图文详解Transformer为什么如此强大

韩冰 AI大模型实验室 2024-04-09 03:03 美国

在过去几年中，Transformer 在自然语言处理（NLP）领域引起了巨大关注。现在，它们在 NLP 之外的领域也得到了成功使用。

Transformer 之所以如此强大，关键在于它的“注意力（Attention）模块”。这个模块能够理解文本序列中的每个词与其他词之间的关系。

但大家最关心的问题是，Transformer 究竟是如何做到这一点的？

在本文中，我们将尝试解答这个问题，并理解为什么 Transformer 会进行这样的计算过程。

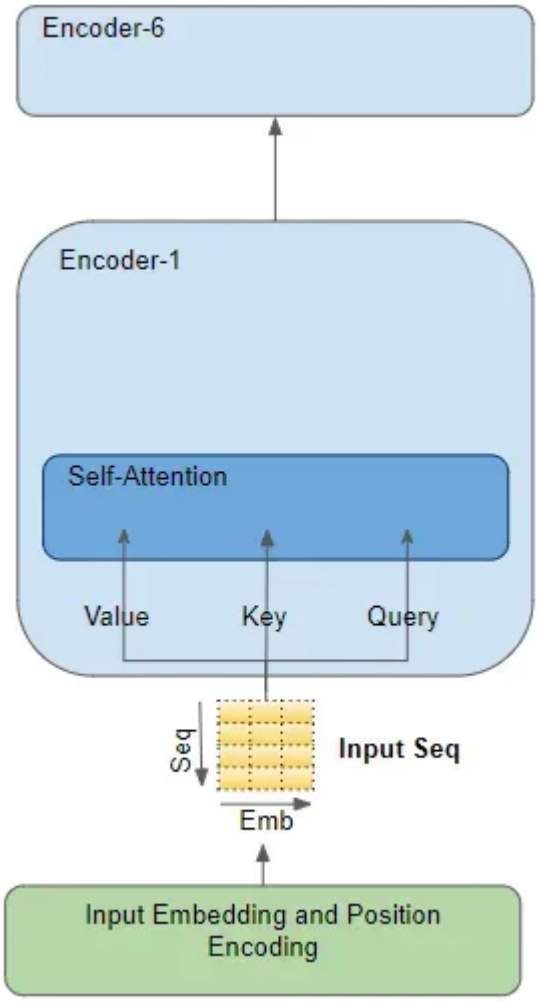
在我之前的一系列关于 Transformer 的文章中，我们已经深入探讨了其架构，并逐步了解了它们在训练和推理阶段的工作原理。我们还深入探索了它们的内部机制，以详细了解它们是如何运作的。

我们追求的目标，不仅是理解事物的工作原理，更要理解它为何以这种方式运作。

要了解 Transformer 的运行原理，我们需要重点关注它的注意力（Attention）机制。我们从探究输入到注意力模块的数据开始，然后分析它是如何处理这些输入的。

#01
输入序列是怎样传送到注意力模块的

注意力模块存在于每个编码器（Encoder）和每个解码器（Decoder）之中。我们先来仔细看看编码器中的注意力机制。



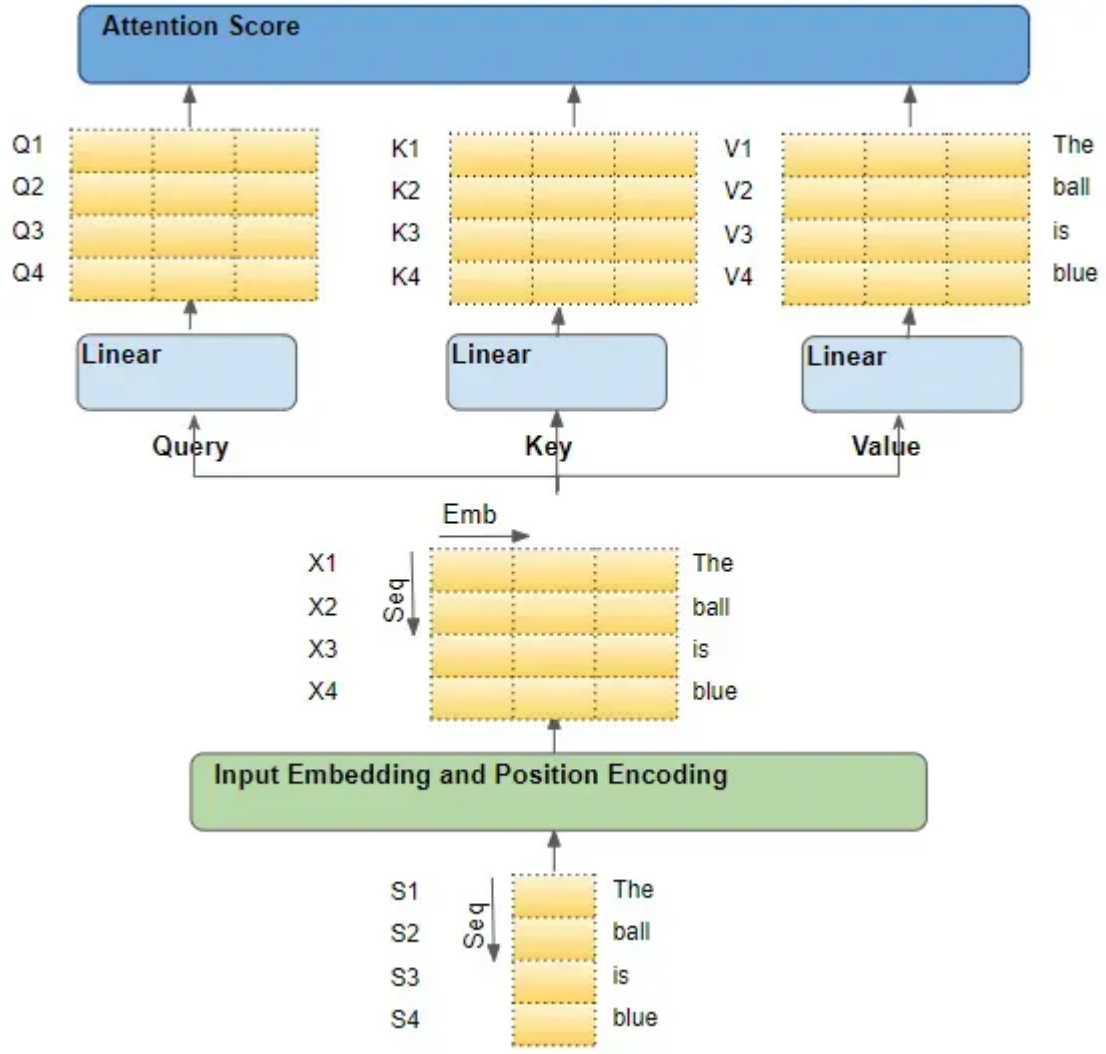
编码器中的注意力机制

以一个例子来说明，假设我们正在进行一项从英语到西班牙语的翻译任务，其中一个样本的源序列是 “The ball is blue”，目标序列是 “La bola es azul”。

源序列首先经过一个嵌入和位置编码层，这个层会为序列中的每个词创建一个嵌入向量。这些嵌入向量随后被送入编码器，首先到达的就是注意力模块。

在注意力模块里，嵌入的序列会通过三个线性层，生成三个不同的矩阵 —— 分别称为查询（Query）、键（Key）和值（Value）。这三个矩阵是用来计算注意力分数的关键。

我们需要明确的是，这些矩阵中的每一行都对应着源序列中的一个词。



源序列的流程

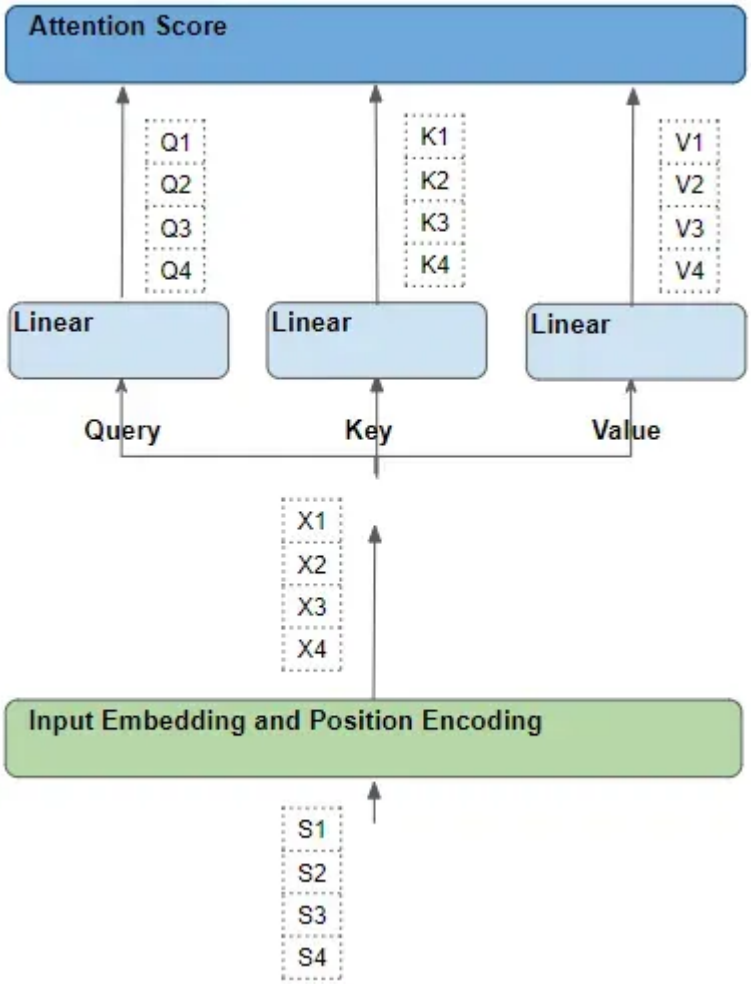
#02
每个输入行都是源序列中的一个词

要深入了解驱动 Transformer 的核心机制，我们需要专注于其注意力（Attention）机制。这涉及从源序列中的每个单词出发，追踪它们在 Transformer 体系中的流转。我们特别关注的是，注意力模块内部发生了什么。

这有助于我们清楚地了解源序列和目标序列中的每个单词是如何与其他单词进行交互的。

在解释过程中，请专注于对每个单词进行的操作，以及每个向量是如何与原始输入单词相映射的。如果某些细节，比如矩阵的形状、算术运算的细节、多个注意力头等，与单词的具体运算路径无直接关联，我们就可以先不考虑它们。

为了简化解释和可视化，我们先忽略嵌入维度，只关注每个单词的行。

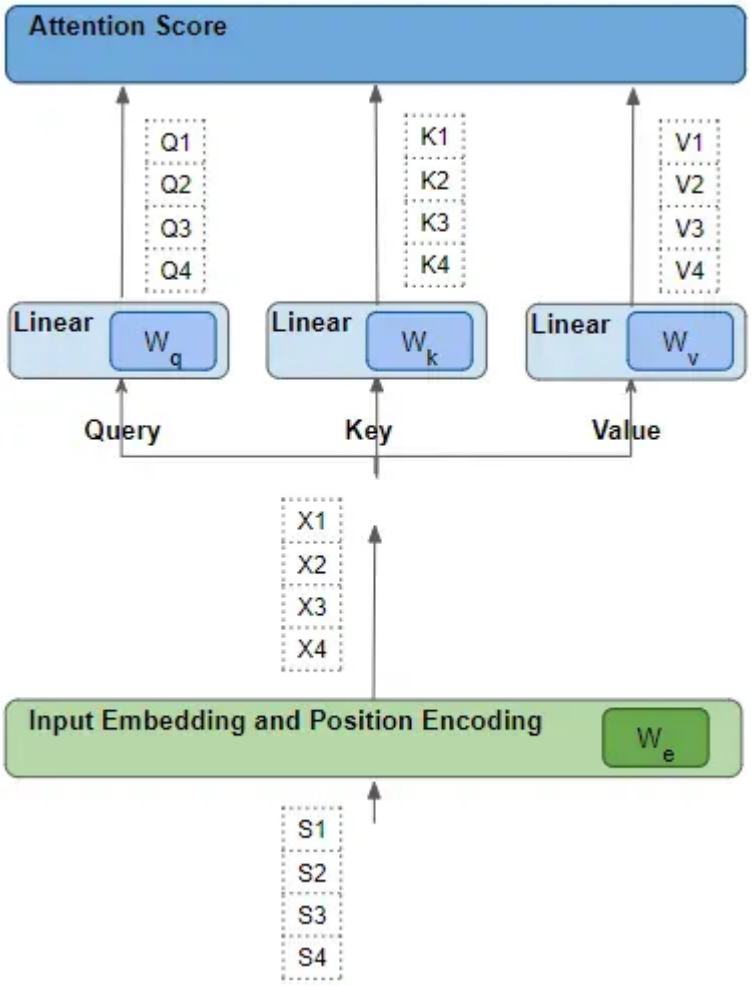


源序列中每个单词的流程

#03
每个单词都经过了一系列可学习的变换

每一行都是由一系列变换生成的，这些变换包括嵌入、位置编码和线性层。

所有这些变换都是可训练的操作。这意味着这些操作中使用的权重并非预设的，而是通过模型学习得到的，以便产生期望的输出预测。



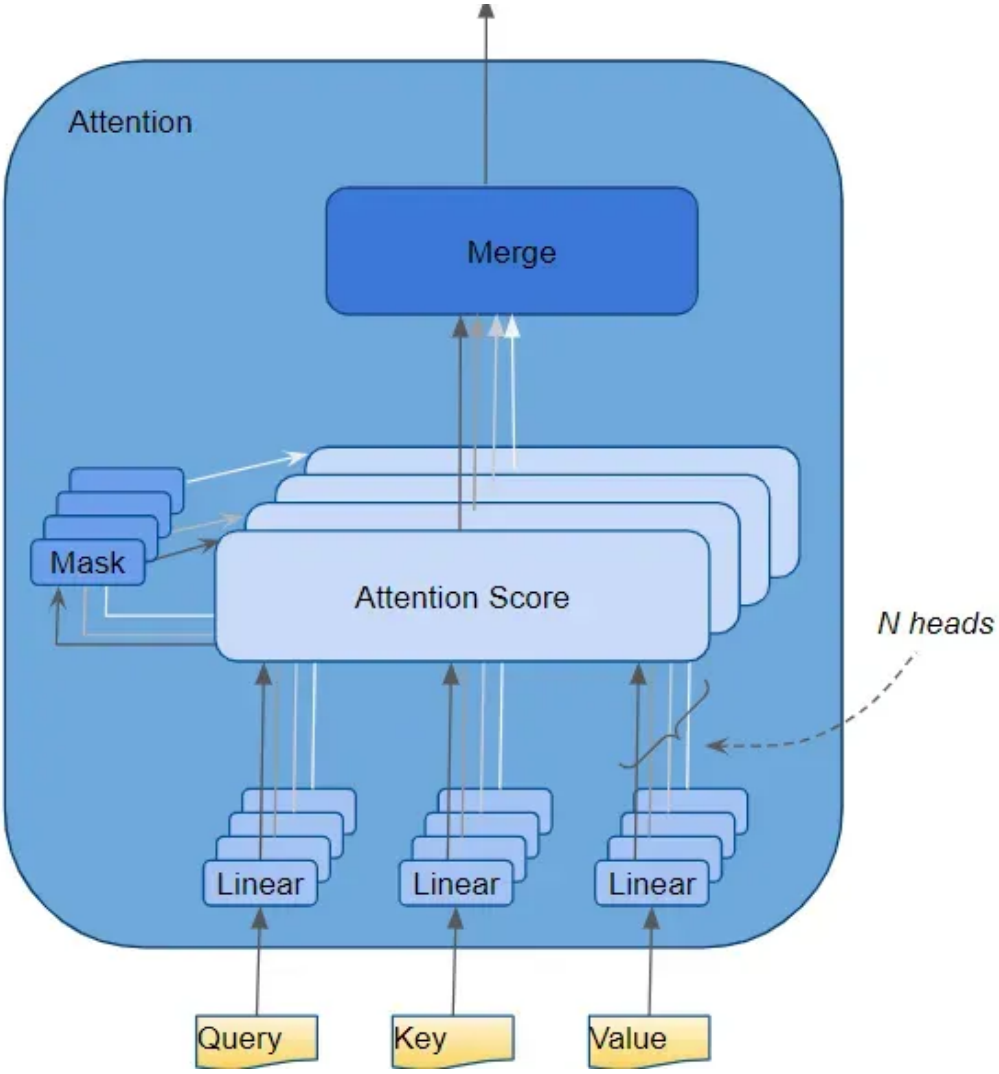
学习线性和嵌入权重

一个关键问题是，Transformer 是如何确定哪一套权重能够获得最佳结果的？这是一个我们稍后需要回过头来深入讨论的问题。

#04

注意力分数：查询（Query）和键（Key）单词之间的点积

注意力机制包含几个步骤，但我们这里主要关注线性层和注意力分数。



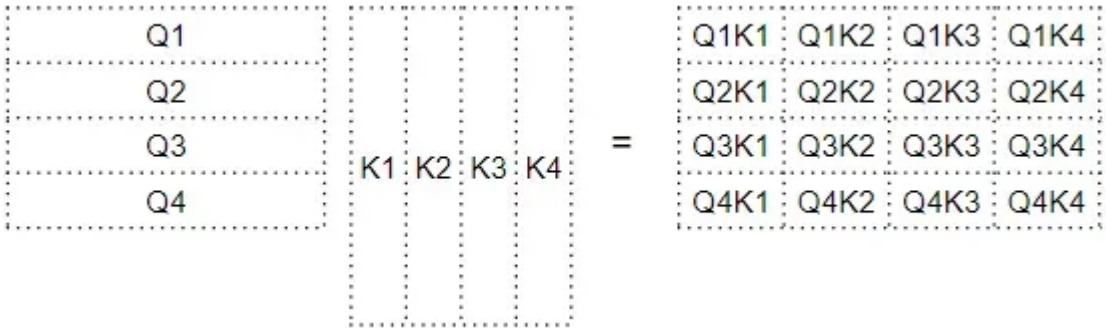
多头注意力

$$Z = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

注意力分数计算

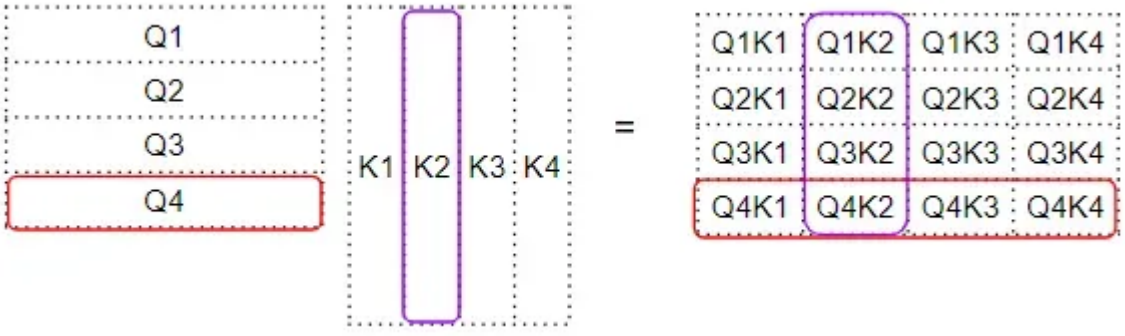
如公式所示，注意力机制的第一步是在查询（Q）矩阵和键（K）矩阵的转置之间进行矩阵乘法（即点积）。观察每个单词的变化。

我们得到了一个中间矩阵（可以称之为“因子”矩阵），其中每个单元是两个单词间的矩阵乘法结果。



查询矩阵和关键矩阵之间的点积

比如说，第四行中的每一列代表了第四个查询单词与每个键单词之间的点积结果。

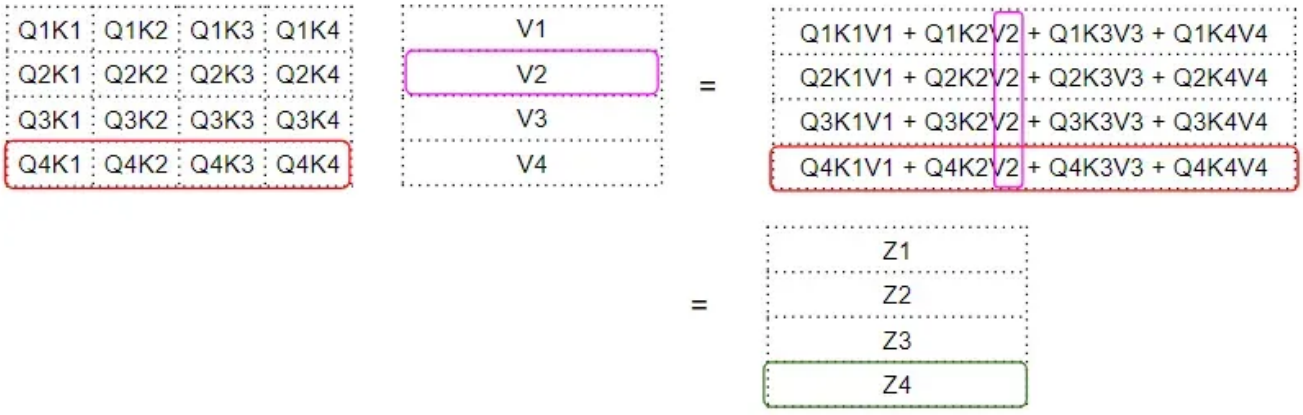


查询矩阵和关键矩阵之间的点积

#05

注意力分数：查询 - 键（Query-Key）和值（Value）单词之间的点积

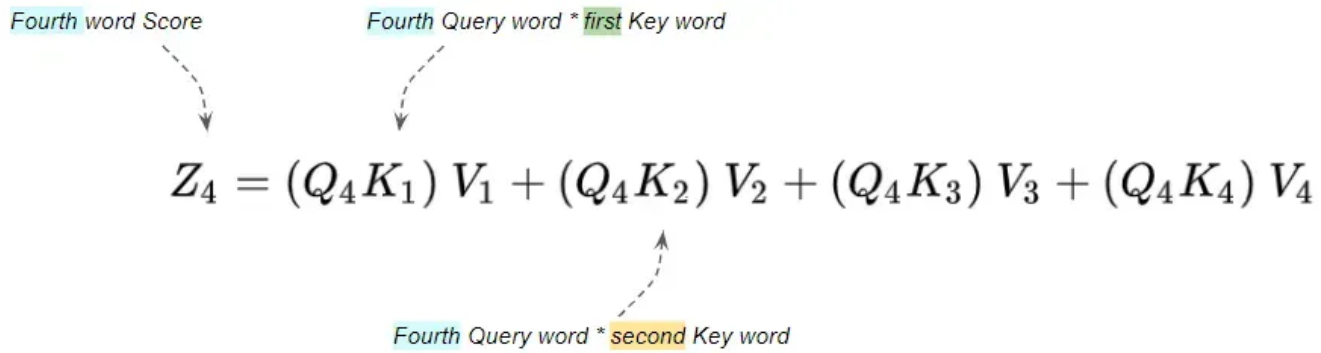
接下来的步骤是在这个中间“因子”矩阵和值（V）矩阵之间进行矩阵乘法，从而产生由注意力模块输出的注意力分数。这里我们可以看到，第四行是第四个查询单词矩阵与所有其它键和值单词的乘积结果。



查询键和值矩阵之间的点积

这样就产生了注意力模块输出的注意力分数向量（Z）。

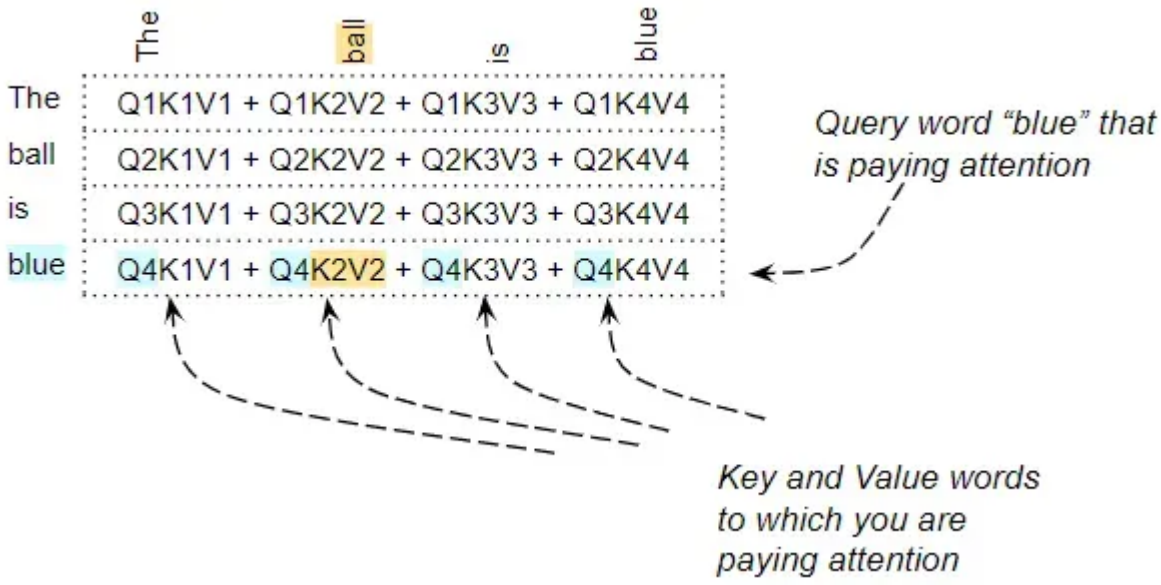
我们可以这样来理解输出分数：对于每个单词，它是“值（Value）”矩阵中每个单词的编码值，经过“因子”矩阵加权处理。这个因子矩阵是针对特定单词的查询（Query）值和所有单词的键（Key）值的点积。



注意力分数是值词的加权总和

#06
查询（Query）、键（Key）和值（Value）单词的作用是什么？

查询单词可以被理解为我们正在计算注意力的那个单词。键和值单词是我们关注的对象，即这些单词对查询单词有多大的相关性。



“blue”一词的注意力分数会关注所有其他单词

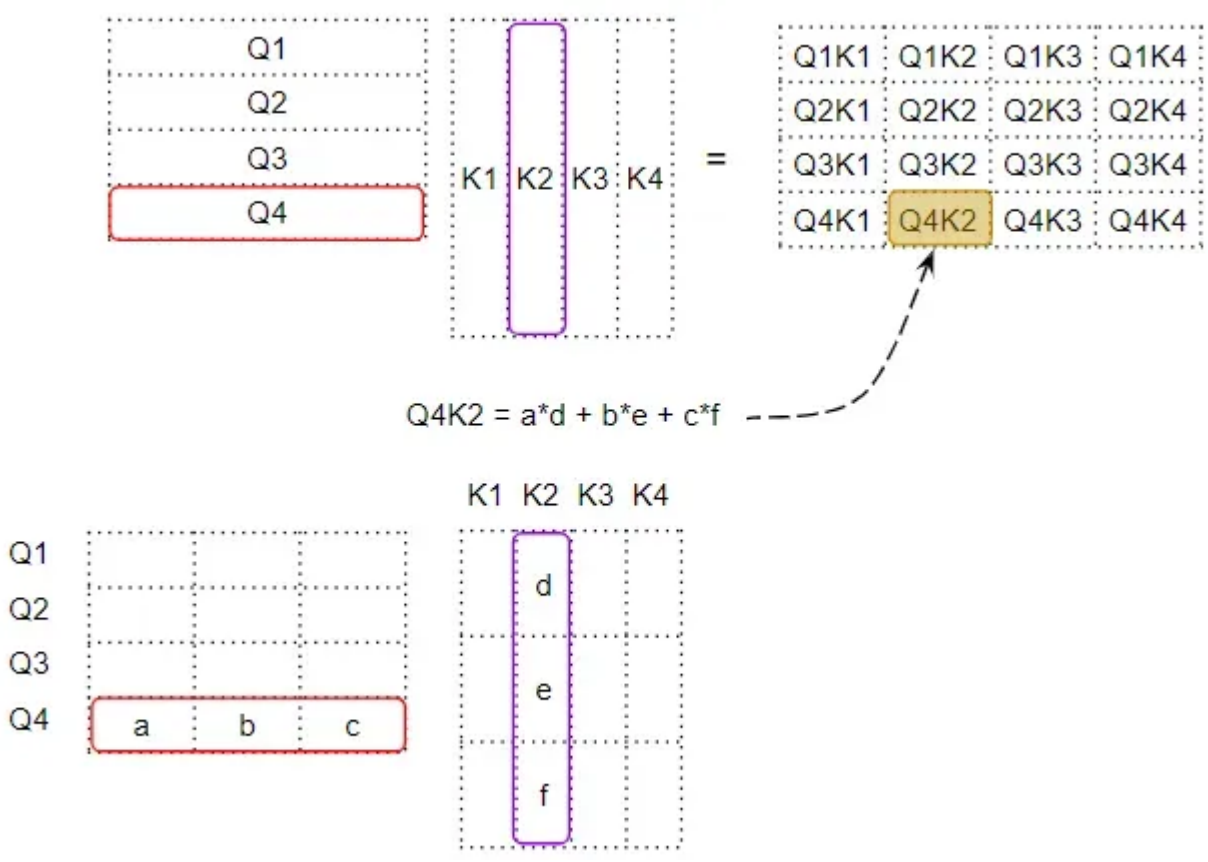
举个例子，对于句子“The ball is blue”，“blue”这个词的行将包含它与其他每个单词的注意力分数。这里，“blue”是查询单词，其他单词则是“键 / 值”。

尽管还有其他操作，如除法和 softmax，但在本文中我们可以暂时忽略它们。它们只是改变矩阵中的数值，但并不影响每个单词行在矩阵中的位置，也不涉及单词之间的互动。

#07
点积揭示了单词间的相似度

我们已经看到，注意力分数捕捉了一个特定单词与句子中其他每个单词之间的交互，通过点积运算后再求和。但矩阵乘法如何帮助 Transformer 判断两个单词之间的相关性呢？

要理解这一点，记住查询、键和值的行实际上是具有嵌入维度的向量。让我们更详细地看看这些向量之间的矩阵乘法是如何计算的。



每个单元格都是两个词向量之间的点积

当我们对两个向量进行点积时，我们会将数对相乘，然后将结果相加。

- 如果两个配对数字（例如‘a’和‘d’）都是正数或都是负数，那么乘积会是正的，从而增加最终总和。
- 如果一个数字是正数而另一个是负数，则乘积会是负的，从而减少最终总和。

- 如果乘积是正数，这两个数字越大，它们对最终总和的贡献越大。

这意味着，如果两个向量中对应的数字的符号相同，那么最终的总和会更大。

#08
Transformer 如何学习单词间的相关性？

点积的原理同样适用于注意力分数。如果两个单词的向量更加对齐，它们的注意力分数就会更高。

那么，我们希望 Transformer 呈现出什么样的行为呢？

我们希望对于句子中彼此相关的两个单词，注意力分数较高。而对于彼此无关的两个单词，则希望分数较低。

比如，在句子 “The black cat drank the milk” 中，单词 “milk” 与 “drank” 非常相关，与 “cat” 稍微不那么相关，而与 “black” 无关。我们希望 “milk” 和 “drank” 之间产生高分数，“milk” 和 “cat” 之间产生略低的分数，而 “milk” 和 “black” 之间的分数则接近于零。

这就是我们希望模型学会输出的结果。

为了做到这一点，“milk” 和 “drank” 的向量必须对齐。“milk” 和 “cat” 的向量将略有差异。而 “milk” 和 “black” 的向量则会相差甚远。

让我们回到之前提到的问题 ——Transformer 如何确定哪一组权重能带来最佳效果？

单词向量是基于单词嵌入和线性层的权重生成的。因此，Transformer 可以通过学习这些嵌入和线性层的权重，来产生所需的单词向量。

换句话说，它会以这样的方式学习这些嵌入和权重：如果句子中的两个单词彼此相关，那么它们的向量就会对齐，从而产生较高的注意力分数。而对于彼此不相关的单词，它们的向量就不会对齐，从而产生较低的注意力分数。

因此，“milk” 和 “drank” 之间的嵌入将非常对齐，产生较高的分数。而 “milk” 和 “cat” 之间的嵌入会有所差异，产生略低的分数；“milk” 和 “black” 之间的嵌入则会相差甚远，产生较低的分数。

这就是注意力模块背后的原理。

#09

总结：是什么让 Transformer 动起来的？

查询（Query）和键（Key）之间的点积计算出每对单词之间的相关性。然后，这种相关性被用作一个“因子”，以计算所有值（Value）单词的加权总和。这个加权总和就是作为注意力分数输出的。

Transformer 以这样一种方式学习嵌入等，即让彼此相关的单词更加对齐。

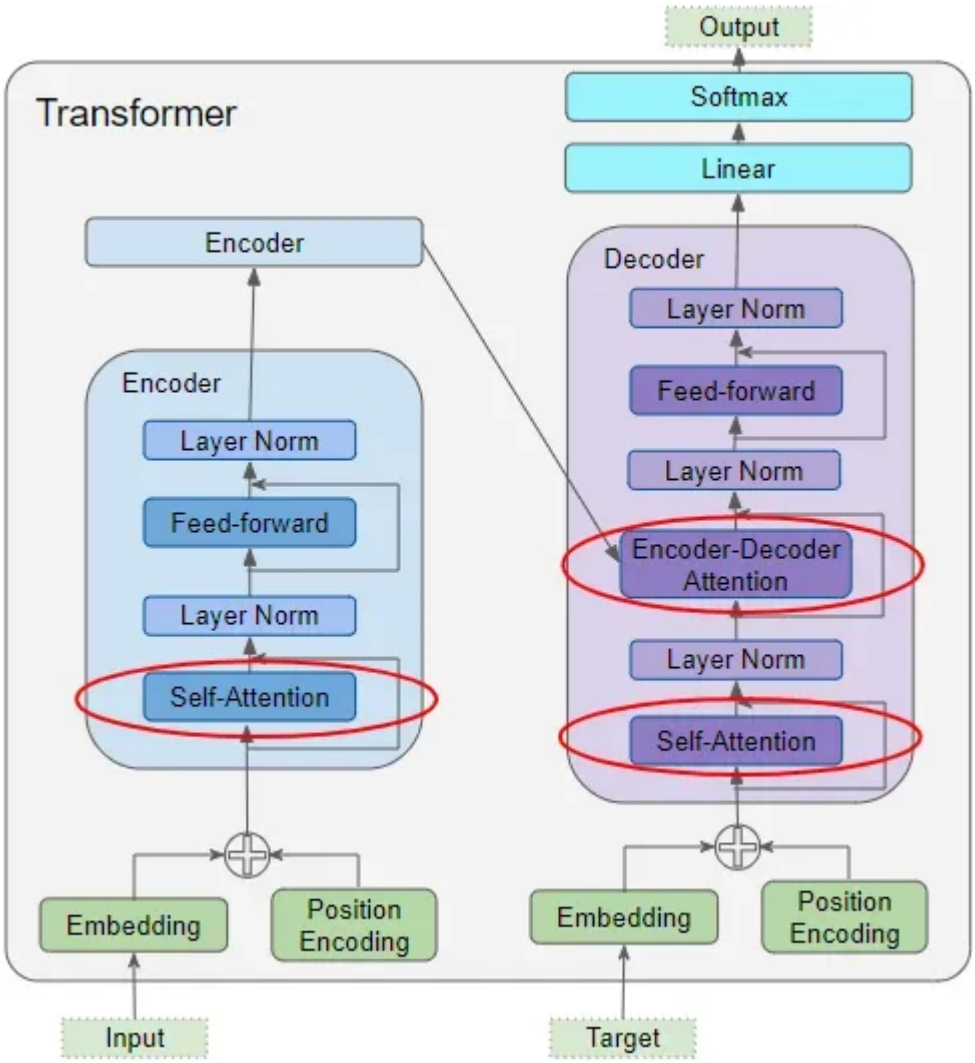
这正是引入三个线性层并为查询、键和值生成输入序列的三个版本的原因之一。这为注意力模块提供了更多可以学习的参数，以优化单词向量的创建。

#10

Transformer 中的编码器自注意力

在 Transformer 中，注意力被用于三个地方：

- 编码器中的自注意力 —— 源序列关注自身
- 解码器中的自注意力 —— 目标序列关注自身
- 解码器中的编码器 - 解码器注意力 —— 目标序列关注源序列



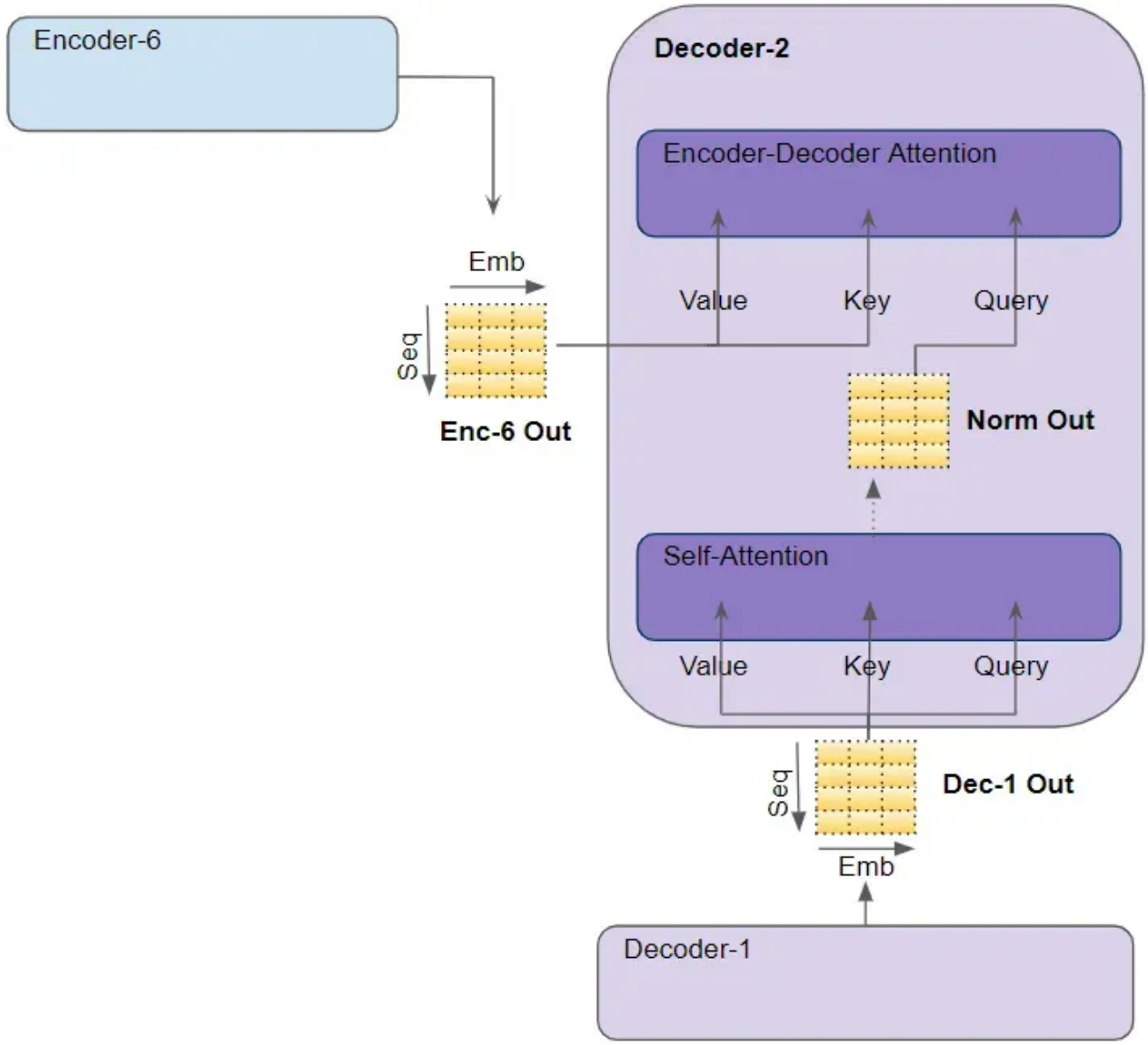
Transformer 中的注意力

在编码器自注意力中，我们计算源句中每个单词与源句中其他单词的相关性。这在所有编码器的堆栈中都会发生。

#11

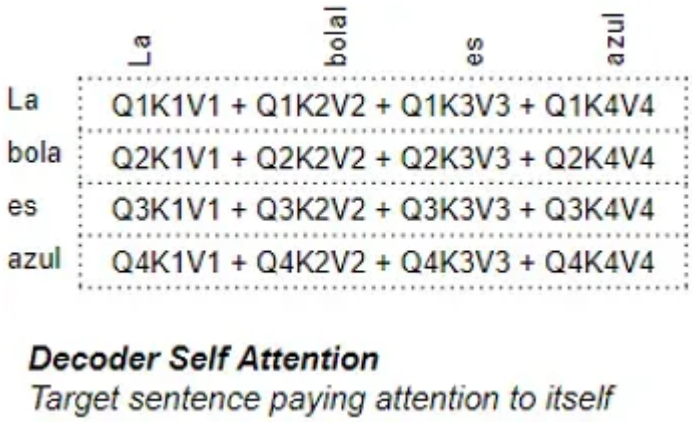
Transformer 中的解码器自注意力

我们刚才在编码器自注意力中所看到的大部分内容也同样适用于解码器中的注意力，但存在一些小的、但重要的差异。



解码器中的注意力

在解码器自注意力中，我们计算目标句中每个单词与目标句中其他单词的相关性。

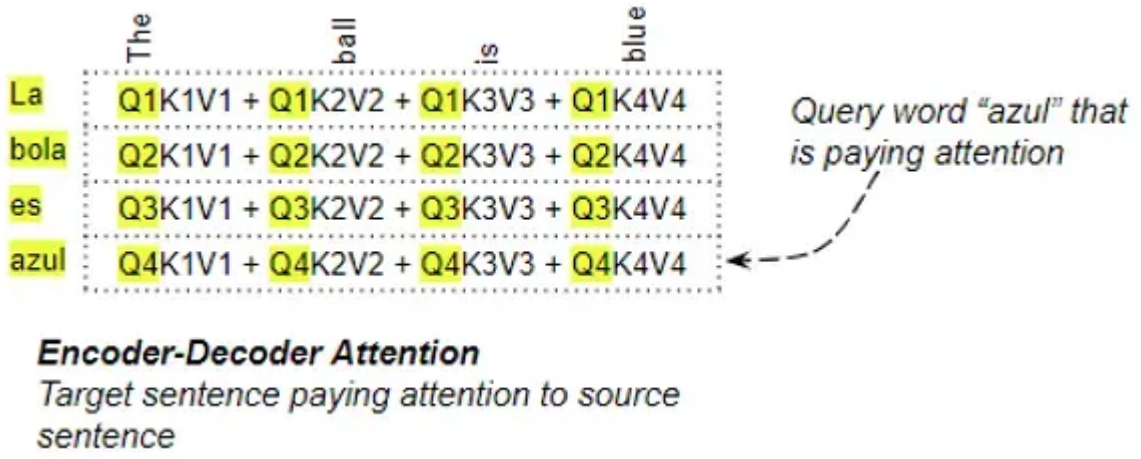


解码器自注意力

#12

Transformer 中的编码器 - 解码器注意力

在编码器 - 解码器注意力中，查询来自目标句，而键 / 值来自源句。因此，它计算了目标句中每个单词与源句中每个单词的相关性。



编码器 - 解码器注意力

#13

结论

希望这篇文章能让你对 Transformer 设计的巧妙之处有一个清晰的理解。也请阅读本系列的其他三篇文章：

- 《图解 Transformer 架构设计》
- 《图解 Transformer 工作原理》
- 《图解 Transformer 多头注意力机制》

以深入了解为什么 Transformer 成为了如此多深度学习应用的首选架构。

原文链接：https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

现在大模型都是基于 Transformer 构建的，要了解大模型必须要了解 Transformer，欢迎进入 AI 大模型实验室微信群一起学习 Transformer。