回顾和复习常见神经网络结构的底层原理——**Transformer**。之前的科研项目没有涉及到Transformer的创新，没有细致的了解，今天查漏补缺一下。21号没整理总结完。

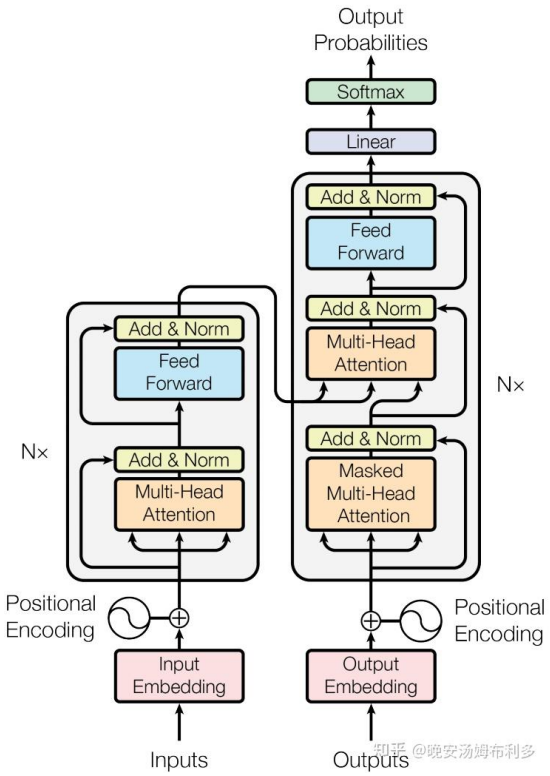
Day4：Transformer（上）

Transformer 是在 Attention is All You Need 论文中提出的，它是Google于2017年提出来算法框架，它使用了Self Attention的结构，取代了以往 NLP 任务中的 RNN 网络结构。

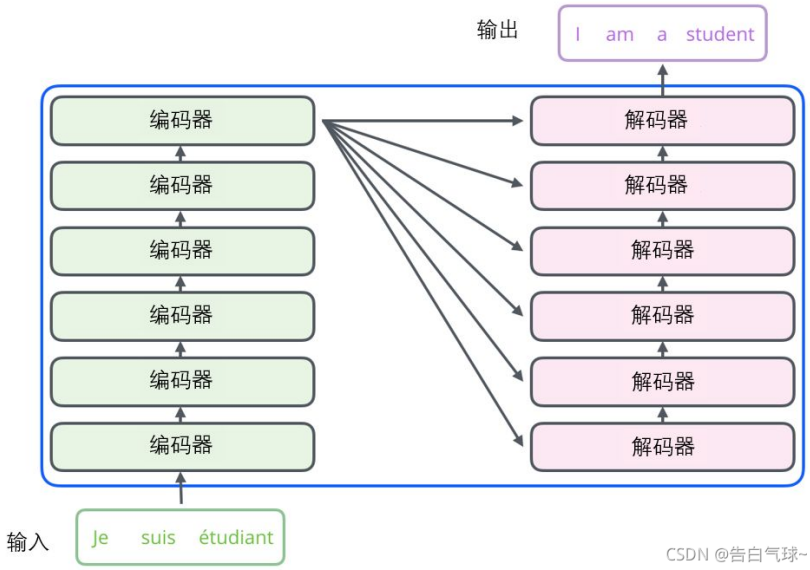
Transformer模型的其中一个优点，就是使得**模型训练过程能够并行计算**。在 RNN 中，每一个 time step 的计算都依赖于上一个 time step 的输出，这就使得所有的 time step 必须串行化，而在 Transformer 中，所有 time step 的数据，都是经过 Self Attention 计算，使得整个运算过程可以并行化计算。

1. Transformer的结构

Encoder block由6个encoder堆叠而成，Decoder block由6个decoder堆叠而成（`N=6`）。



编码器和解码器在结构上是相同的，但是它们之间并没有共享参数。



(1) 输入

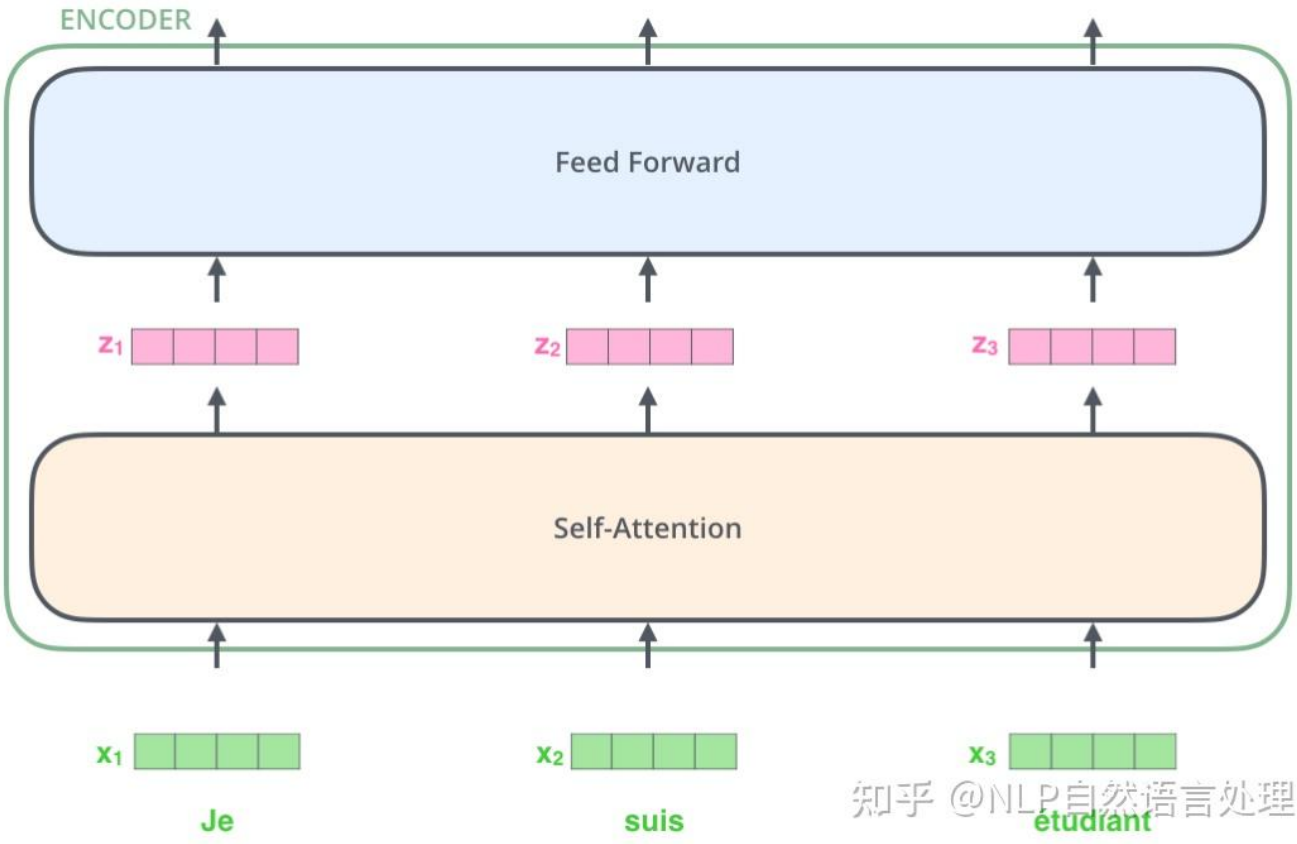
和通常的 NLP 任务一样，我们首先会使用词嵌入算法（embedding algorithm），将每个词转换为一个词向量。实际中向量一般是 256 或者 512 维。**为了简化起见，这里将每个词的转换为一个 4 维的词向量。**

在实际中，每个句子的长度不一样，我们会取一个适当的值，作为向量列表的长度。如果一个句子达不到这个长度，那么就填充全为 0 的词向量；如果句子超出这个长度，则做截断。句子长度是一个超参数，通常是训练集中的句子的最大长度，实验可以尝试不同长度的效果。编码器（Encoder）接收的输入都是一个**向量列表**。

本文以语音识别为例，输入为 `X` 和 `X_lens` 。 `X` 为输入特征，尺寸为 `(B,T,d)` ， `B` 为batch-size (一个训练轮次中模型接收的样本数量） ， `T` 为当前batch中各音频序列的最大长度， `d` 为特征维度。 `X_lens` 为 `B` 个样本各自的序列长度，尺寸为 `(B,)` 。

（2）编码器

每一个编码器在结构上都是一样的，但它们的权重参数是不同的。第一个编码器的输入是词向量，而后面的编码器的输入是上一个编码器的输入。



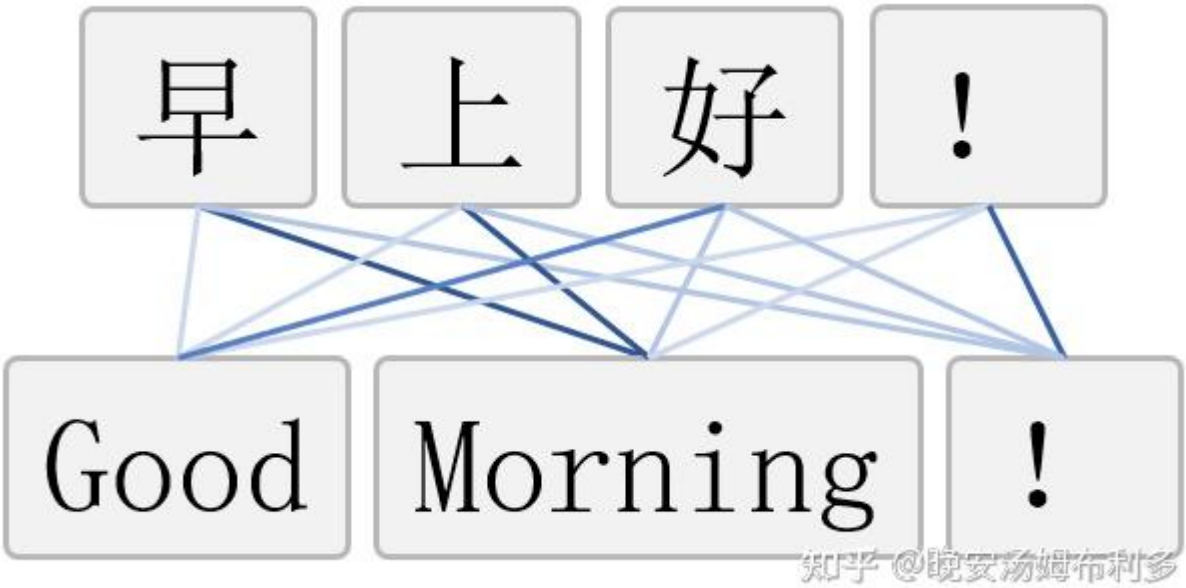
每一个编码器里面，可以分为 2 层：Self-Attention 层（Muti-Head Attention模块，是由多个Self-Attention组成）和Feed Forward Neural Network层。每个位置的词向量经过编码器都有自己单独的路径。具体来说，在 Self-Attention 层中，这些路径之间是有依赖关系的；而在 Feed Forward （前馈神经网络）层中，这些路径之间是没有依赖关系的。输入编码器的文本数据，首

先会经过一个 Self Attention 层，这个层处理一个词的时候，不仅会使用这个词本身的信息，也会使用句子中其他词的信息。

（3）Self-Attention

模型训练的时候却并不一定要求模型直接学习到模型输出和输入之间的注意力，比如CNN中，我们仅仅只是通过反向传播的方式来更新卷积核的权重，但是并没有任何和注意力直接有关的约束。而Transformer，则是**直接以注意力机制为主要构成模块的主干网络。**

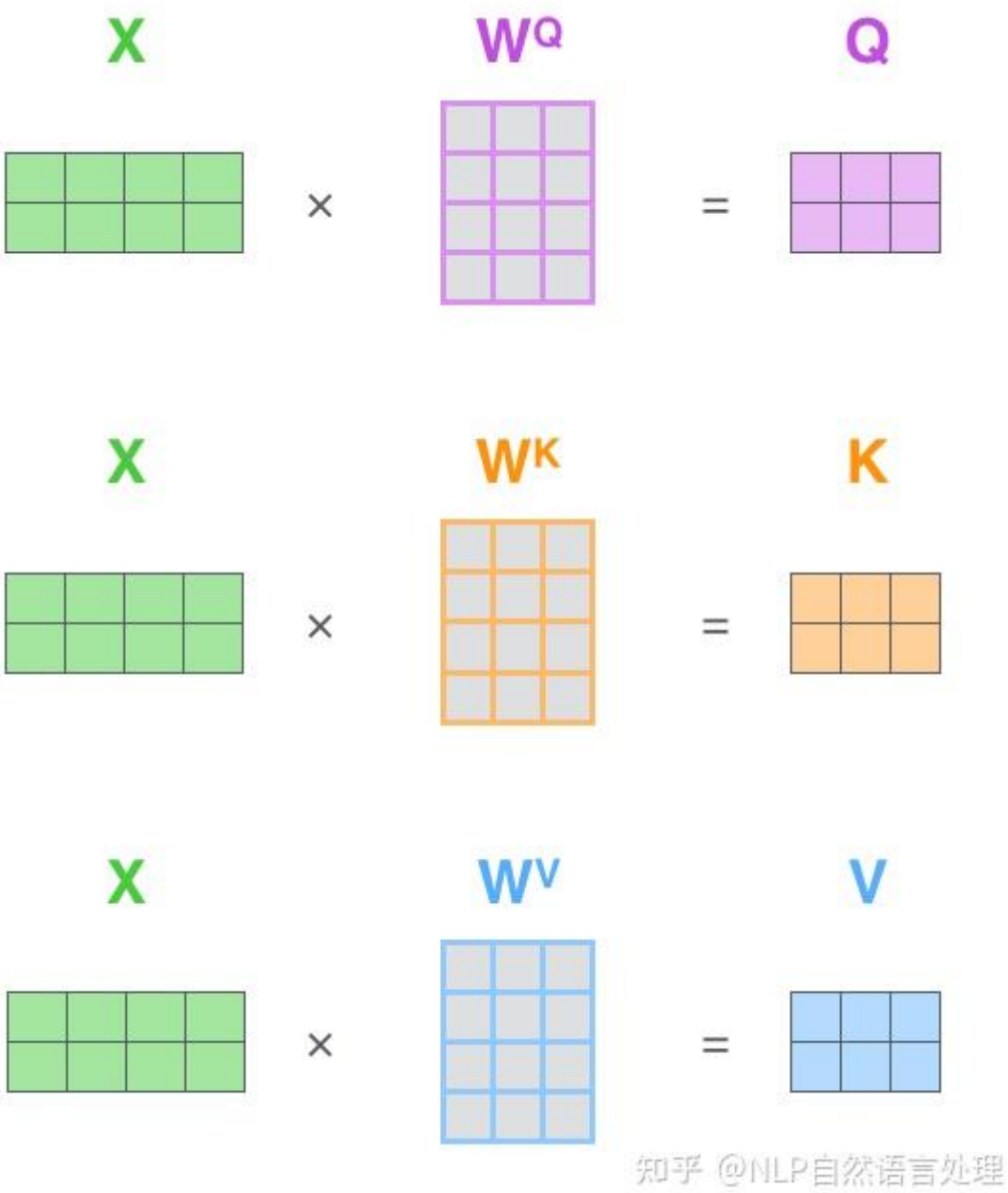
- 注意力的理解



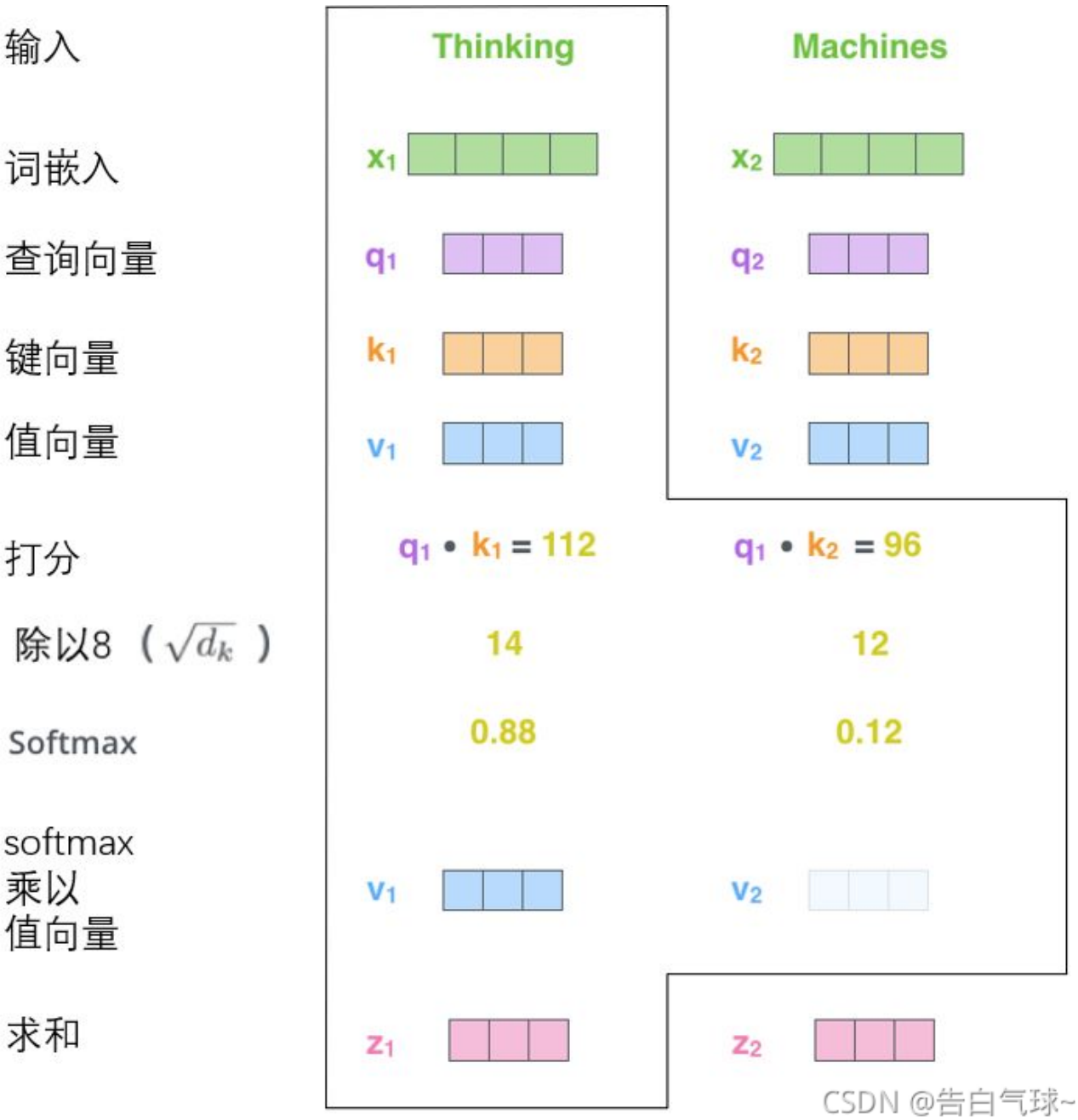
假如我们想把“早上好！”这句中文翻译成对应的英文“Good Morning！”。我们现在把“早上好！”作为模型输入，“Good Morning！”。具体来讲，“Good”和“好”的关联性最强，和“早上”以及“！”的关联性较弱；“Morning”和“早上”的关联性最强，和“好”以及“！”的关联性较弱；“！”和“！”的关联性最强，和“早上”以及“好”的关联性较弱。

- Transformer的注意力机制

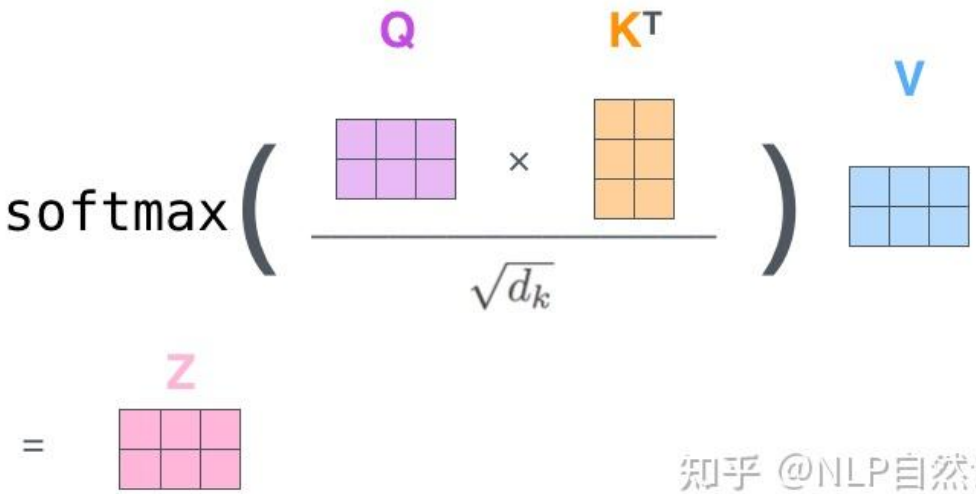
Transformer中用的注意力机制包括Query(**Q**)，Key (**K**)和Value(**V**)三个组成部分（学习过数据库的同学对这三个名词应该比较熟悉）。可以这样理解，V是我们手头已经有的所有资料，K可以作为规则库；Q是我们待查询的东西，我们希望把V中和Q有关的信息都找出来；而K是V这个知识库的钥匙（键值），V中每个位置的信息对应于一个K。对于V中的每个位置的信息而言，如果Q和对应钥匙K的匹配程度越高，那么就可以从该条信息中找到和Q更多的内容。Q(查询)、K(键值)、V(值)矩阵，是通过输入矩阵X和权重矩阵 W^Q,W^K,W^V 相乘得到的。



得到Q、K、V之后就可以计算出Self-Attention的输出，包括注意力打分$Q \times K^T$和将分数除以8(8是论文中使用的键向量的维数64的平方根，这会让梯度更稳定。这里也可以使用其它值，8只是默认值，这样做是为了防止内积过大。)，然后通过softmax传递结果（归一化指数函数，它将每一个元素的范围都压缩到（0，1）之间，并且所有元素的和为1）。

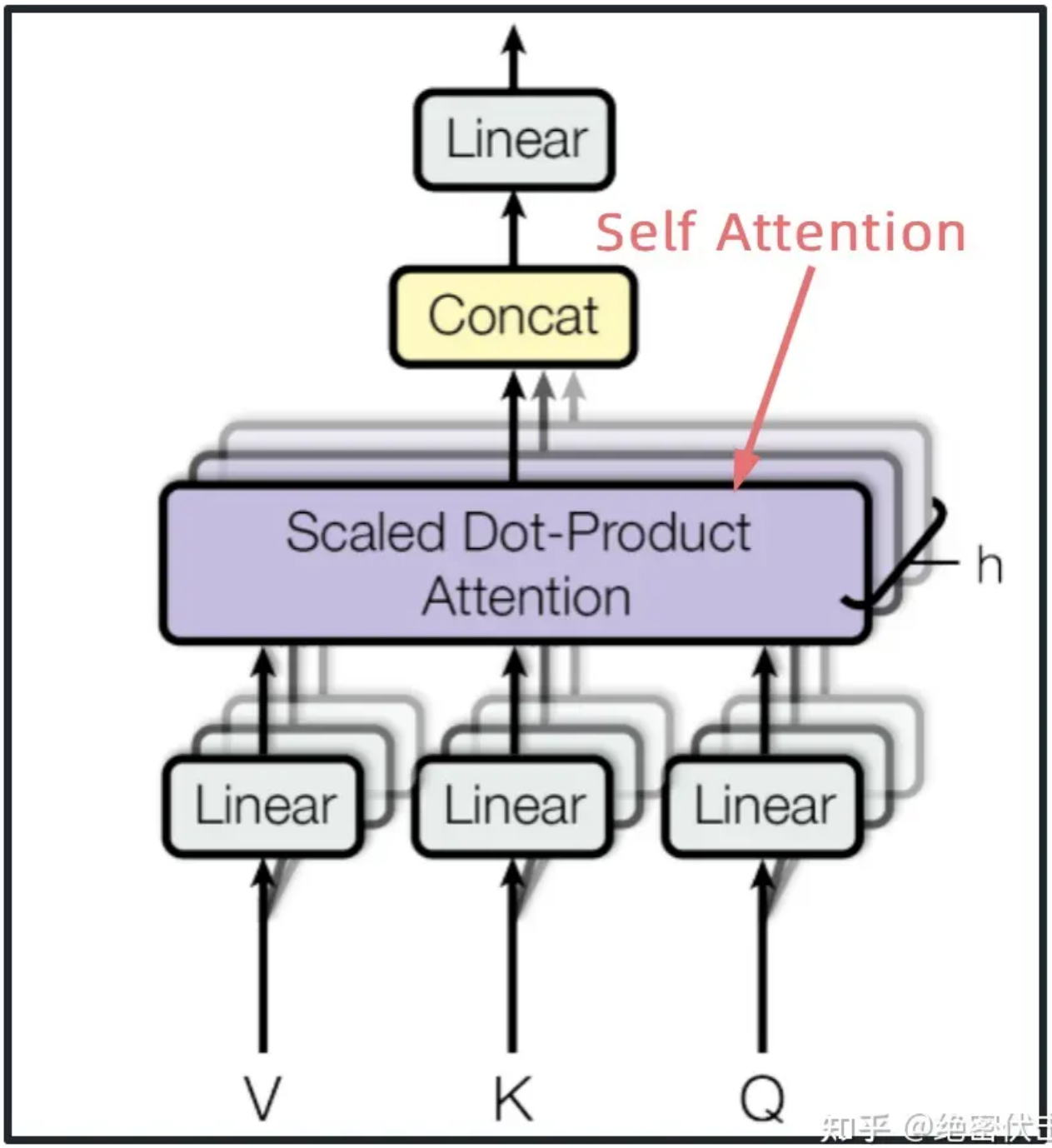


这个softmax分数决定了每个单词对编码当下位置（“Thinking”）的贡献。随着模型处理输入序列的每个单词，自注意力会关注整个输入序列的所有单词，帮助模型对本单词更好地进行编码。但这样每次只能计算一个位置的输出向量，在实际的代码实现中，Self Attention 的计算过程是使用矩阵来实现的，这样可以加速计算，一次就得到所有位置的输出向量：



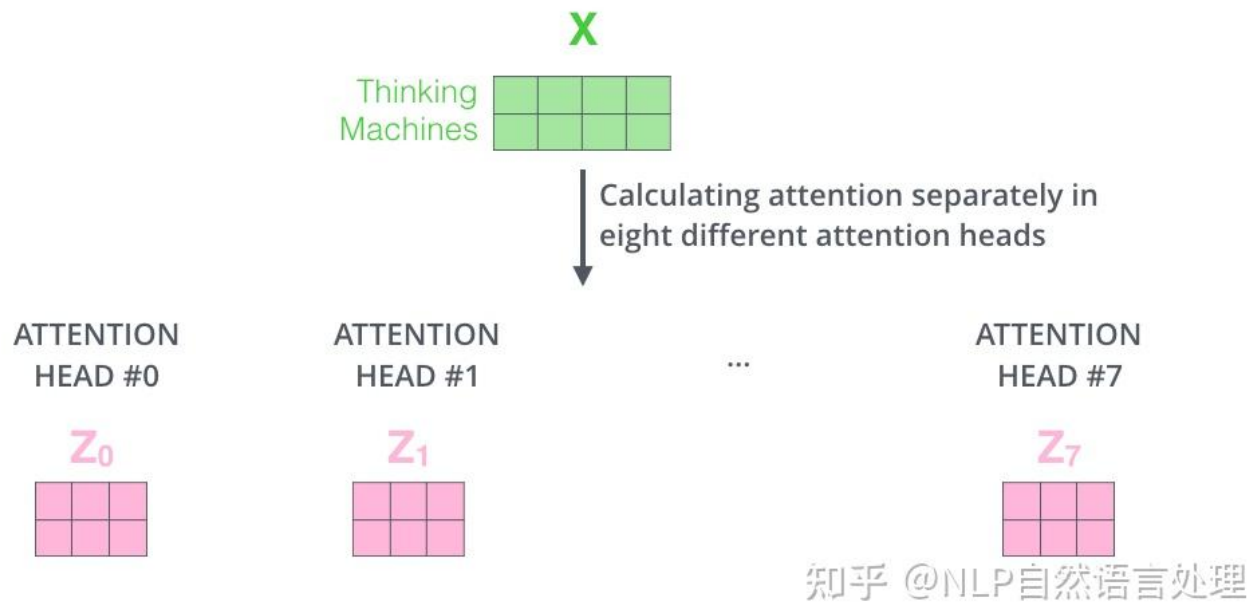
（4）多头注意力机制（multi-head attention）

Transformer 的论文通过增加多头注意力机制（一组注意力称为一个 attention head），进一步完善了 Self Attention 层。而Multi-Head Attention是由多个Self-Attention组合形成的，下图是论文中Multi-Head Attention的结构图。

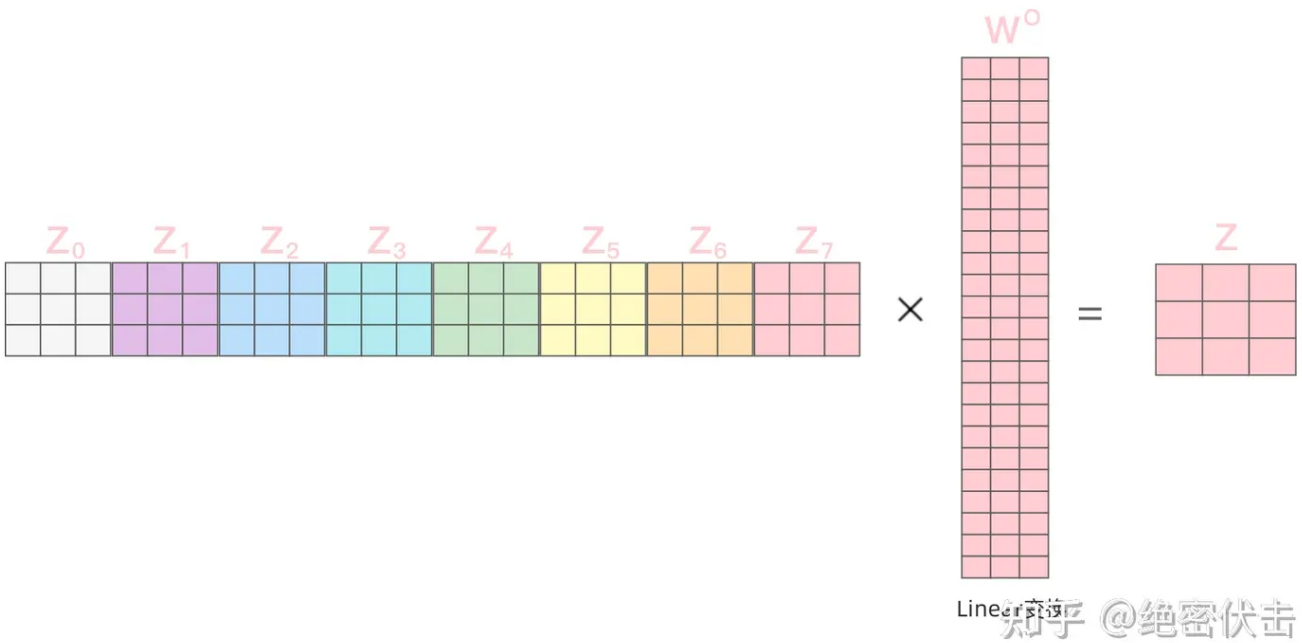


- 它扩展了模型关注不同位置的能力。在上面的例子中，第一个位置的输出 z1 包含了句子中其他每个位置的很小一部分信息，但 z1 可能主要是由第一个位置的信息决定的。当我们翻译句子：The animal didn't cross the street because **it** was too tired时，我们想让机器知道其中的it指代的是什么。这时，多头注意力机制会有帮助。
- 多头注意力机制赋予 attention 层多个"子表示空间"。下面我们会看到，多头注意力机制会有多组的权重矩阵（在 Transformer 的论文中，使用了 8 组注意力（attention heads）。因此，接下来我也是用 8 组注意力头 （attention heads））。每一组注意力的的权重矩阵都是随机初始化的。经过训练之后，每一组注意力可以看作是把输入的向量映射到一个"子表示空间"。

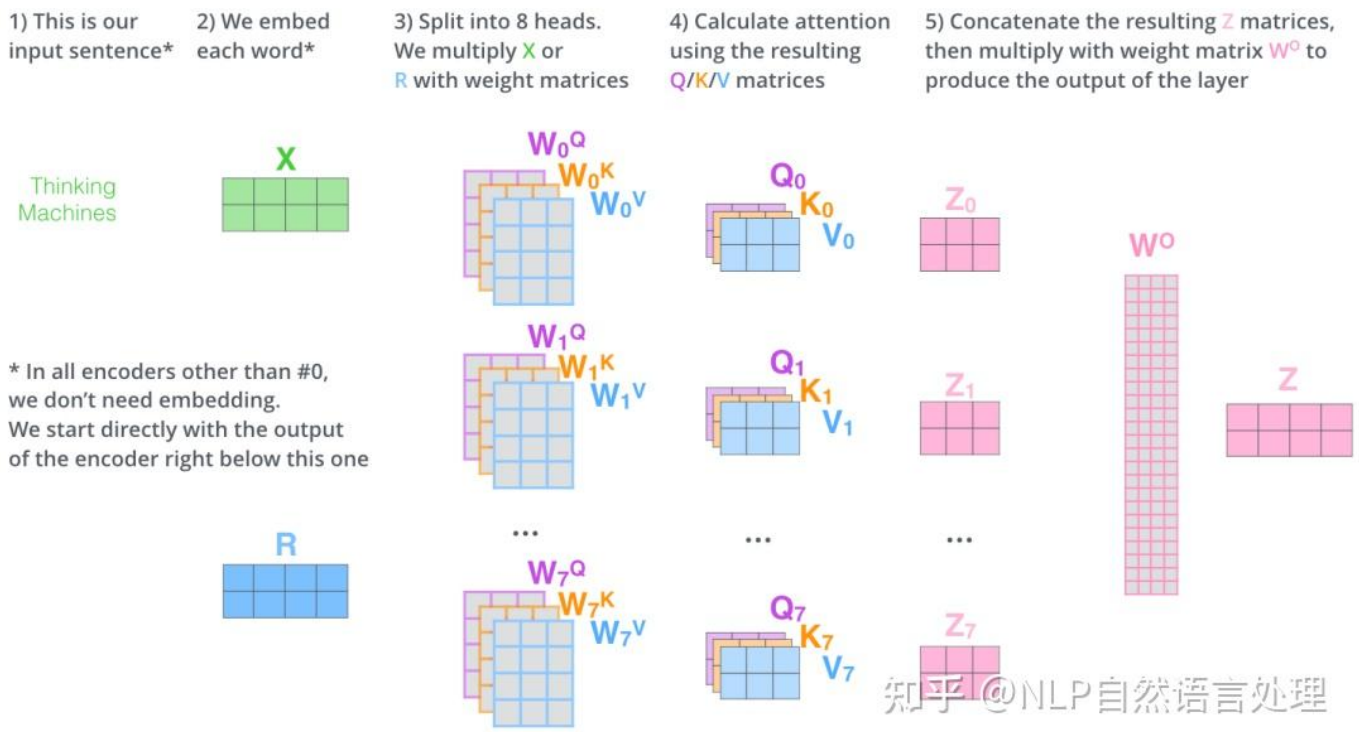
在多头注意力机制中，我们为每组注意力维护单独的W^Q, W^K, W^V权重矩阵。我们把每组 K, Q, V 计算得到每组的 Z 矩阵，就得到h个 Z 矩阵。下图是 h=8 的情况：



得到8个输出矩阵$Z_0 \sim Z_7$后，Multi-Head Attention将它们拼接在一起（Concat），然后传入一个Linear层（把拼接后的矩阵和W_0权重矩阵相乘），得到Multi-Head Attention最终的输出矩阵Z，该矩阵包含了所有 attention heads（注意力头） 的信息。这个矩阵会输入到 FFNN (Feed Forward Neural Network)层。



出现了相当多的矩阵。下面我把所有的内容都放到一张图中，这样你可以总揽全局，在这张图中看到所有的内容。



既然我们已经谈到了多头注意力，现在让我们重新回顾之前的翻译例子，看下当我们编码单词it时，不同的 attention heads （注意力头）关注的是什么部分。当我们编码单词"it"时，其中一个 attention head （注意力头）最关注的是"the animal"，另外一个 attention head 关注的是"tired"。因此在某种意义上，"it"在模型中的表示，融合了"the animal"和"tired"的部分表达。