注意力机制到底在做什么，Q/K/V怎么来的？一文读懂Attention注意力机制

PP鲁

+ 关注他

> 本文同时发布于我的个人网站，公式图片显示效果更好，欢迎访问：lulaoshi.info/machine-l...

Transformer[^1]论文中使用了注意力Attention机制，注意力Attention机制的最核心的公式为：

$$Attention(Q,K,V)=Softmax(\frac{QK^\top}{\sqrt{d_k}})V$$

这个公式中的 `Q` 、 `K` 和 `V` 分别代表Query、Key和Value，他们之间进行的数学计算并不容易理解。

从向量点乘说起

我们先从$Softmax(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$这样一个公式开始。

首先需要复习一下向量点乘（Dot Product）的概念。对于两个行向量$\mathbf{x}$和$\mathbf{y}$：

$$\mathbf{x}=[x_0,x_1,\cdots,x_n]$$

$$\mathbf{y}=[y_0,y_1,\cdots,y_n]$$

$$\mathbf{x}\cdot\mathbf{y}=x_0y_0+x_1y_1+\cdots+x_ny_n$$

向量点乘的几何意义是：向量$\mathbf{x}$在向量$\mathbf{y}$方向上的投影再与向量$\mathbf{y}$的乘积，能够反应两个向量的相似度。向量点乘结果大，两个向量越相似。

一个矩阵$\mathbf{X}$由n行向量组成。比如，我们可以将某一行向量$\mathbf{x}_i$理解成一个词的词向量，共有n个行向量组成$n\times n$的方形矩阵：

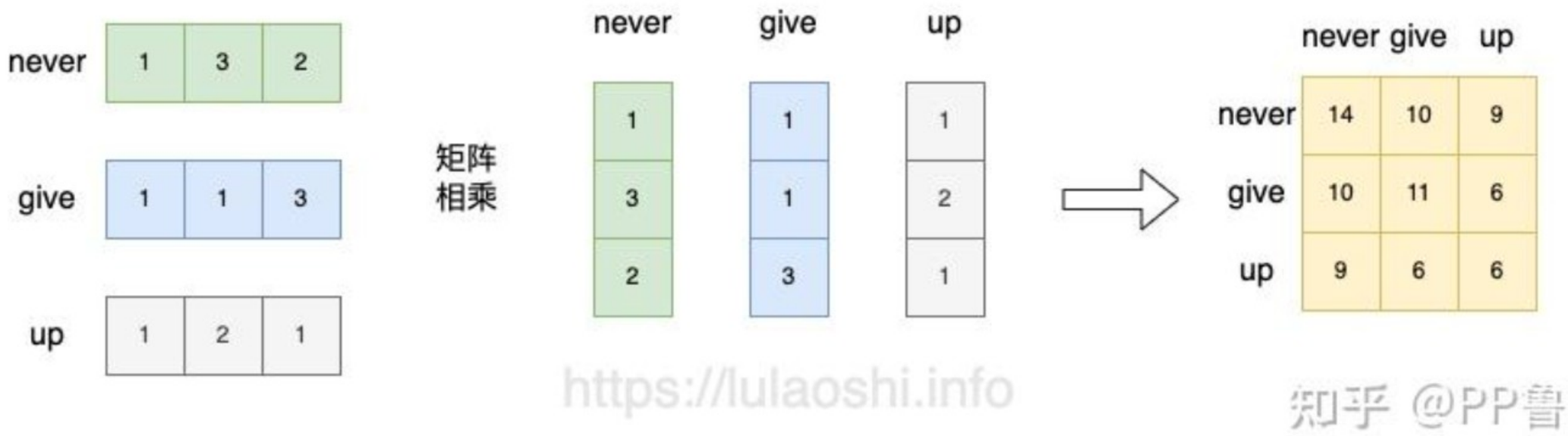
$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \quad \mathbf{X}^\top = \begin{bmatrix} \mathbf{x}_0^\top & \mathbf{x}_1^\top & \cdots & \mathbf{x}_n^\top \end{bmatrix}$$

矩阵$\mathbf{X}$与矩阵的转置$\mathbf{X}^\top$相乘，$\mathbf{X}$中的每一行与$\mathbf{X}^\top$的每一列相乘得到目标矩阵的一个元素，$\mathbf{X}\mathbf{X}^\top$可表示为：

$$\mathbf{X}\mathbf{X}^\top = \begin{bmatrix} \mathbf{x}_0 \cdot \mathbf{x}_0 & \mathbf{x}_0 \cdot \mathbf{x}_1 & \cdots & \mathbf{x}_0 \cdot \mathbf{x}_n \\ \mathbf{x}_1 \cdot \mathbf{x}_0 & \mathbf{x}_1 \cdot \mathbf{x}_1 & \cdots & \mathbf{x}_1 \cdot \mathbf{x}_n \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_n \cdot \mathbf{x}_0 & \mathbf{x}_n \cdot \mathbf{x}_1 & \cdots & \mathbf{x}_n \cdot \mathbf{x}_n \end{bmatrix}$$

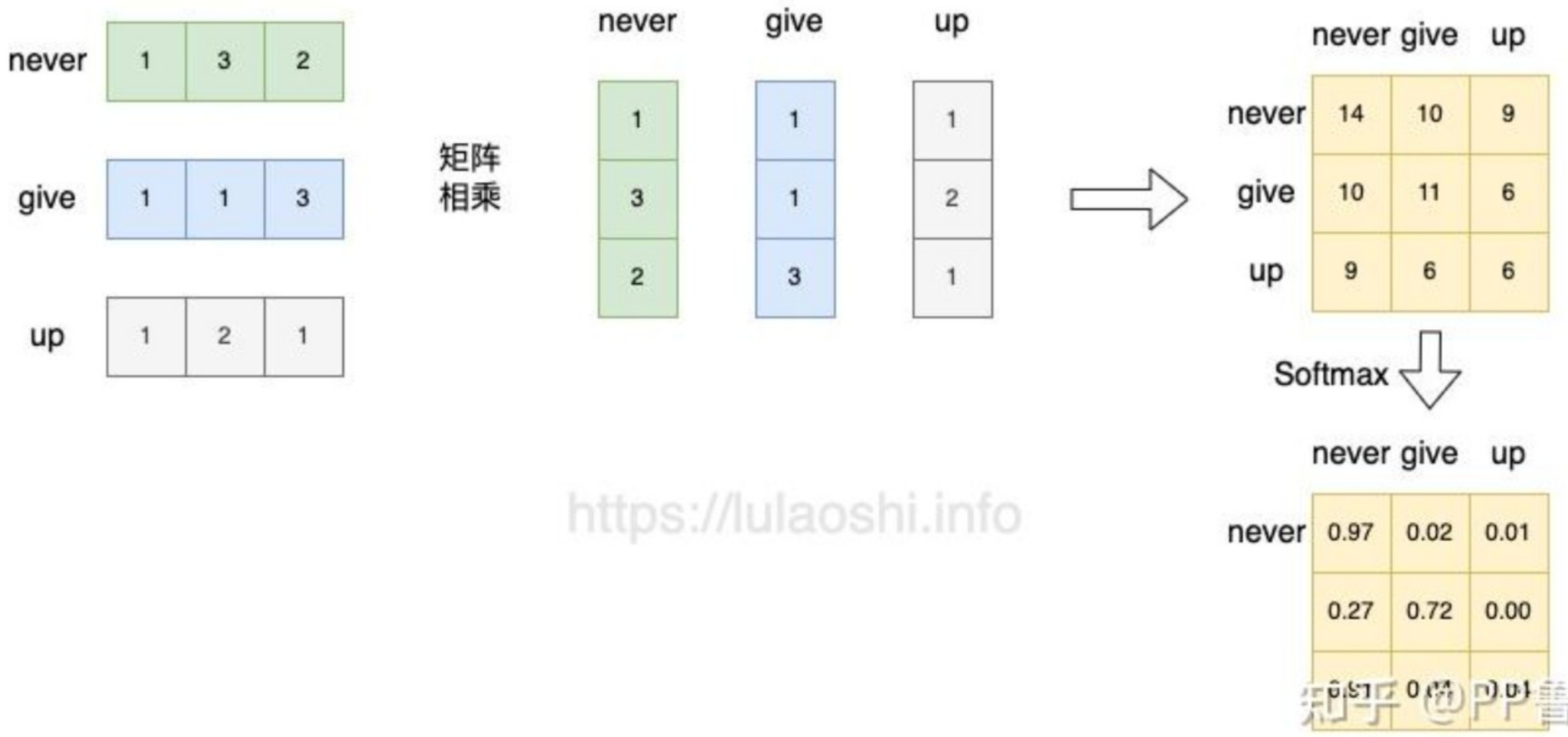
以$\mathbf{X}\mathbf{X}^\top$中的第一行第一列元素为例，其实是向量$\mathbf{x}_0$与$\mathbf{x}_0$自身做点乘，其实就是$\mathbf{x}_0$自身与自身的相似度，那第一行第二列元素就是$\mathbf{x}_0$与$\mathbf{x}_1$之间的相似度。

下面以词向量矩阵为例，这个矩阵中，每行为一个词的词向量。矩阵与自身的转置相乘，生成了目标矩阵✦，目标矩阵其实就是一个词的词向量与各个词的词向量的相似度。

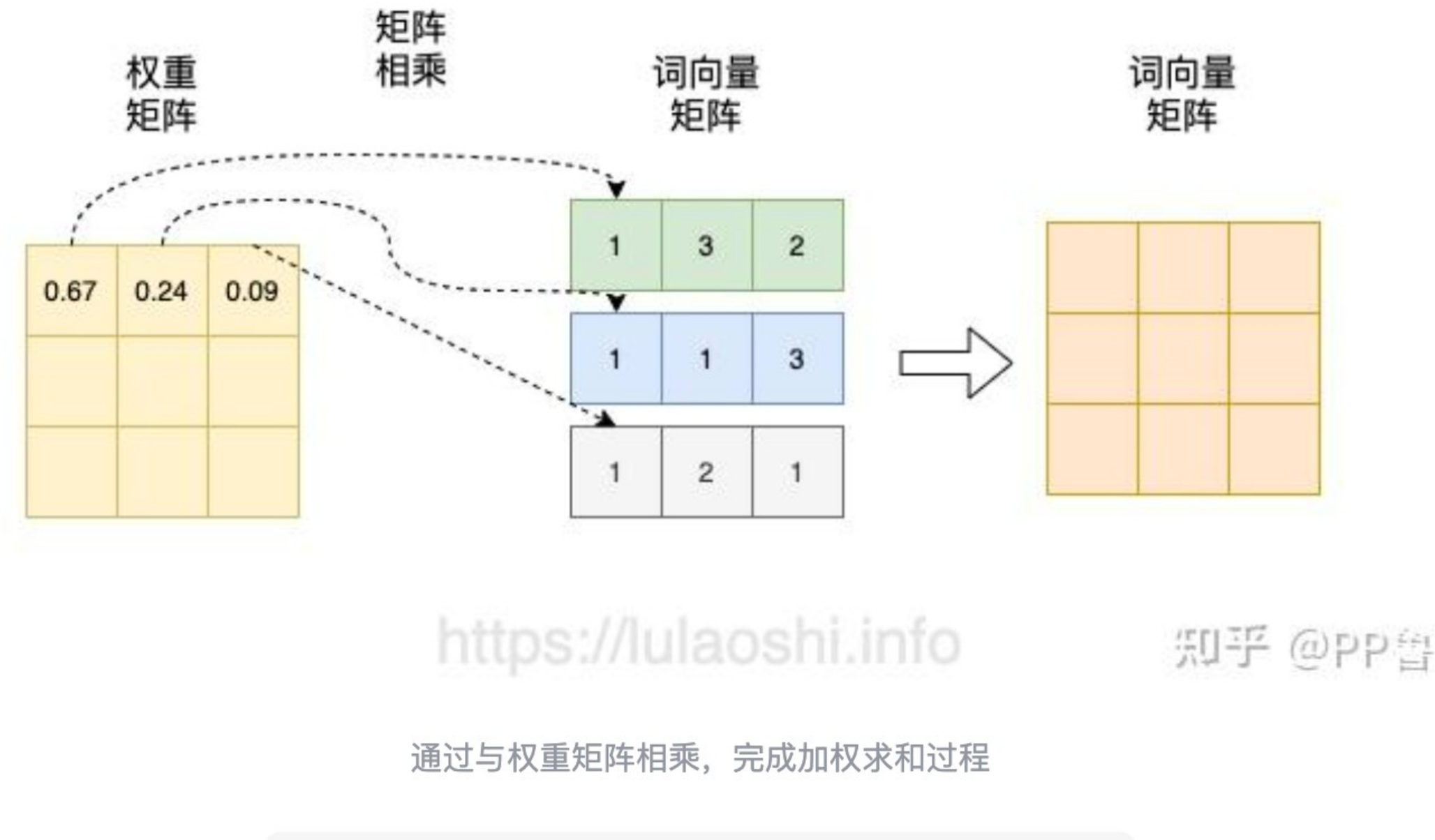


词向量矩阵相乘

如果再加上Softmax呢？我们进行下面的计算：$Softmax(\mathbf{X}\mathbf{X}^\top)$。Softmax的作用是对向量做归一化，那么就是对相似度的归一化，得到了一个归一化之后的权重矩阵，矩阵中，某个值的权重越大，表示相似度越高。



在这个基础上，再进一步：$Softmax(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$，将得到的归一化的权重矩阵与词向量矩阵相乘。权重矩阵中某一行分别与词向量的一列相乘，词向量矩阵的一列其实代表着不同词的某一维度。经过这样一个矩阵相乘，相当于一个加权求和的过程，得到结果词向量是经过加权求和之后的新表示，而权重矩阵是经过相似度和归一化计算得到的。



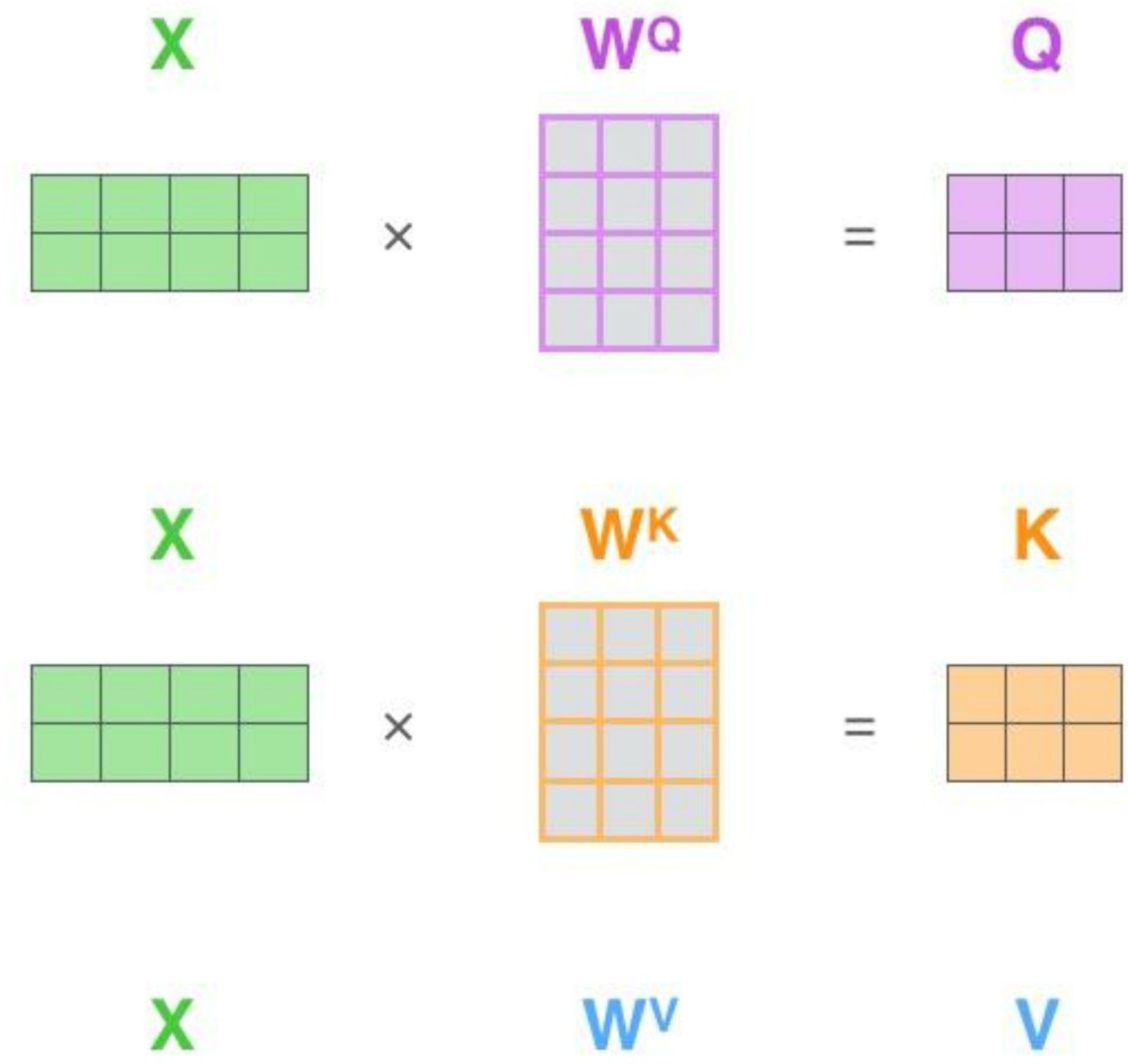
通过与权重矩阵相乘，完成加权求和过程

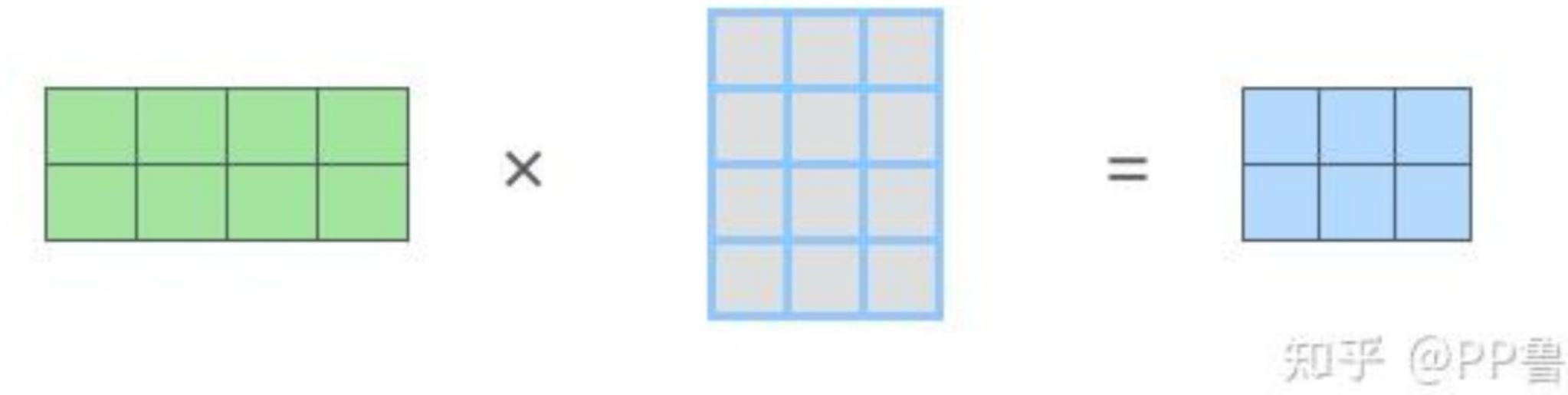
Q、K、V

注意力Attention机制的最核心的公式为：$Softmax(\frac{QK^\top}{\sqrt{d_k}})V$，与我们刚才分析的$Softmax(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$有几分相似。Transformer[^1]论文中将这个Attention公式描述为：Scaled Dot-Product Attention。其中，Q为Query、K为Key、V为Value。Q、K、V是从哪儿来的呢？Q、K、V其实都是从同样的输入矩阵X线性变换✦而来的。我们可以简单理解成：

$$Q = XW^Q$$
$$K = XW^K$$
$$V = XW^V$$

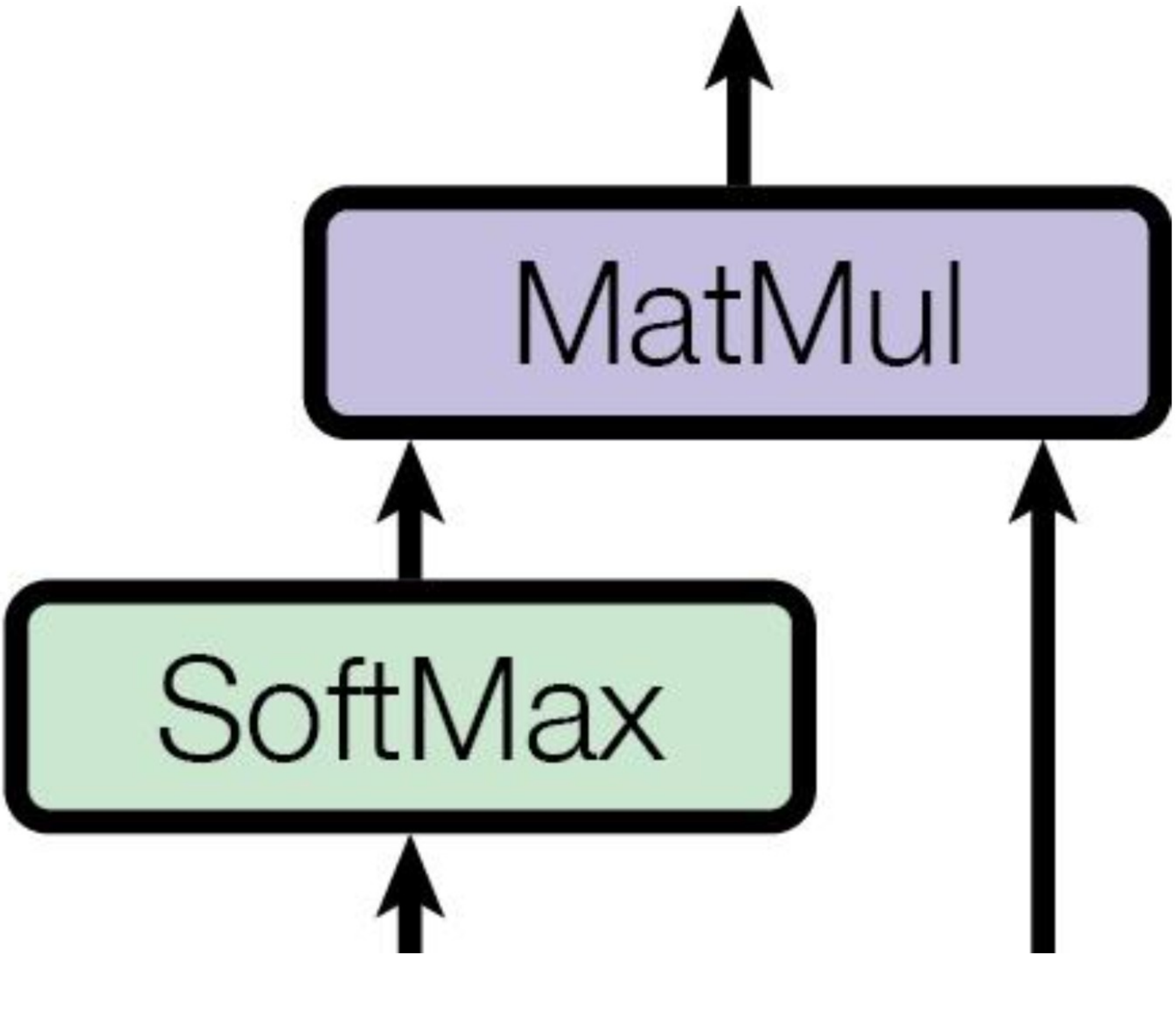
用图片演示为：

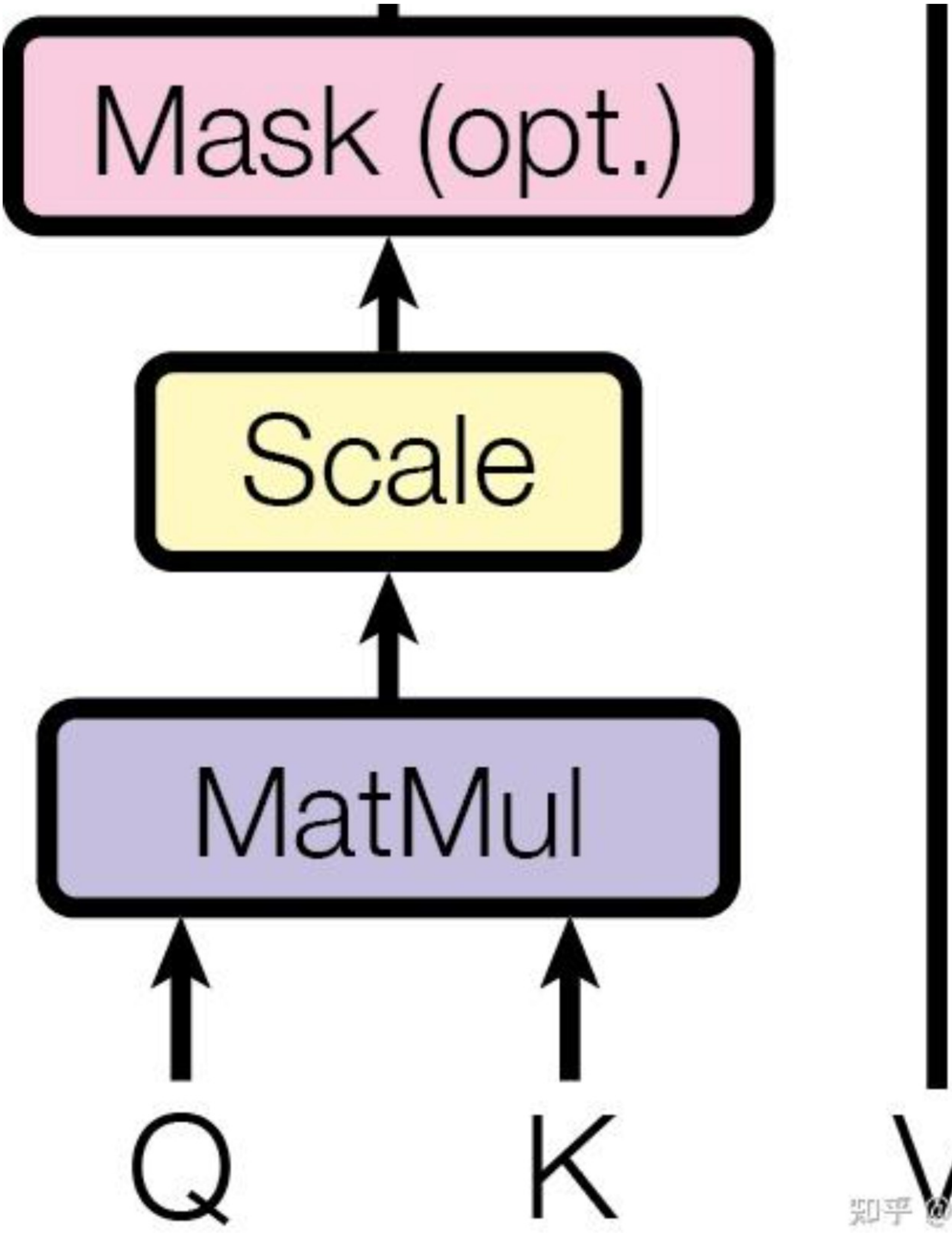




X分别乘以三个矩阵，生成Q、K、V矩阵

其中，W^Q，W^K和W^V是三个可训练的参数矩阵。输入矩阵X分别与W^Q，W^K和W^V相乘，生成Q、K和V，相当于经历了一次线性变换。Attention不直接使用X，而是使用经过矩阵乘法✦生成的这三个矩阵，因为使用三个可训练的参数矩阵✦，可增强模型的拟合能力。





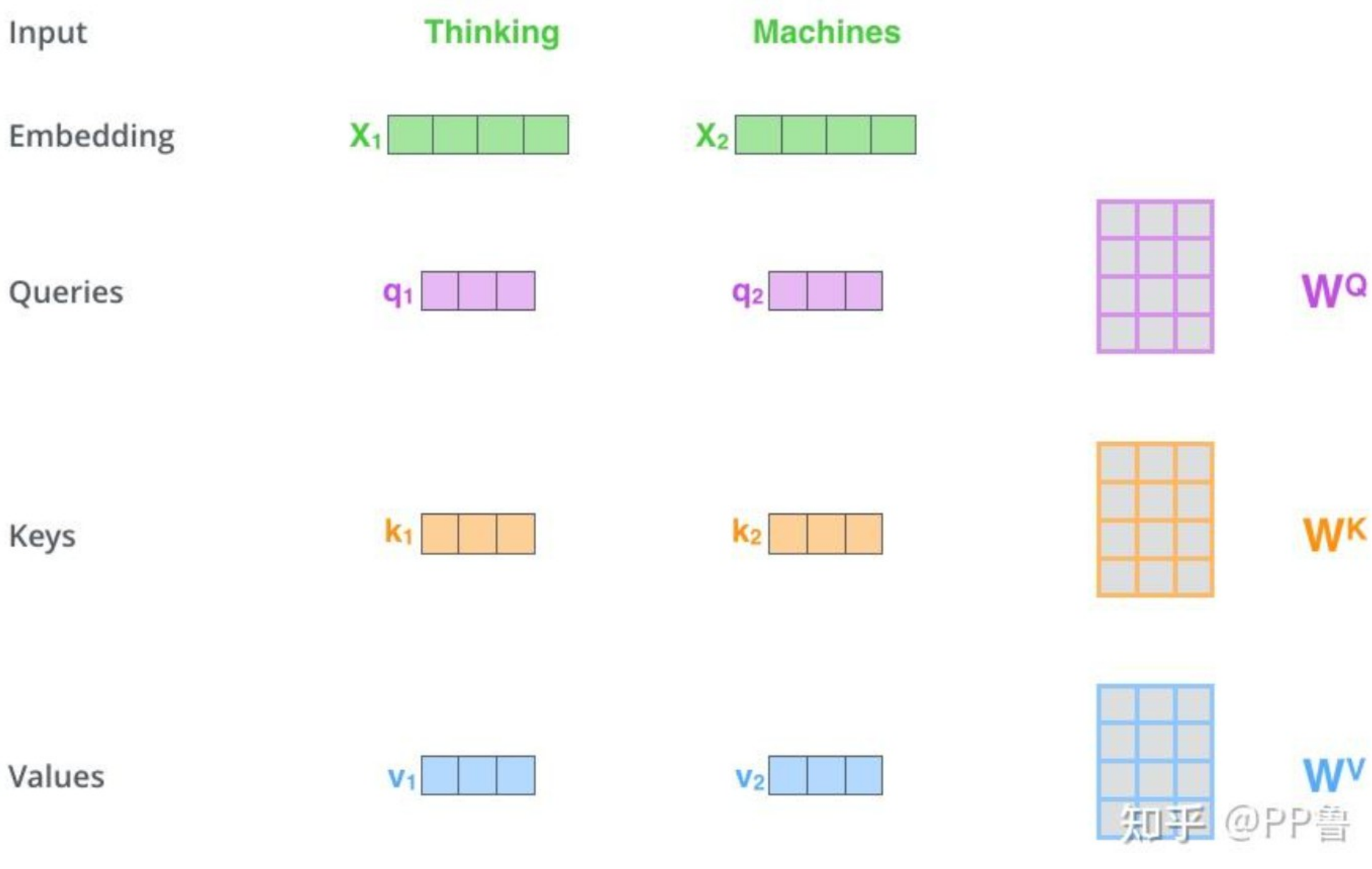
Scaled Dot-Product Attention

在这张图中，Q与$K^\top$经过MatMul，生成了相似度矩阵✦。对相似度矩阵每个元素除以$\sqrt{d_k}$，d_k为K的维度大小。这个除法被称为Scale。当d_k很大时，$QK^\top$的乘法结果方差✦变大，进行Scale可以使方差变小，训练时梯度更新更稳定。

Mask是机器翻译等自然语言处理✦任务中经常使用的环节。在机器翻译等NLP场景中，每个样本句子的长短不同，对于句子结束之后的位置，无需参与相似度的计算，否则影响Softmax的计算结果。

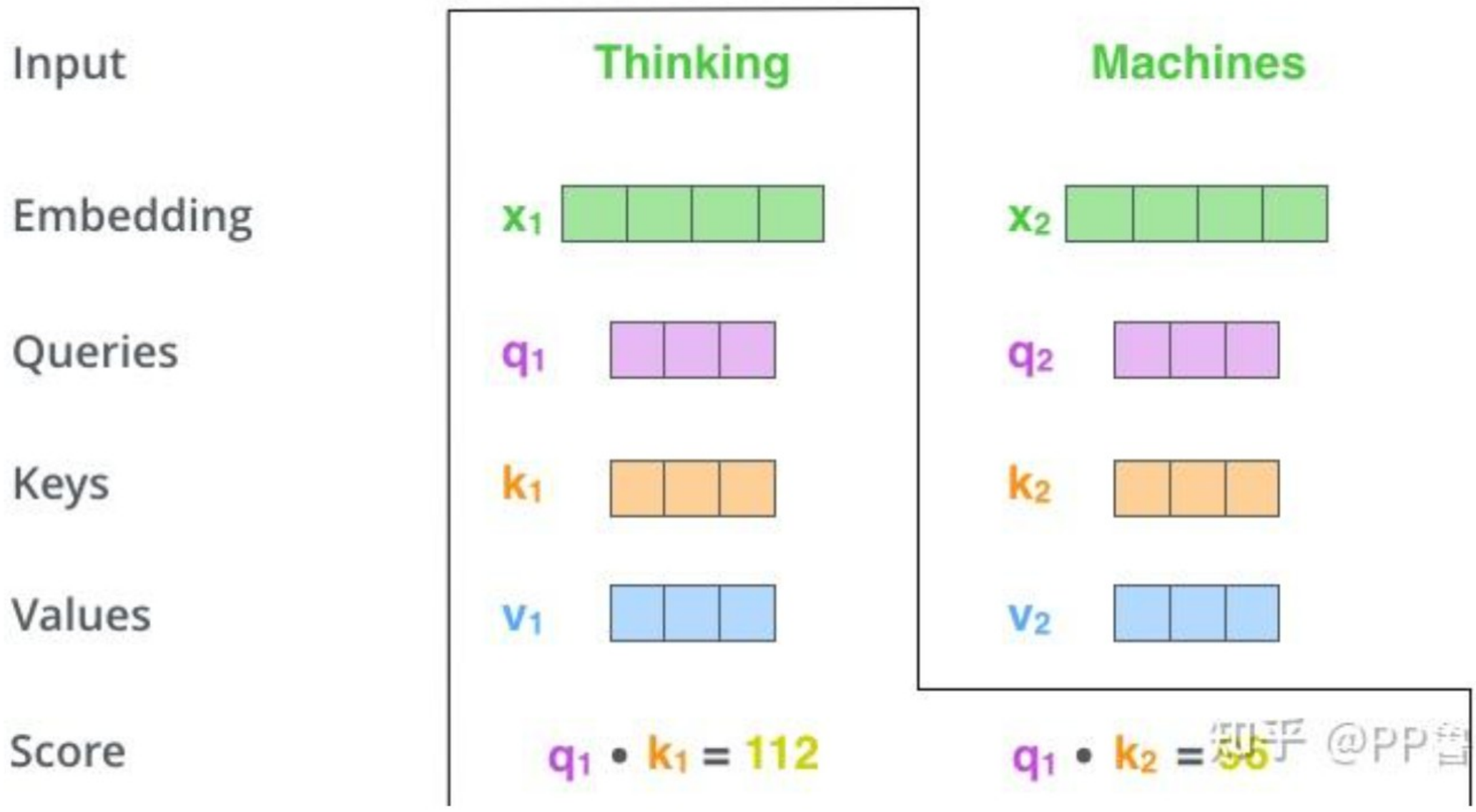
我们用国外博主Transformer详解博文[^2]中的例子来将上述计算串联起来解释。

输入为词向量矩阵X，每个词为矩阵中的一行，经过与W进行矩阵乘法，首先生成Q、K和V。 `q1 = X1 * WQ` ， `q1` 为 `Q` 矩阵中的行向量， `k1` 等与之类似。



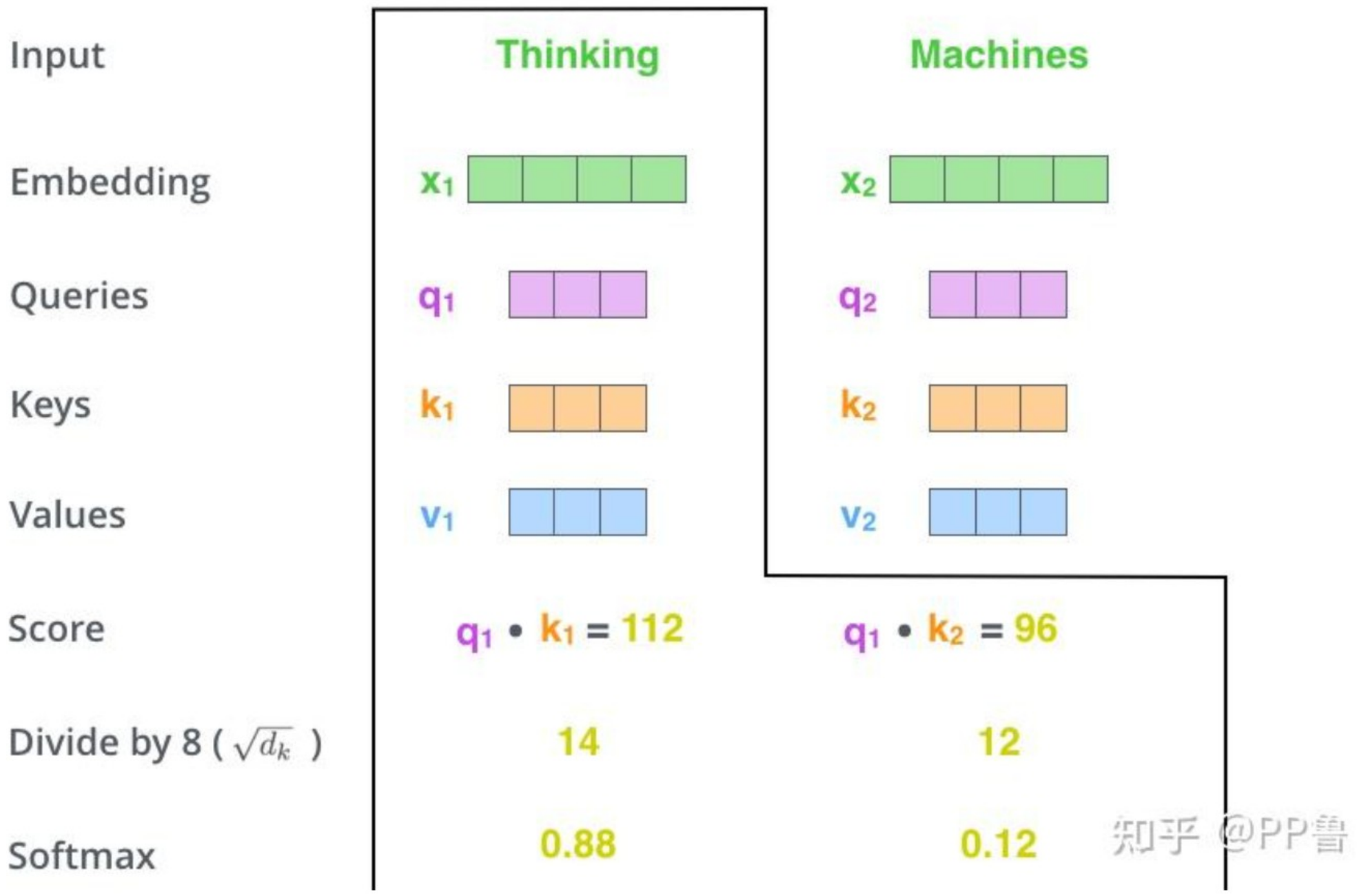
从词向量到Q、K、V

第二步是进行$QK^\top$计算，得到相似度。



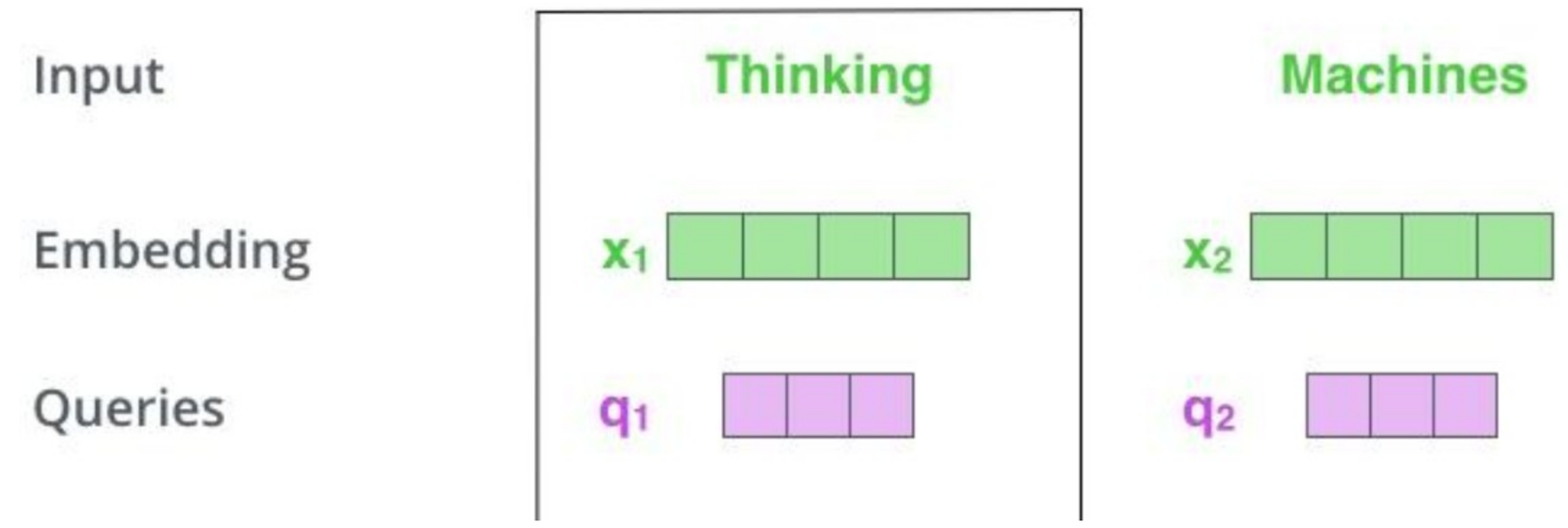
Q与K相乘，得到相似度

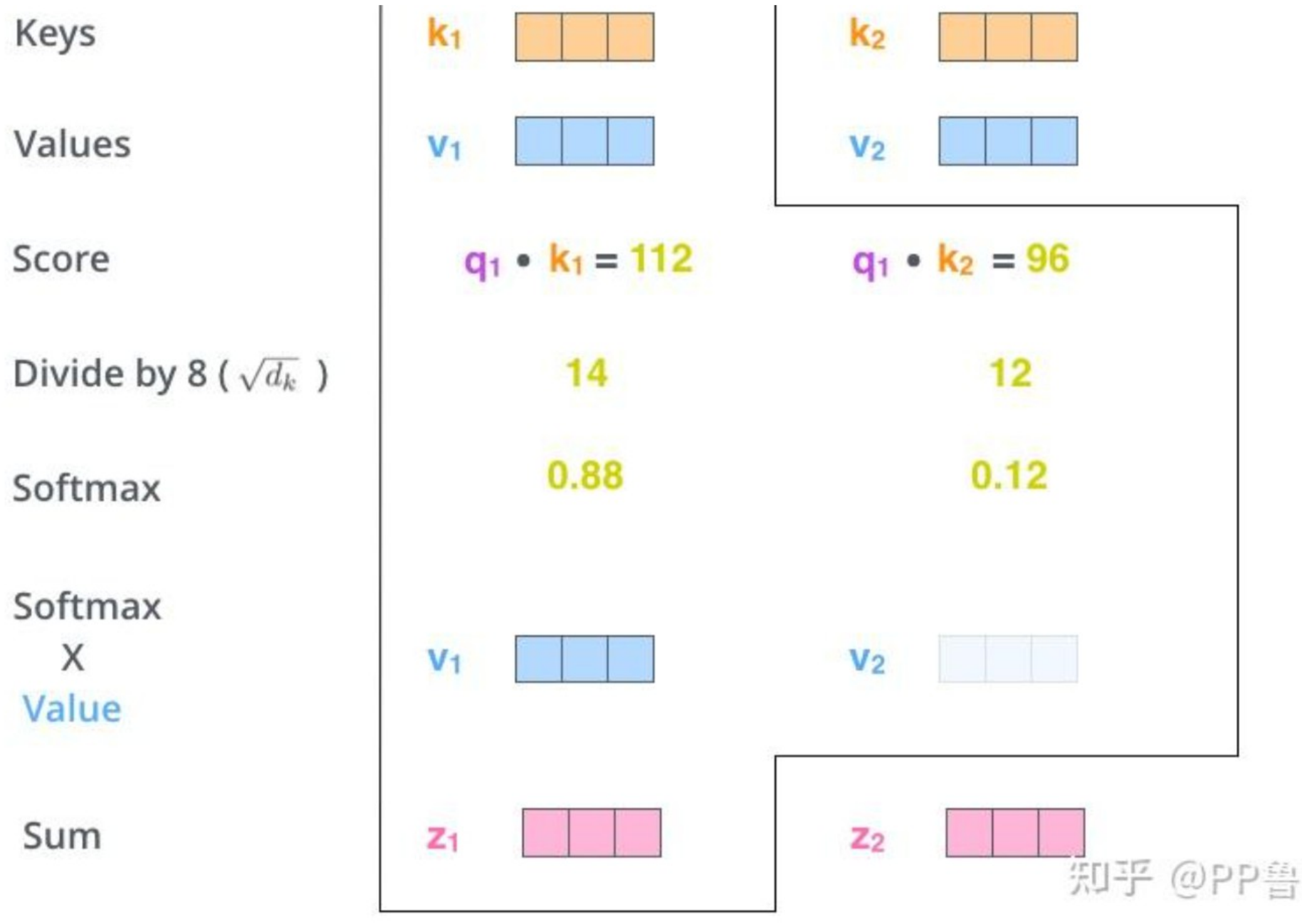
第三步，将刚得到的相似度除以$\sqrt{d_k}$，再进行Softmax。经过Softmax的归一化后，每个值是一个大于0小于1的权重系数✦，且总和为0，这个结果可以被理解成一个权重矩阵。



Scale & Softmax

第四步是使用刚得到的权重矩阵，与V相乘，计算加权求和。

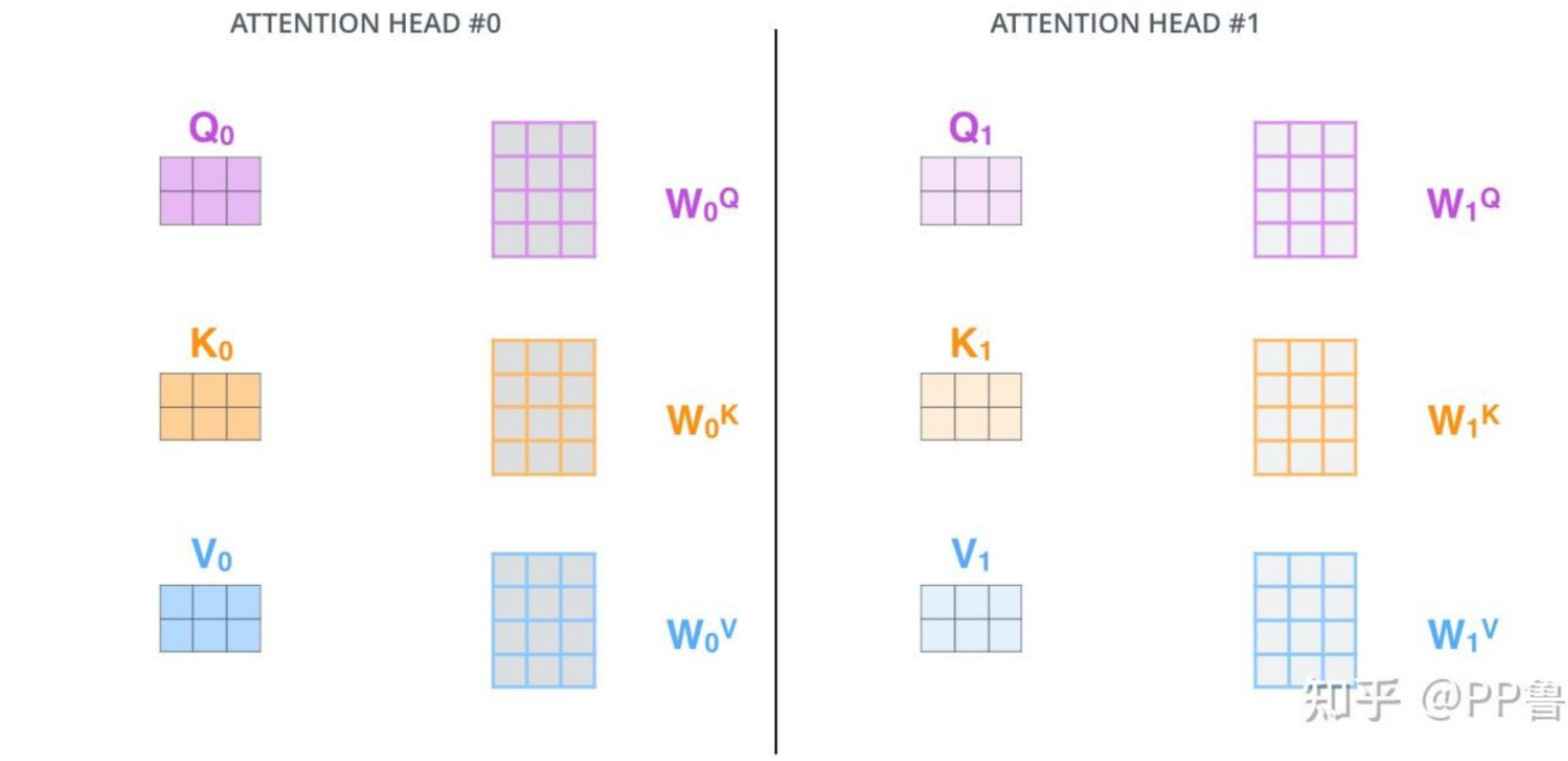




使用权重矩阵与V相乘，得到加权求和

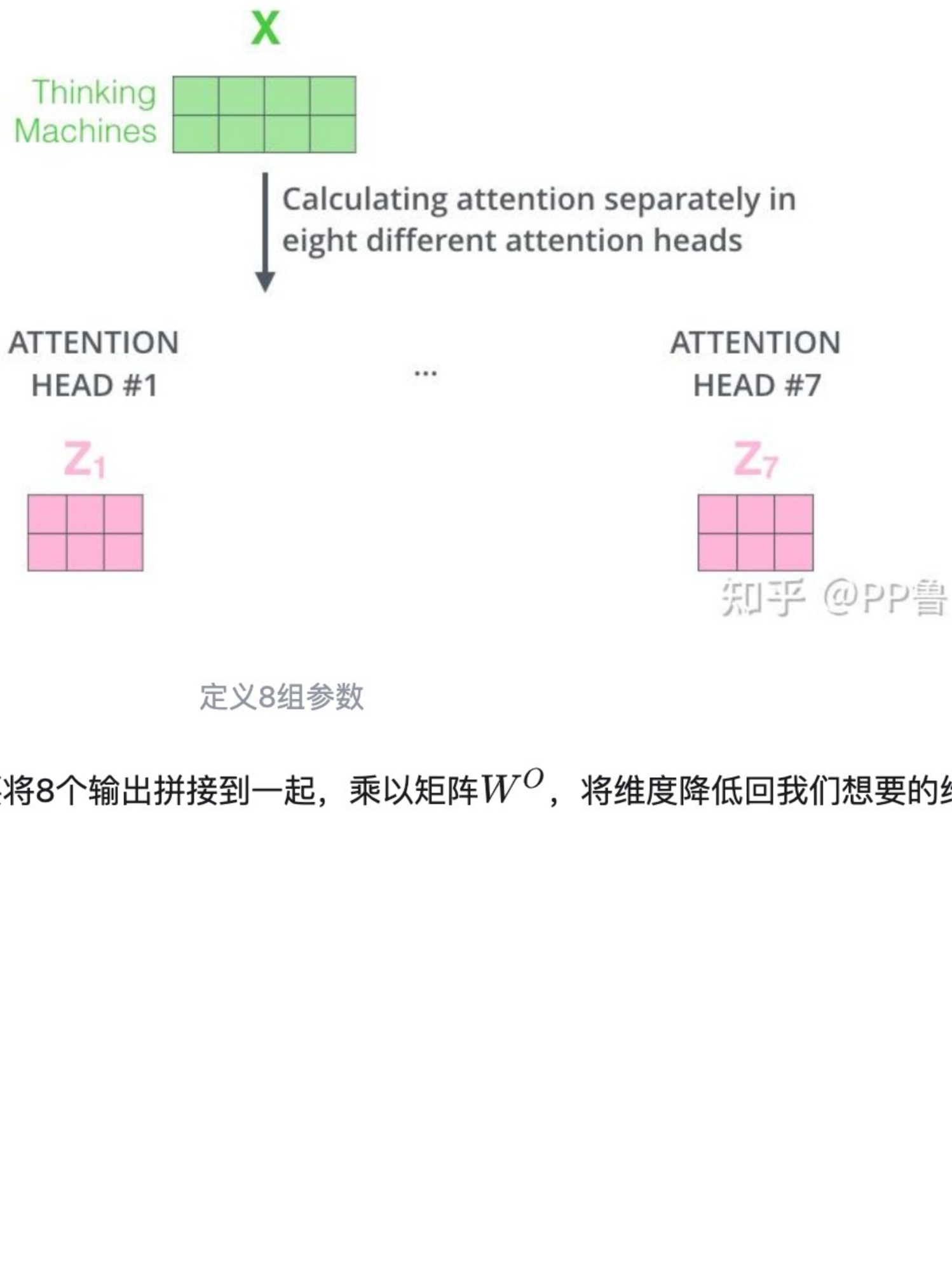
多头注意力

为了增强拟合性能，Transformer对Attention继续扩展，提出了多头注意力（Multiple Head Attention）。刚才我们已经理解了，Q、K、V是输入X与W^Q、W^K和W^V分别相乘得到的，W^Q、W^K和W^V是可训练的参数矩阵。现在，对于同样的输入X，我们定义多组不同的W^Q、W^K、W^V，比如W_0^Q、W_0^K、W_0^V，W_1^Q、W_1^K和W_1^V，每组分别计算生成不同的Q、K、V，最后学习到不同的参数。



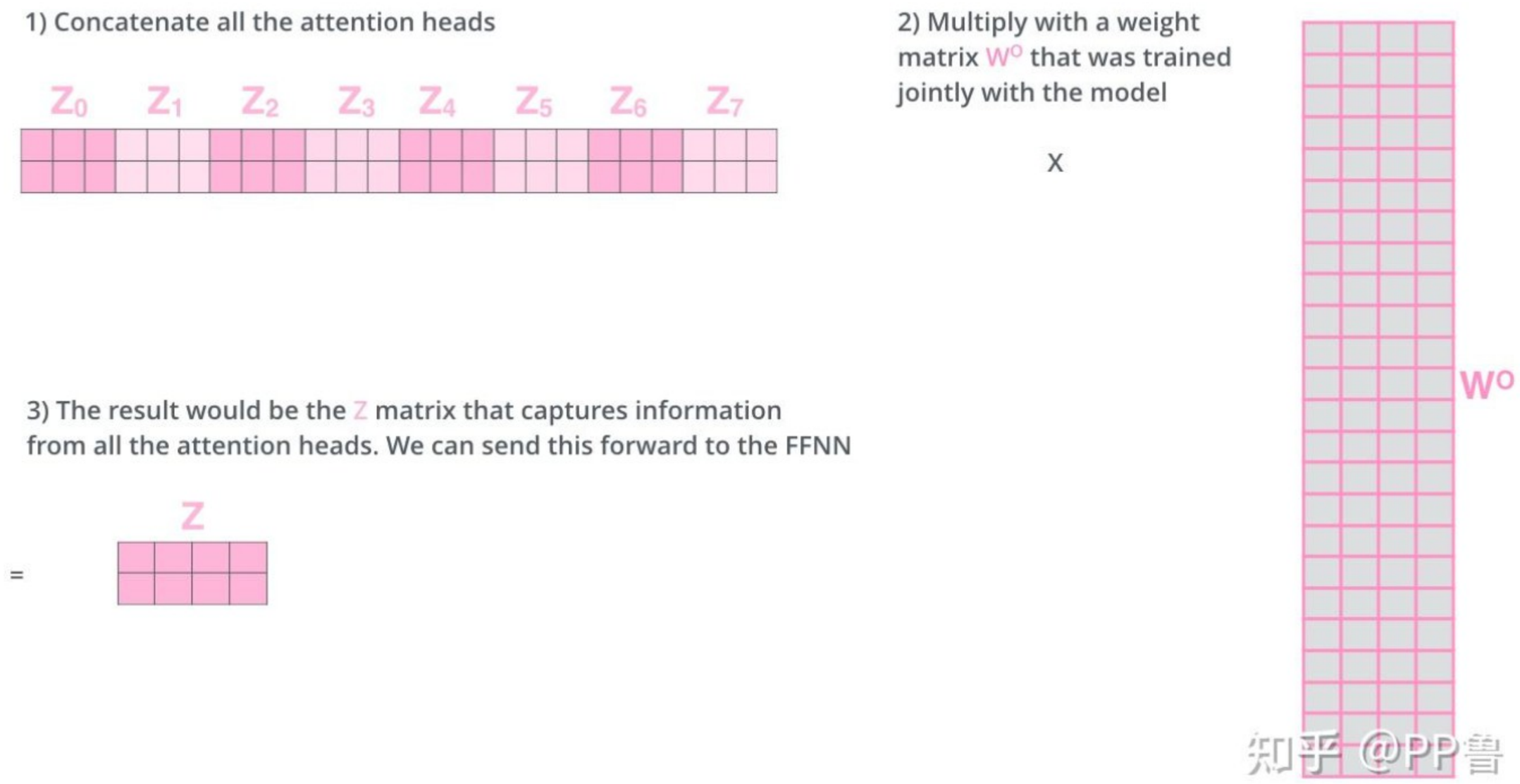
定义多组W，生成多组Q、K、V

比如我们定义8组参数，同样的输入X，将得到8个不同的输出Z_0到Z_7。



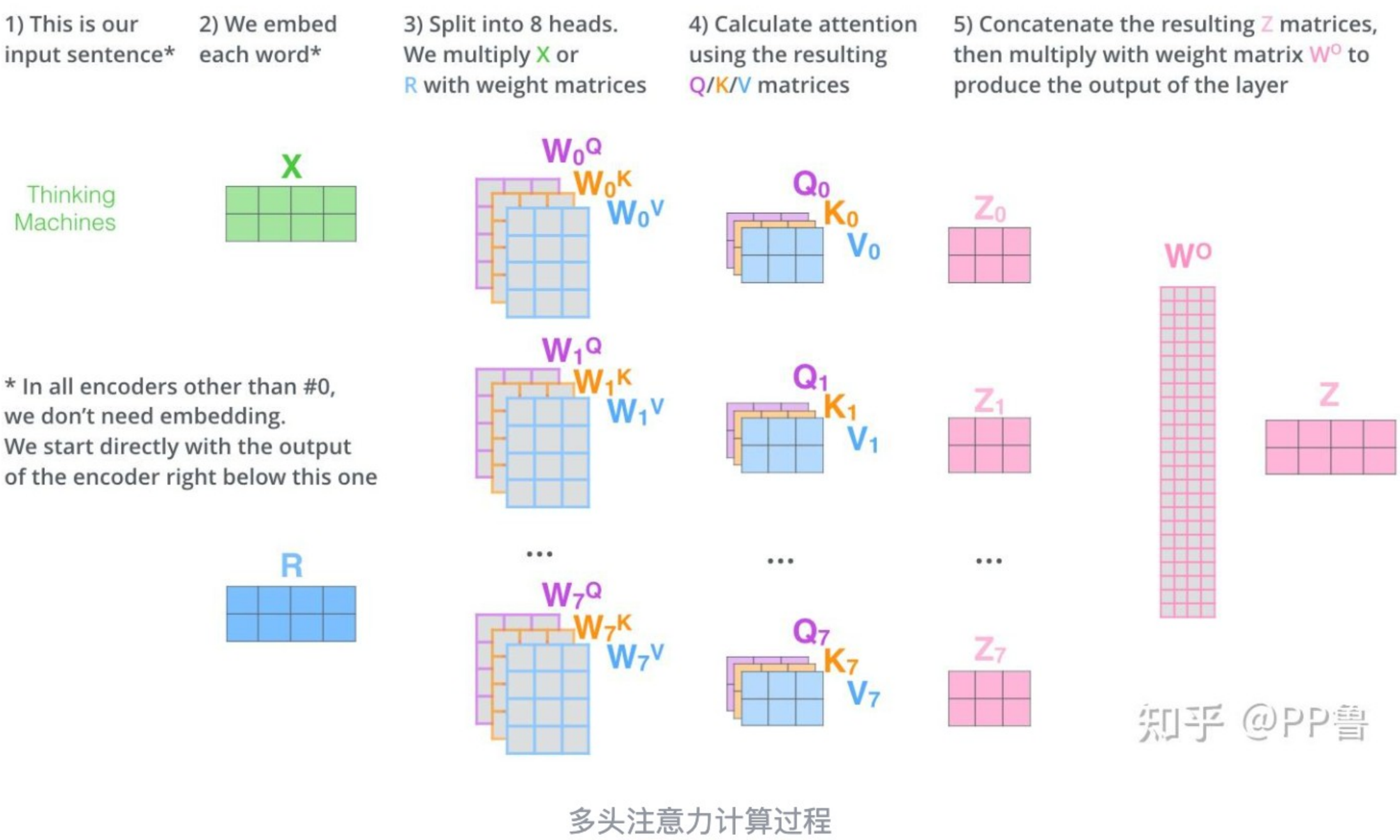
定义8组参数

在输出到下一层前，我们需要将8个输出拼接到一起，乘以矩阵W^O，将维度降低回我们想要的维度。



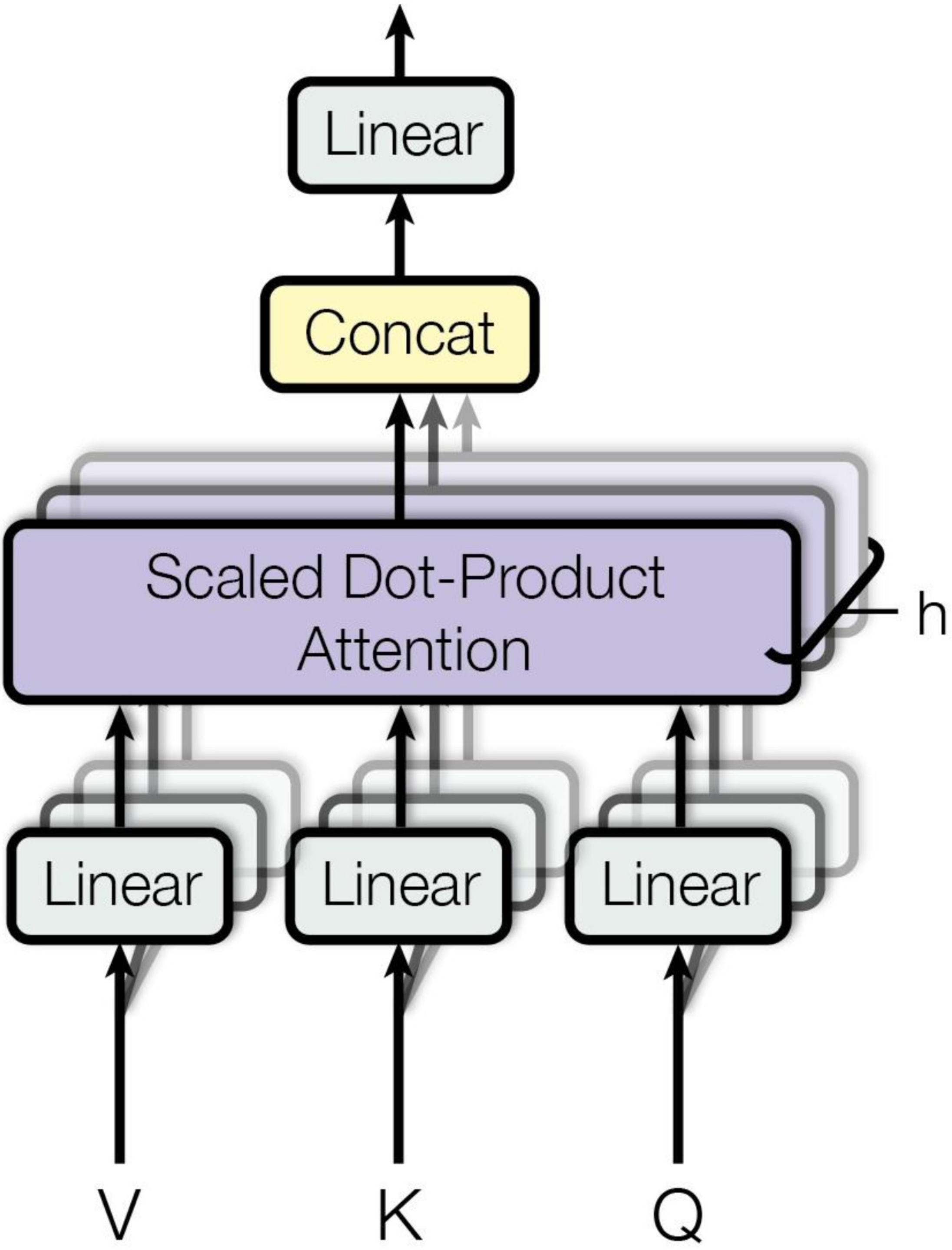
将多组输出拼接后乘以矩阵Wo以降低维度

多头注意力的计算过程如下图所示。对于下图中的第2）步，当前为第一层时，直接对输入词进行编码，生成词向量X；当前为后续层时，直接使用上一层输出。



多头注意力计算过程

再去观察Transformer论文中给出的多头注意力图示，似乎更容易理解了：



Transformer论文给出的多头注意力图示

[^1]: Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. 31st Conference on Neural Information Processing Systems 2017(NIPS 2017). Long Beach, CA, USA: 2017: 5998–6008.

[^2]: **jalammar.github.io/illu...**