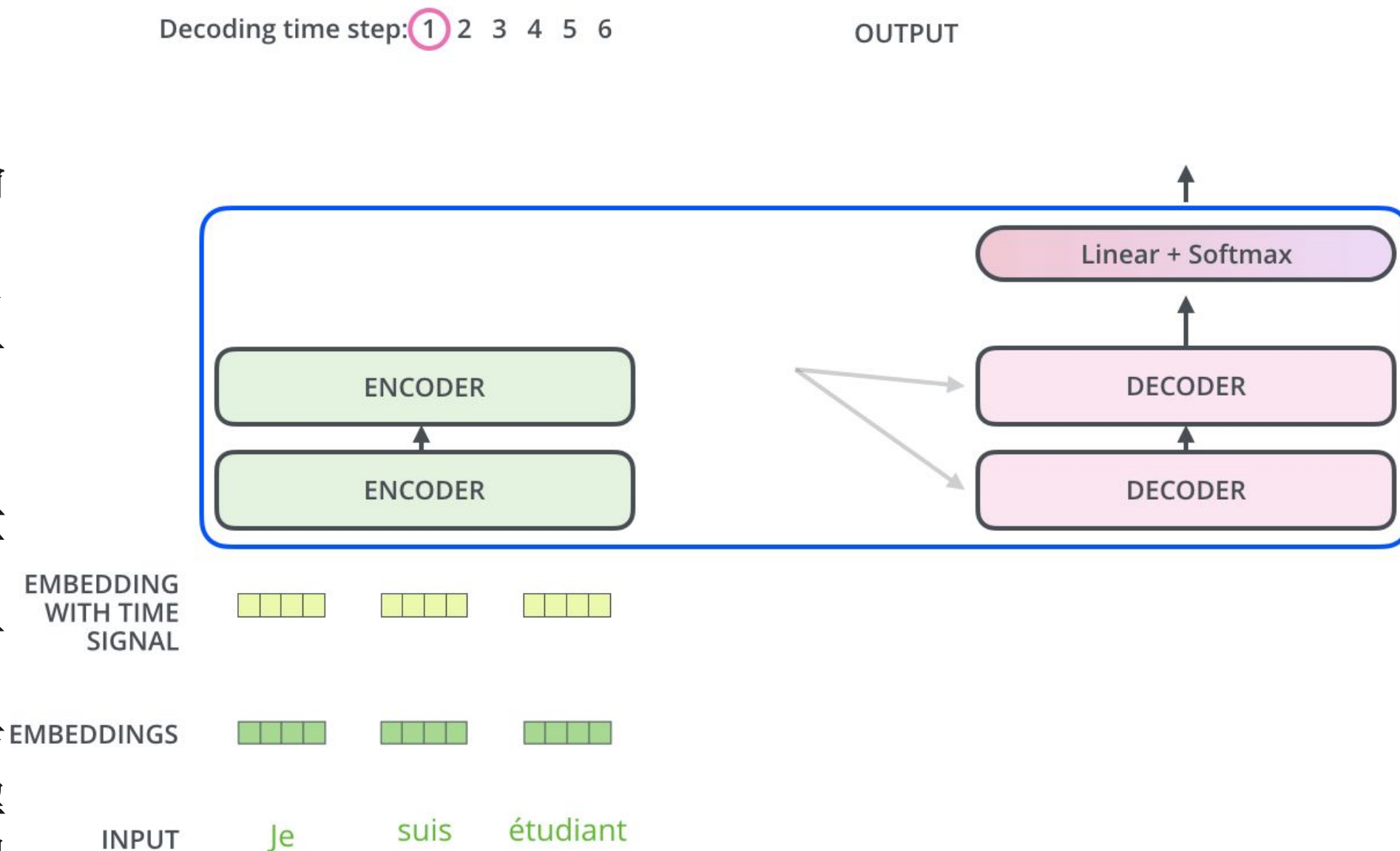


编码器1



decoder

编码器通过处理输入序列开启工作。顶端编码器的输出之后会变转化为一个包含向量 K （键向量）和 V （值向量）的注意力向量集。这些向量将被每个解码器用于自身的“编码-解码注意力层”，而这些层可以帮助解码器关注输入序列哪些位置合适：



在完成编码阶段后，则开始解码阶段。解码阶段的每个步骤都会输出一个输出序列（在这个例子中，是英语翻译的句子）的元素

