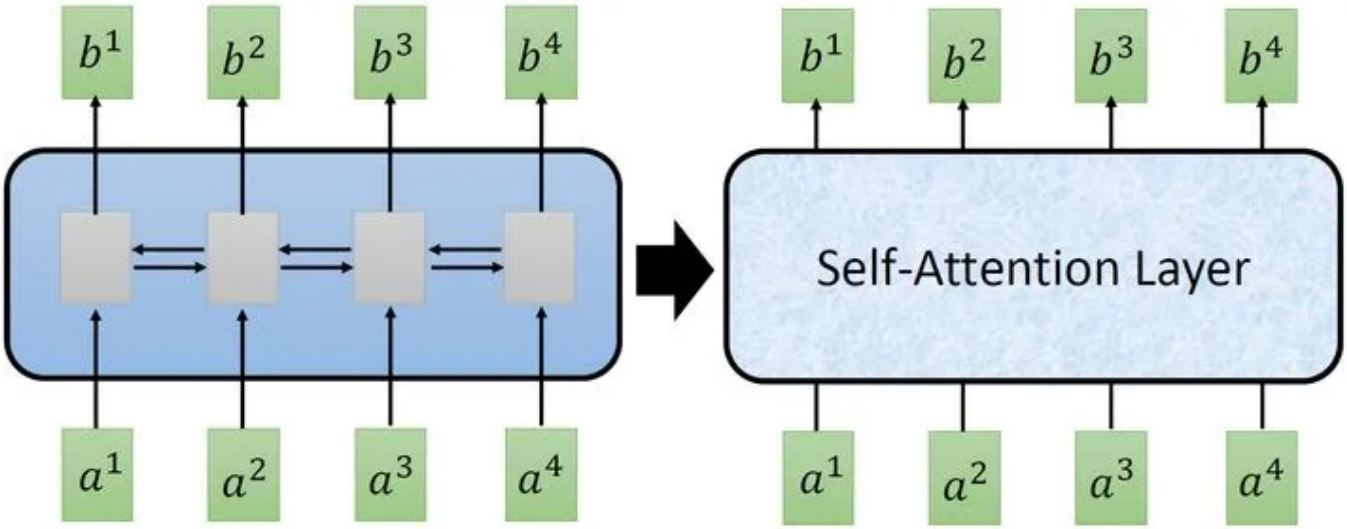
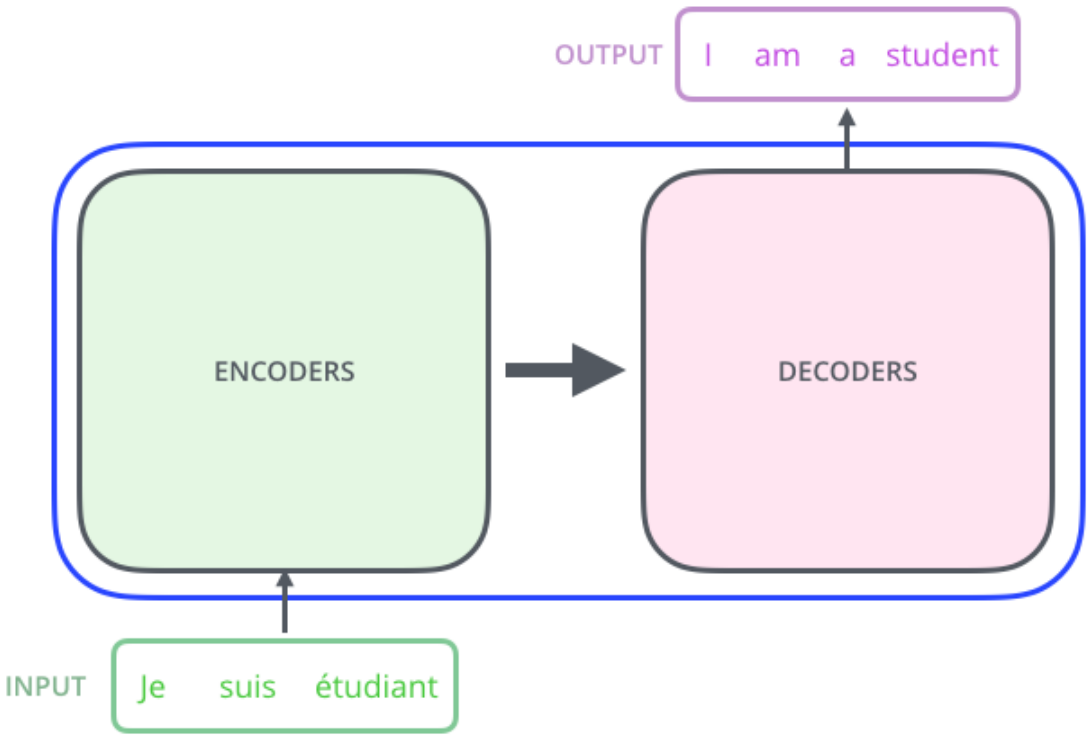
Attention is All You Need

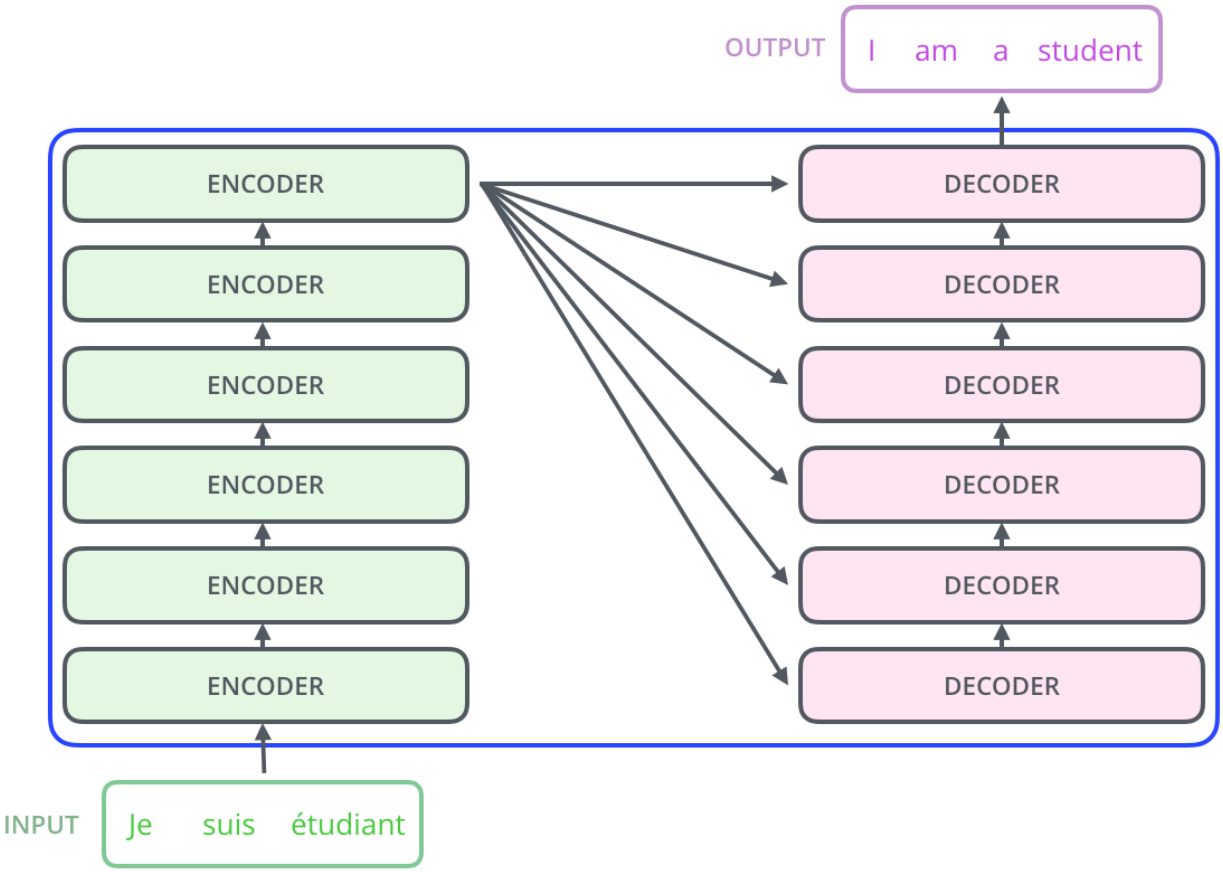
论文：https://arxiv.org/abs/1706.03762 仓库：Official TensorFlow，PyTorch

A High-Level Look

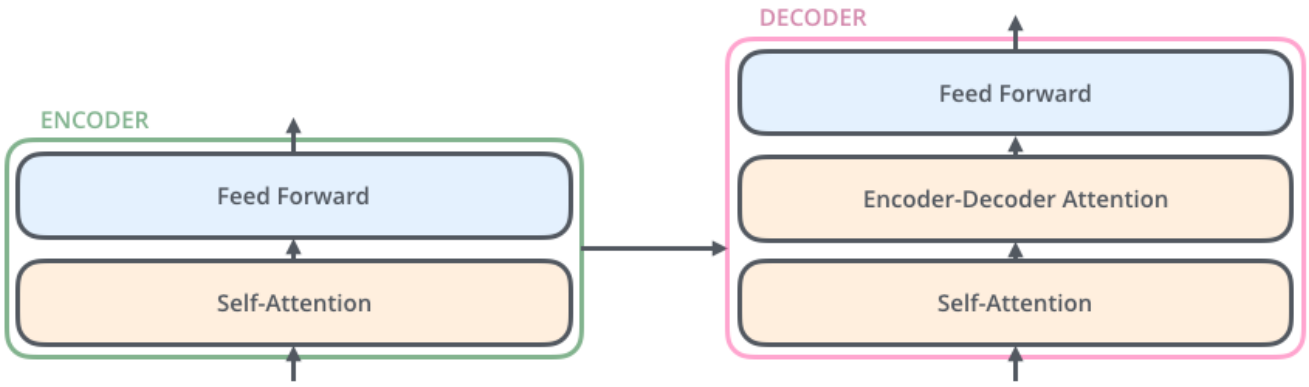


一种新的layer，叫self-attention，它的输入和输出和RNN是一模一样的，输入一个sequence，输出一个sequence，它的每一个输出都看过了整个的输入sequence，这一点与bi-directional RNN相同。但是它的每一个输出都可以并行化计算。





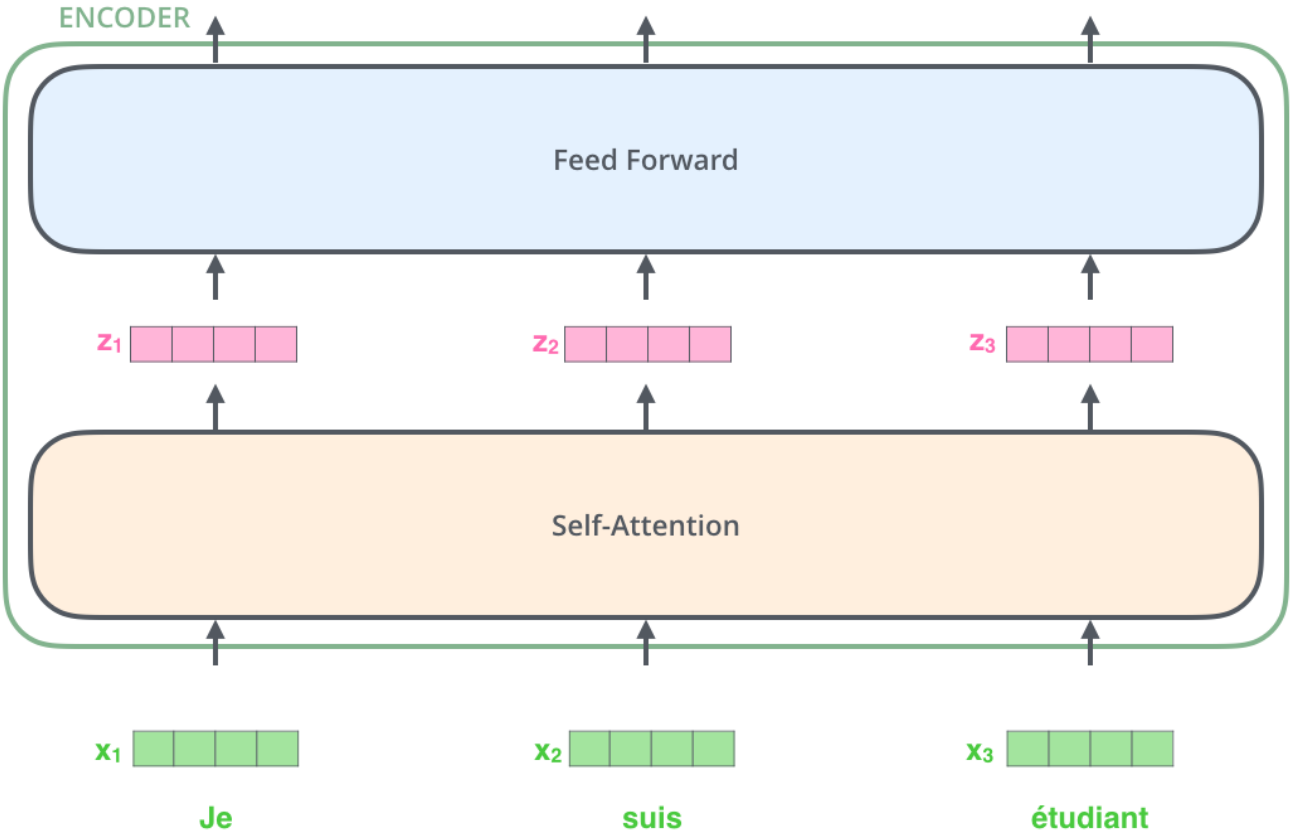
文章中采用6个Encoder/Decoder块堆叠的方式。

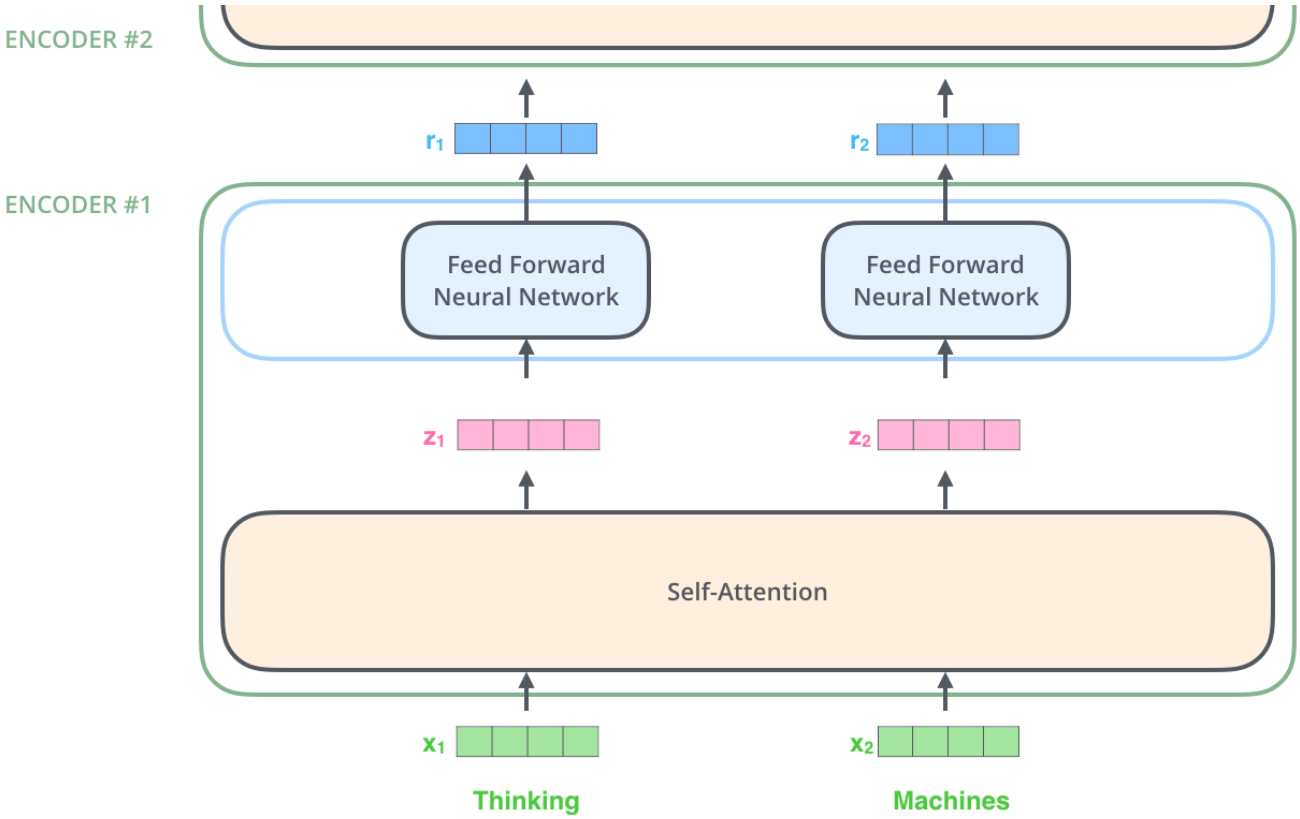


一个单词embedding之后传入Self-Attention层，来计算该词与其他词之间的相关性，然后进入前向传播模块。解码器中多了一个Attention模块，使decoder可以更加专注于输入句子中与当前输入的最相关部分。

Attention中的张量流

将输入的单词用embedding algorithm转化成512维向量。

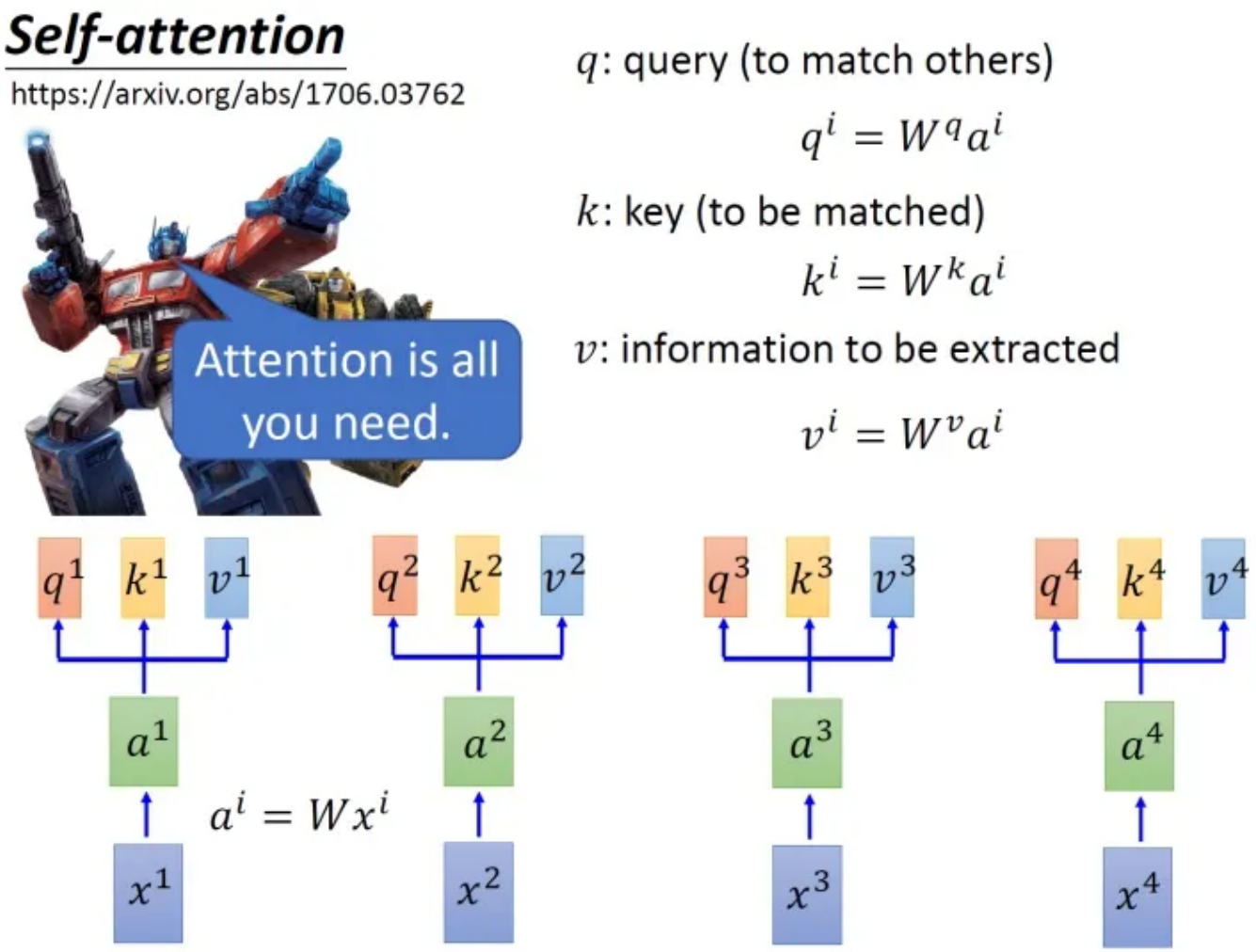




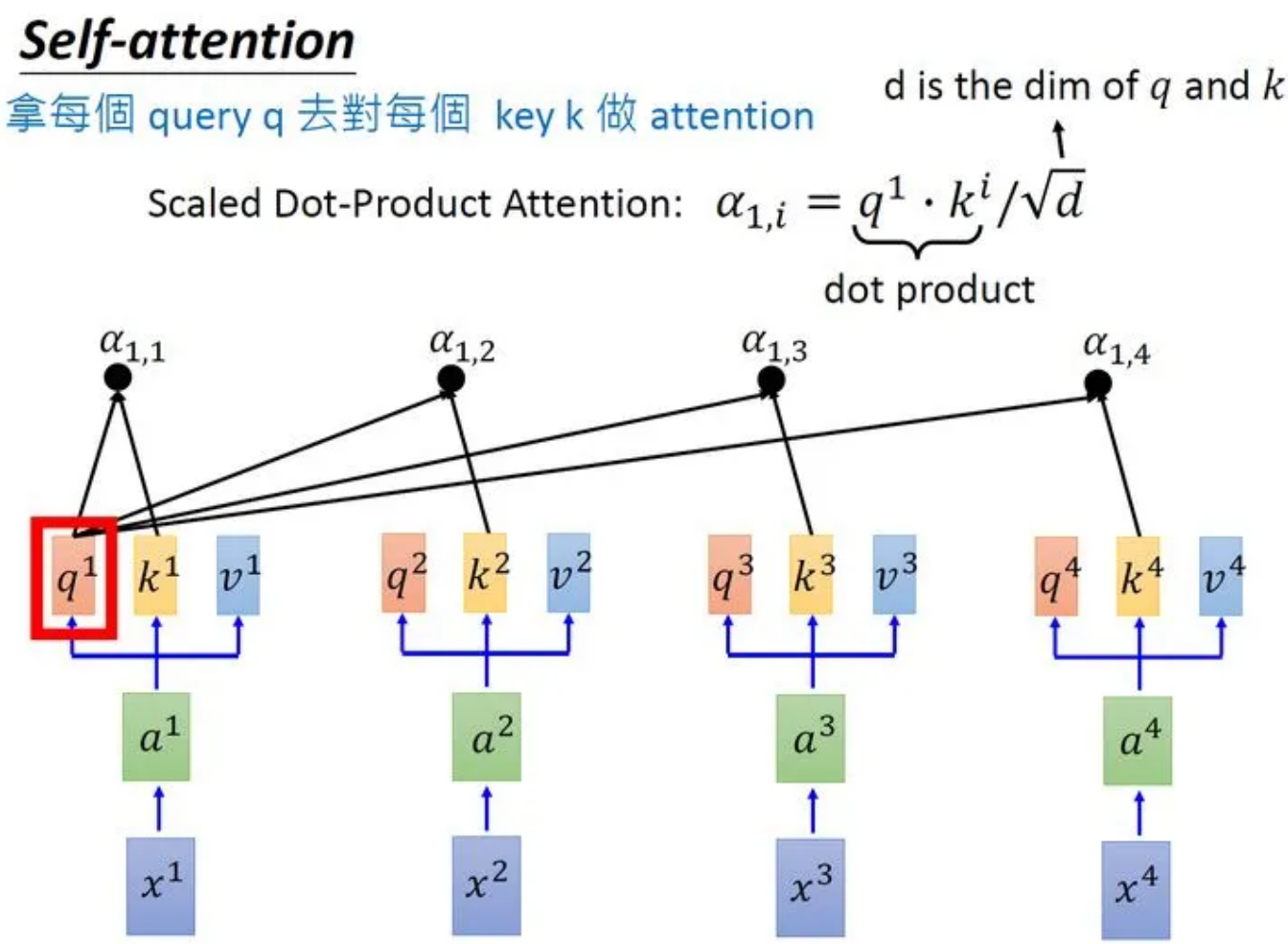
完整encoder流程如上图，其中Self-Attention层会计算输入之间的相关性，而前向传播中则不计算。Attention可视化，可以看到不同层的attention模块关注的句子位置。

Self-Attention机制与Q，K，V的理解

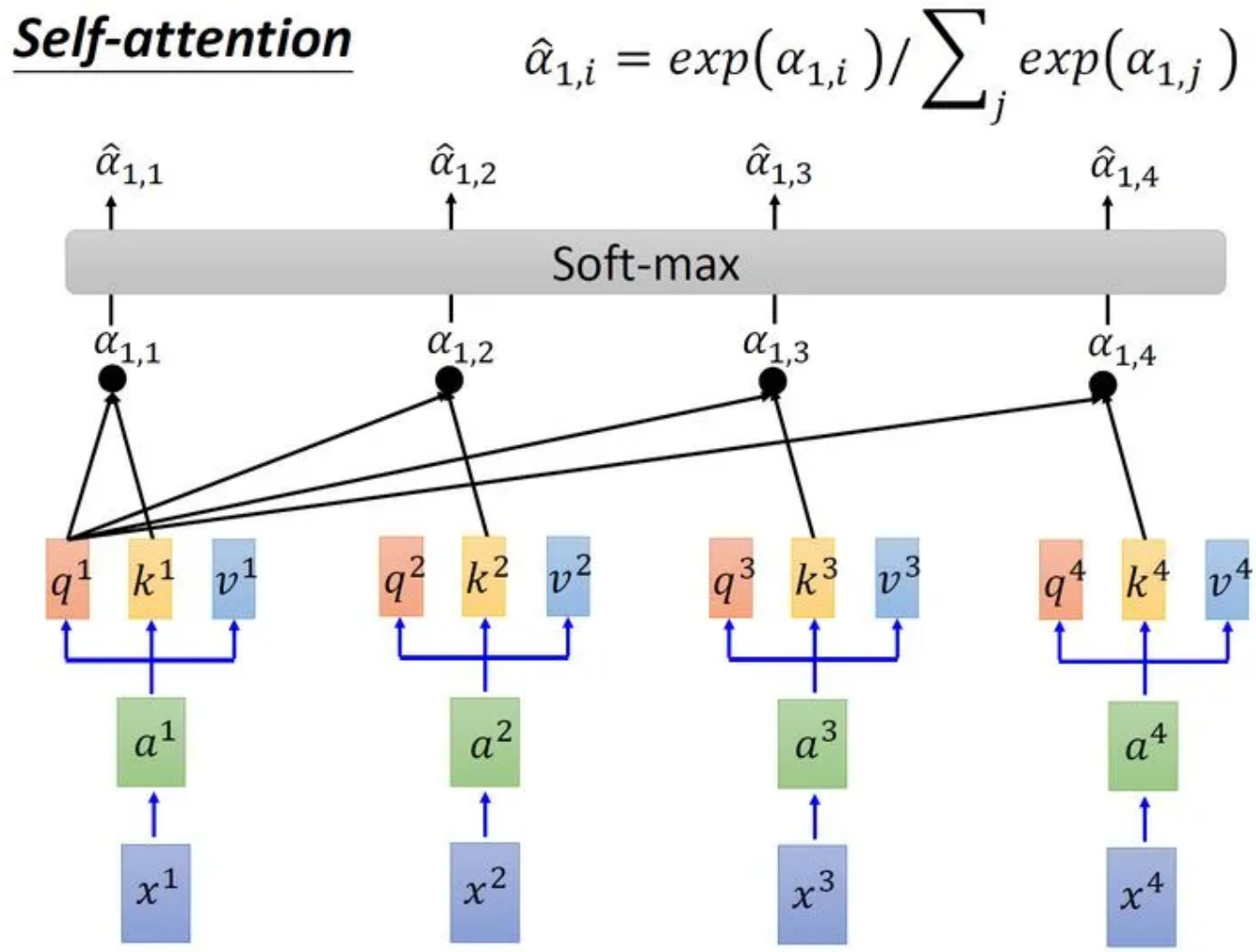
High Level



其中Q代表Query，“代表了”当前词汇，用于和其他单词做匹配。K代表Key，用于被其他单词匹配。在对当前单词做运算时，计算出的Q和其他单词的K做运算，得到相关性。在对其他单词运算时，当前单词计算出的K用于和其他单词的Q做运算，得到相关性。即Q是"to match others“，K是"to be matched“。V代表了当前单词提取出的信息，用于后期加权。 上图中x^i是一个sequence input vector，embedding得到a^i，乘上三个不同的转移矩阵W_q，W_k，W_v得到q^i，k^i，v^i。



接下来使用每个query去对每个key做attention，attention就是匹配这2个向量有多接近。将当前单词的q与其他单词的k做内积，再除以根号q和k的维度（q和k维度相同），得到αᵢ。因为q·k的数值会随着dimension的增大而增大，所以要除以$\sqrt{dimension}$，相当于归一化的效果。

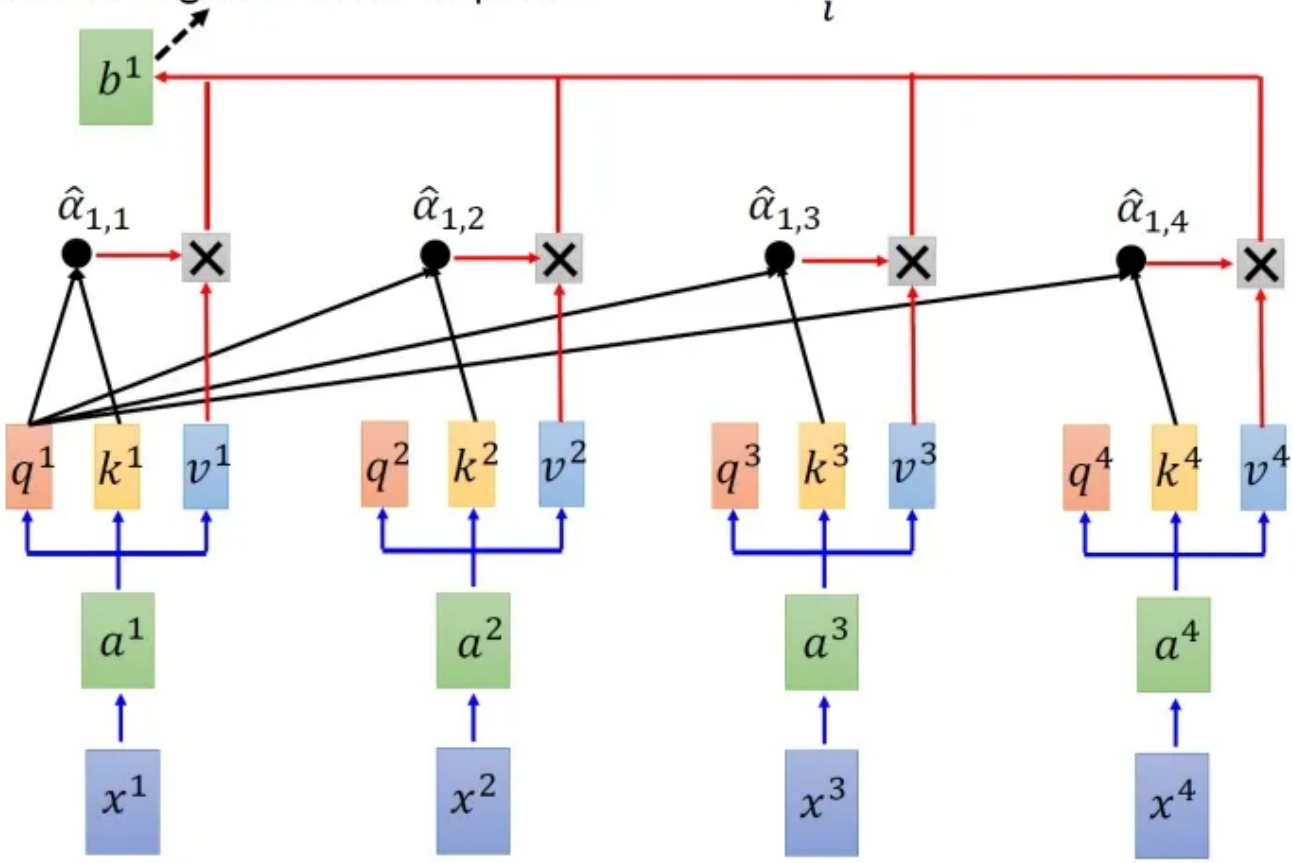


将αᵢ取Softmax操作后，得到$\hat{\alpha}$ᵢ，之后和所有v^{i}相乘，结果得到b^i，即产生b^{i}的过程用到了输入的全部信息，并且可以并行计算。如果要考虑local information，则只需要学习出相应的$\hat{\alpha}$ᵢ=0即可。考虑global information，则只需要学习出相应的$\hat{\alpha}$ᵢ不为0即可。

Self-attention

Considering the whole sequence

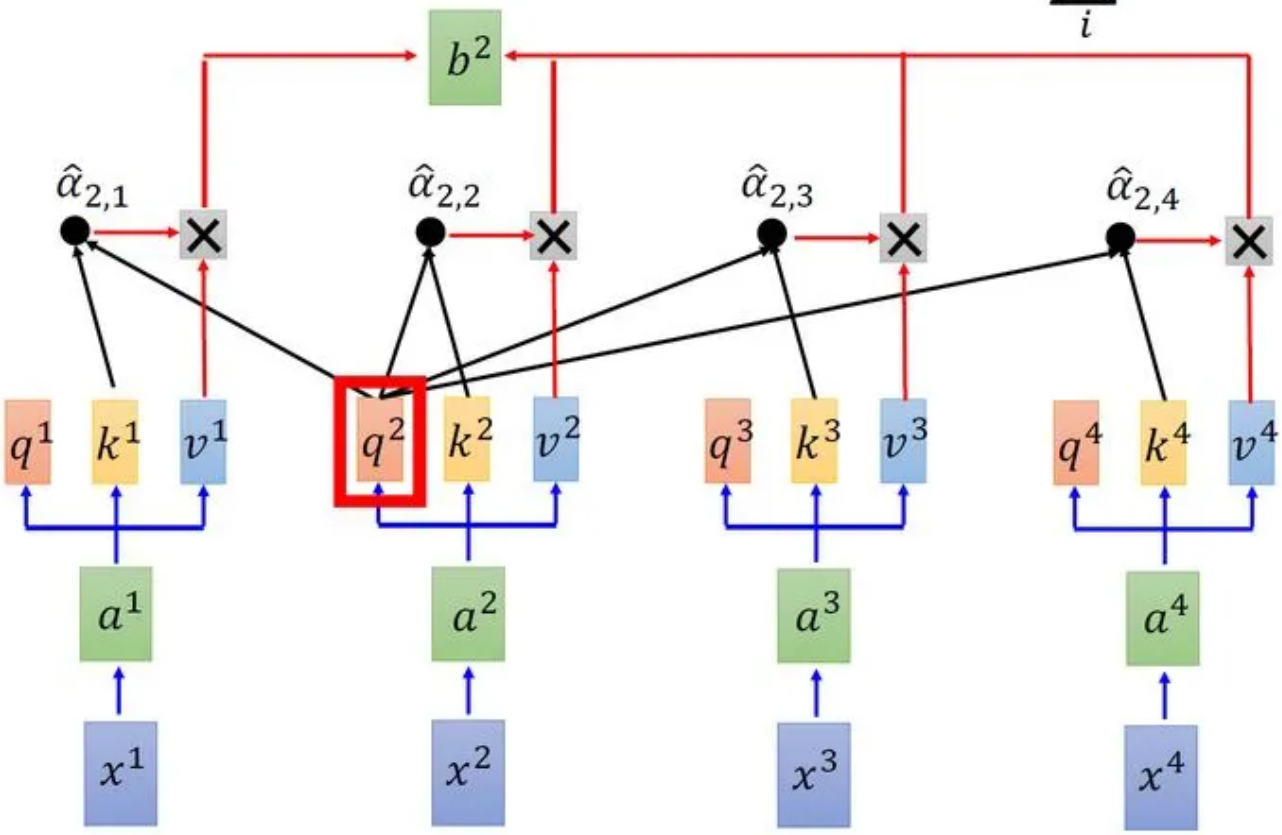
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



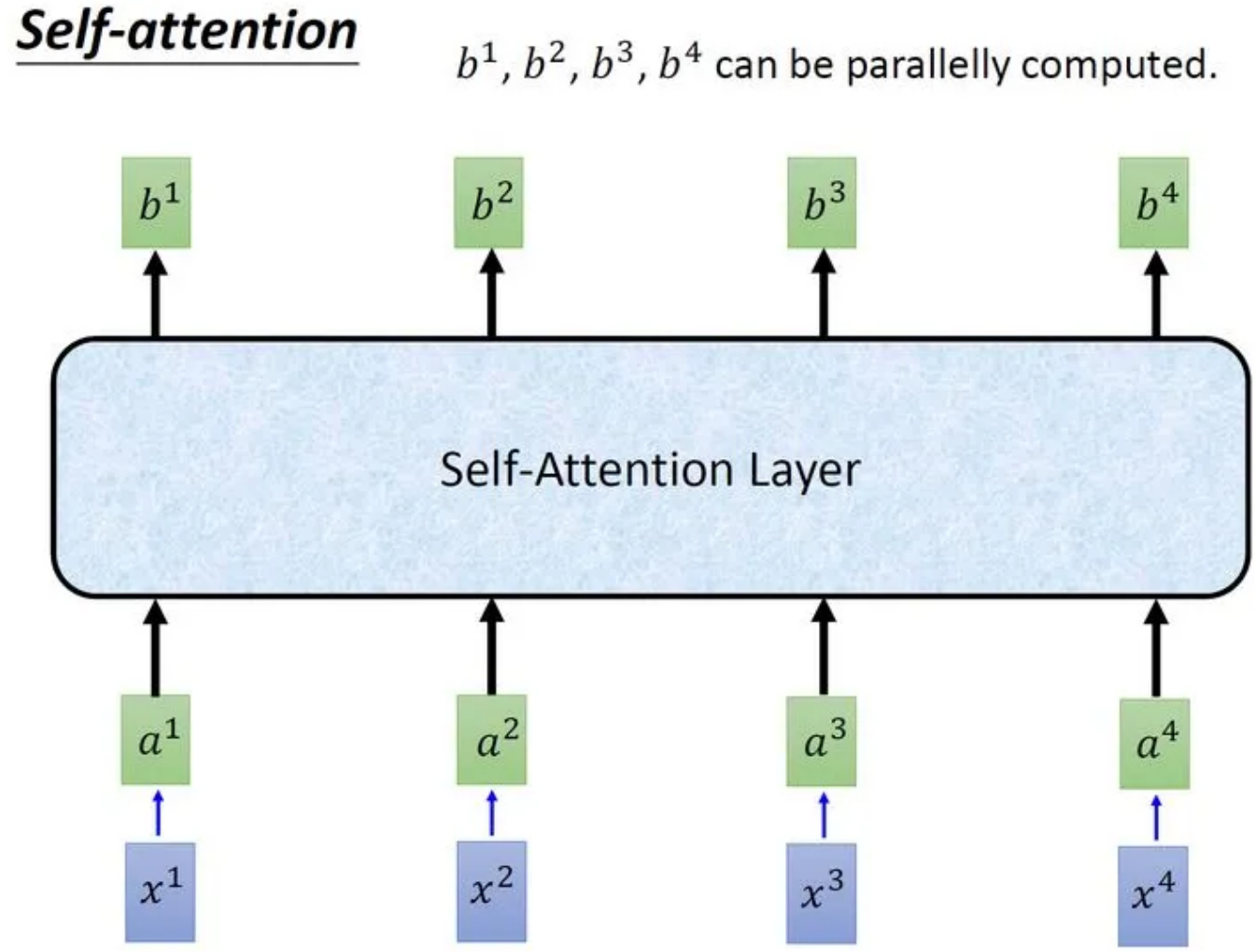
Self-attention

拿每個 query q 去對每個 key k 做 attention

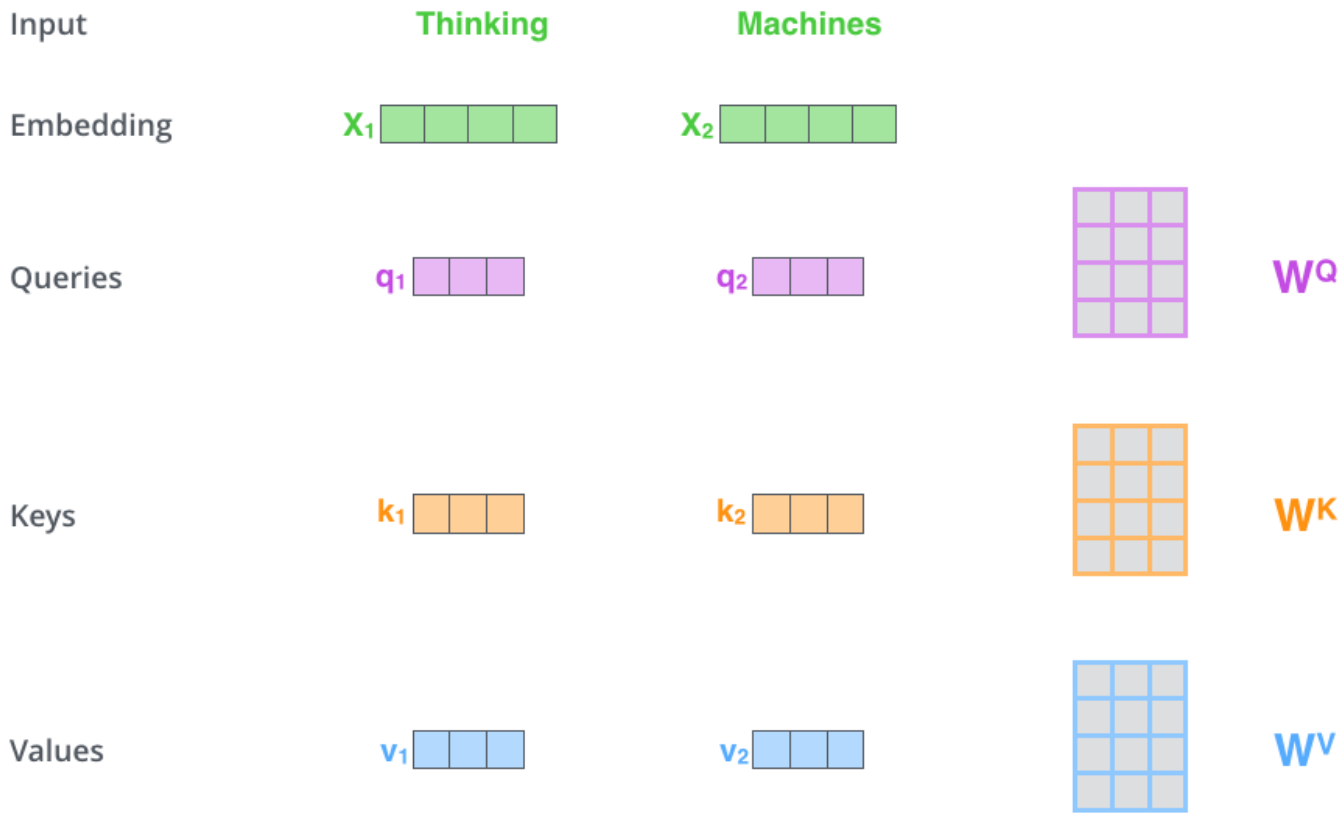
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



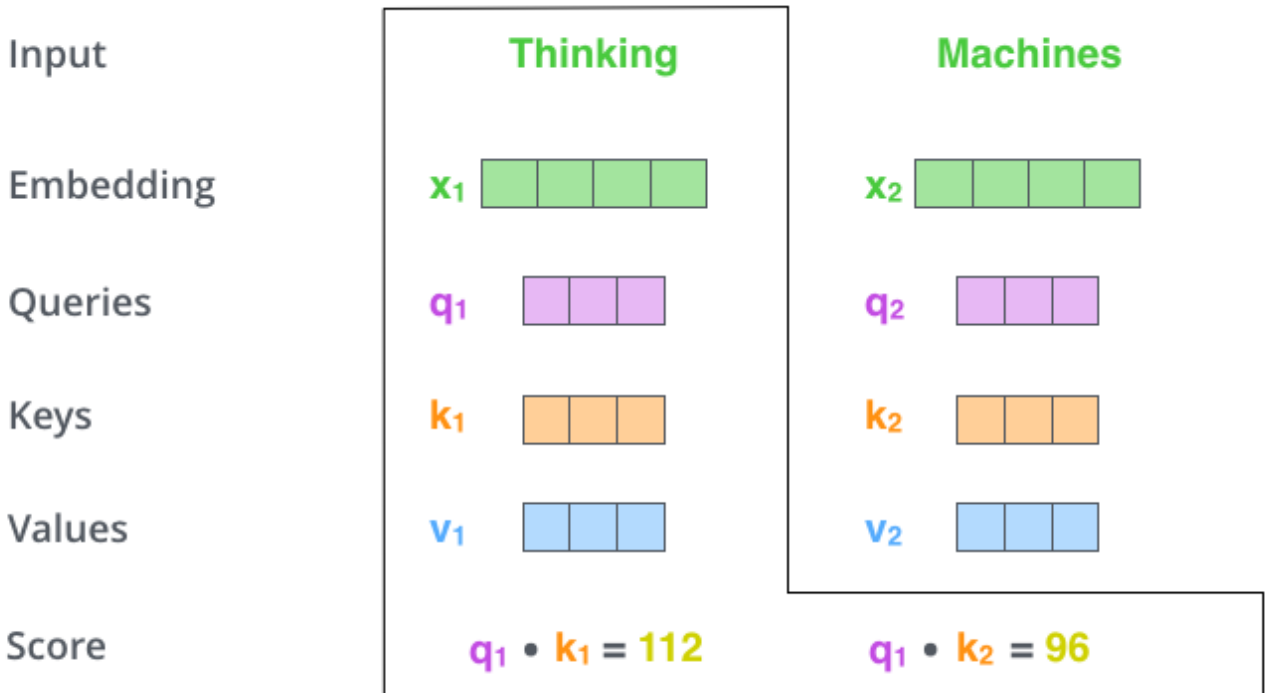
其余b^i的计算过程同理，输入x^i即可以得到b^i。最终Attention的High-level look如下图。



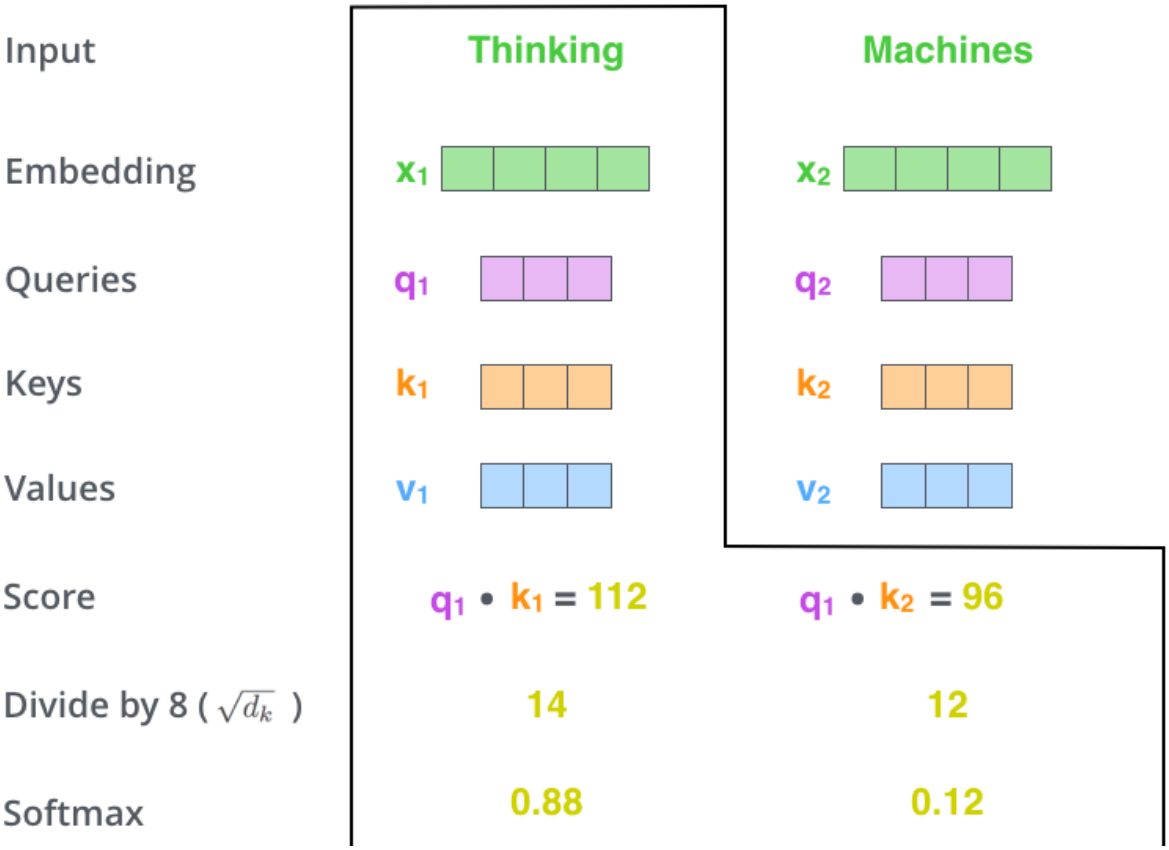
Details in Vectors and Matrices



第一步将输入词汇embedding之后，通过W^Q、W^K、W^V三个矩阵得到每个输入单词对应的Q、K、V三个向量。Q、K、V维度比embedding要小，在论文中Q、K、V是64维，而embedding是512维，维度只是一种architecture choice。

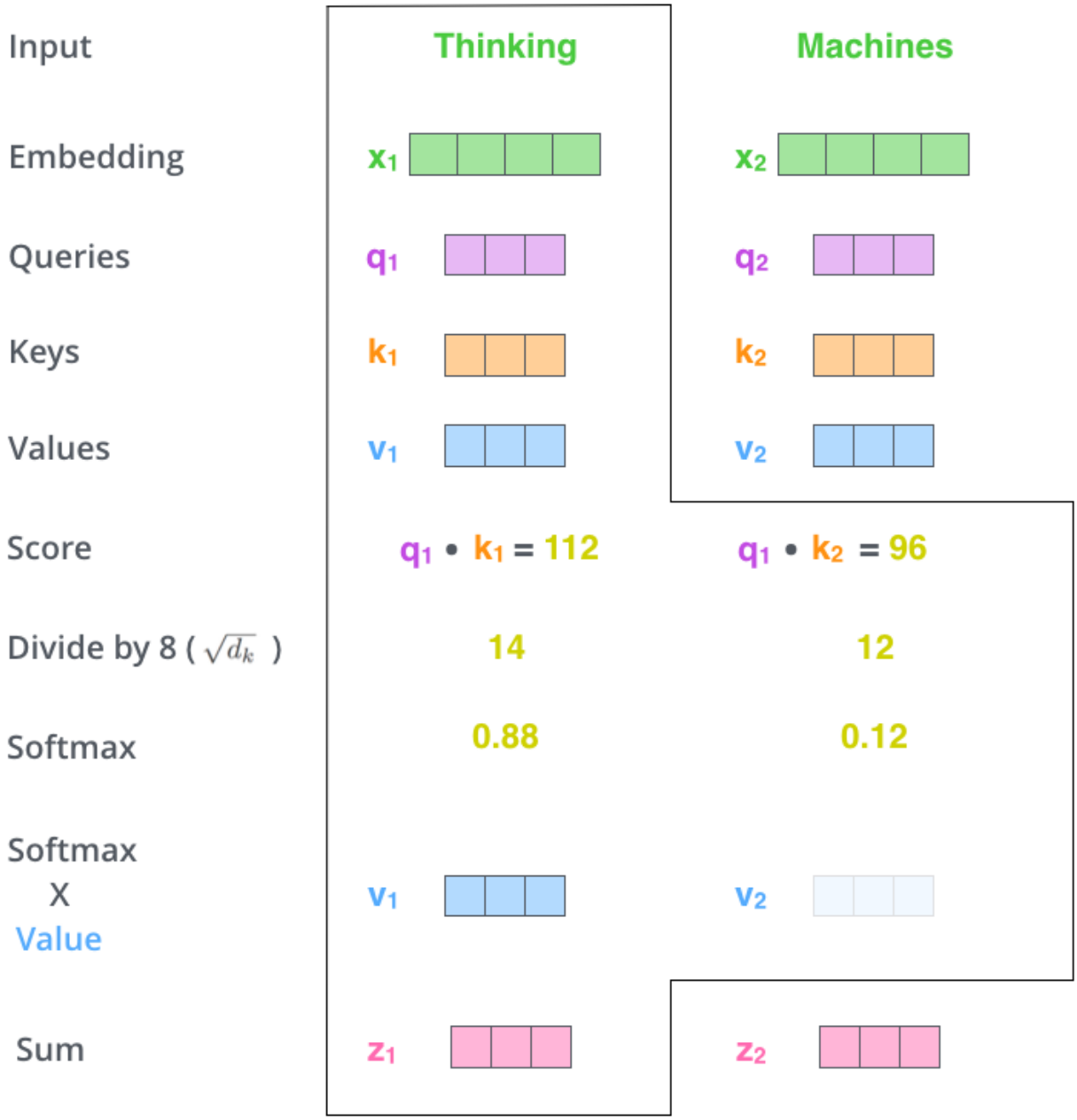


第二步计算每个单词相对当前单词的分数。用当前单词的query（to match others）点乘以每个单词的key（to be matched），即可。



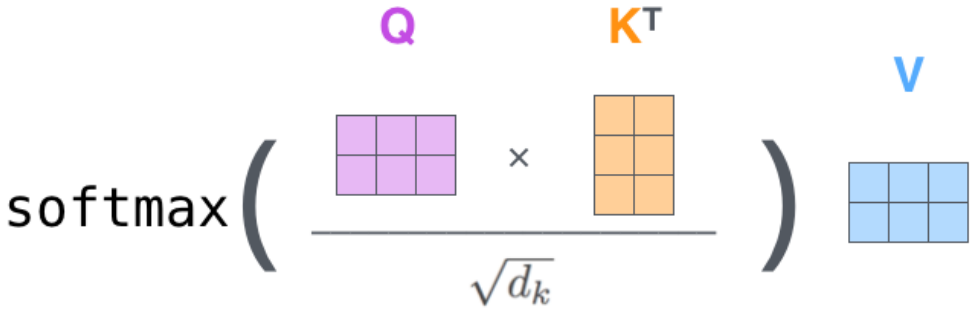
第三步将计算出的分数除以8，即$\sqrt{d_k}$（$d_k=64$），第四步则将计算出来的全部结果取softmax操

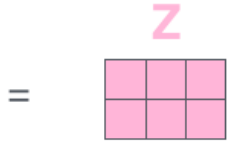
作，使所有值大于零且相加等于1。此处softmax的结果可以理解为单词之间的相关度，越大则相关度越高。



最后将softmax得到的结果与value（information to be extracted）相乘，可以发现与当前单词越相关的单词，value的权重越大。

将全部输入堆叠成矩阵，流程如下列图所示。





Self-attention

$$q^i = W^q a^i$$

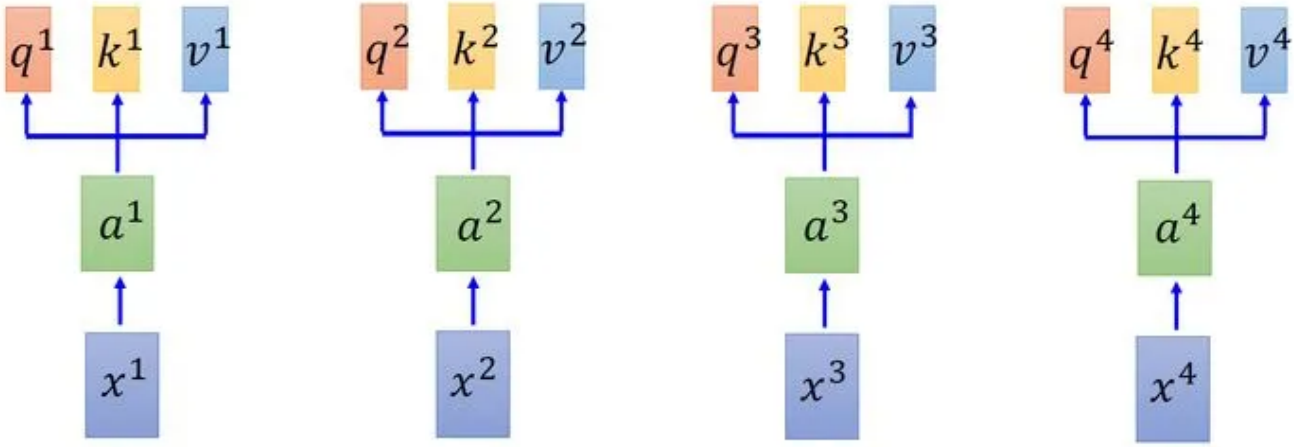
$$k^i = W^k a^i$$

$$v^i = W^v a^i$$

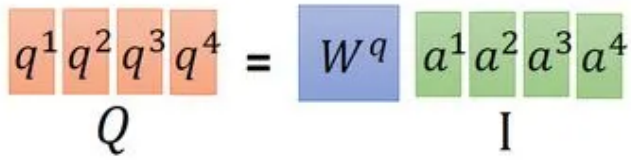
$$\underbrace{q^1 q^2 q^3 q^4}_{Q} = W^q \underbrace{a^1 a^2 a^3 a^4}_{I}$$

$$\underbrace{k^1 k^2 k^3 k^4}_{K} = W^k \underbrace{a^1 a^2 a^3 a^4}_{I}$$

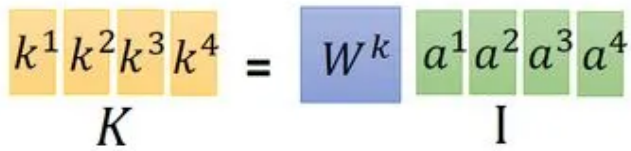
$$\underbrace{v^1 v^2 v^3 v^4}_{V} = W^v \underbrace{a^1 a^2 a^3 a^4}_{I}$$

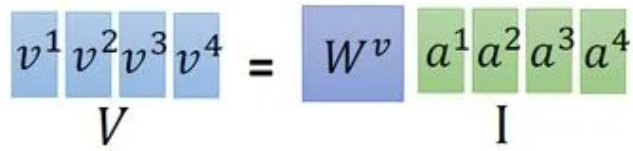


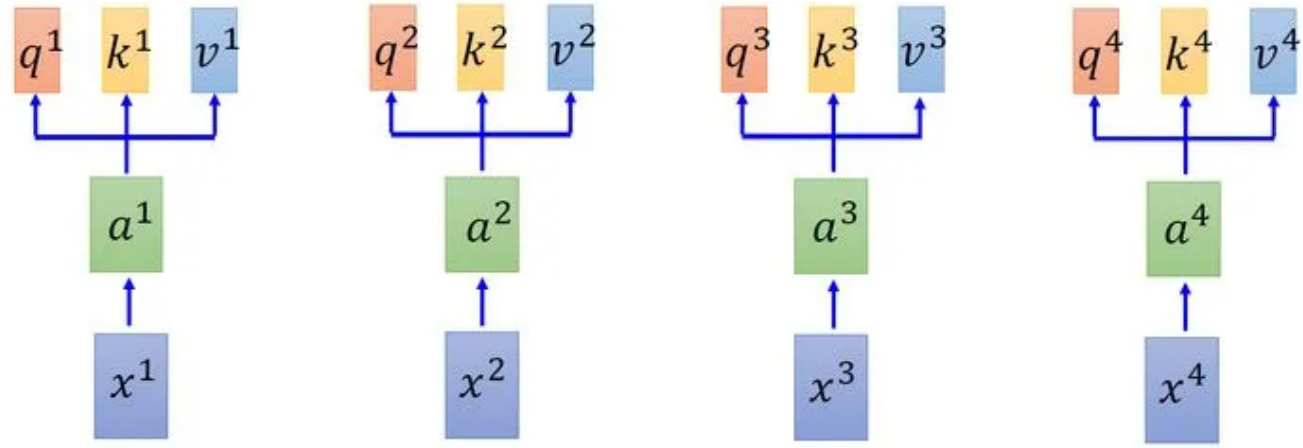
Self-attention

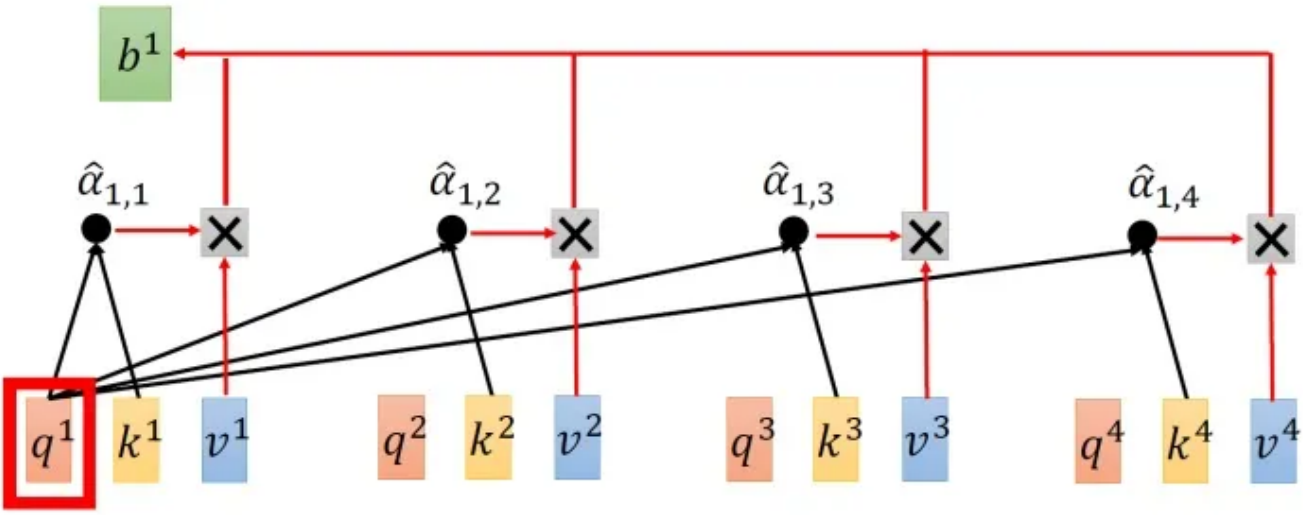


$$q^i = W^q a^i$$
$$k^i = W^k a^i$$
$$v^i = W^v a^i$$





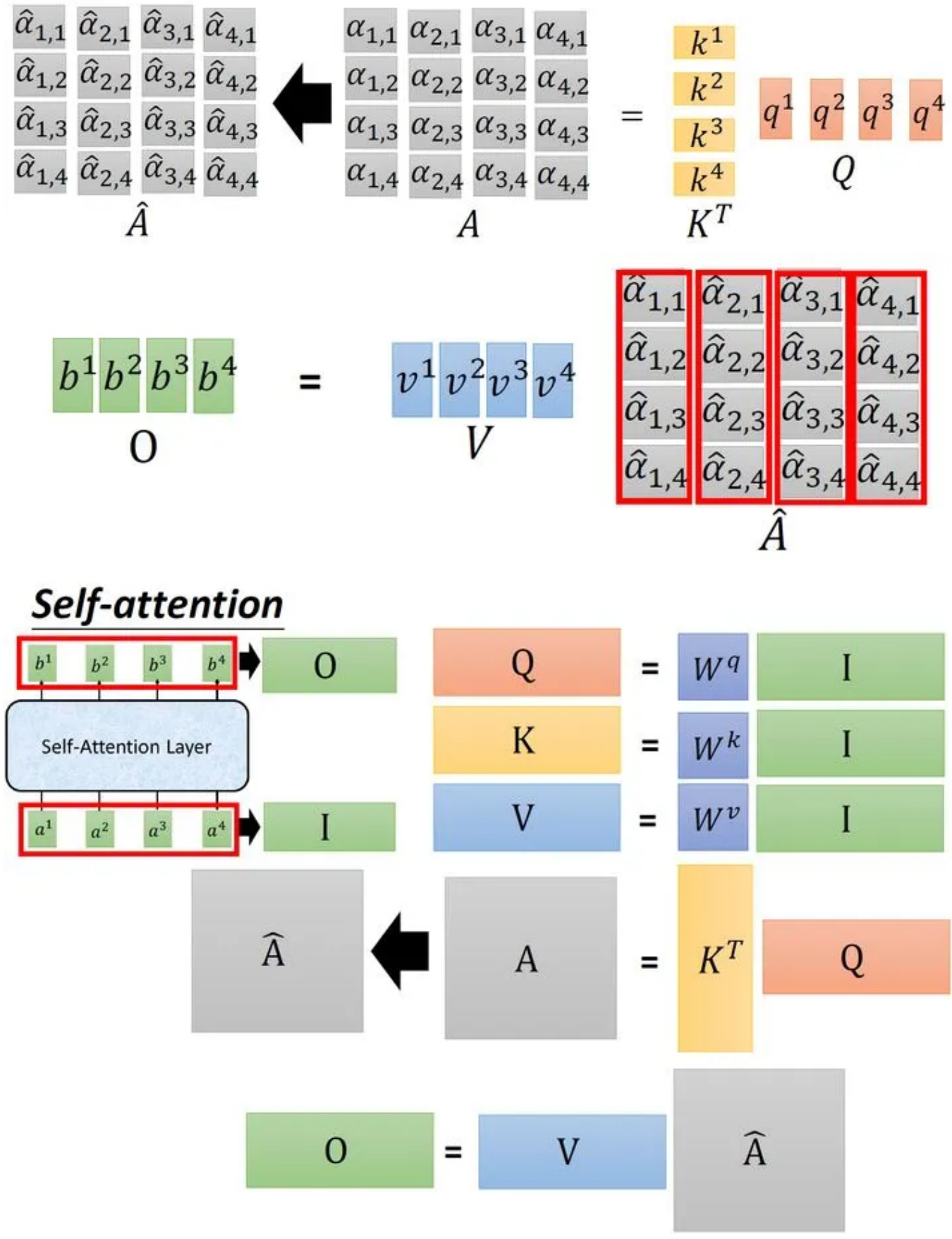




$$\alpha_{1,1} = k^1 q^1 \quad \alpha_{1,2} = k^2 q^1$$
$$\alpha_{1,3} = k^3 q^1 \quad \alpha_{1,4} = k^4 q^1$$

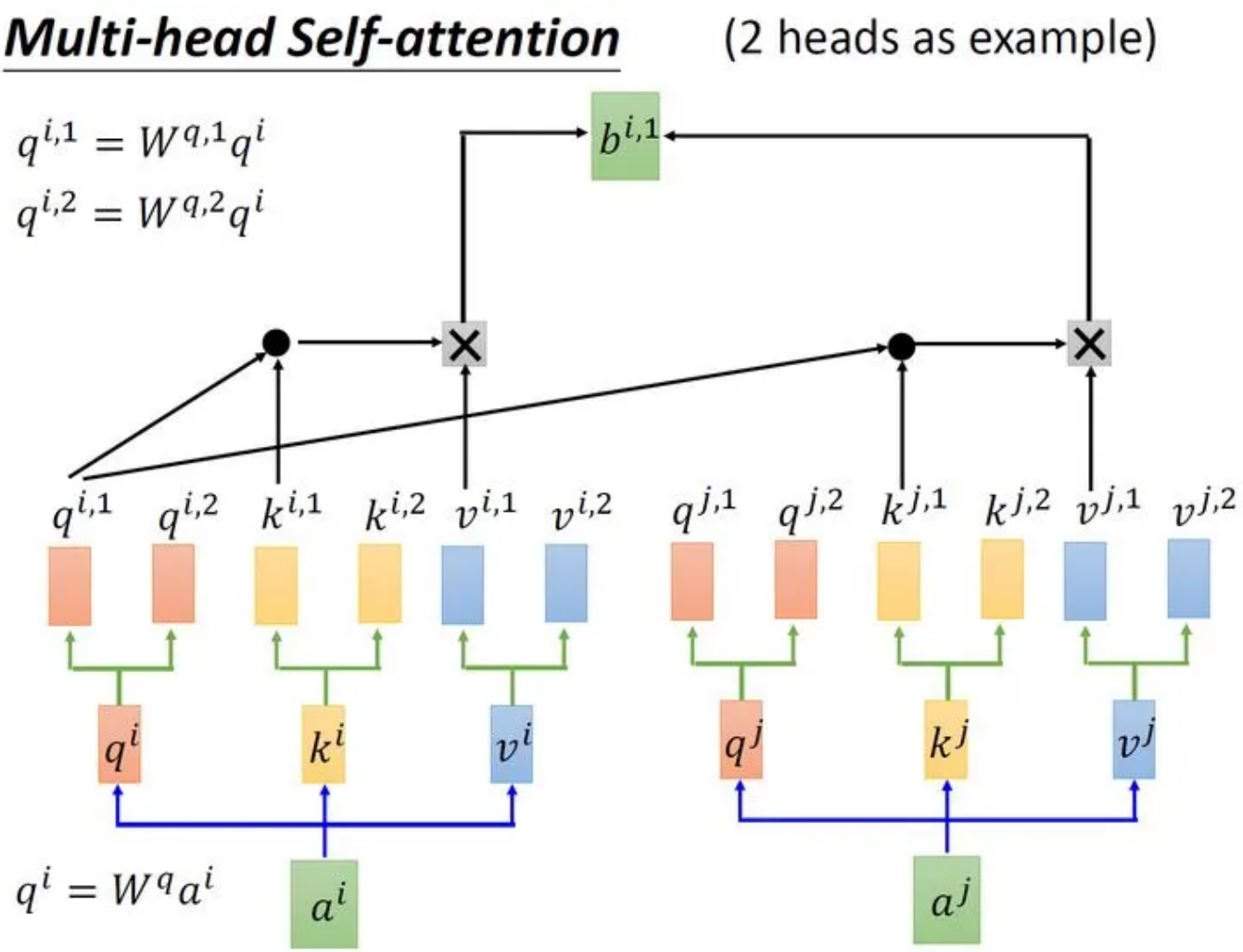
(ignore $\sqrt{d}$ for simplicity)

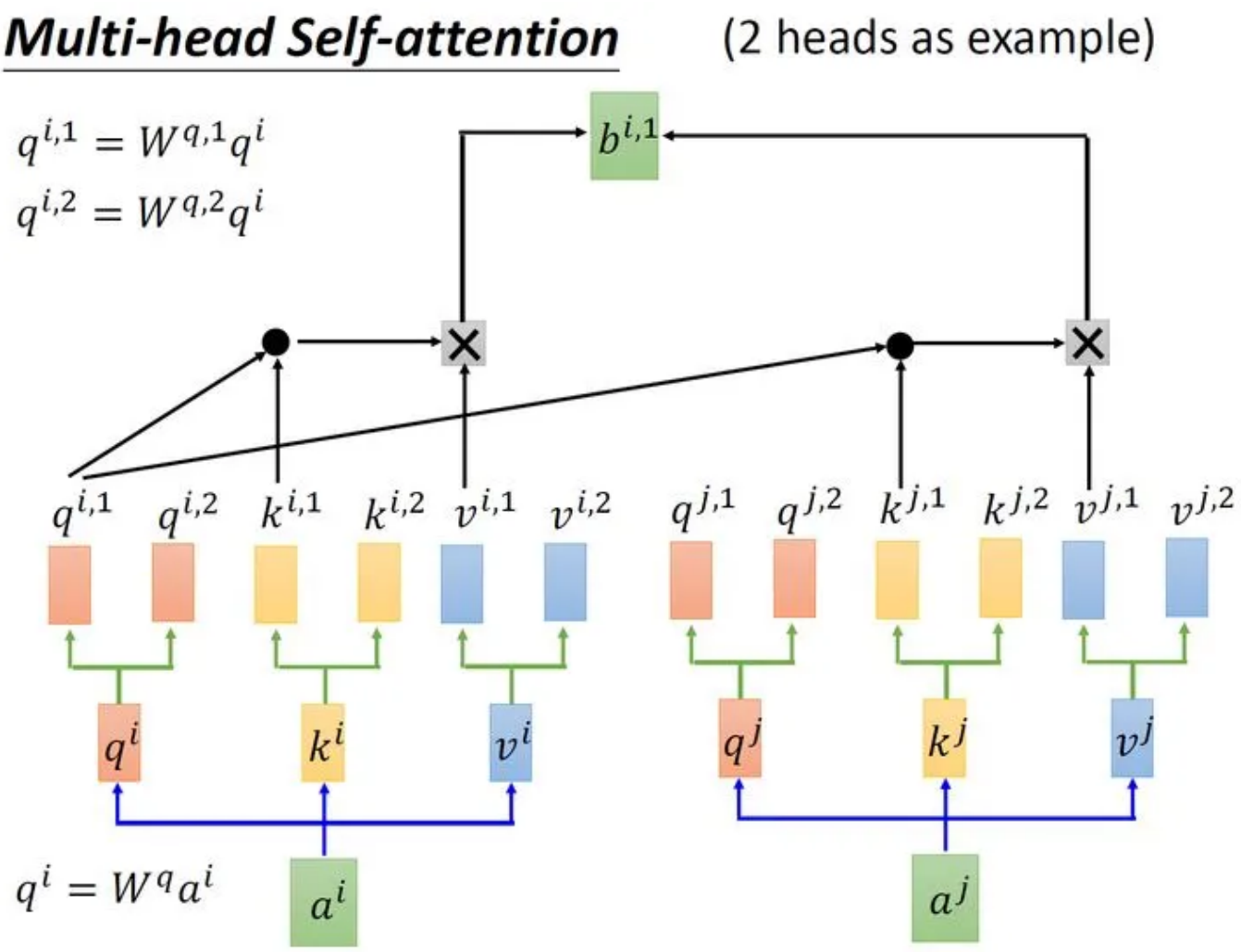
$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} q^1$$

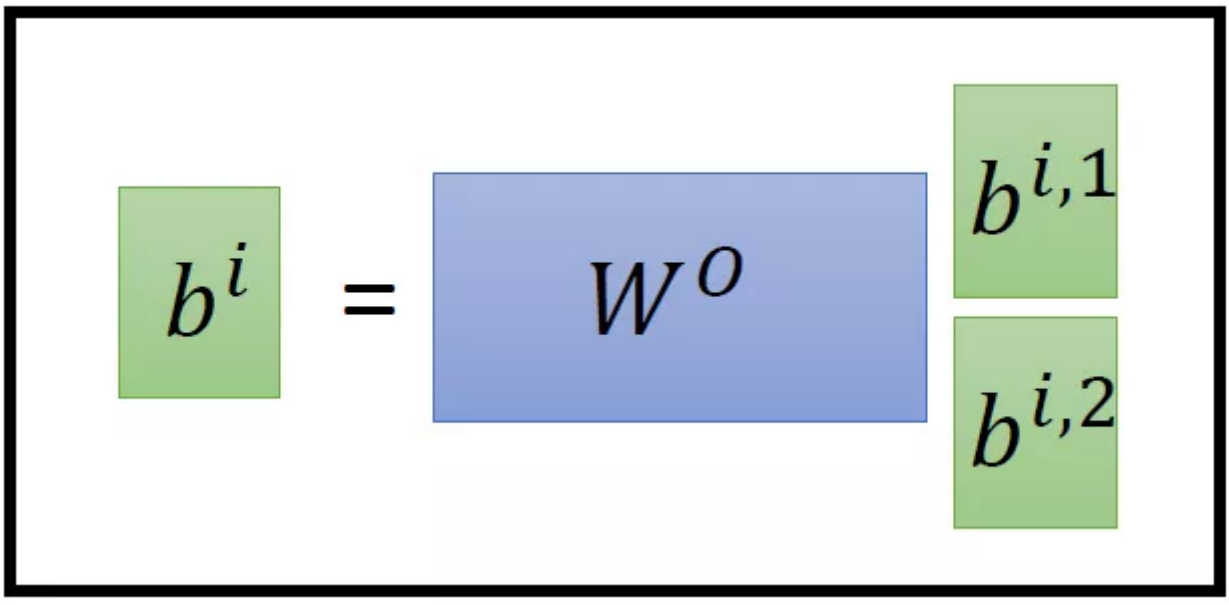


Multi-head Self-Attention

Multi-head和single-head没有本质区别，只是在计算Q、K、V矩阵时会计算出多个矩阵。下图给出2-head的示例。在最后会有一个转移矩阵W^O来将b的维度调整至与原来相同。

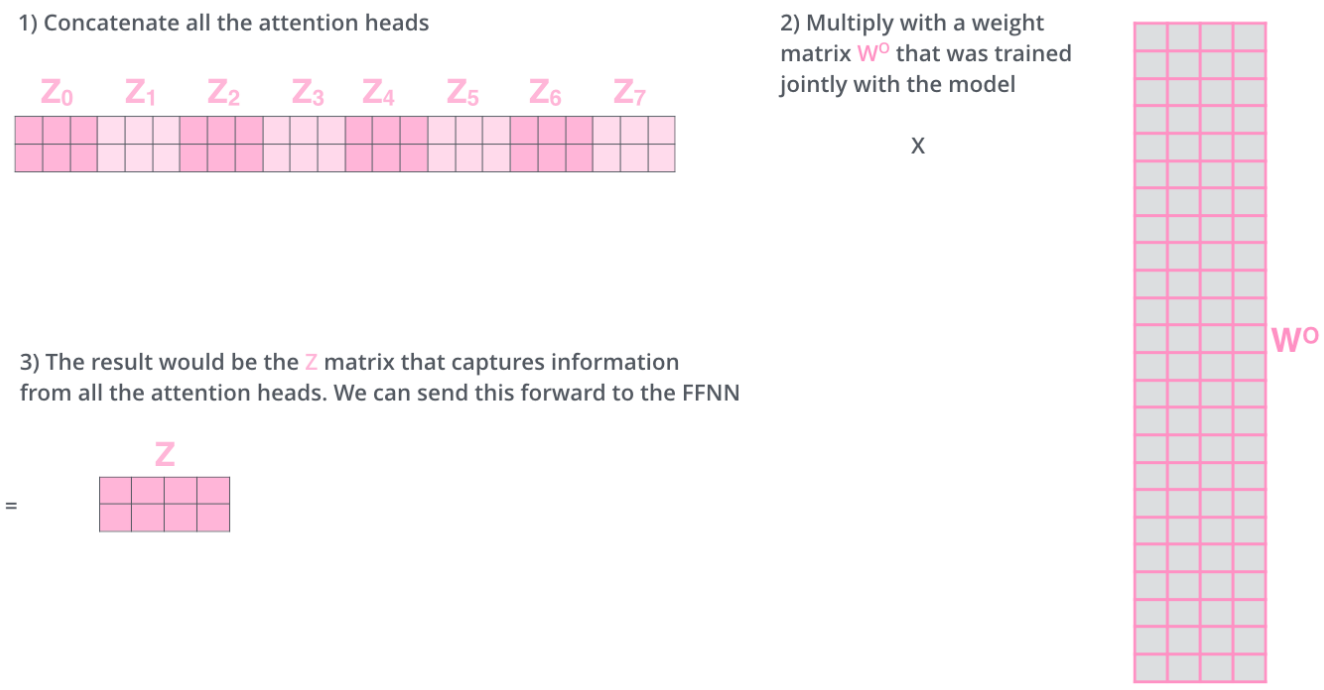

Multi-head Self-attention
(2 heads as example)
$q^{i,1} = W^{q,1} q^i$
$q^{i,2} = W^{q,2} q^i$
$b^{i,1}$
$q^{i,1}$ $q^{i,2}$ $k^{i,1}$ $k^{i,2}$ $v^{i,1}$ $v^{i,2}$
$q^{j,1}$ $q^{j,2}$ $k^{j,1}$ $k^{j,2}$ $v^{j,1}$ $v^{j,2}$
q^i k^i v^i
q^j k^j v^j
$q^i = W^q a^i$
a^i
a^j

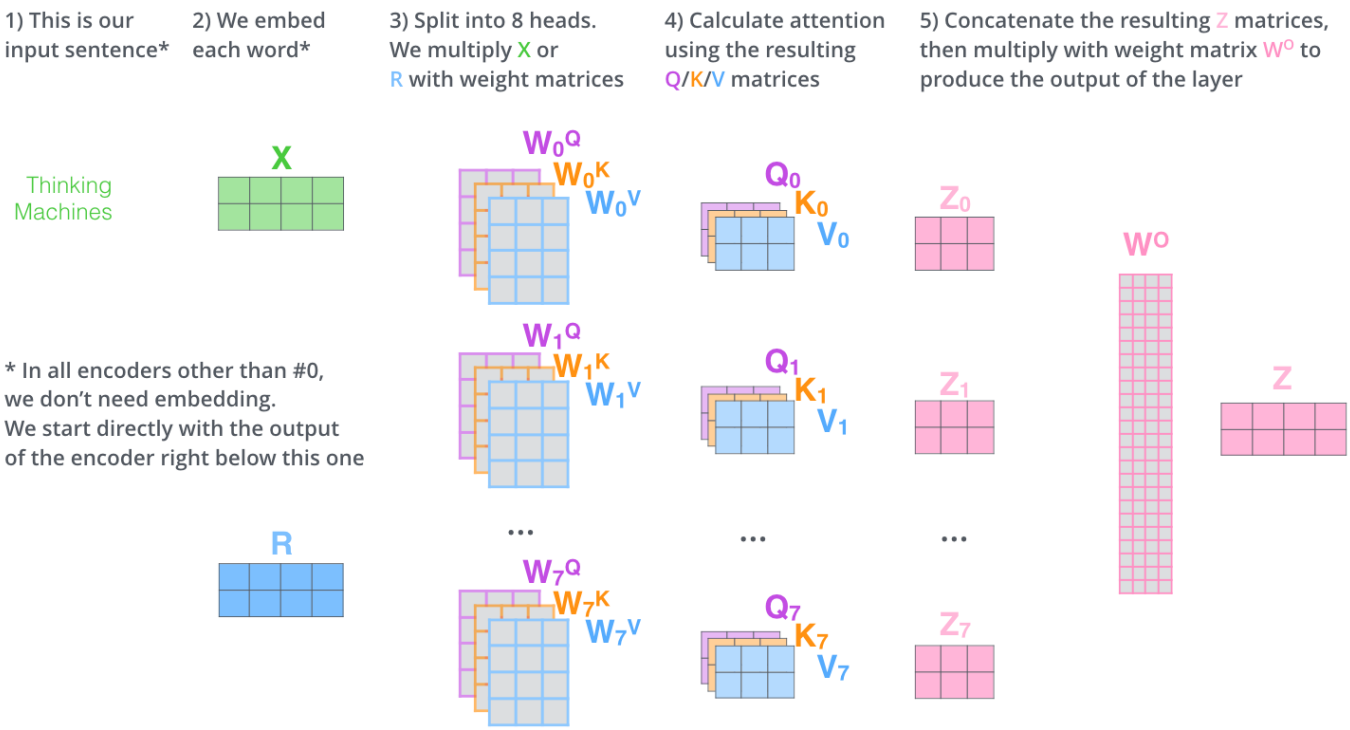

Multi-head Self-attention
(2 heads as example)
$q^{i,1} = W^{q,1} q^i$
$q^{i,2} = W^{q,2} q^i$
$b^{i,1}$
$q^{i,1}$ $q^{i,2}$ $k^{i,1}$ $k^{i,2}$ $v^{i,1}$ $v^{i,2}$
$q^{j,1}$ $q^{j,2}$ $k^{j,1}$ $k^{j,2}$ $v^{j,1}$ $v^{j,2}$
q^i k^i v^i
q^j k^j v^j
$q^i = W^q a^i$
a^i
a^j


b^i = W^o $b^{i,1}$ $b^{i,2}$

下列图也是一组示例。


X
Thinking
Machines
ATTENTION HEAD #0
ATTENTION HEAD #1
Q0
K0
V0
W0Q
W0K
W0V
Q1
K1
V1
W1Q
W1K
W1V
X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7



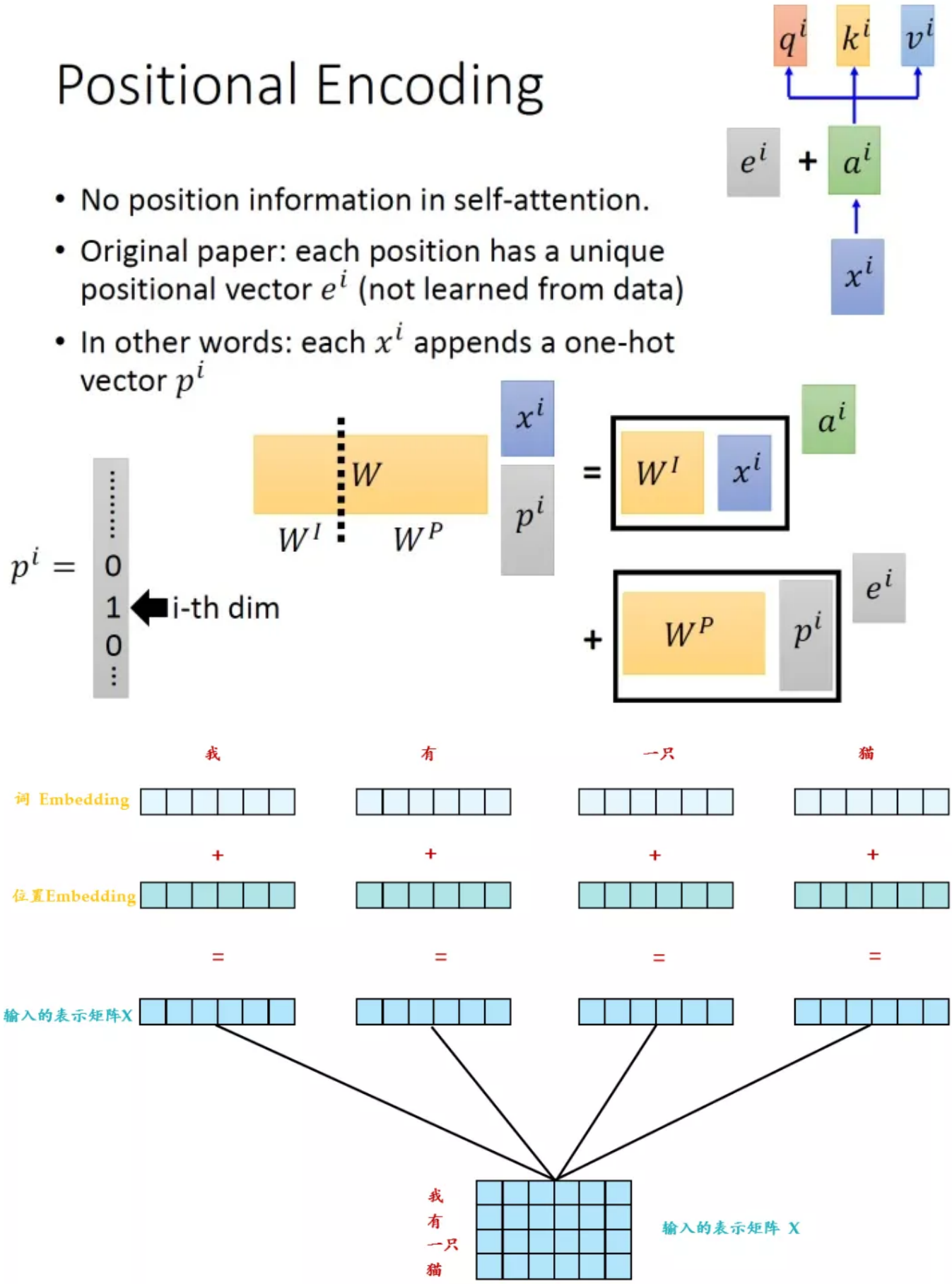


Demo

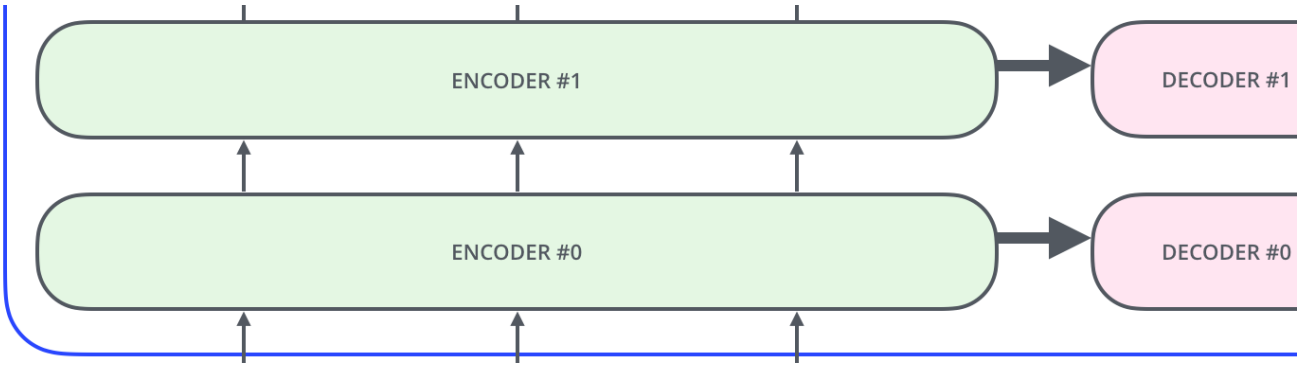
Tensor2Tensor Notebook

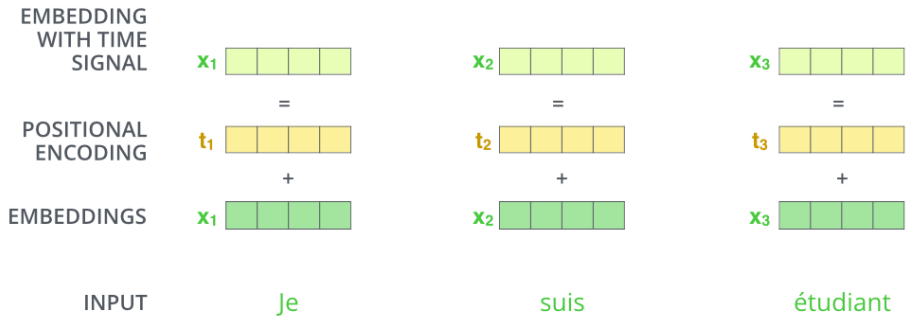
Positional Encoding

以上是multi-head self-attention的原理，但是还有一个问题是：现在的self-attention中没有位置的信息，一个单词向量的"近在咫尺"位置的单词向量和"远在天涯"位置的单词向量效果是一样的，没有表示位置的信息(No position information in self attention)。所以输入"A打了B"或者"B打了A"的效果其实是一样的，因为并没有考虑位置的信息。

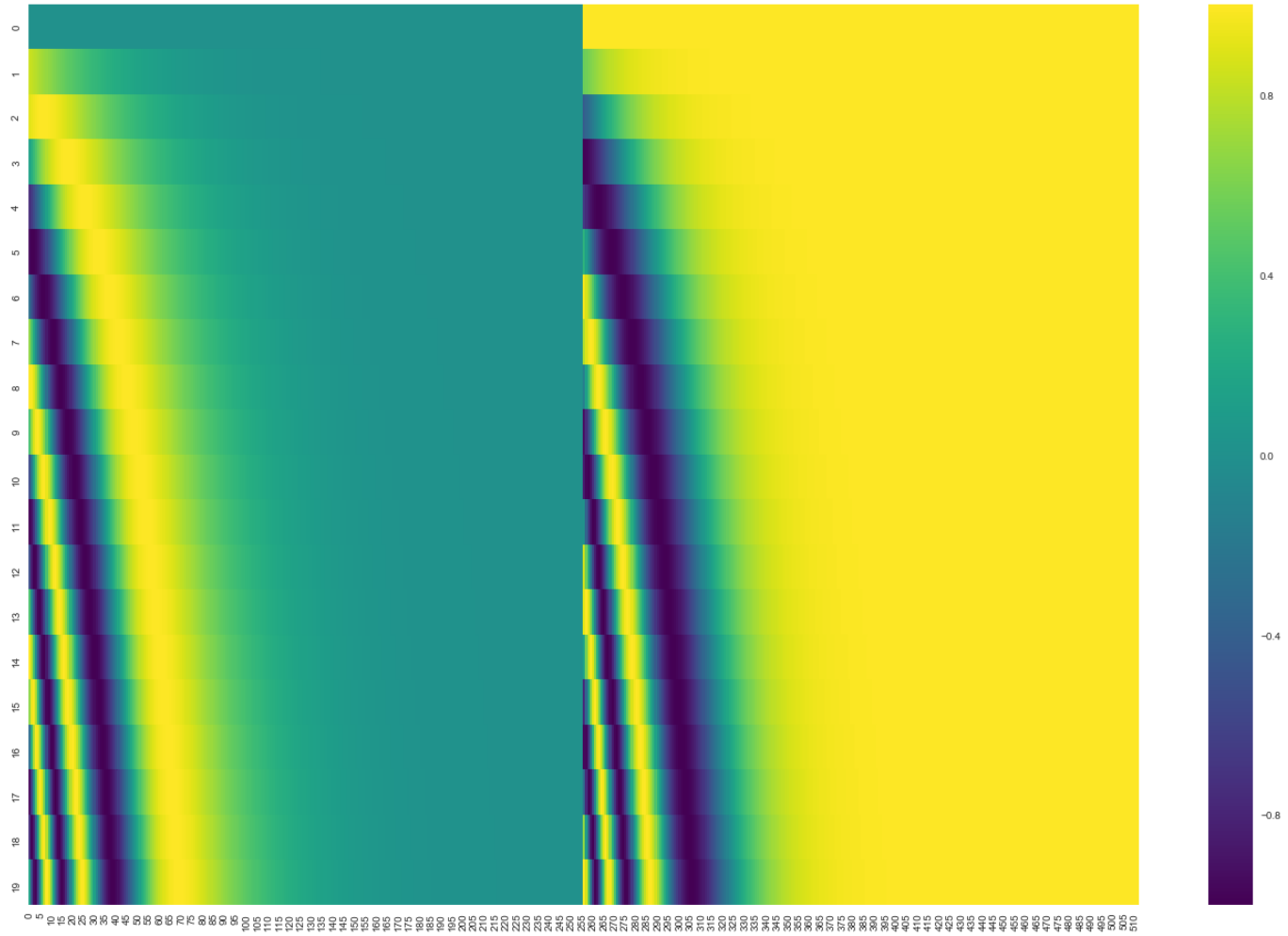


具体的做法是：给每一个位置规定一个表示位置信息的向量e_i，让它与a_i加在一起之后作为新的a_i参与后面的运算过程，但是这个向量e_i是由人工设定的，而不是神经网络学习出来的。每一个位置都有一个不同的e_i。

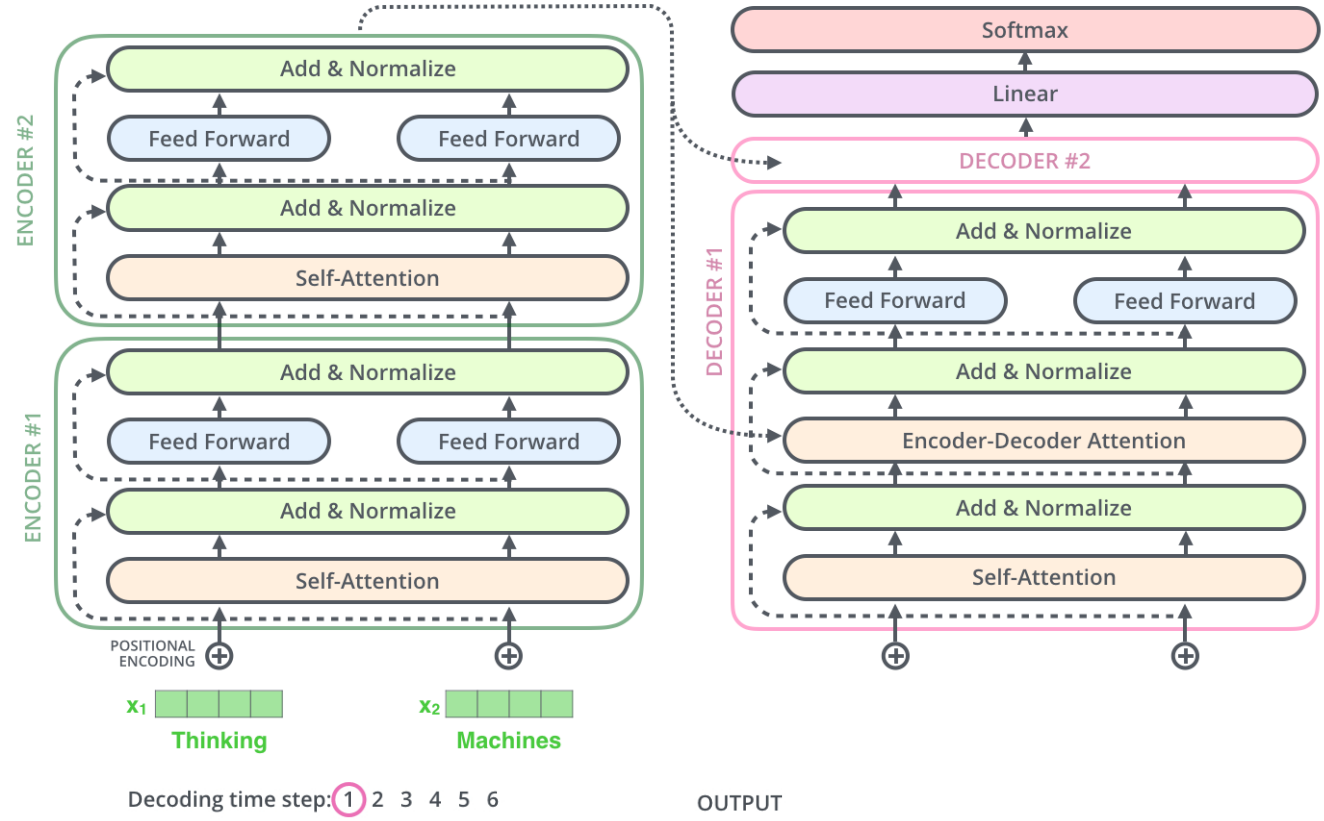


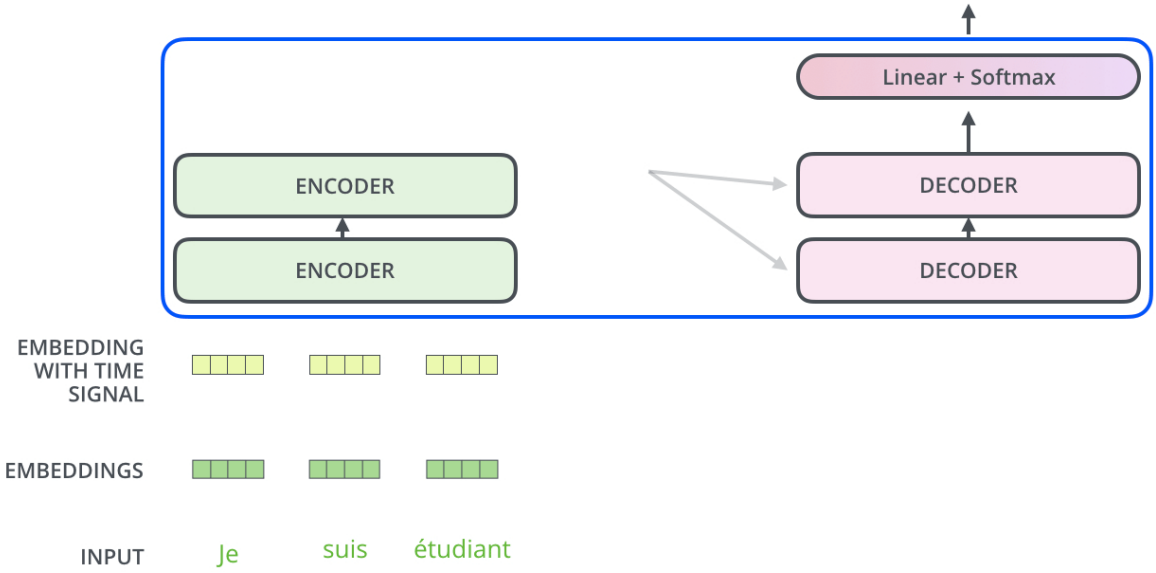


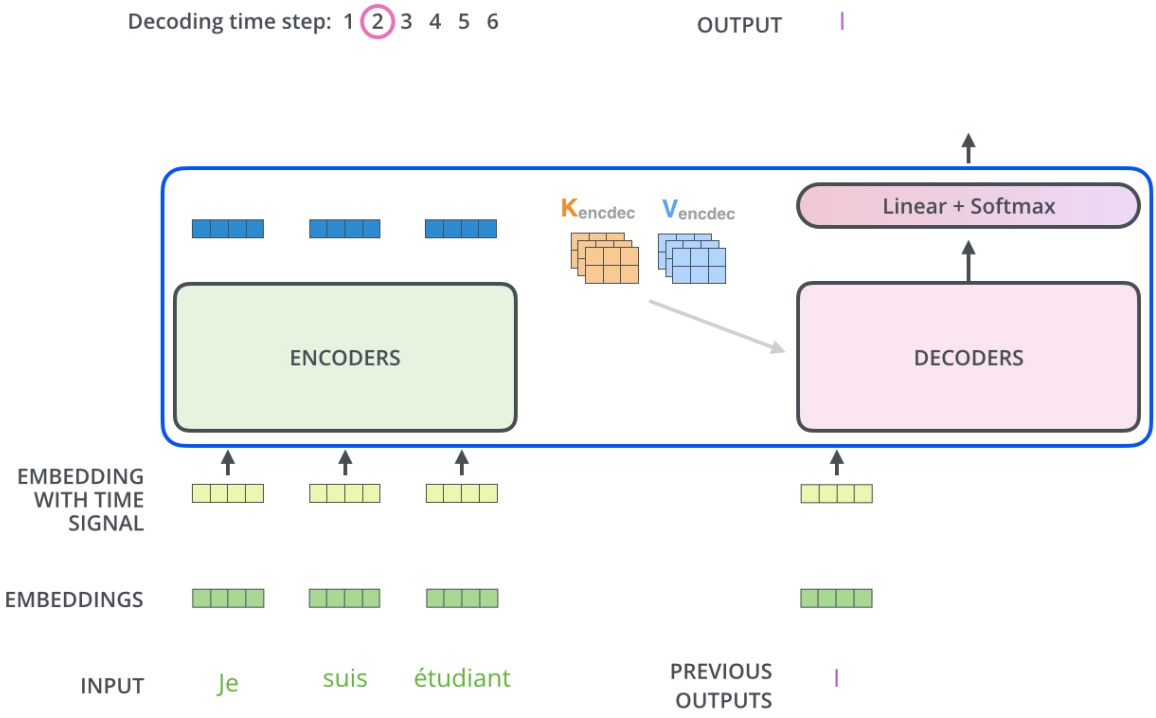
位置编码设定类似下图。



完整流程


Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #2
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
x2
Thinking
Machines
Decoding time step: 1 2 3 4 5 6
OUTPUT


Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant



Reference

[1] 一篇英文博客 [2] Transformer中文综述