手把手教你了解Transformer —Part 2

Tim在路上
北京邮电大学 计算机学硕士

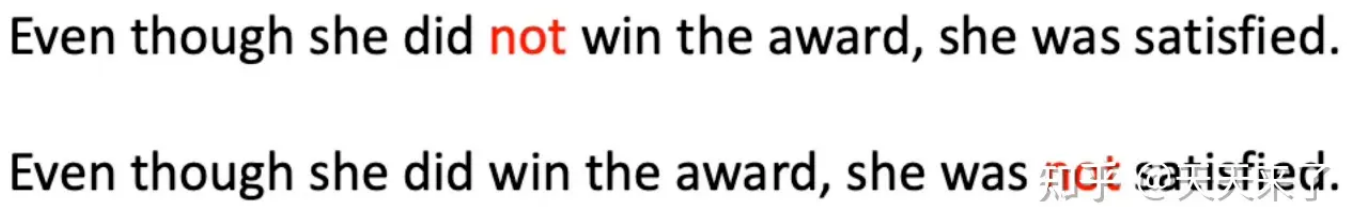
Transformer是由论文《Attention is All You Need》提出，它的并行化的能力使得许多神经网络模型可以更快速的和更宽泛的输入大量的数据。从最初论文里的文本翻译，到现在已经扩展和应用到了各个场景。

Transformer在其结构上包括Encoder与Decoder组成部分，在Part 1中，（手把手教你了解Transformer —Part 1），我们详细介绍了Transformer中Encoder最开始的部分—Inputs and Positional Encoding，它主要包括定义数据、计算词汇量、编码和生成单词的嵌入向量与位置的嵌入向量等部分。此外，我们简单介绍了单个自注意力机制的执行过程，包括Q，K，V的计算，以及self-Attention的输出等。

在本Part中，我们先来介绍和回顾下上述执行过程的意义，也就是为什么结构式这样的，这么做的原因是什么？

Positional Embeddings的意义是什么？

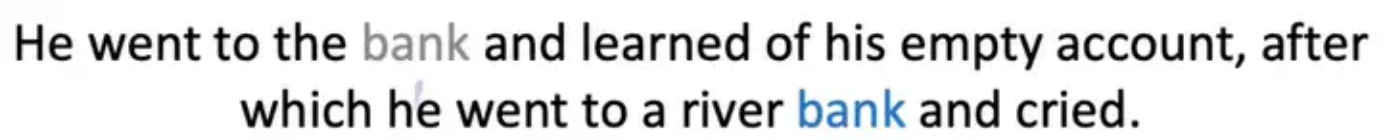
那么为什么说Positional Embeddings或者说文本翻译中单词的顺序是非常重要的呢？我们可以先来看下下图：



在上述语句中，单词“not”的位置不同，整个句子的情绪和语义就完全不同了。所以在训练过程中要将文本语句中单词的位置引入到整个训练过程中。

self-Attention模型的作用是什么？

其实注意力模型在2014年就已经被提出，但一般来说注意力模型计算的是输入序列与额外的上下文之间的联系，自注意力模型计算的是输入序列内不同位置之间的关联性。



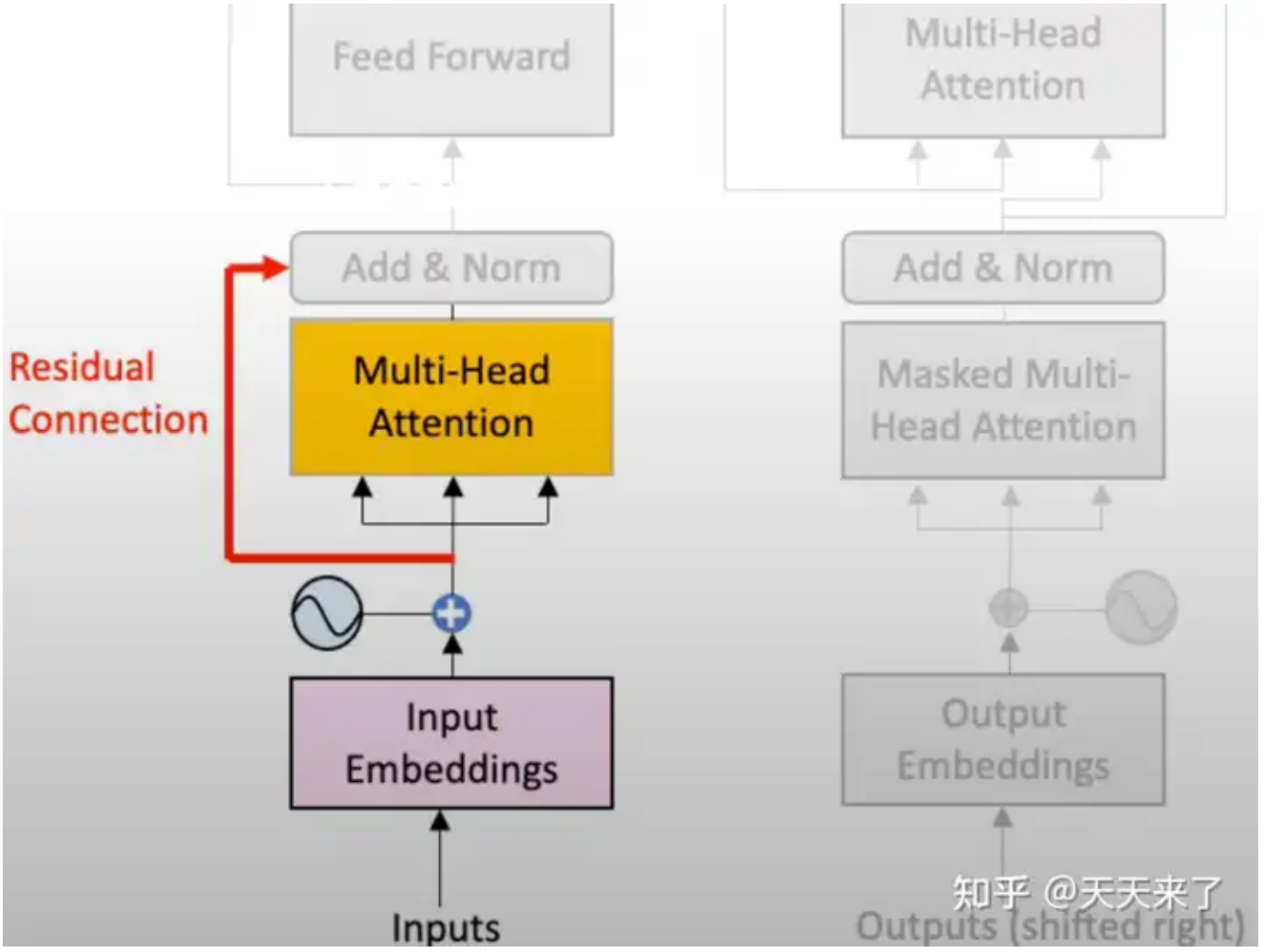
我们来看下上面的句子，在这个句子中有两个“bank”，但是其代表的意义是完全不同的，第一个“bank”代表的是银行，第二个 “bank”代表的是岸边。

那么模型算法是如何知道“bank”所代表的具体意思的呢？

在这里模型可以根据“his empty account” 来推断第一个”bank”为银行，根据“river”可以推断第二个“bank”为岸边。而self-attention指代的就是通过计算输入序列内不同位置之间的关联性得到权重，然后将这些权重应用于输入序列的每个位置，以获取位置自适应的表示。

接下来我们继续分析Transformer的网络结构。

步骤二（Residual Connections）



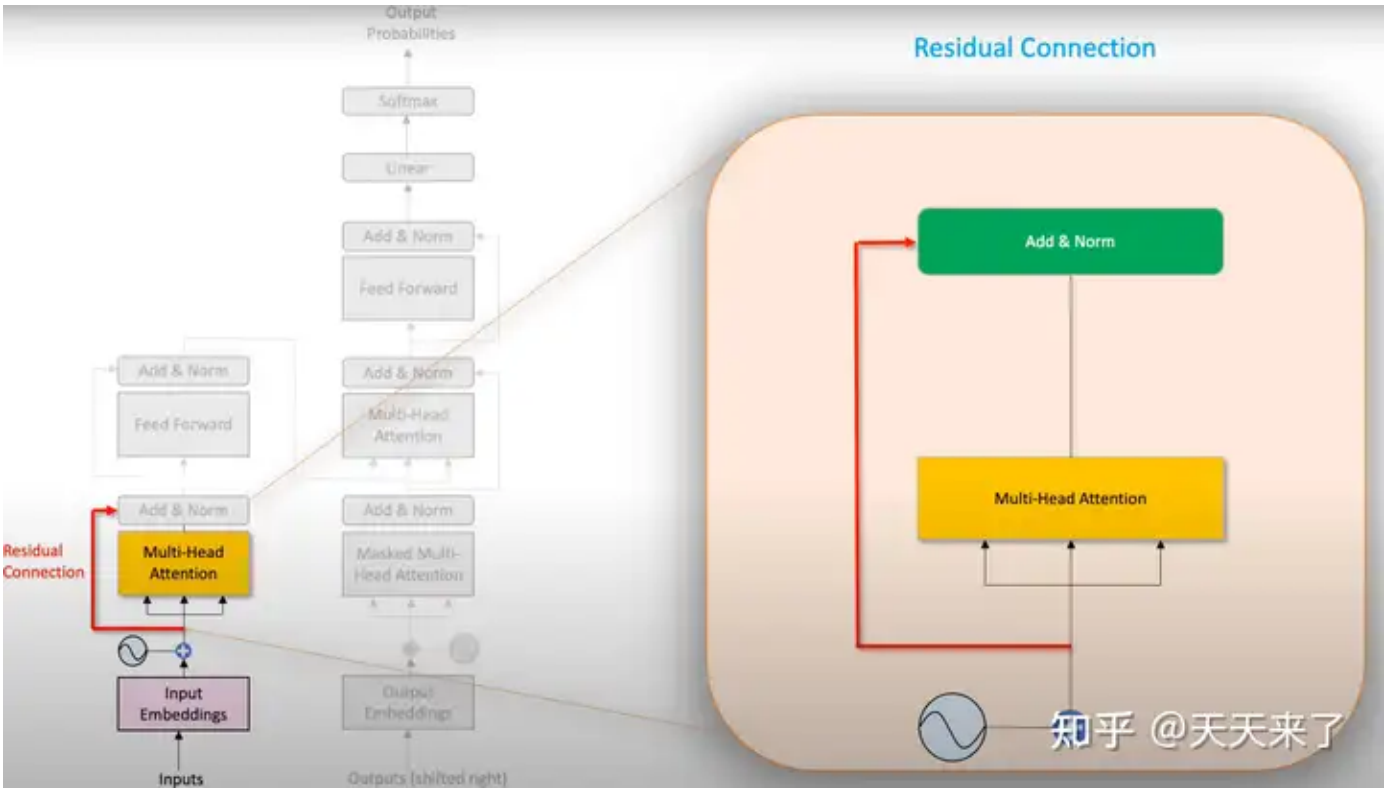
从上图中我们可以看到，模型在计算完Multi-Head Attention多头注意力网络后，将其输出与多头注意力的输入一同连接到了下一层。这个操作其实是类似于是一种残差连接（ResNet）。

Residual Connections 其主要解决了两个问题：

保留知识问题（knowledge preservation）
梯度消失问题（Vanishing Gradient Problem）

随着网络层数的增加，越深的网络层，其保留的原始信息越少，通过在网络中跨越多个层级直接传递信息，可以使得后续的网络可以提取有效的高层次信息，避免信息的丢失。

此外，梯度信息在反向传播过程中很容易出现消失的情况。这是由于梯度在多层网络中传递时会受到多次权重矩阵的连续乘法操作，导致梯度逐渐变小，使得网络参数无法得到有效更新。通过将前一层的特征直接添加到后续层级，避免了梯度消失问题，同时也有助于加速收敛和优化过程。



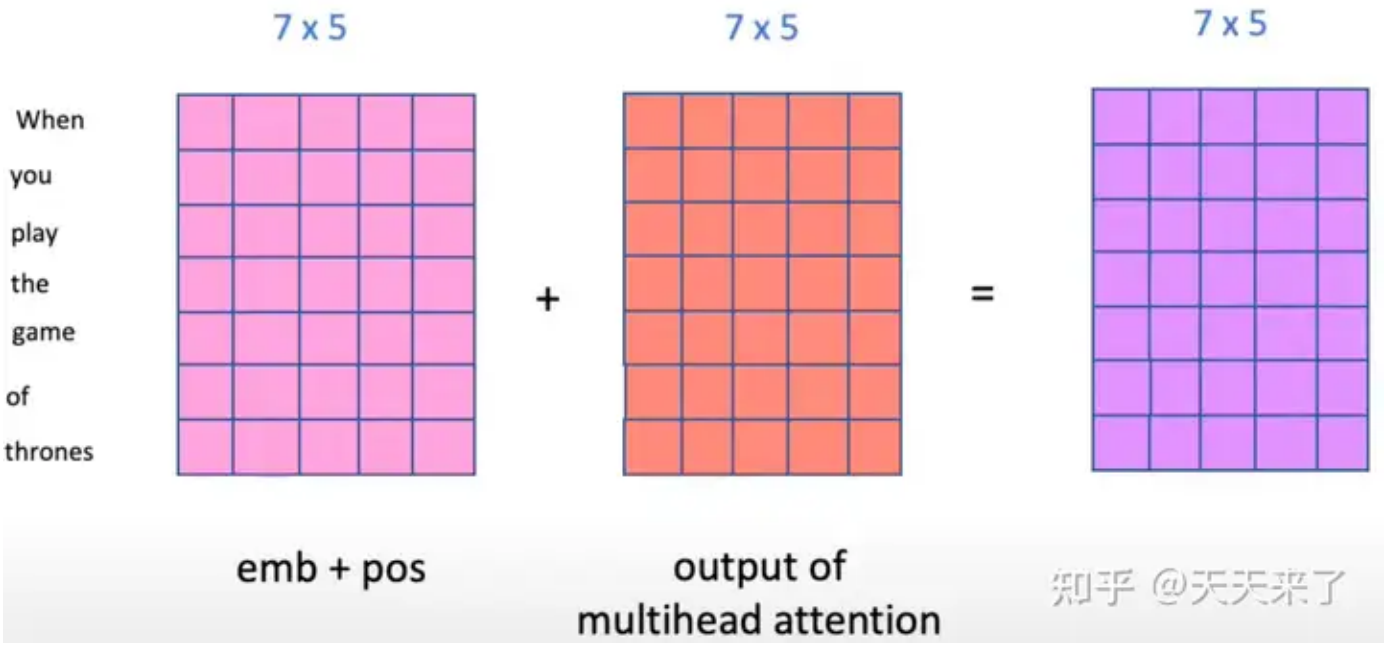
步骤三（Add & Norm）

Add & Norm 层由 Add 和 Norm 两部分组成，其计算公式如下：

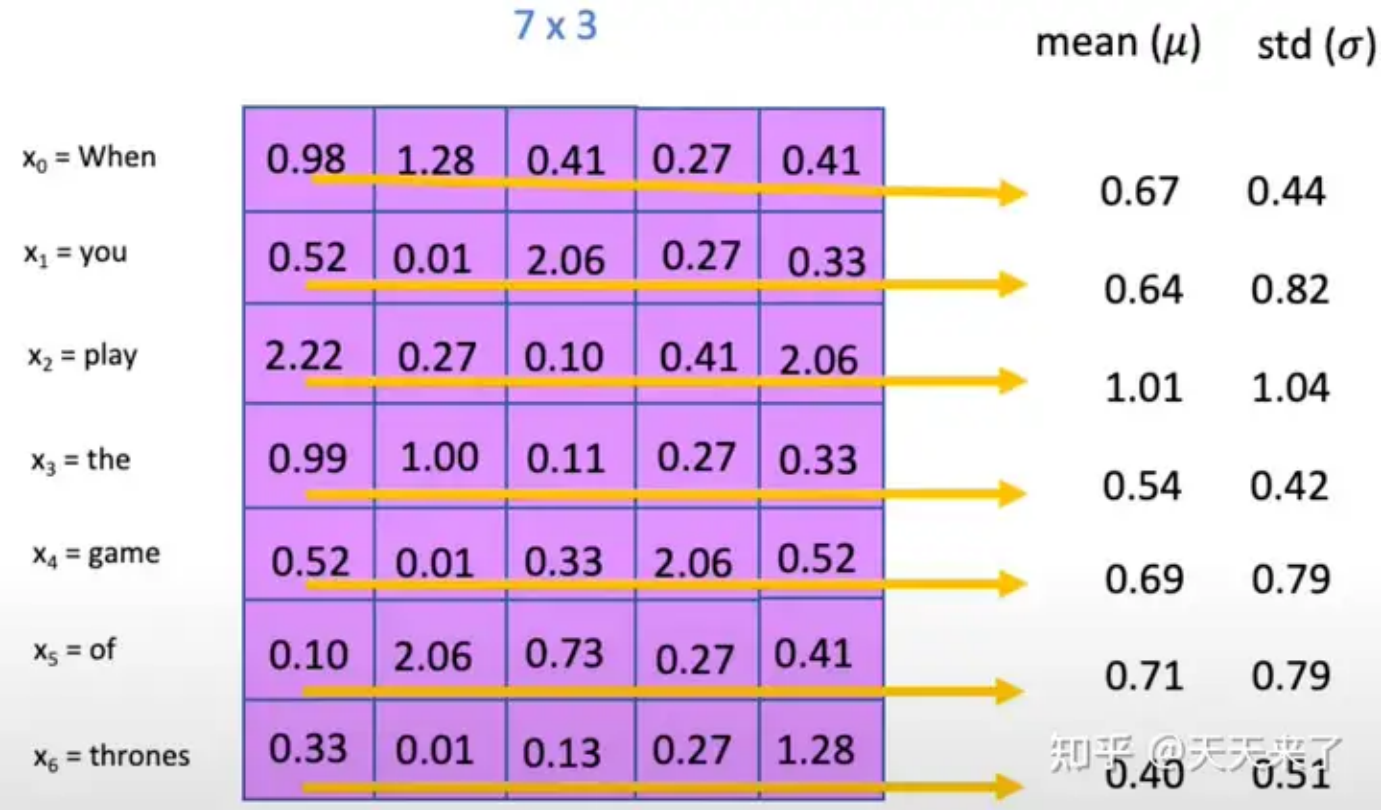
$$\mathrm{LayerNorm}\left(X + \mathrm{MultiHeadAttention}(X)\right)$$

$$\mathrm{LayerNorm}\left(X + \mathrm{FeedForward}(X)\right)$$

其中X代表的是Multi-Head Attention前的输入。



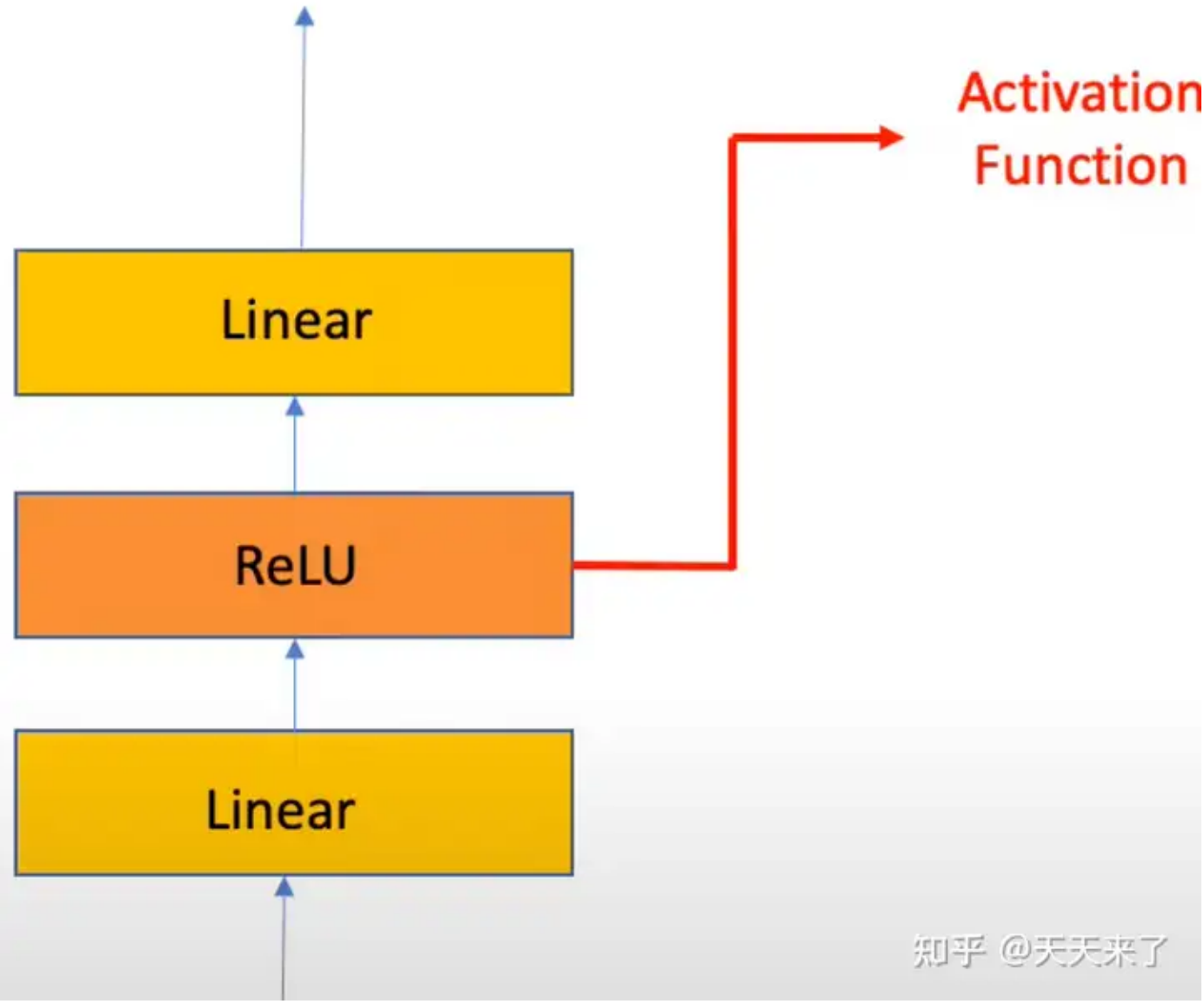
其中Add指的就是**X**+MultiHeadAttention(**X**)，它是残差连接的一部分。如上图所示，将emb + pos的嵌入向量矩阵与MultiHeadAttention输出进行相加，得到结果在输入到LayerNorm中。



Layer Normalization 会将每一层神经元的输入都转成均值方差都一样的，这样可以使得网络模型更快收敛以及更加稳定。

步骤四（Feed Forward）

Feed Forward 层比较简单，是一个两层的全连接层，第一层的激活函数为 Relu，第二层不使用激活函数。最后其输出再经过一层Add & Norm。

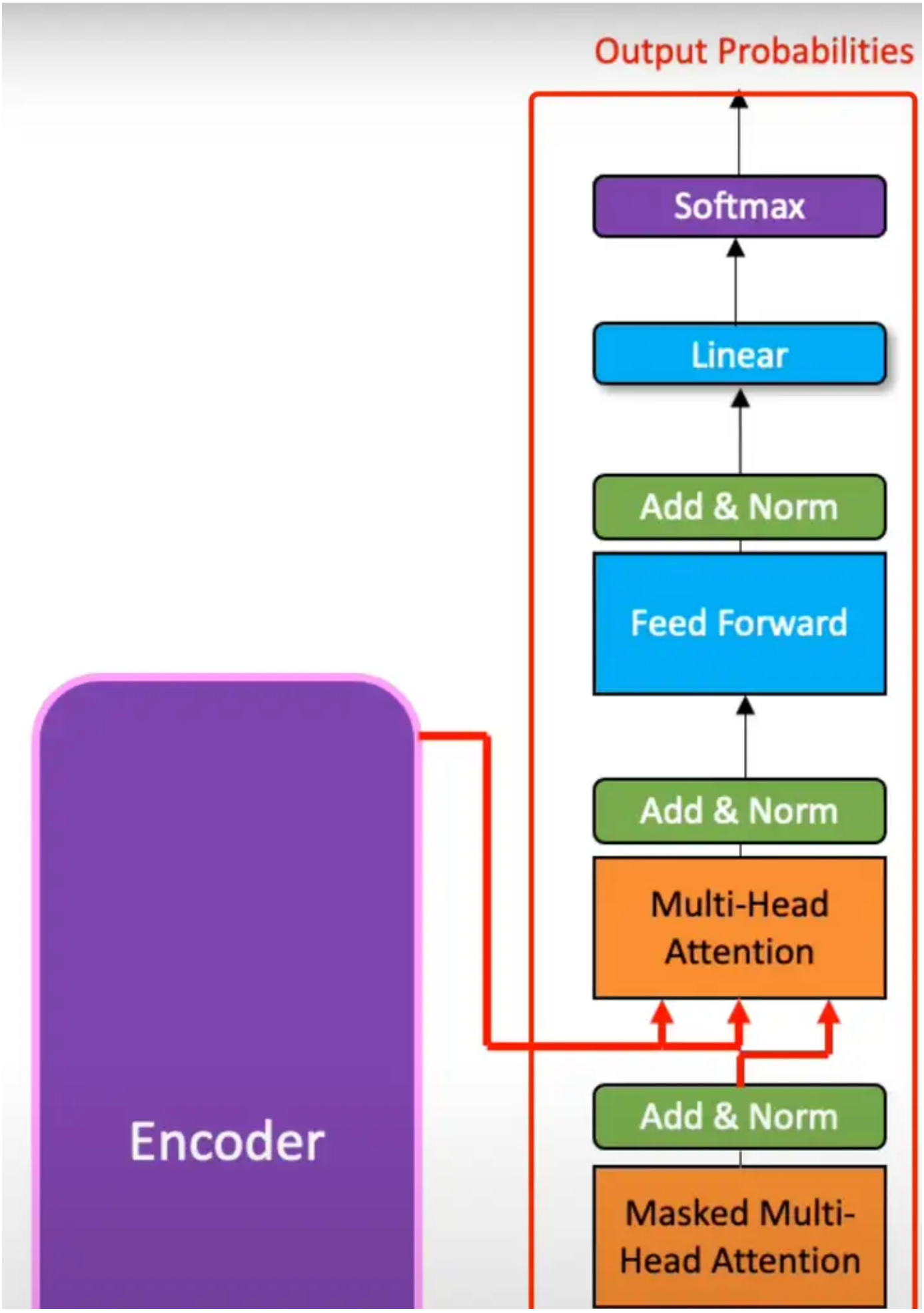


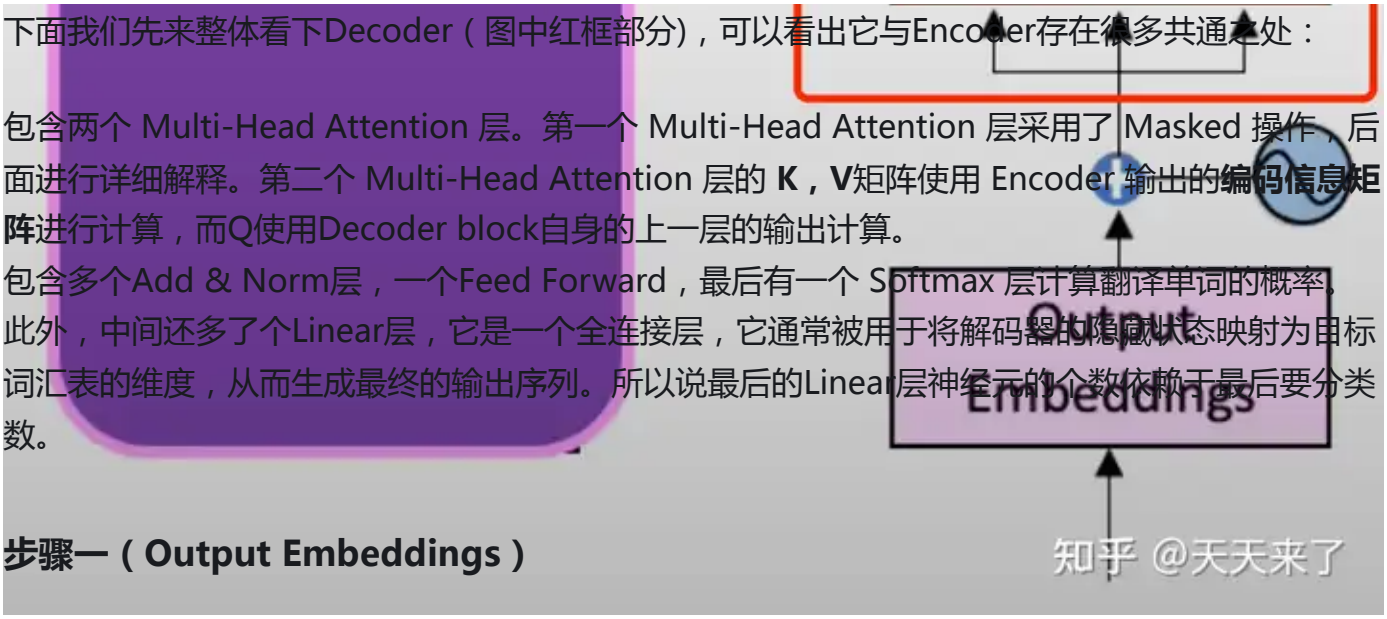
步骤五（组成 Encoder）

由上面的描述可知，一个Encode block是由Multi-Head Attention, Add & Norm, Feed Forward, Add & Norm组成一个 Encoder block。通过多个Encoder block可以组成一个Encoder。具体过程如下：

第一个 Encoder block 的输入为句子单词的表示向量矩阵，后续 Encoder block 的输入是前一个 Encoder block 的输出，最后一个 Encoder block 输出的矩阵就是**编码信息矩阵** ，这一矩阵后续会用到 Decoder 中。

3. 解码器（Decoder）


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Encoder



下面我们先来整体看下Decoder（图中红框部分），可以看出它与Encoder存在很多共通之处：

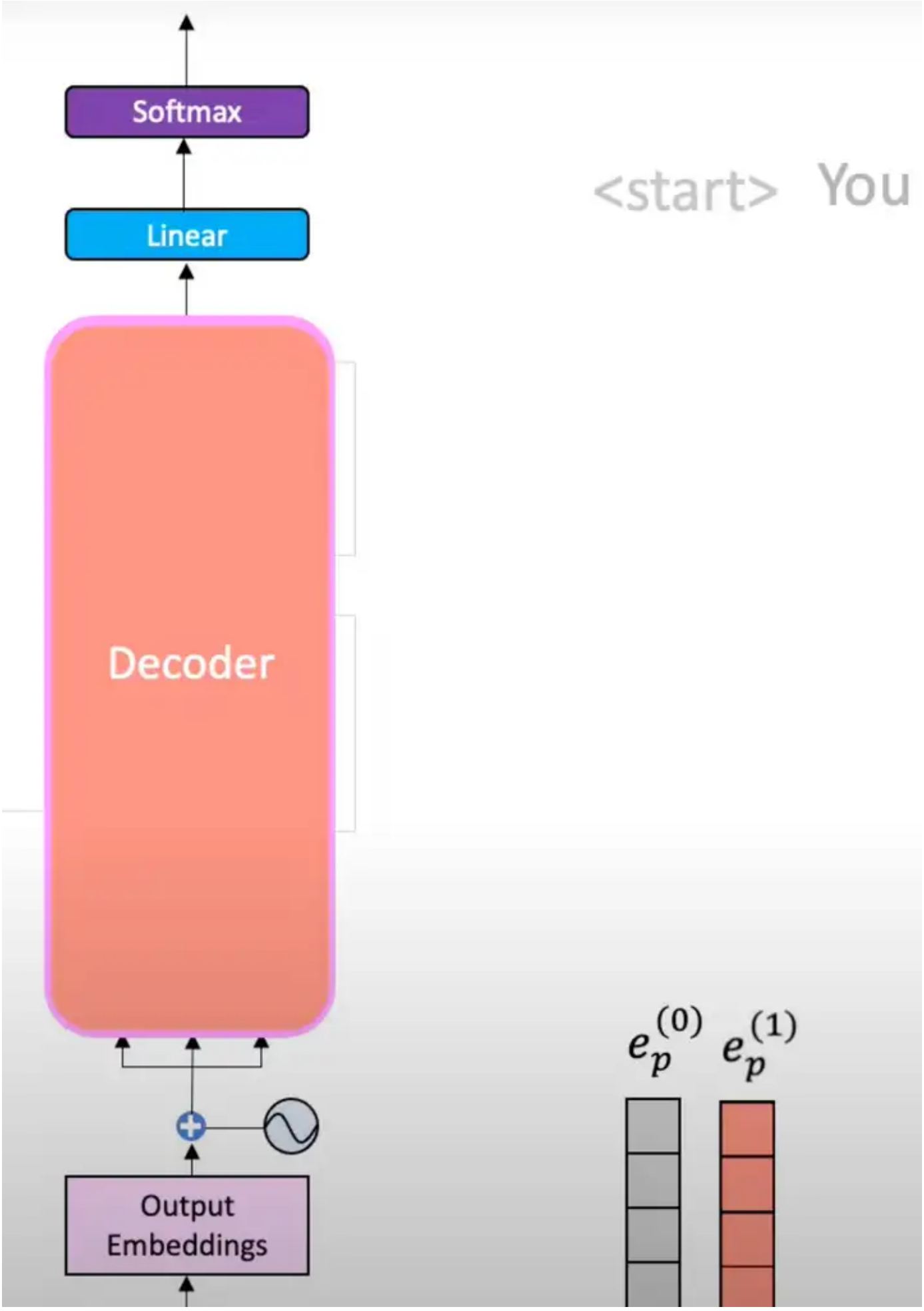
包含两个 Multi-Head Attention 层。第一个 Multi-Head Attention 层采用了 Masked 操作，后面进行详细解释。第二个 Multi-Head Attention 层的 **K，V**矩阵使用 Encoder 输出的**编码信息矩阵**进行计算，而Q使用Decoder block自身的上一层的输出计算。

包含多个Add & Norm层，一个Feed Forward，最后有一个 Softmax 层计算翻译单词的概率。

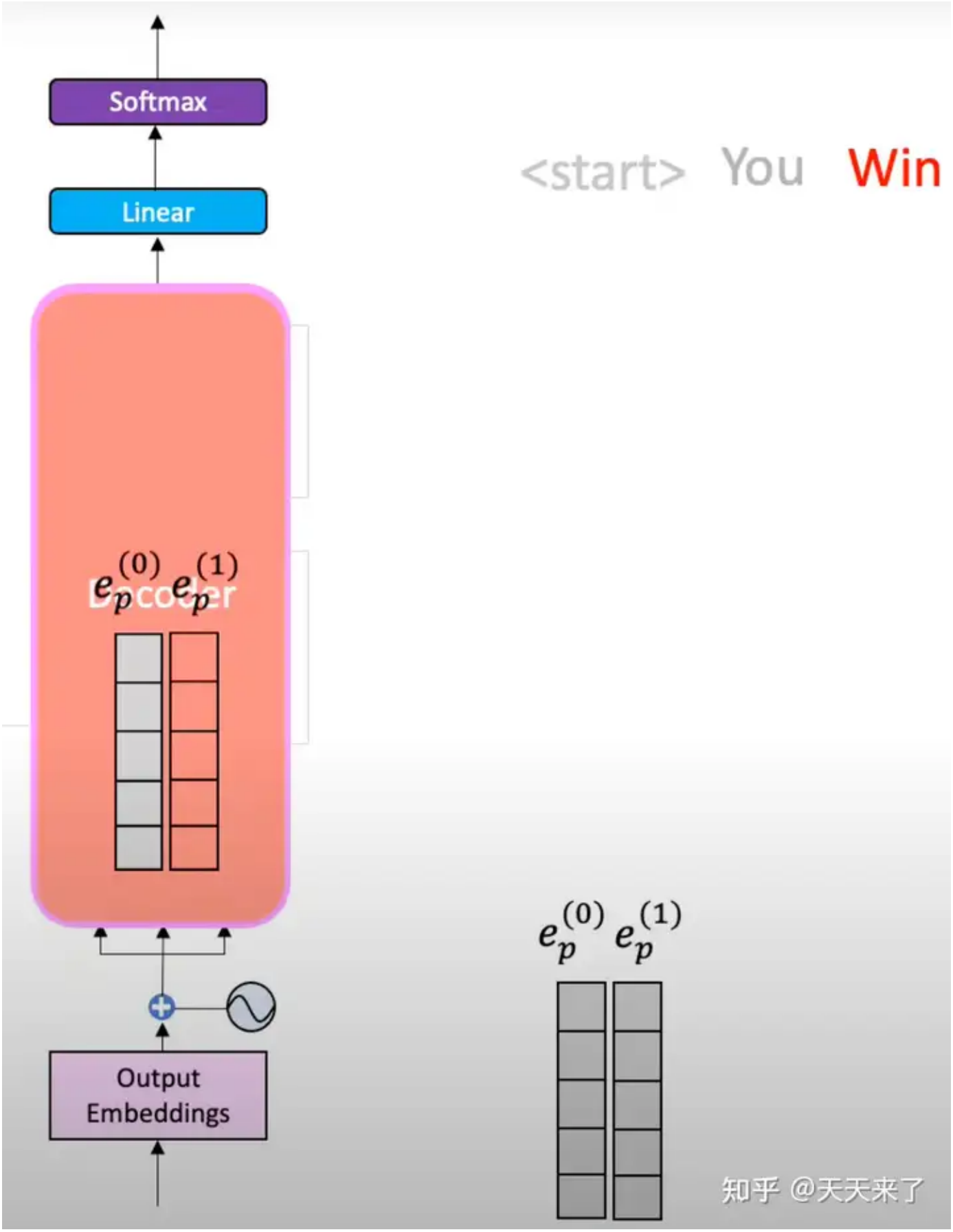
此外，中间还多了个Linear层，它是一个全连接层，它通常被用于将解码器的隐藏状态映射为目标词汇表的维度，从而生成最终的输出序列。所以说最后的Linear层神经元的个数依赖于最后要分类数。

步骤一（ Output Embeddings ）

因为在对话的过程中单词是顺序生成的，即生成完第 i 个单词，才可以生成第 i+1 个单词。输出嵌入矩阵Output Embeddings就是依次的输出结果转换为对应的Embedding。


Softmax
Linear
Decoder
Output Embeddings
<start> You
$e_p^{(0)}$
$e_p^{(1)}$

如上图所示，在开始时我们将初始的输出设为<start>， 将<start>生成其Embedding 作为输入送到Decoder矩阵中，输出预测的输出“You”。

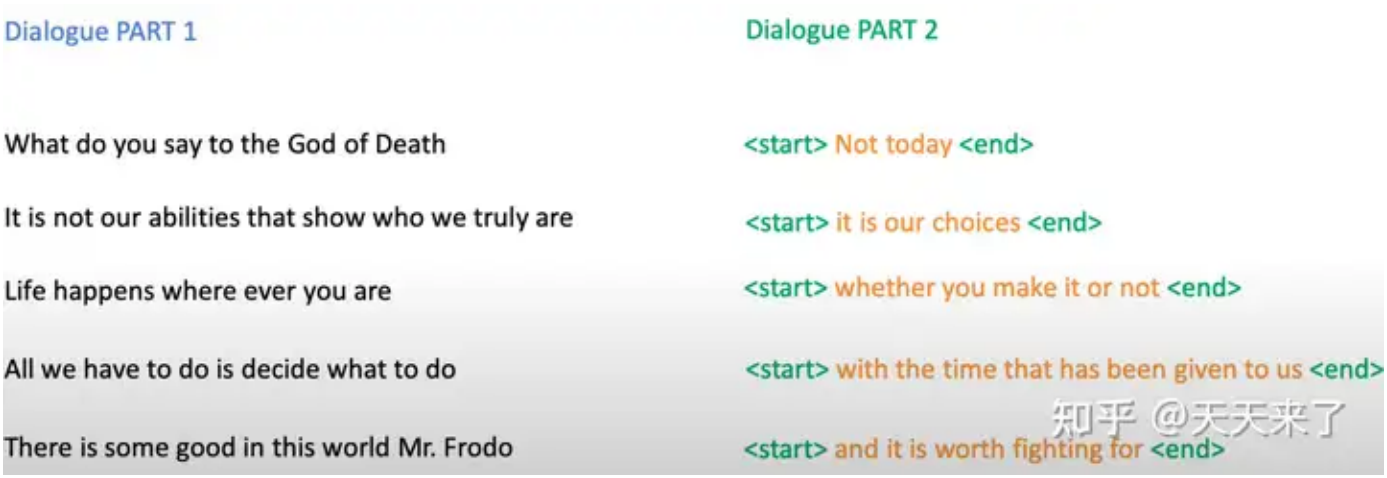


然后”You”转换为对应的Embedding矩阵e1，和<start>的矩阵在一起送入Decoder中，预测出下一个输出为”Win “，如此往复直到最后一个单词。

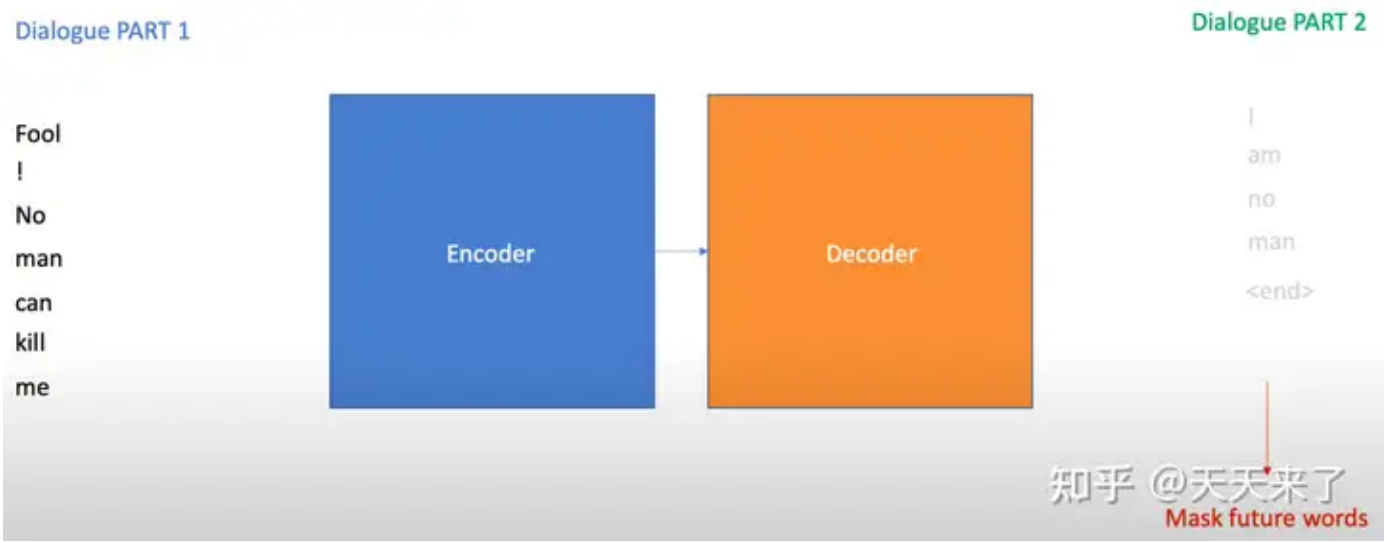
步骤二（Masked Multi-Head Attention）

在进行介绍Masked多头注意力模块前，我们先来了解一个概念。模型分为训练和推理两个阶段，用户先通过数据完成模型的训练后，对最优的模型状态进行保存，基于保存的模型进行推理。

首先，我们需要对模型进行训练，不同于推理阶段，训练时完整的对话数据都是可见的，如下图所示：



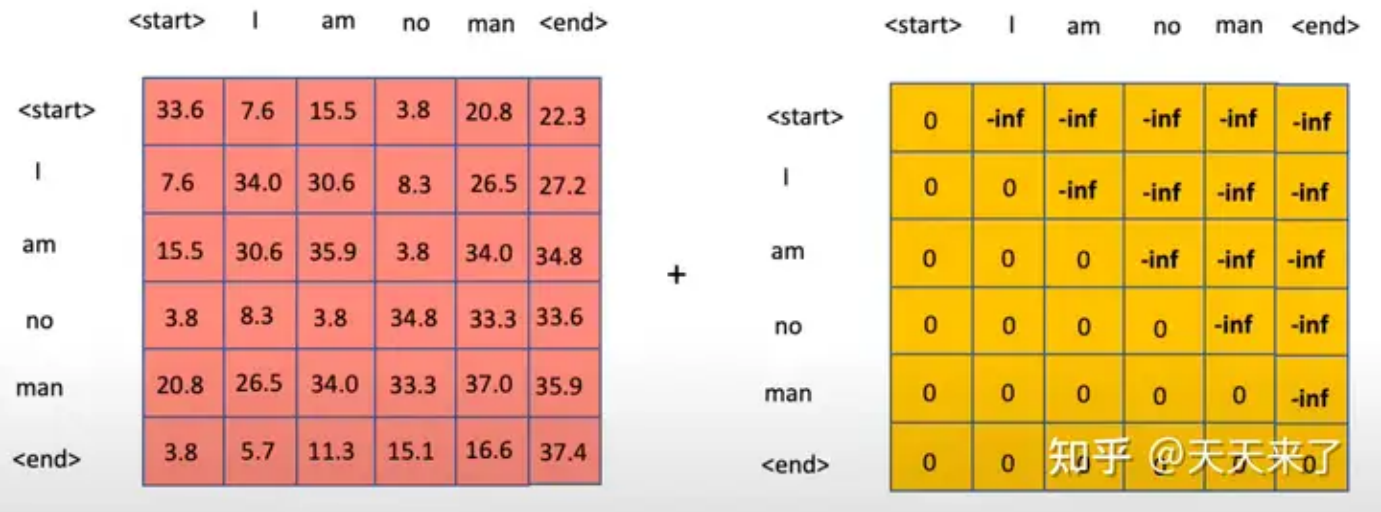
上面是《权力的游戏》中的对话数据集，在回答阶段我们再数据集上增加<start><end>方便数据更好送入Output Embeddings阶段。



例如，我们将对话” Fool! No man can kill me “” I am mo man “ 送入模型训练，此时我们是知道对话的结果的，就像模拟考试一样，我们需要先将回话部分给mask，防止你抄答案。

在mask对话的结果后，我们将对话” Fool! No man can kill me“送到模型Encoder中进行训练，同时将<start>送入Decoder， 此时可能给出一个预测的单侧为” You “，将其与标准答案”I“之间计算loss，并调整模型的参数。最终使得模型预测的结果可以更靠近标准的结果”I am mo man “。

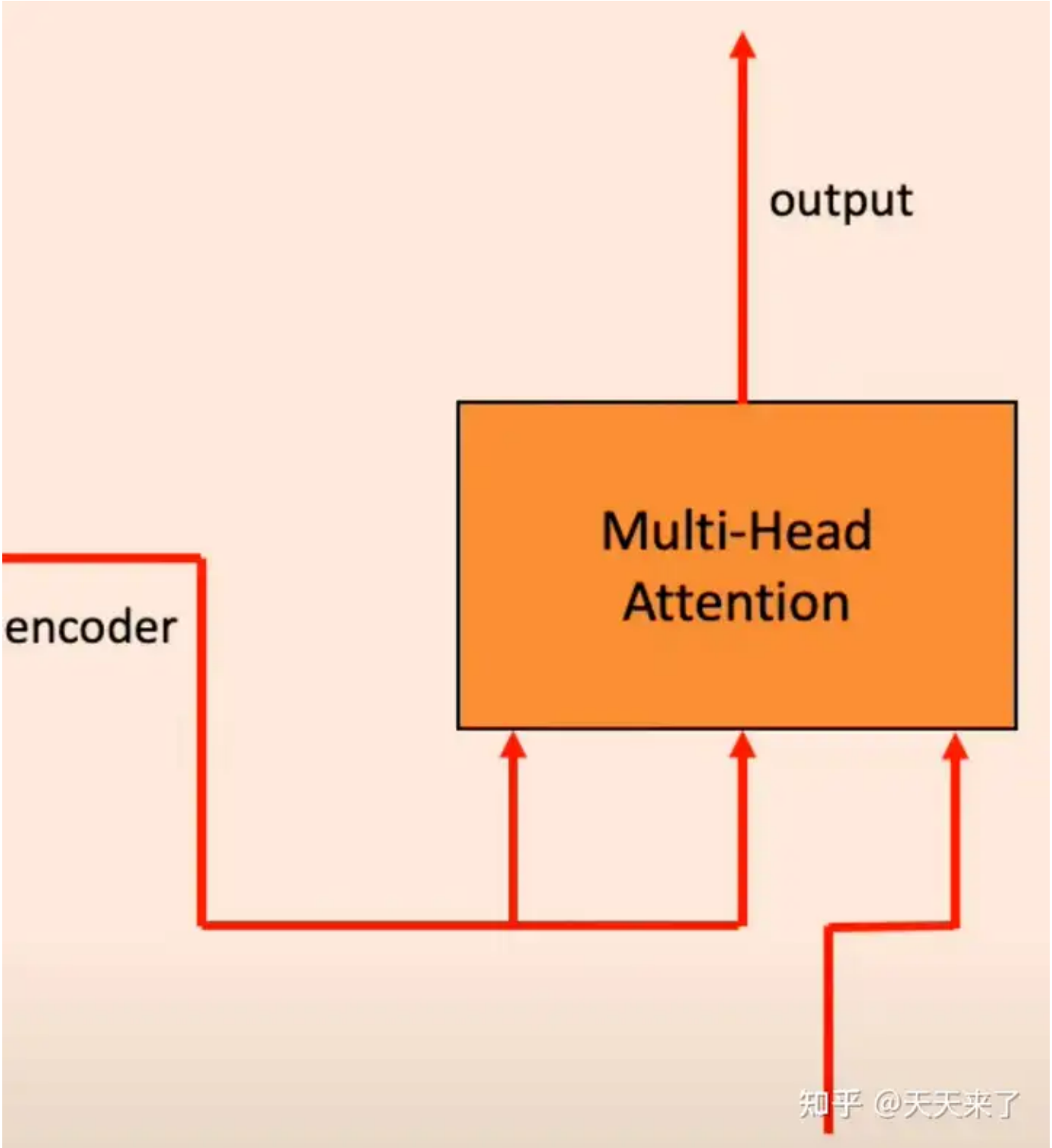
这就是我们为什么需要Masked Multi-Head Attention。



但是注意，这里的mask只是mask将要预测输出的单词以及之后。如上图所示，我们使用-inf作为mask区域，0作为非mask区域，然后将两个矩阵进行相加。值为0的区域，不会改变其原值。

然后接下来的计算就与Part1中self-Attention的计算过程一致了。最后在经过一层Add&Norm后，我们将其输出与Encoder的输出一起送入下一个**Multi-Head Attention。**

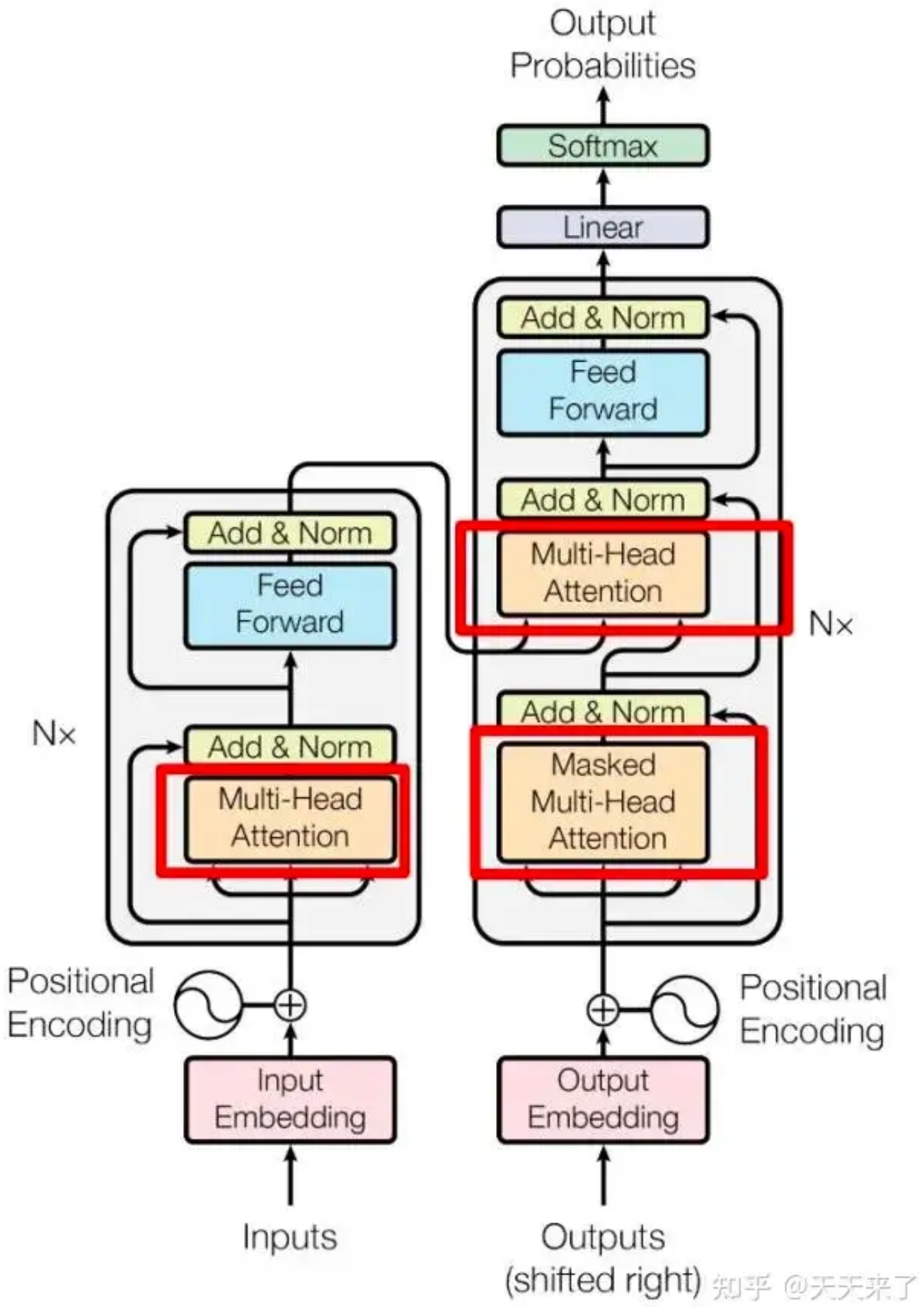
步骤三（Multi-Head Attention）



这个Multi-Head Attention与Encoder中的基本没什么区别，主要的区别在于其中 Self-Attention 的 **K, V**矩阵不是使用 上一个 Decoder block 的输出计算的，而是使用 **Encoder 的编码信息矩阵 C** 计算的。

根据 Encoder 的输出 **C**计算得到 **K, V**，根据上一个 Decoder block 的输出 **Z** 计算 **Q** (如果是第一个 Decoder block 则使用输入矩阵 **X** 进行计算)，后续的计算方法与之前描述的一致。

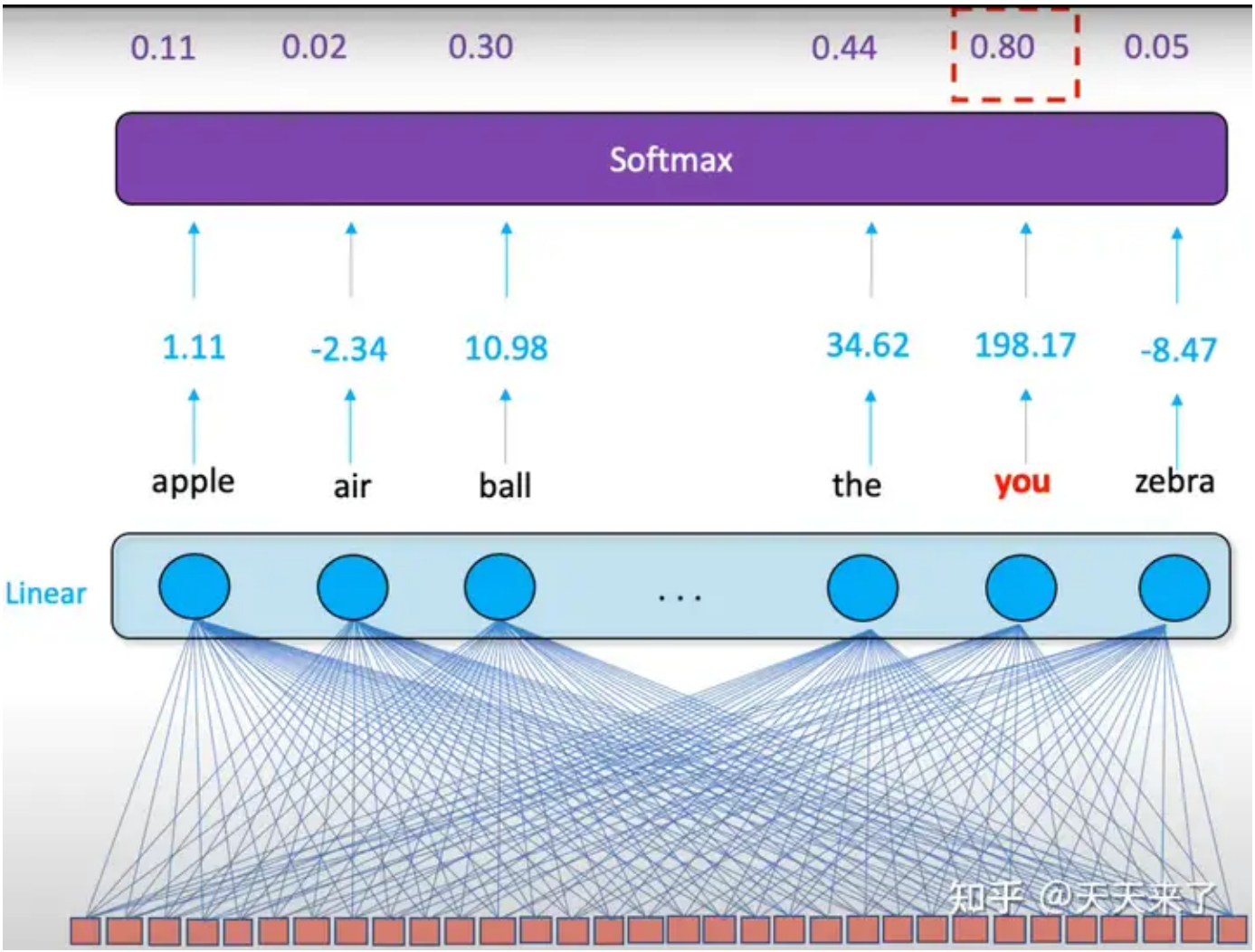
这样做的好处是在 Decoder 的时候，每一位单词都可以利用到 Encoder 所有单词的信息 (这些信息无需 **Mask**)。


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
知乎 @天天来了

由上图可知，这个部分直到Linear之前与Encoder部分完全一样，所以细节就不展开了。

步骤四 (Linear & Softmax 预测输出单词)

首先，解码器的最后一个位置有一个Linear层，它是一个全连接层，通过将解码器的隐藏状态传入线性层进行线性变换。这个线性变换将解码器的隐藏状态映射为一个与目标词汇表大小相同的向量，相当于将输出的结果矩阵与词汇表向量相乘，然后可以通过 softmax 操作将其转化为概率分布，得出词汇表中每一个单词成为预测单词的概率。



如上图所示，词汇表中单词”You“的概率最大，那么它将会成为这次的输出。

此外，与 Encoder 一样，Decoder 是由多个 Decoder block 组合而成。

4. 总结

Transformer 与 RNN 不同，可以比较好地并行训练。

Transformer 本身是不能利用单词的顺序信息的，因此需要在输入中添加位置 Embedding，否则 Transformer 就是一个词袋模型了。

Transformer 的重点是 Self-Attention 结构，其中用到的 **Q, K, V**矩阵通过输出进行线性变换得到。

Transformer 中 Multi-Head Attention 中有多个 Self-Attention，可以捕获单词之间多种维度上的相关系数 attention score。

编辑于 2023-07-22 06:18 · IP 属地北京