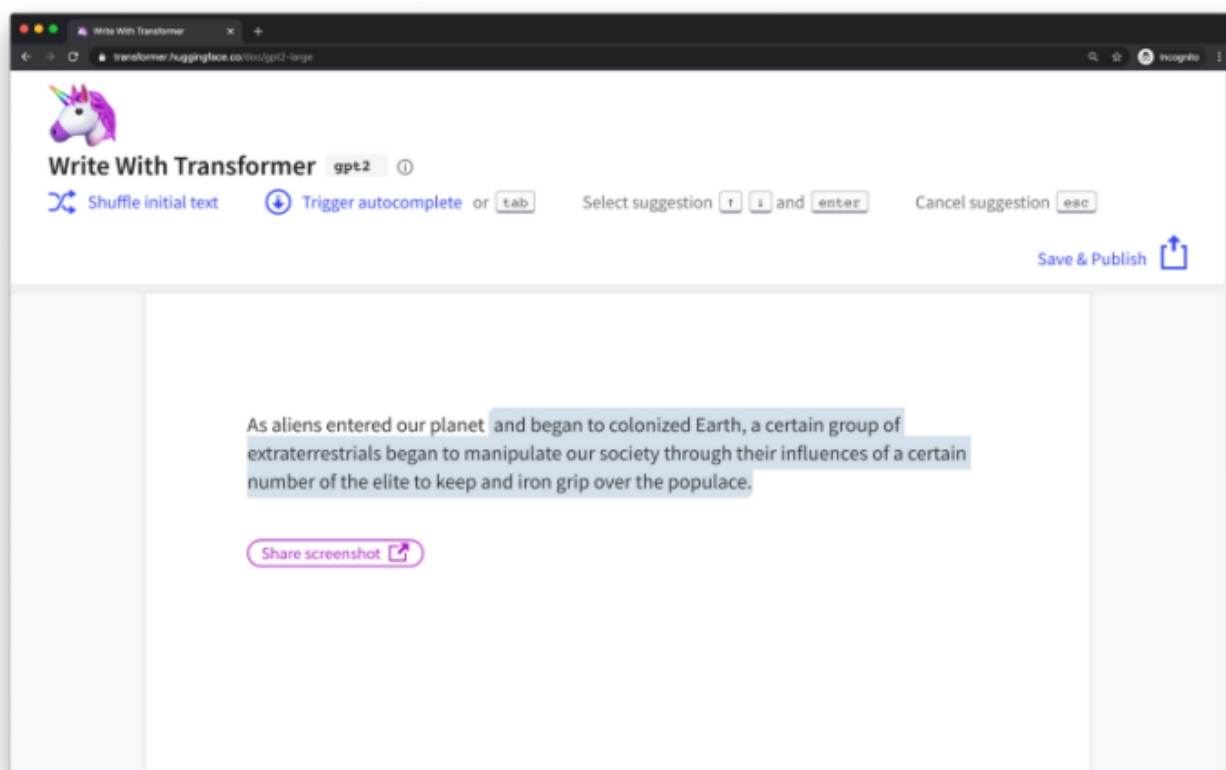


1. 引言

RNN由于其顺序结构训练速度常常受到限制，而且由于梯度下降，它难以看到较远处的信息。既然Attention模型本身可以看到全局的信息，那么一个自然的疑问是我们能不能去掉RNN结构，仅仅依赖于Attention模型，这样我们可以使训练并行化，同时拥有全局信息？

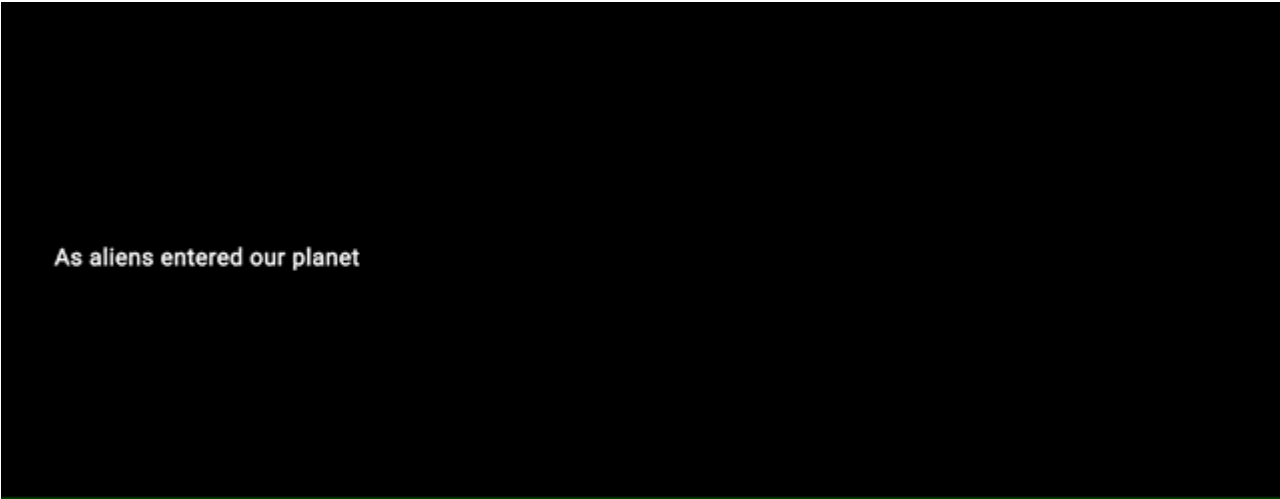
你可能听说过不同的著名Transformer模型，如 BERT、GPT。在这篇文章中，我们将精读谷歌的这篇 [Attention is All you need](#) 论文来回顾一下仅依赖于Self-Attention机制的Transformer架构。

在开始之前，先让我们看一个好玩的例子。Transformer可以根据你写的一句开头，写出一段科幻小说！链接：[Hugging Face – The AI community building the future.](#)



Our input: “As Aliens entered our planet”. Transformer output: “and began to colonized Earth, a certain group of extraterrestrials began to manipulate our society through their influences of a certain number of the elite to keep and iron grip over the populace.”

这真是一个黑暗的故事...但有趣的是研究模型是如何生成这个黑暗故事的。当模型逐字生成文本时，它可以“关注”与生成的单词相关的单词(attention)。知道要关注哪些单词的能力也是在训练过程中通过反向传播学到的。



As aliens entered our planet

Transformer的优势在于，它可以不受梯度消失的影响，能够保留**任意长**的长期记忆。而RNN的记忆窗口很短；LSTM和GRU虽然解决了一部分梯度消失的问题，但是它们的记忆窗口也是有限的。

Recurrent neural networks (RNN) are also capable of looking at previous inputs too. But the power of the attention mechanism is that it doesn't suffer from short term memory. RNNs have a shorter window to reference from, so when the story gets longer, RNNs can't access words generated earlier in the sequence. This is still true for Gated Recurrent Units (GRU) and Long-short Term Memory (LSTM) networks, although they do a bigger capacity to achieve longer-term memory, therefore, having a longer window to reference from. The attention mechanism, in theory, and given enough compute resources, have an infinite window to reference from, therefore being capable of using the entire context of the story while generating the text.



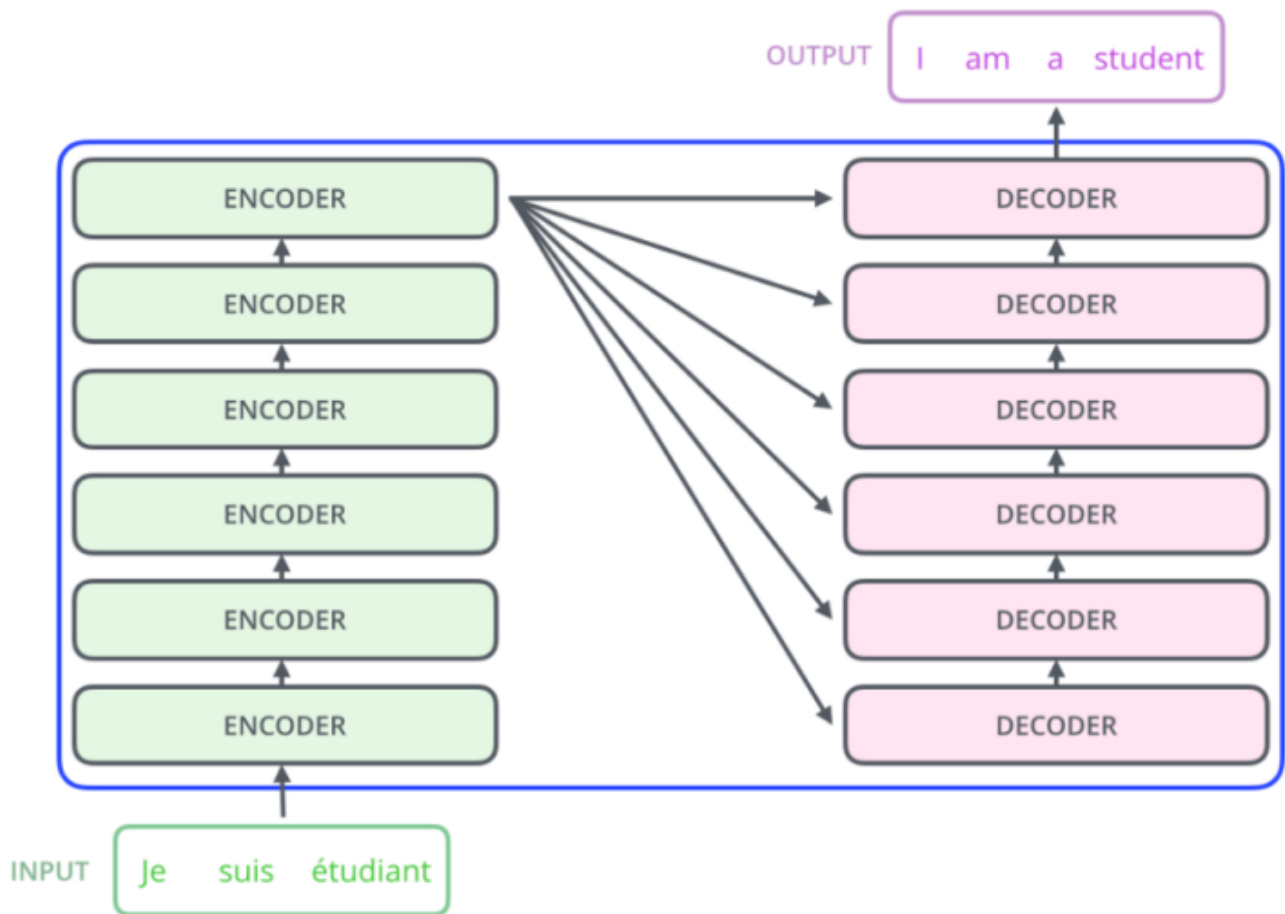
Attention Mechanism has an infinite reference window

As aliens entered our planet and began to colonize earth a certain group of extraterrestrials ...

2.Attention Is All You Need — Step by Step Walkthrough

2.1 总体结构

Transformer的结构采用 Encoder-Decoder 架构。论文中Encoder层由6个Encoder堆叠在一起，Decoder层也是6层。



先从整体来看：

- Encoder将输入序列做了一个复杂的、高阶的embedding；
- Decoder使用了Encoder的这个高阶embedding，同时还根据之前Decoder的输出，一步一步地产生输出。

每一个Encoder和Decoder的内部结构如下图：

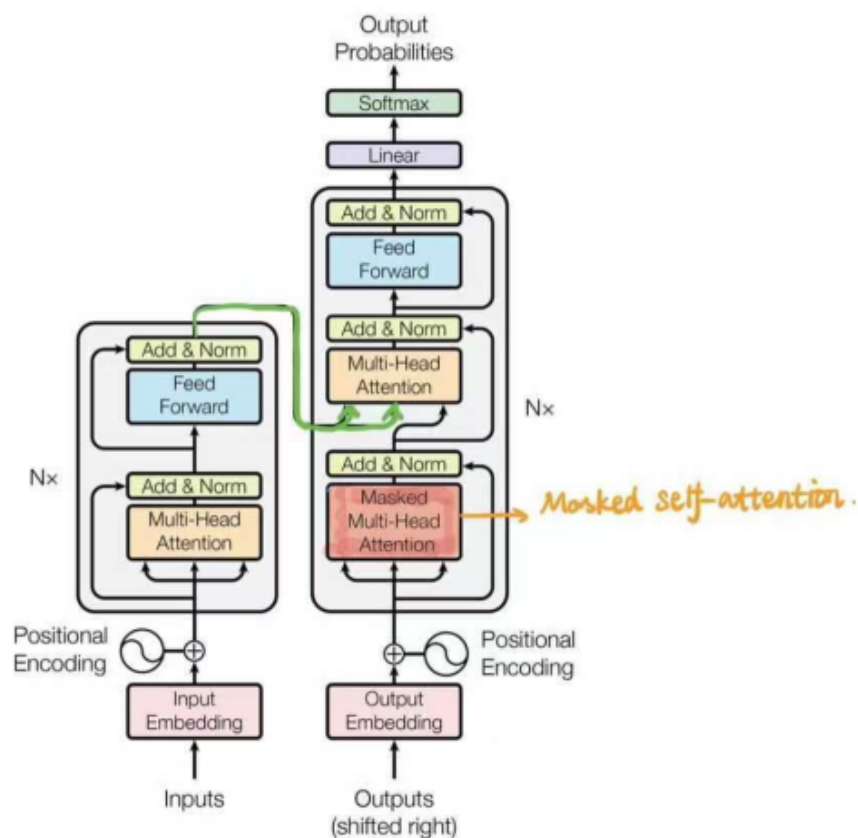


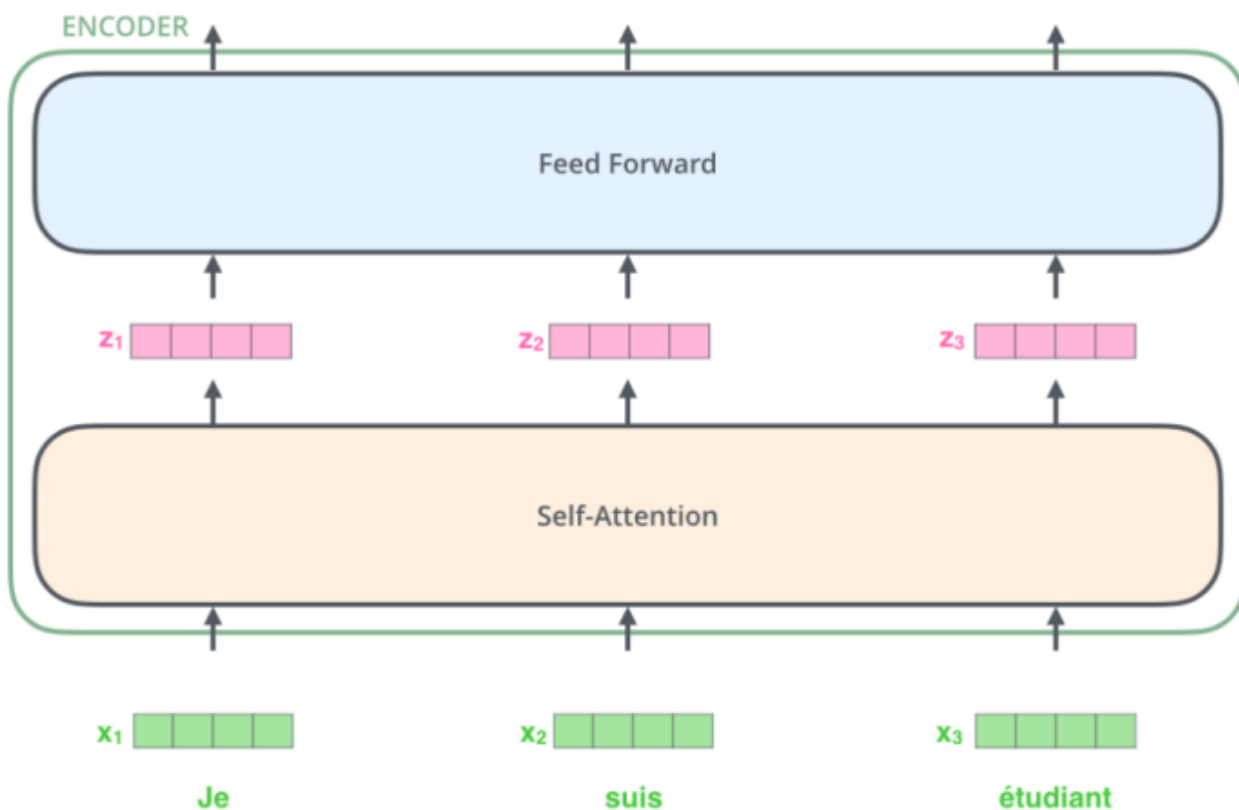
Figure 1: The Transformer - model architecture.

先看Encoder。Encoder包含两层：

- 多头Self-attention层，帮助当前节点不仅仅只关注当前的词，从而能获取到上下文的语义。
- 前馈神经网络层(feed forward)，提供非线性。

2.2 Encoder层详细说明

首先，模型需要对输入的数据进行一个embedding操作，并输入到Self-attention层，处理完数据后把数据送给前馈神经网络，前馈神经网络的计算可以并行，得到的输出会输入到下一个Encoder。大致结构如下：



x_1, x_2 就是embedding, z_1, z_2 是经过self-attention之后的输出, r_1, r_2 是经过feed forward网络之后的输出, 它们会被输入到下一层encoder中去。

2.2.1 Embedding层

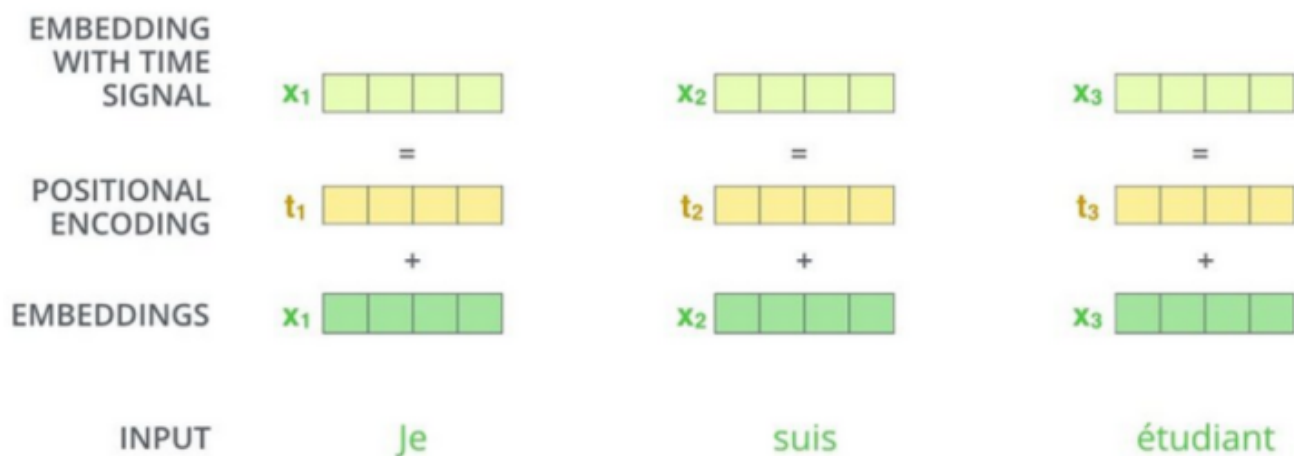
Transformer模型中缺少一种解释输入序列中单词顺序的方法。为了处理这个问题, Transformer给Encoder层和Decoder层的输入添加了一个额外的向量Positional Encoding, 维度和embedding的维度一样。这个位置向量的具体计算方法有很多种, 论文中的计算方法如下:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

其中 pos 是指当前词在句子中的位置, i 是指向量中每个值的 index, 可以看出, 在偶数位置, 使用正弦编码, 在奇数位置, 使用余弦编码。

所以, 最终一个词的embedding, 就是它的语义信息embedding+序列信息embedding (positional encoding):



2.2.2 Self-attention层

让我们从宏观视角看自注意力机制，精炼一下它的工作原理。

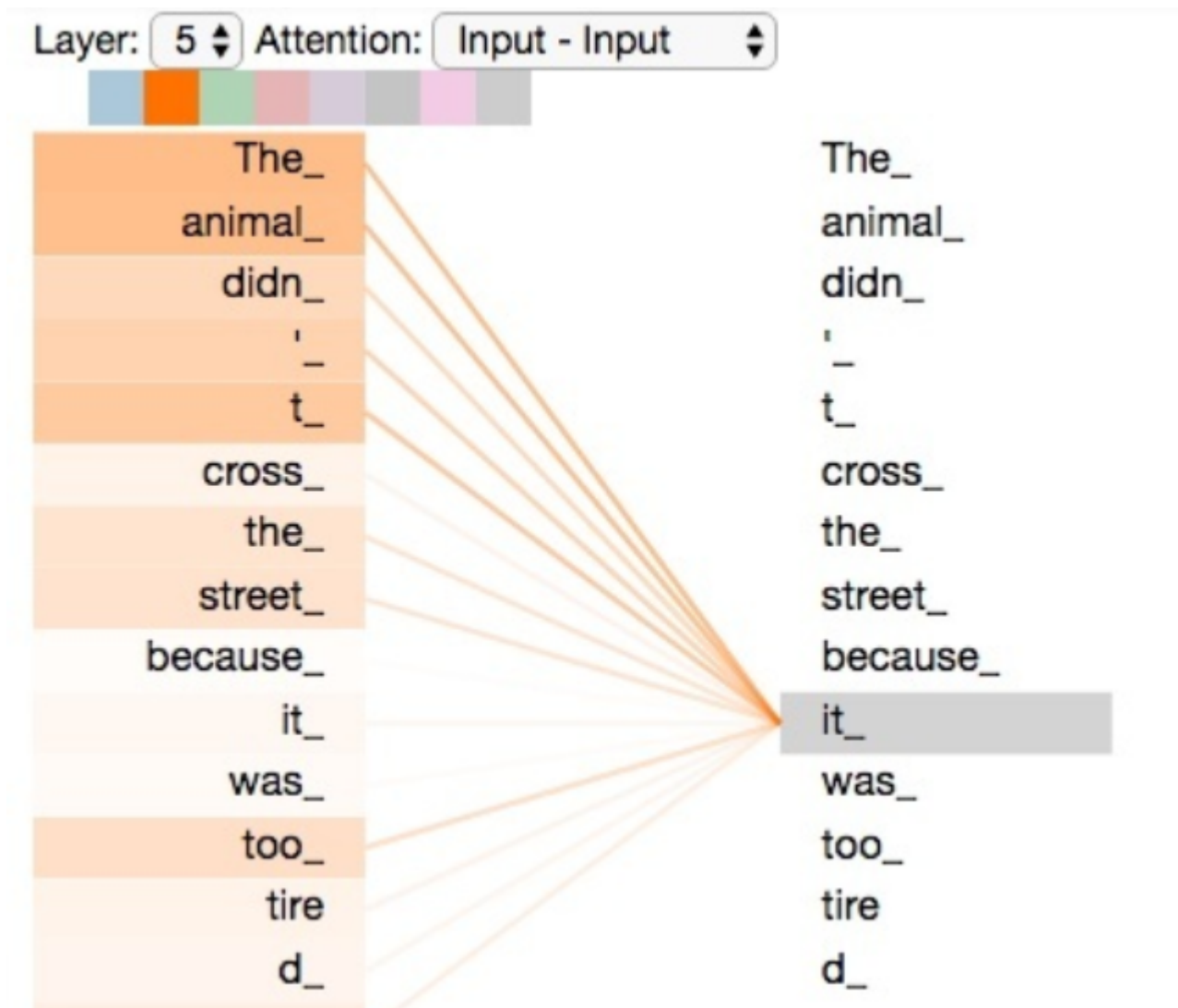
例如，下列句子是我们想要翻译的输入句子：

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。

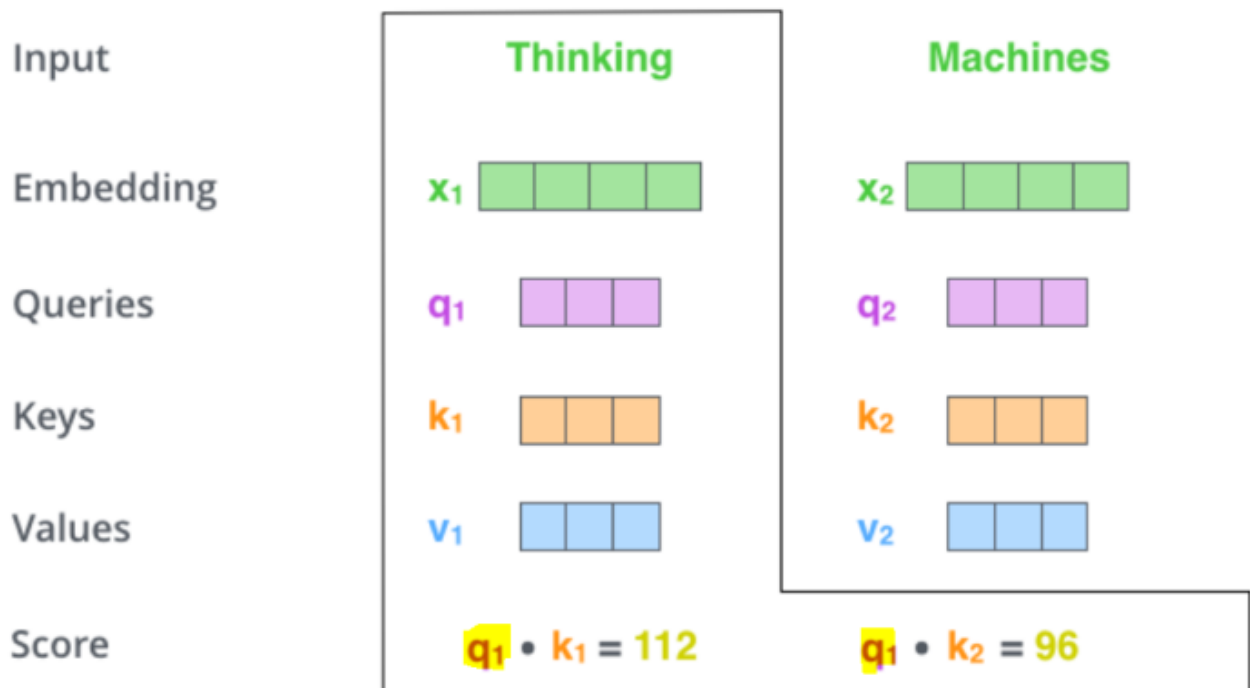
随着模型处理输入序列的每个单词，自注意力会关注整个输入序列的**所有单词**，帮助模型对本单词更好地进行编码(embedding)。



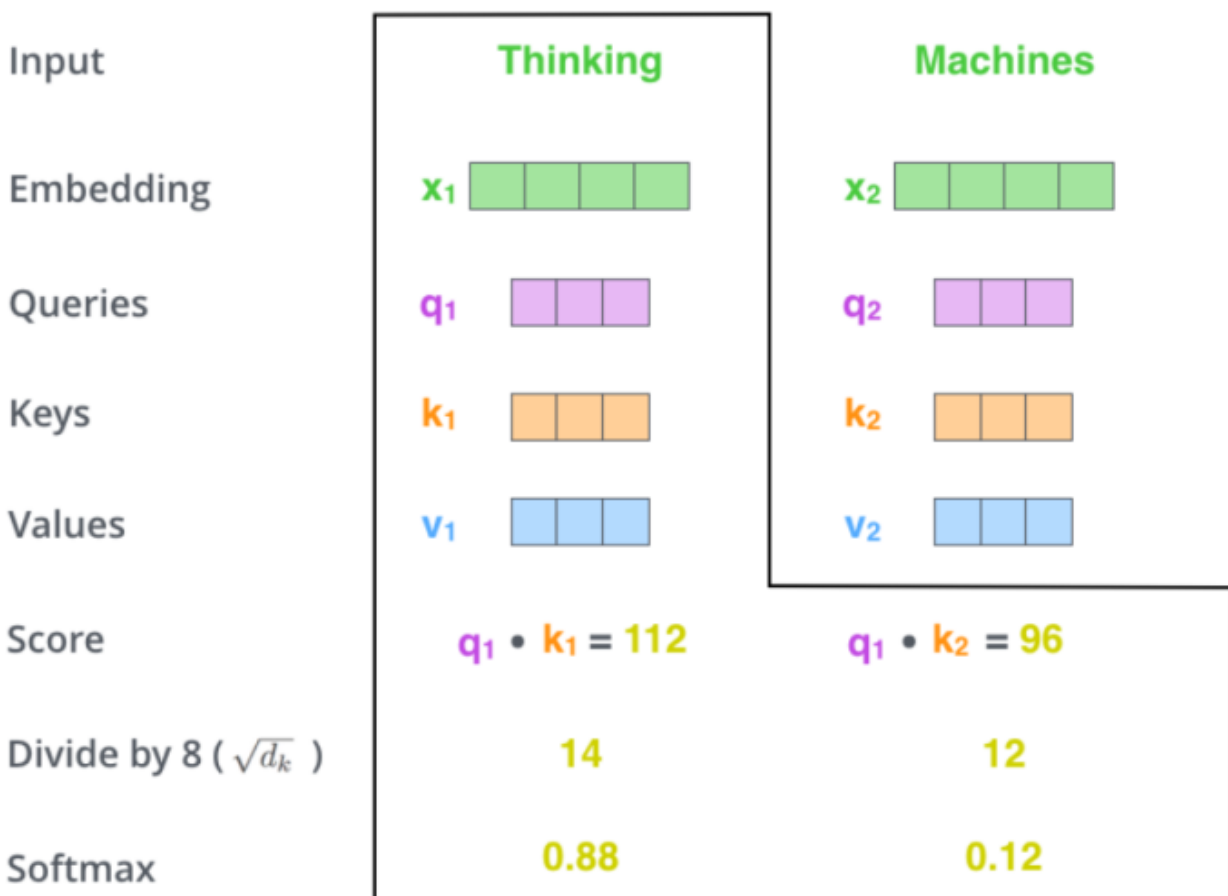
如上图所示，当我们在编码器#5（栈中最上层编码器）中编码“it”这个单词的时，注意力机制的部分会去关注“The Animal”，将它的表示的一部分编入“it”的编码中。接下来我们看一下Self-Attention详细的处理过程。

step1: 首先，对于输入序列的每个单词，它都有三个向量编码，分别为： $Query$ 、 Key 、 $Value$ 。这三个向量是用embedding向量与三个矩阵（ W^Q, W^K, W^V ）相乘得到的结果。这三个矩阵的值在BP的过程中会一直进行更新。

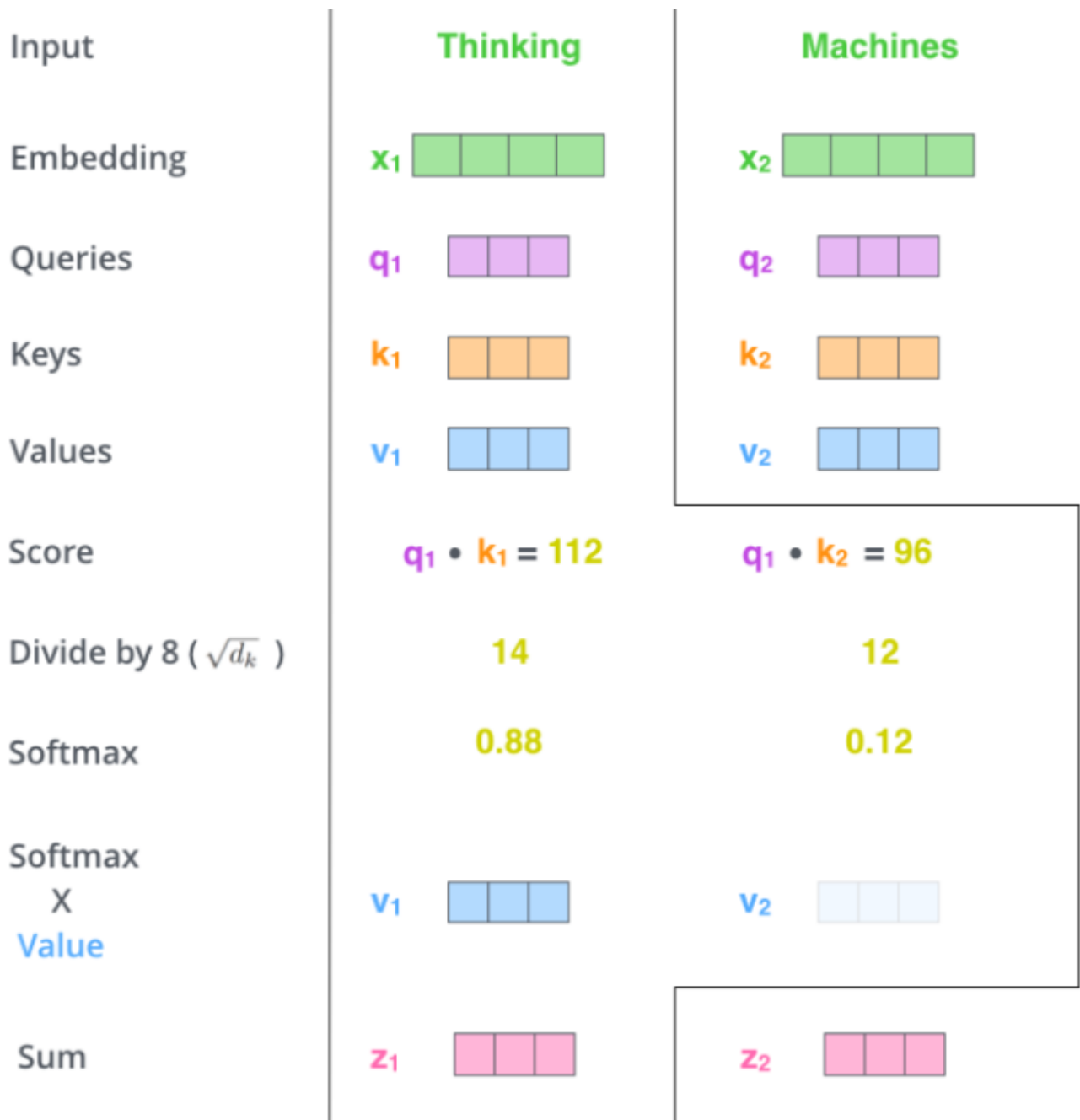
step2: 第二步计算Self-Attention的分数值，该分数值决定了当我们在某个位置encode一个词时，对输入句子的其他部分的关注程度。这个分数值的计算方法是用该词语的 $Query$ 与句子中其他词语的 Key 做点乘。以下图为例，假设我们在为这个例子中的第一个词“Thinking”计算自注意力向量，我们需要拿输入句子中的每个单词对“Thinking”打分。这些分数决定了在编码单词“Thinking”的过程中重视句子其它token的程度。



step3: 再对每个分数除以 \sqrt{d} (d 是embedding维度) , 之后做softmax进行归一化。



step4: 把每个 *Value* 向量和softmax得到的权重值进行相乘, 进行加权求和, 得到的结果即是一个词语的self-attention embedding值。这个embedding值已经是融合了句子中其他token的了。



这样，自注意力的计算就完成了。得到的向量就可以传递给前馈神经网络。

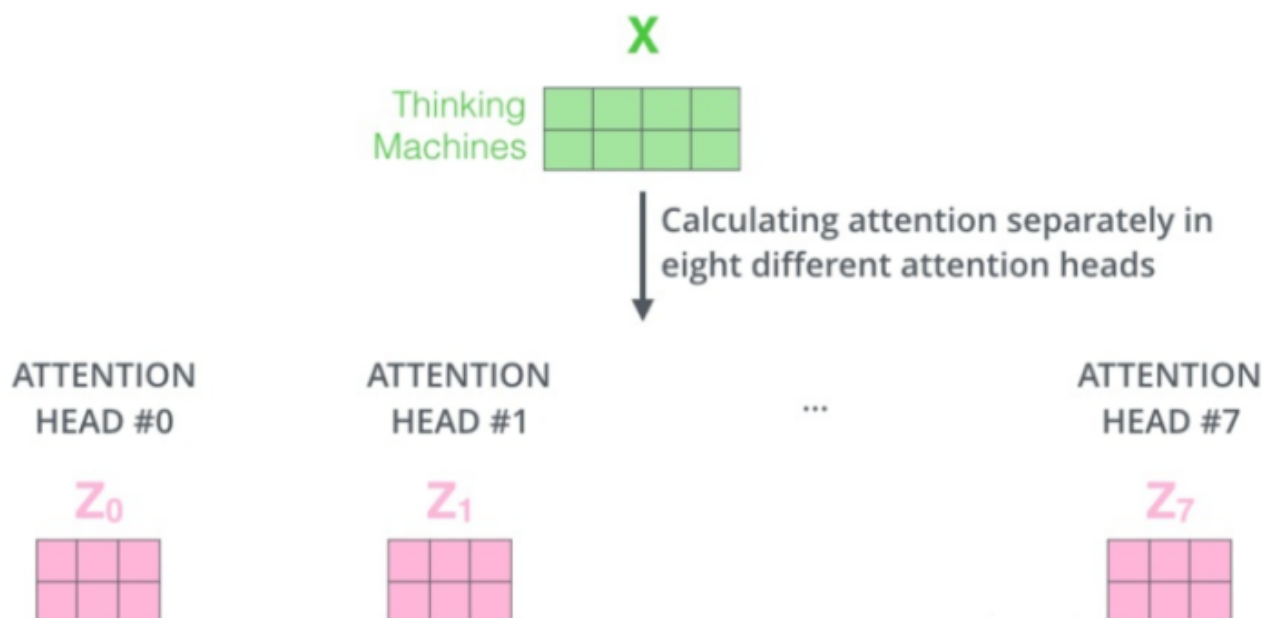
2.2.3 Multi-Headed Attention

通过增加一种叫做“多头”注意力 (“multi-headed”attention) 的机制，论文进一步完善了自注意力层。

接下来我们将看到，对于“多头”注意力机制，我们有多组Query/Key/Value权重矩阵集

$$W^Q, W^K, W^V$$

(Transformer使用八个注意力头)。



现在对于每一个词语，我们有了八个向量

$$z_0, \dots, z_7$$

，它们分别由八个head产生。但是对于下一个feed-forward层，我们应该把每个词语都用一个向量来表示。所以下一步，我们需要把这八个向量压缩成一个向量。

可以直接把这些矩阵拼接在一起，然后用一个附加的权重矩阵

$$W^O$$

与它们相乘：

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^O that was trained jointly with the model

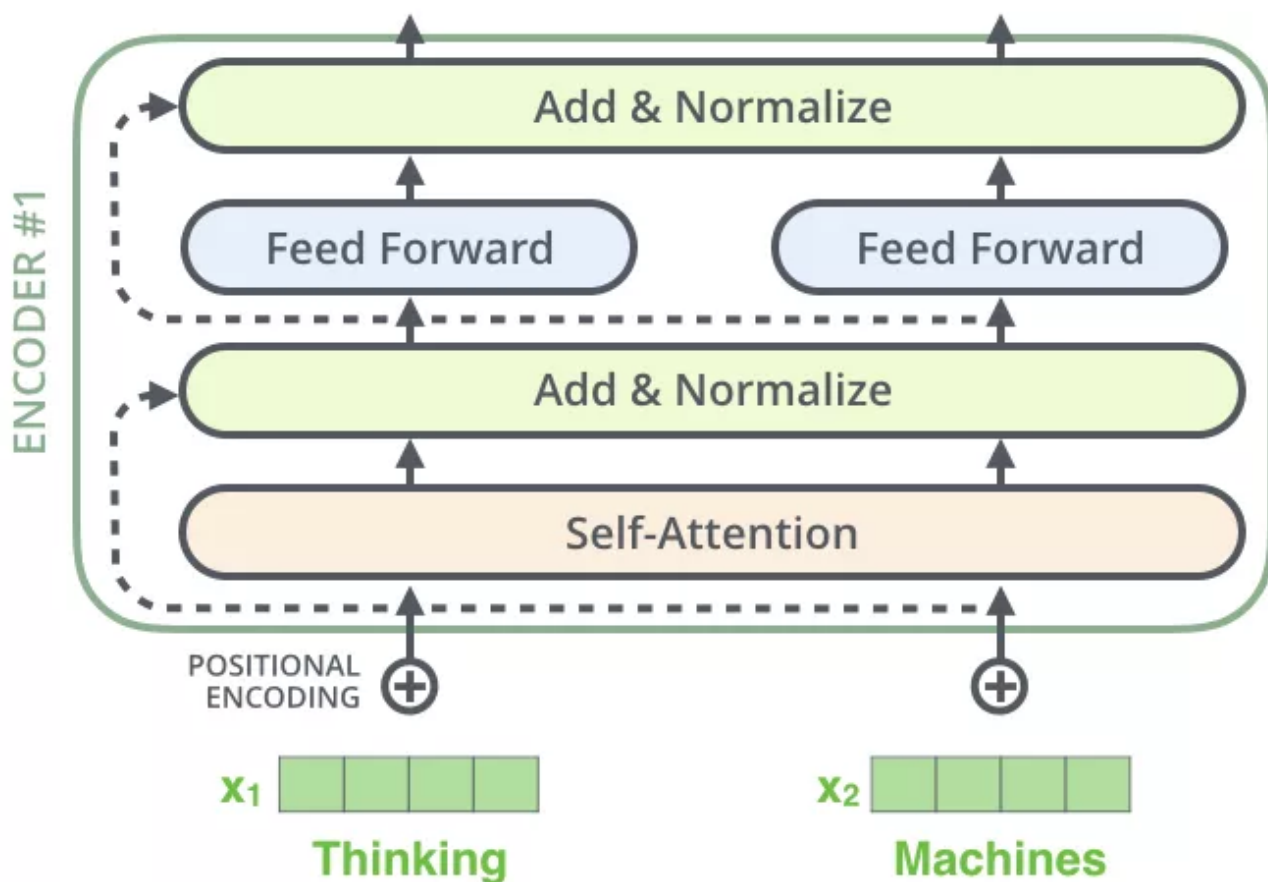
X

3) The result would be the **Z** matrix that captures information from all the attention heads. We can send this forward to the FFNN



2.2.4 The Residuals and Layer normalization

在继续进行下去之前，我们需要提到一个encoder中的细节：在每个encoder中都有一个残差连接，并且都跟随着一个Layer Normalization（层-归一化）步骤。



Layer-Norm也是归一化数据的一种方式，不过 Layer-Norm 是在**每一个样本上**计算均值和方差，而不是 Batch-Norm 那种在**批**方向计算均值和方差！

Batch Normalization

batch			Same for all training examples	
			mean	std
1	3	6	3	3
2	2	2	2	0
0	1	5	3	3
4	6	1	4	3
5	2	3	3	2
1	0	1	1	1

Layer Normalization

batch					
1	3	6			
2	2	2			
0	1	5			
4	6	1			
5	2	3			
1	0	1			
mean	2	3	3		
std	2	2	2		

Same for all feature dimensions

2.2.5 小结

Encoder就是用来给input一个比较好的embedding，使用self-attention来使一个词的embedding包含了**上下文**的信息，而不是简单的查look-up table。Transformer使用了多层(6层)的Encoder是为了把握一些**高阶**的信息。

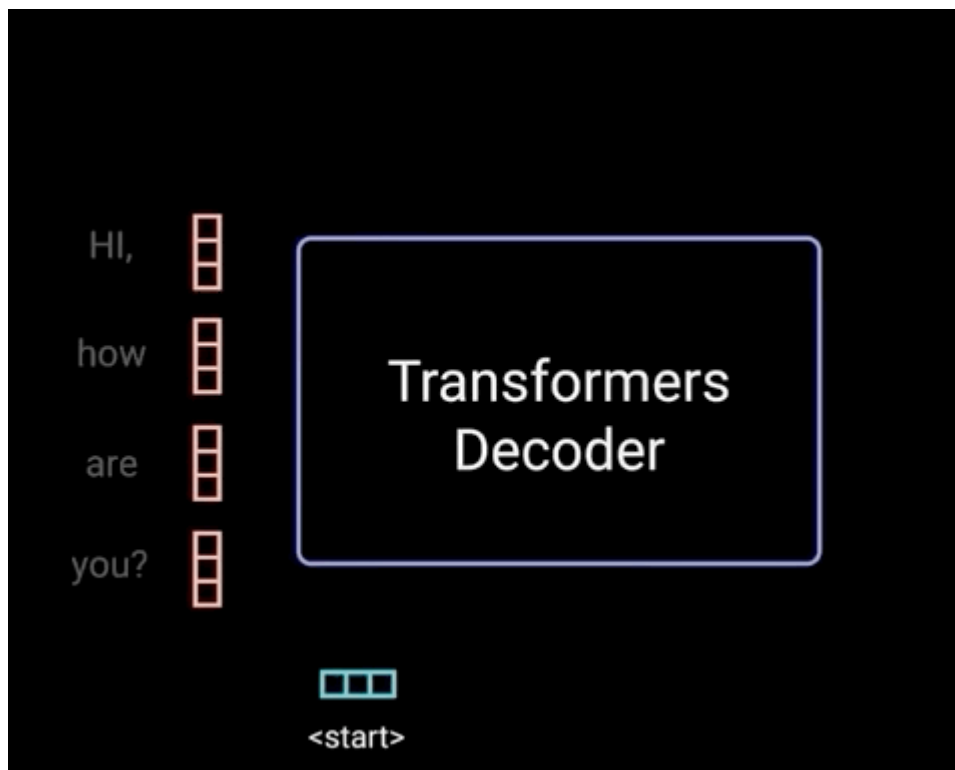
2.3 Decoder层

2.3.1 简介

从更高的角度来看，Transformer的Decoder作用和普通seq2seq一样：从<Start>开始，基于**之前的Decoder输出**，以及**Encoder的embedding**，来预测**下一个词的概率分布**。

以对话系统为例：

Our Input: "Hi how are you" Transformer Output: "I am fine"



下面我们来详细介绍Decoder的内部结构。

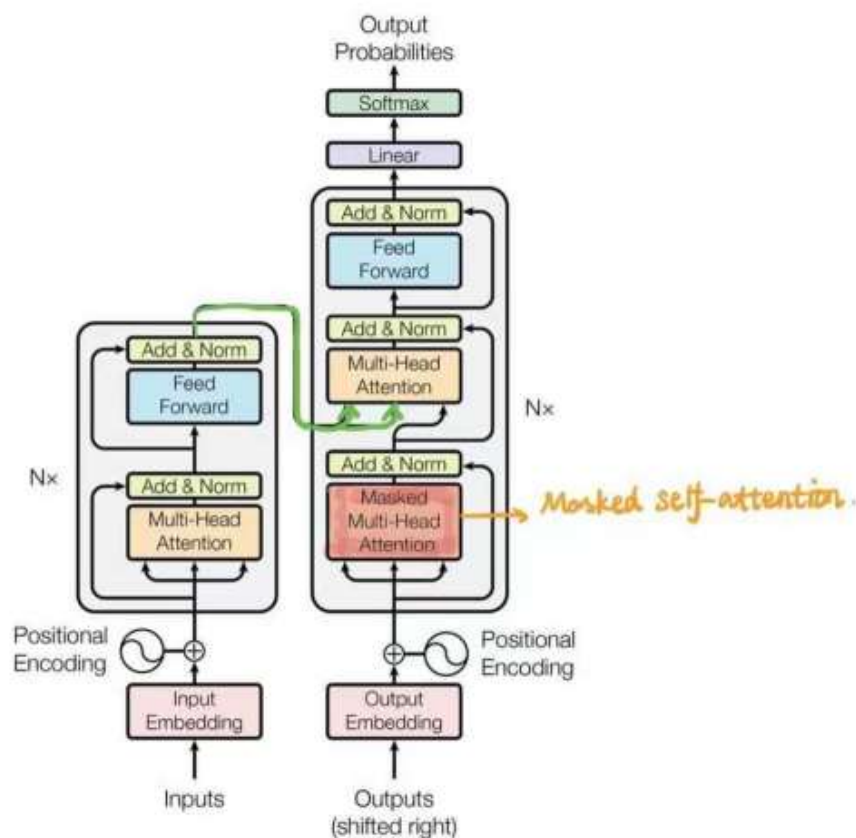
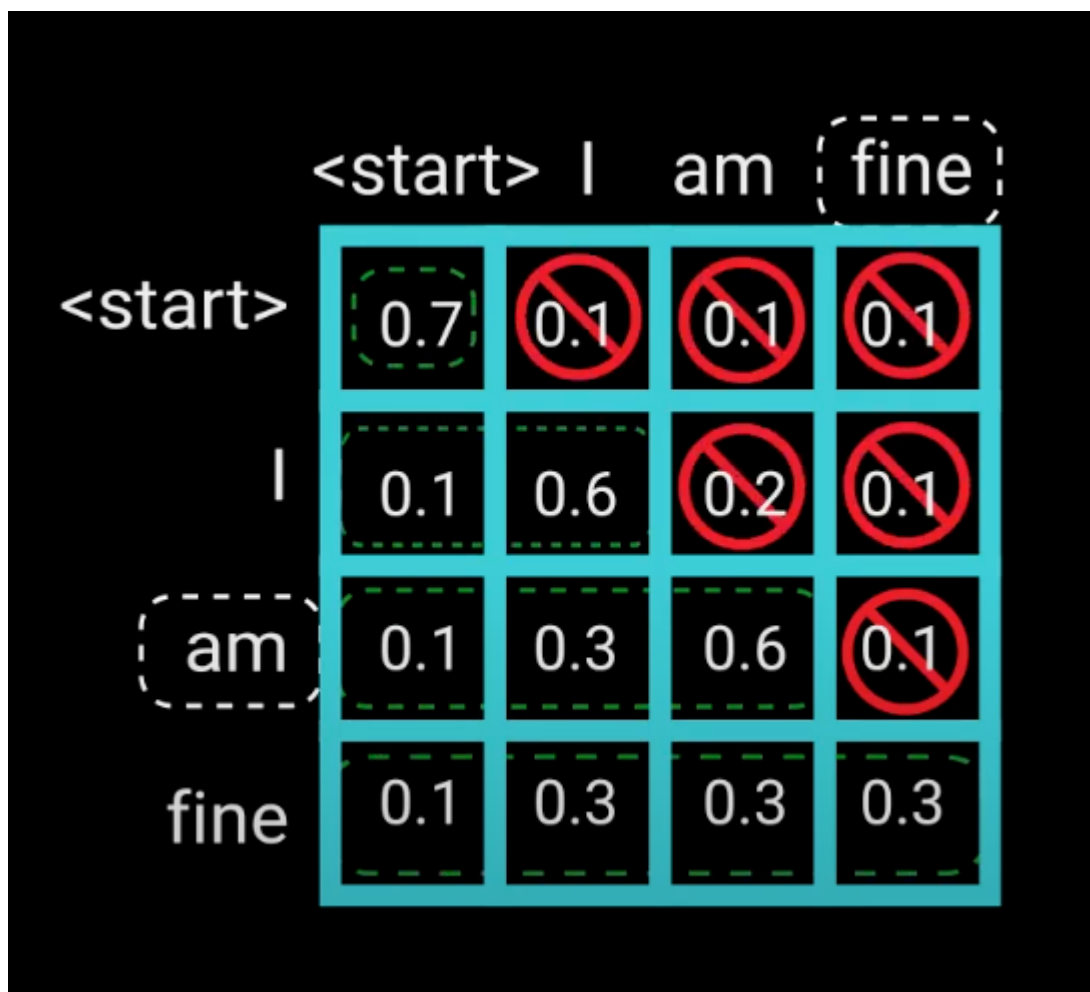


Figure 1: The Transformer - model architecture.

2.3.2 Masked Multi-Head Attention

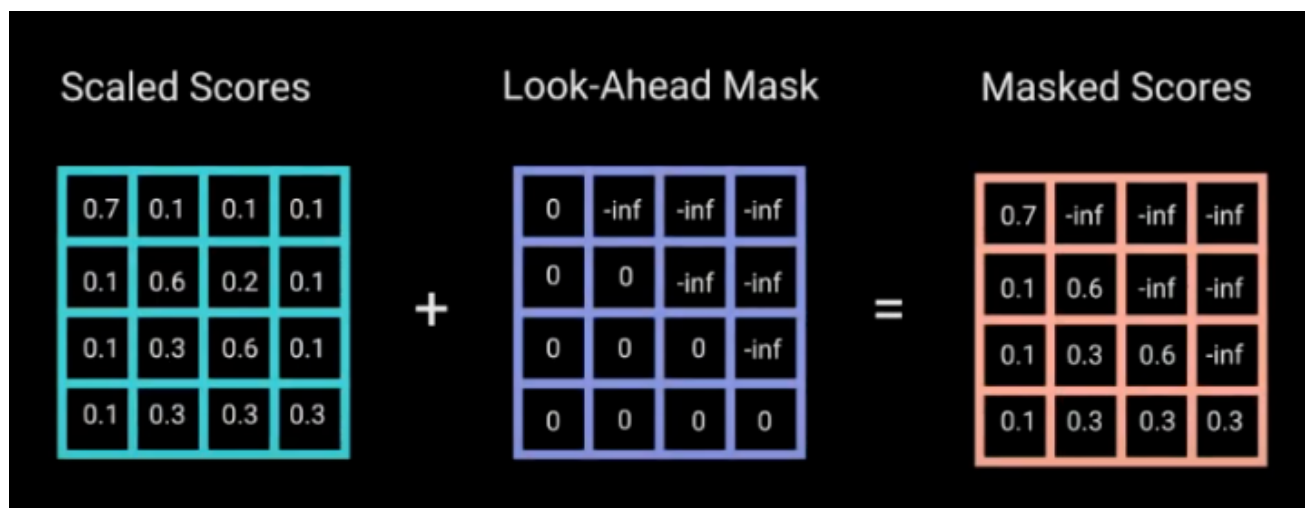
和Encoder一样，Decoder先经过embedding+positional encoding之后得到了一个embedding。然后，输入到**Masked** Multi-Head Attention中。和前面不同的是，Decoder的self-attention层其实是**masked** multi-head attention。mask表示掩码，它对某些值进行掩盖。这是为了防止Decoder在计算某个词的attention权重时“看到”这个词后面的词语。

Since the decoder is **auto-regressive** and generates the sequence word by word, you need to prevent it from conditioning to future tokens. For example, when computing attention scores on the word "am", you should not have access to the word "fine", because that word is a future word that was generated after. **The word "am" should only have access to itself and the words before it.** This is true for all other words, where they can only attend to previous words.

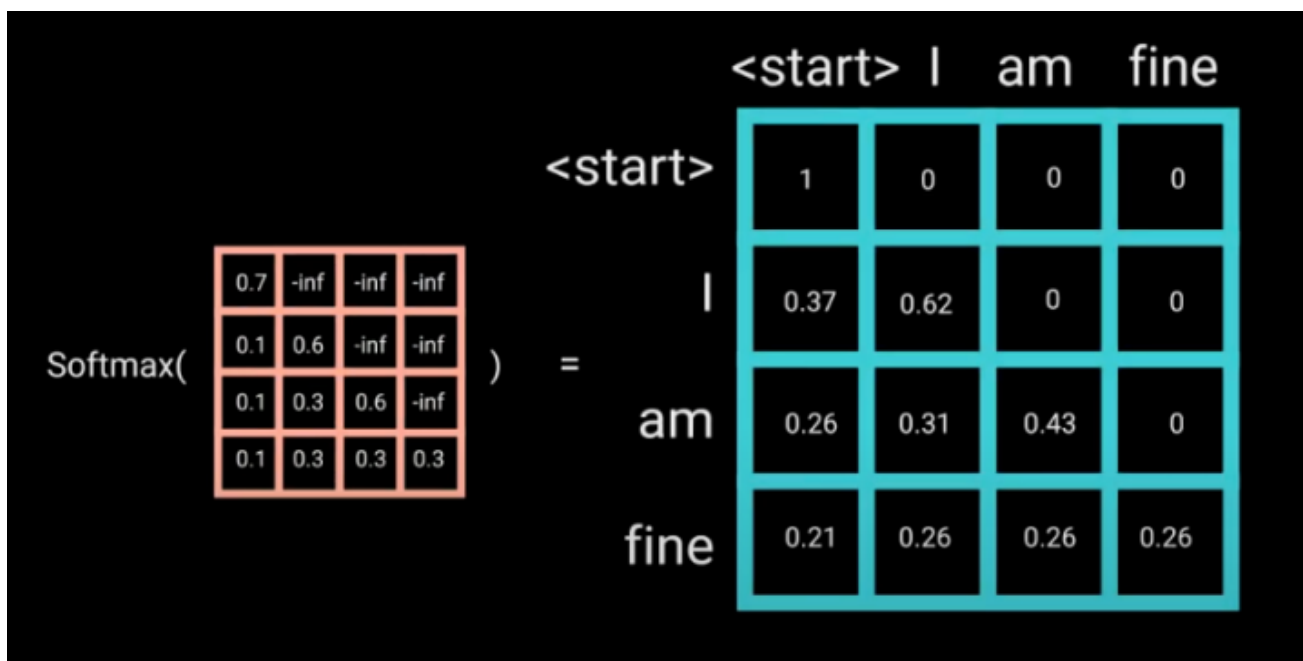


Mask 是为了使得 decoder 不能看见未来的信息。也就是对于一个序列，在 time_step 为 t 的时刻，我们的 Decoder 只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。因此我们需要想一个办法，把 t 之后的信息给隐藏起来。

那么具体怎么做呢？也很简单：产生一个上三角矩阵，上三角的值全为 $-\infty$ 。



加上 $-\infty$ 的目的是，做 softmax 之后 $-\infty$ 会变成 0：

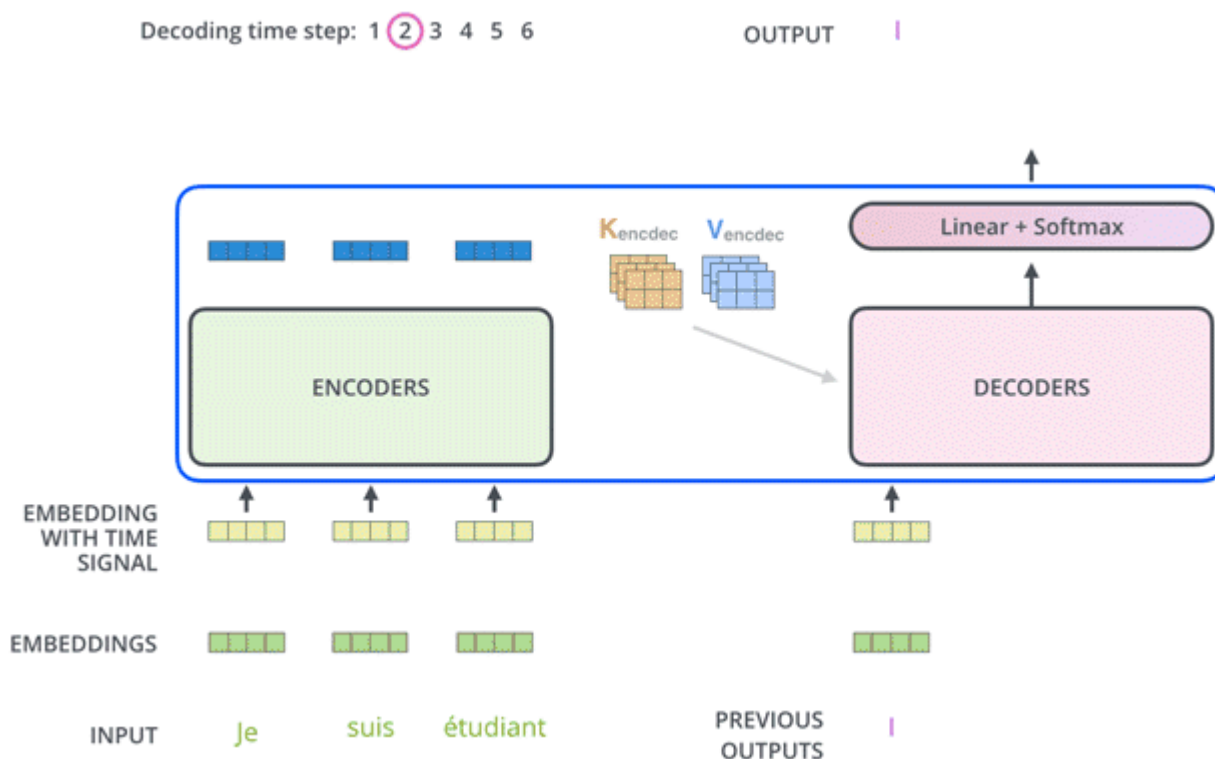


这个mask是Decoder中self-attention和Encoder中的self-attention唯一有区别的地方。经过了masked self-attention之后，decoder的每一个词都得到了**包含其以前信息的高阶融合embedding**。

2.3.3 第二个Multi-head Attention

For this layer, the encoder's outputs are keys and values, and the first multi-headed attention layer outputs are the queries. This process matches the encoder's input to the decoder's input, allowing the decoder to decide which encoder input is relevant to put a focus on.

每个decoder的masked self-attention输出的token embedding都去和encoder输出的高阶融合embedding做self-attention：



3. Q/A

(1) Transformer为什么需要进行Multi-head Attention?

原论文中说进行Multi-head Attention的原因是将模型分为多个头，形成多个子空间，可以让模型去关注不同方面的信息，最后再将各个方面的信息综合起来。其实直观上也可以想到，如果自己设计这样的一个模型，必然也不会只做一次attention，多次attention综合的结果至少能够起到增强模型的作用，也可以类比CNN中同时使用多个卷积核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。

(2) Transformer相比于RNN/LSTM，有什么优势？

- RNN系列的模型，并行计算能力很差，因为 T 时刻的计算依赖 $T-1$ 时刻的隐层计算结果。
- Transformer的特征抽取能力也比RNN系列的模型要好，使用了self-attention和多头机制来让源序列和目标序列自身的embedding表示所蕴含的信息更加丰富。

(3) Transformer如何并行化的？

Transformer的并行化我认为主要体现在self-attention模块。对于某个序列 $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n$ ，self-attention模块可以直接计算 $\boldsymbol{x}_i, \boldsymbol{x}_j$ 的点乘结果，并行得到每个token的表示。而RNN系列的模型就必须按照顺序从 \boldsymbol{x}_1 计算到 \boldsymbol{x}_n 。