图解Transformer工作原理

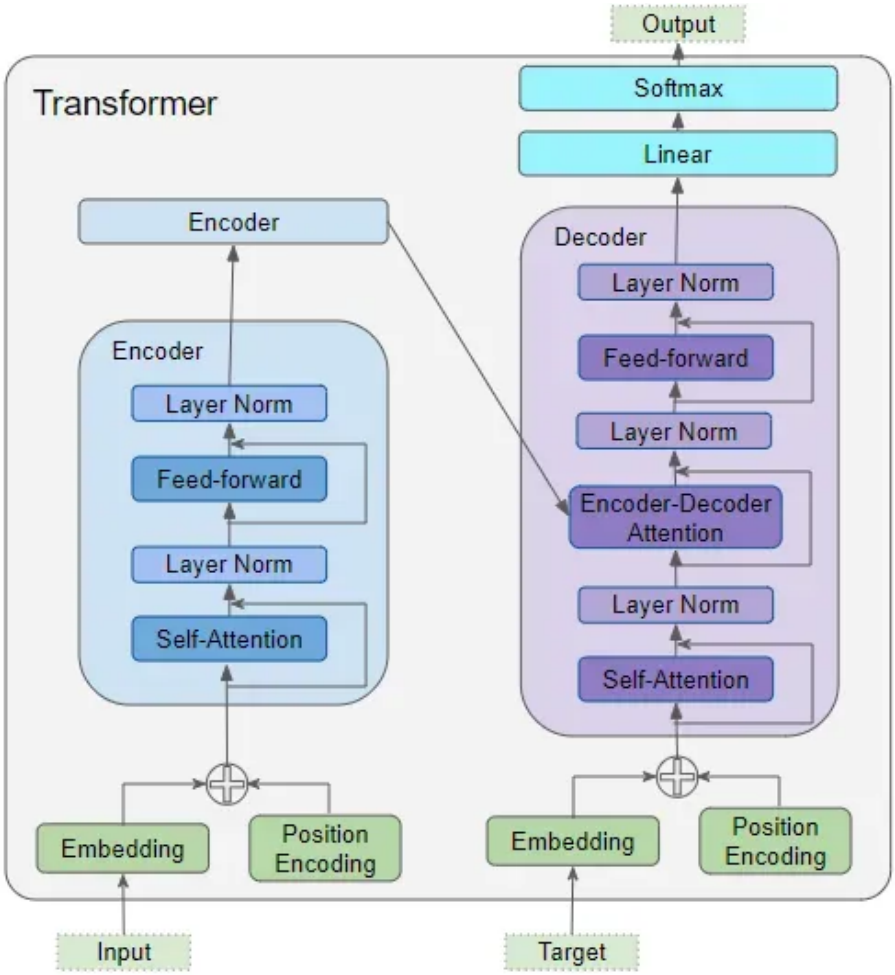
韩冰 AI大模型实验室 2024-03-27 07:06 美国

这是关于 Transformer 系列文章的第二篇。在首篇中，我们了解了 Transformer 的基本功能、使用场景、高级架构及其优点。

本文深入探讨 Transformer 的内部工作机制。我们将探究数据是如何在这个系统中流动的，包括它们的矩阵表示形式以及在每个阶段进行的计算。

#01
架构概览

正如我们在第一部分所见，Transformer 的架构主要由以下部分构成：



编码器（Encoder）和解码器（Decoder）的数据输入，包括：

- 嵌入层（Embedding layer）
- 位置编码层（Position Encoding layer）

编码器堆栈含有若干编码器，每个编码器包含：

- 多头注意力层（Multi-Head Attention layer）
- 前馈层（Feed-forward layer）

解码器堆栈也含有若干解码器，每个解码器包含：

- 两个多头注意力层
- 前馈层

输出部分（位于右上方）—— 负责生成最终输出，包括：

- 线性层（Linear layer）
- Softmax 层

为了更好地理解每个组件的作用，我们将跟随 Transformer 在解决翻译问题的训练过程中，逐步探索其工作原理。我们将以一个训练样本为例，该样本包含一组输入序列（英语中的 “You are welcome”）和目标序列（西班牙语中的 “De nada”）。

#02
嵌入和位置编码

作为一个自然语言处理（NLP）模型，Transformer 需要了解每个单词的两个方面 —— 单词的含义和它在序列中的位置。

- 嵌入层用于编码单词的含义。
- 位置编码层则用来表示单词的位置。

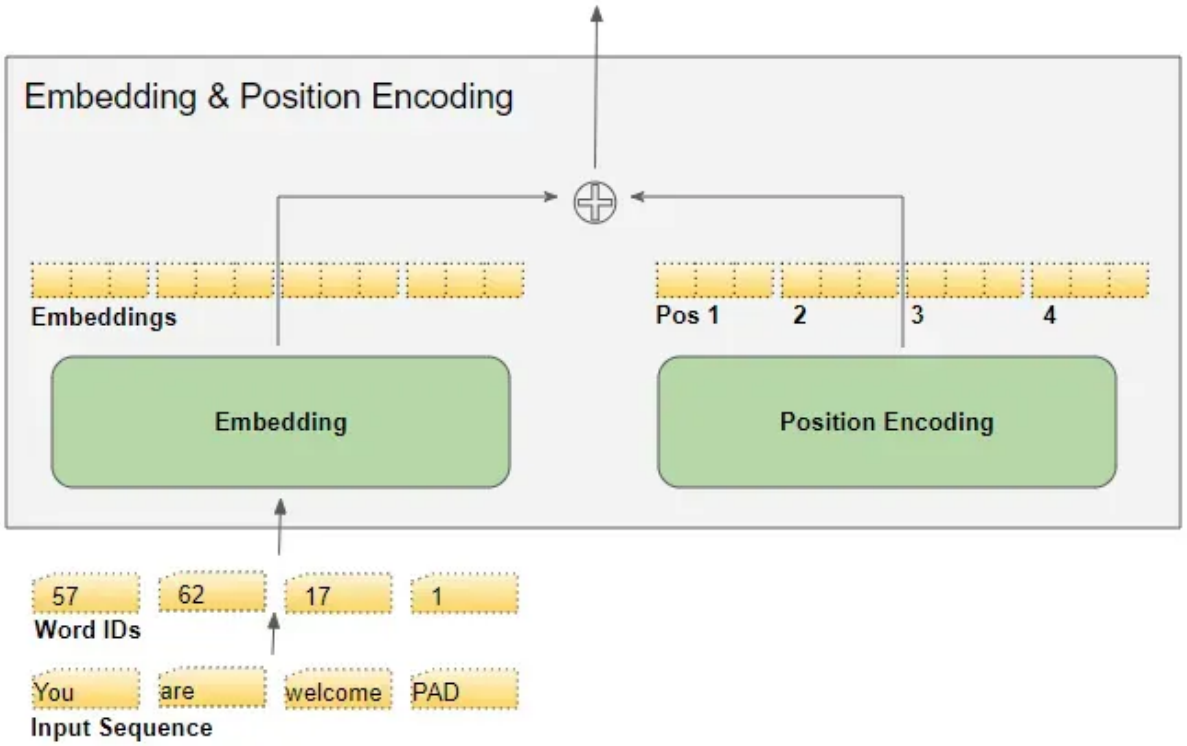
Transformer 通过相加的方式将这两种编码结合起来。

嵌入过程

Transformer 设计了两个嵌入层。输入序列首先进入被称为输入嵌入层的第一个嵌入层。

而目标序列则在向右移动一个位置并在开头插入一个开始标记后，输入到第二个嵌入层。需要注意的是，在推理阶段，由于没有目标序列，我们会将输出序列循环地输入到这第二个嵌入层，正如我们在第一篇文章中了解到的。因此，这个嵌入层被称为输出嵌入层。

文本序列首先被转换为数字化的单词 ID，这一过程是基于我们的词汇表完成的。然后，嵌入层会将每个输入单词转换成一个嵌入向量，这个向量更丰富地表达了该单词的含义。



位置编码

在递归神经网络（RNN）中，由于每个单词是按顺序逐个输入的，网络可以隐式地识别出每个单词的位置。

但 Transformer 不使用 RNN，而是将序列中的所有单词并行输入。这是其相比于 RNN 架构的一大优势，但也导致了位置信息的丢失，因此需要通过额外的方式补充这一信息。

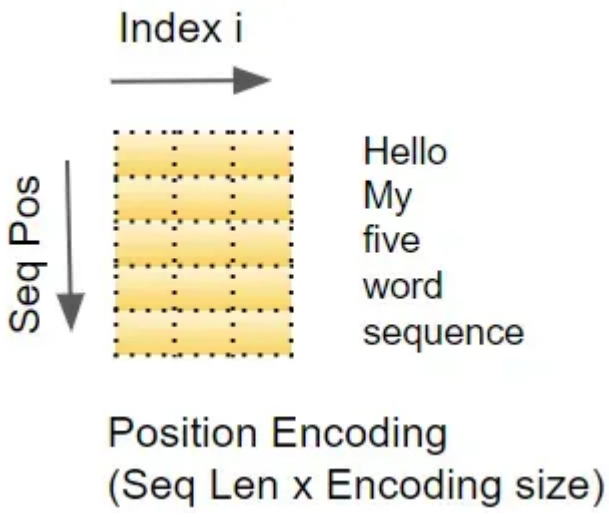
Transformer 同样设计了两个位置编码层，这些位置编码与输入序列无关，它们是根据序列的最大长度而固定的。举例来说：

- 第一项编码指示序列中的第一个位置
- 第二项编码指示第二个位置
- 以此类推

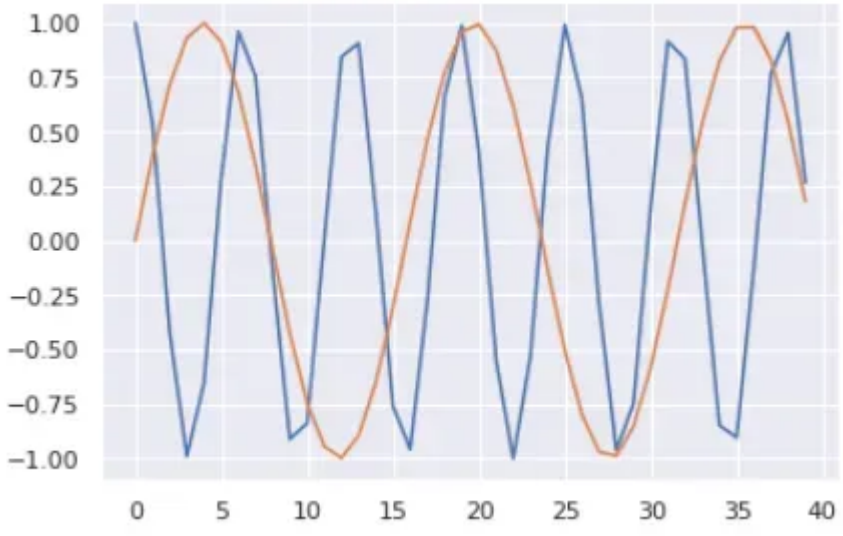
这些位置编码是通过以下公式计算得出的：

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- pos 表示序列中单词的位置
- d_model 是编码向量的长度（与嵌入向量长度相同）
- i 是向量中的索引值



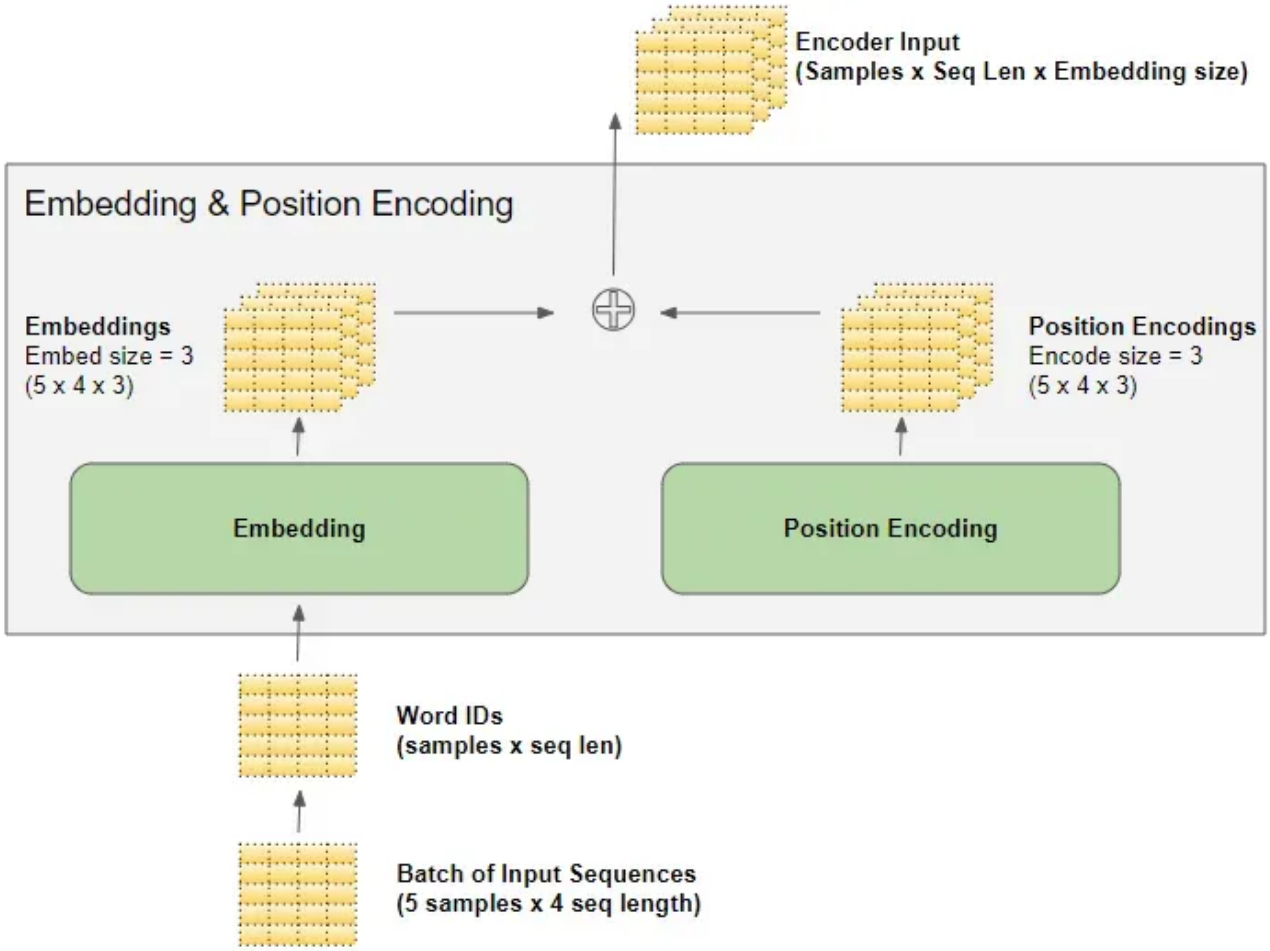
换句话说，这个编码过程就像是将正弦波和余弦波交织在一起，偶数索引处使用正弦值，奇数索引处使用余弦值。比如，对于一个包含 40 个单词的序列，我们可以看到几个不同位置和编码索引组合的编码值。



蓝色曲线展示了所有 40 个单词位置的第 0 索引的编码情况，而橙色曲线则显示了第 1 索引的编码情况。对于其他索引值，也存在类似的曲线。

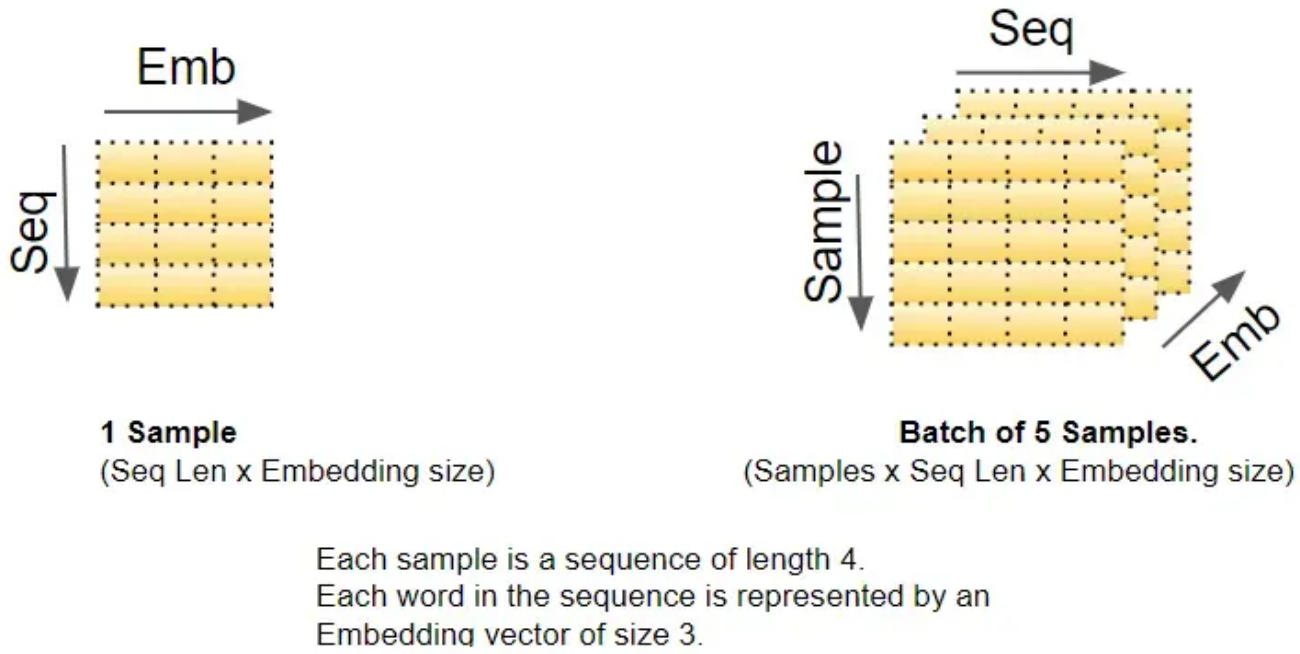
#03
矩阵维度

深度学习模型通常一次处理一批训练样本。嵌入层和位置编码层操作的是代表一批序列样本的矩阵。嵌入层接收形状为（样本数， 序列长度）的单词 ID 矩阵，并将每个单词 ID 编码成一个词向量，这些向量的长度即嵌入大小，从而形成一个（样本数，序列长度，嵌入大小）的输出矩阵。位置编码层使用与嵌入层相同大小的编码，因此产生一个形状相同的矩阵，可以与嵌入矩阵相加。



这种（样本数，序列长度，嵌入大小）的矩阵形状在整个 Transformer 中保持不变，数据会流经编码器和解码器堆栈，直到最终的输出层进行形状调整。

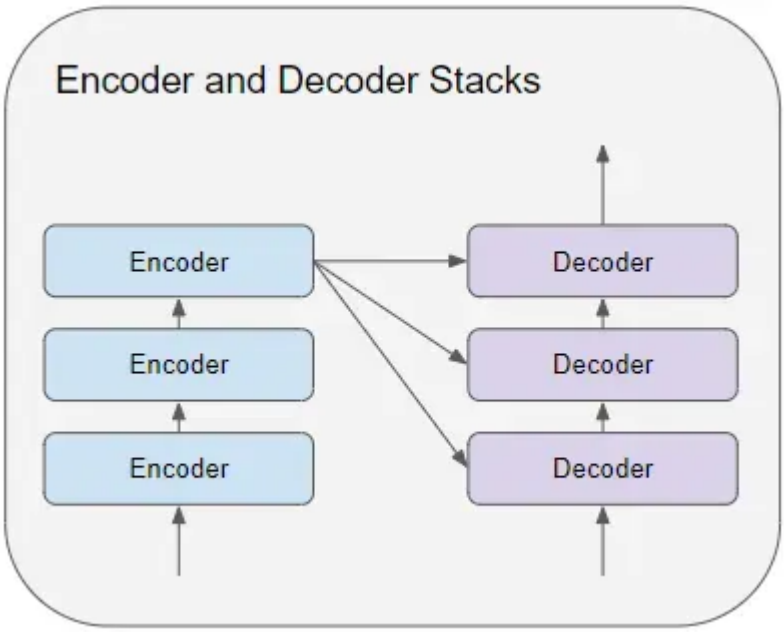
这段描述帮助我们理解了 Transformer 中三维（3D）矩阵的维度构成。然而，为了使可视化过程更为简单明了，从这里开始，我们将不再考虑第一个维度（即样本的维度），转而使用二维（2D）的方式来展示单个样本。



输入嵌入将其输出发送到编码器。类似地，输出嵌入输入到解码器。

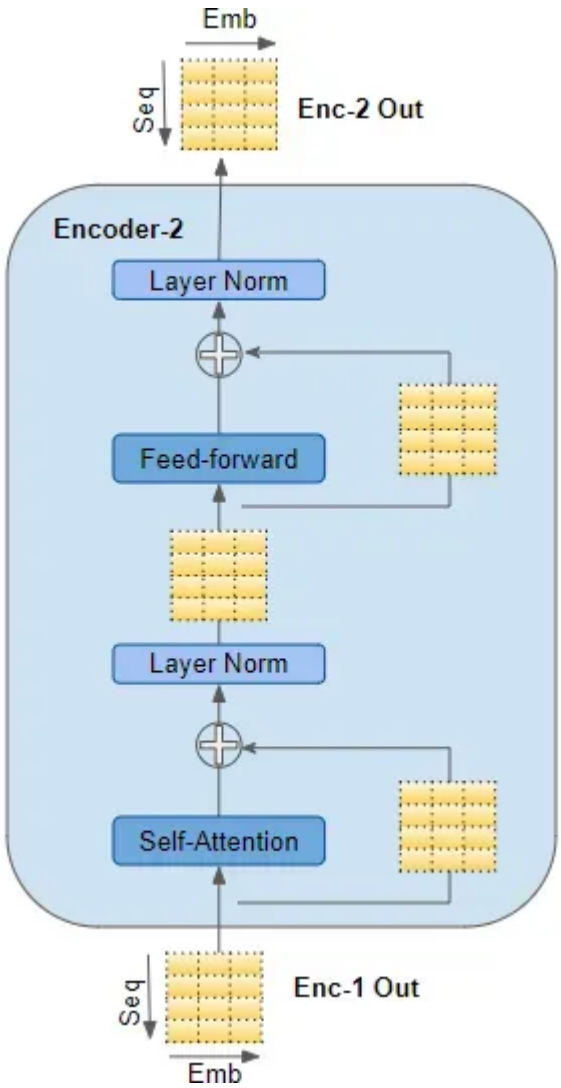
#04
编码器

编码器和解码器堆栈由数个（通常为六个）编码器和解码器依次串联而成。



在 Transformer 的编码器堆栈中，第一个编码器从嵌入层和位置编码层接收输入，而其余的编码器则从上一个编码器接收输入。

编码器首先将输入送入一个多头自注意力层，该层的输出随后传递到一个前馈层，进而将输出传递给下一个编码器。



这些自注意力和前馈子层周围都包含了残差跳跃连接，接着是层归一化处理。

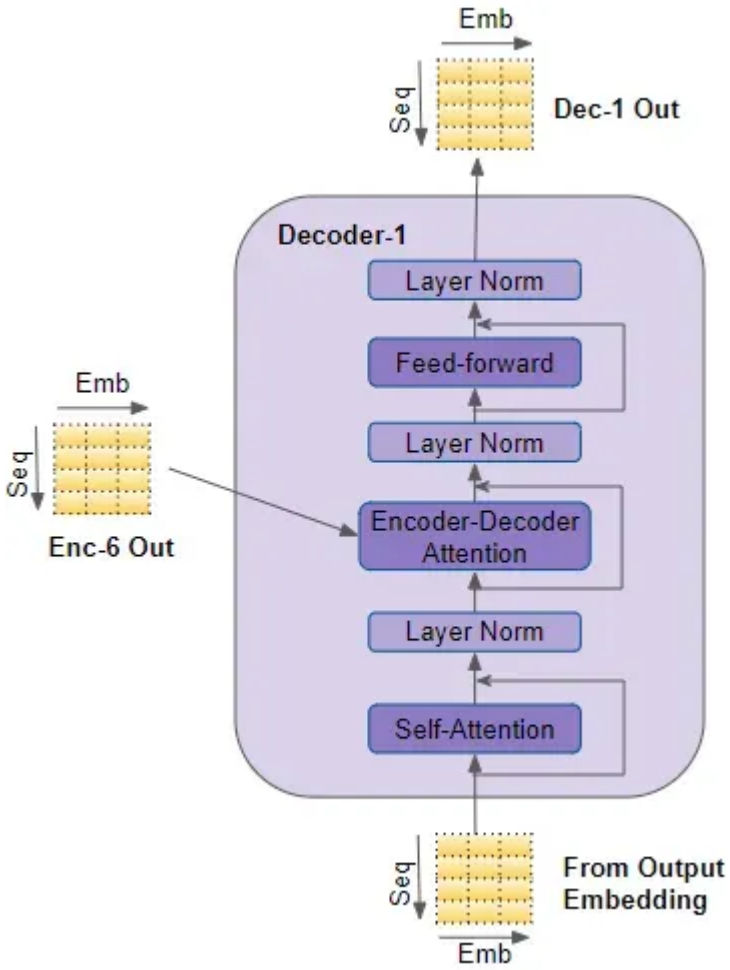
最后一个编码器的输出会被送入解码器堆栈中的每一个解码器，如下所述。

#05
解码器

解码器的结构与编码器相似，但也有一些区别。

与编码器一样，堆栈中的第一个解码器从输出嵌入层和位置编码层接收输入。堆栈中的其他解码器从前一个解码器接收输入。

解码器将其输入传递到一个多头自注意力层。这与编码器中的自注意力层略有不同。解码器的自注意力层只允许关注序列中较早的位置。这是通过对未来位置进行掩码处理来实现的，我们将很快讨论这一点。



与编码器不同，解码器增加了一个称为编码器 - 解码器注意力层（Encoder-Decoder attention layer）的第二多头注意力层。这一层的工作方式与自注意力（Self-attention）相似，但它结合了两种输入：位于其下方的自注意力层的输出以及编码器堆栈的输出。

自注意力层的输出先被送入一个前馈层（Feed-forward layer），然后前馈层再将输出传递到下一个解码器层。

在这些子层中，无论是自注意力、编码器 - 解码器注意力还是前馈层，它们都配备了一种称为残差跳跃连接（residual skip-connection）的机制，紧接着是一层用于数据标准化的层正规化处理（Layer-Normalization）。

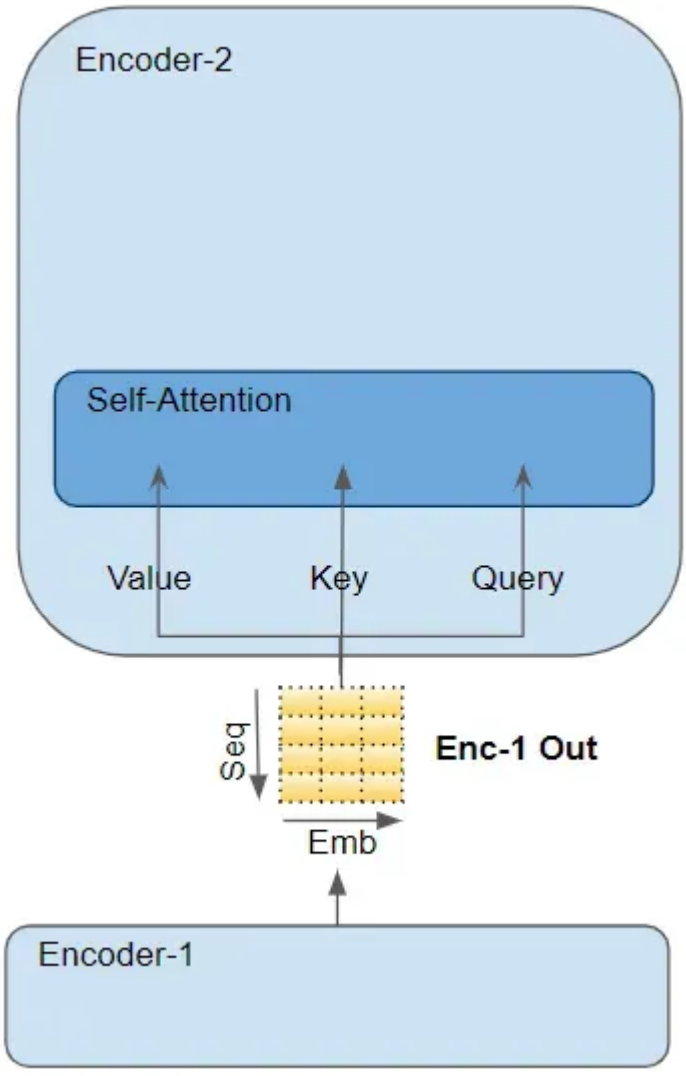
#06
注意力机制

在第一部分中，我们探讨了在处理序列数据时，为何注意力机制（Attention）至关重要。在 Transformer 模型中，注意力机制有三个主要的应用场景：

- 编码器的自注意力：输入序列对自身进行深入分析。
- 解码器的自注意力：目标序列对自己进行分析。
- 解码器中的编码器 - 解码器注意力：目标序列聚焦于输入序列。

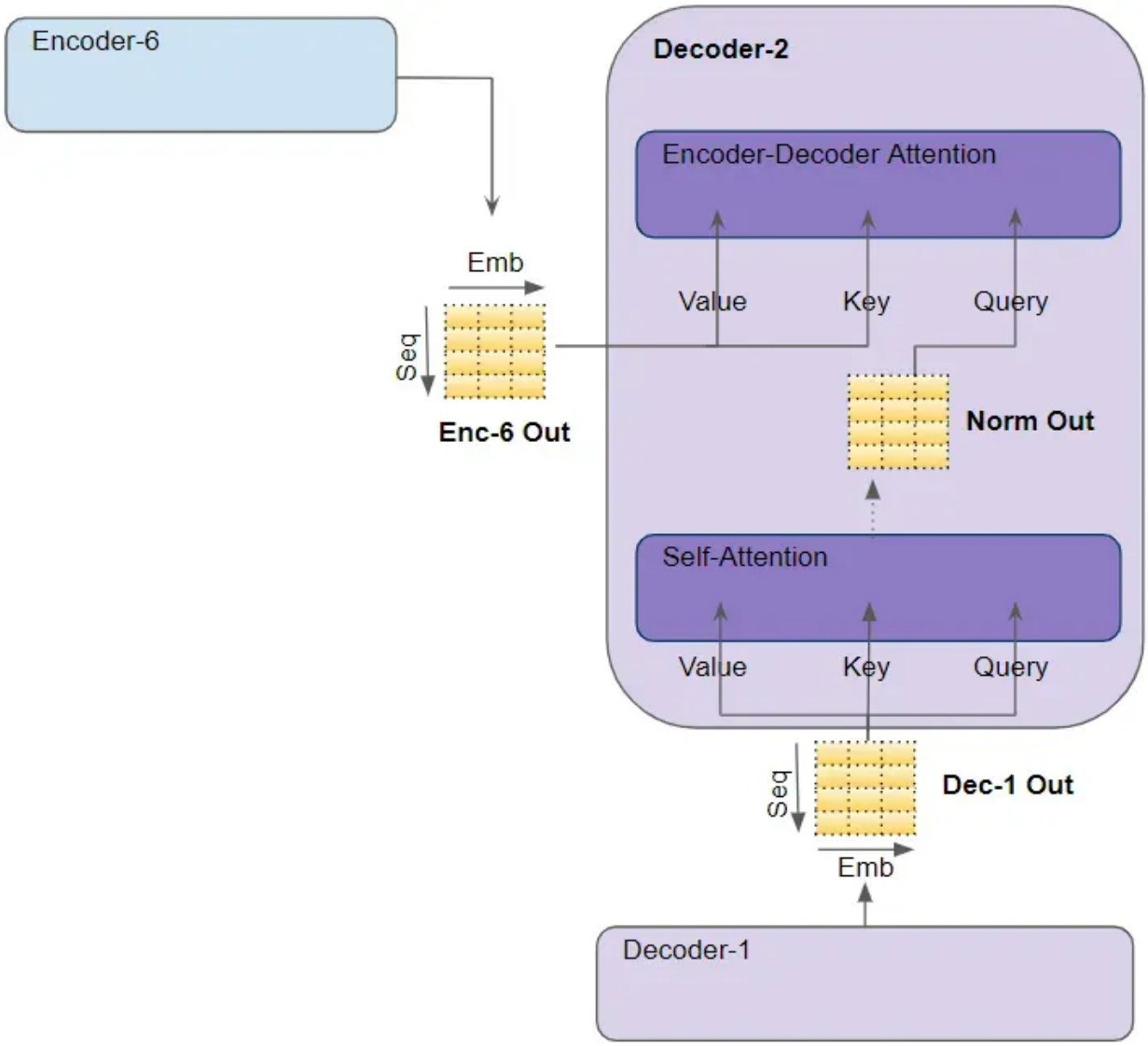
注意力层通过三个关键参数接收输入，这些参数分别是查询（Query）、键（Key）和值（Value）。

- 在编码器的自注意力机制中，编码器的输入同时作为查询、键和值的输入，实现了对信息的全方位分析。



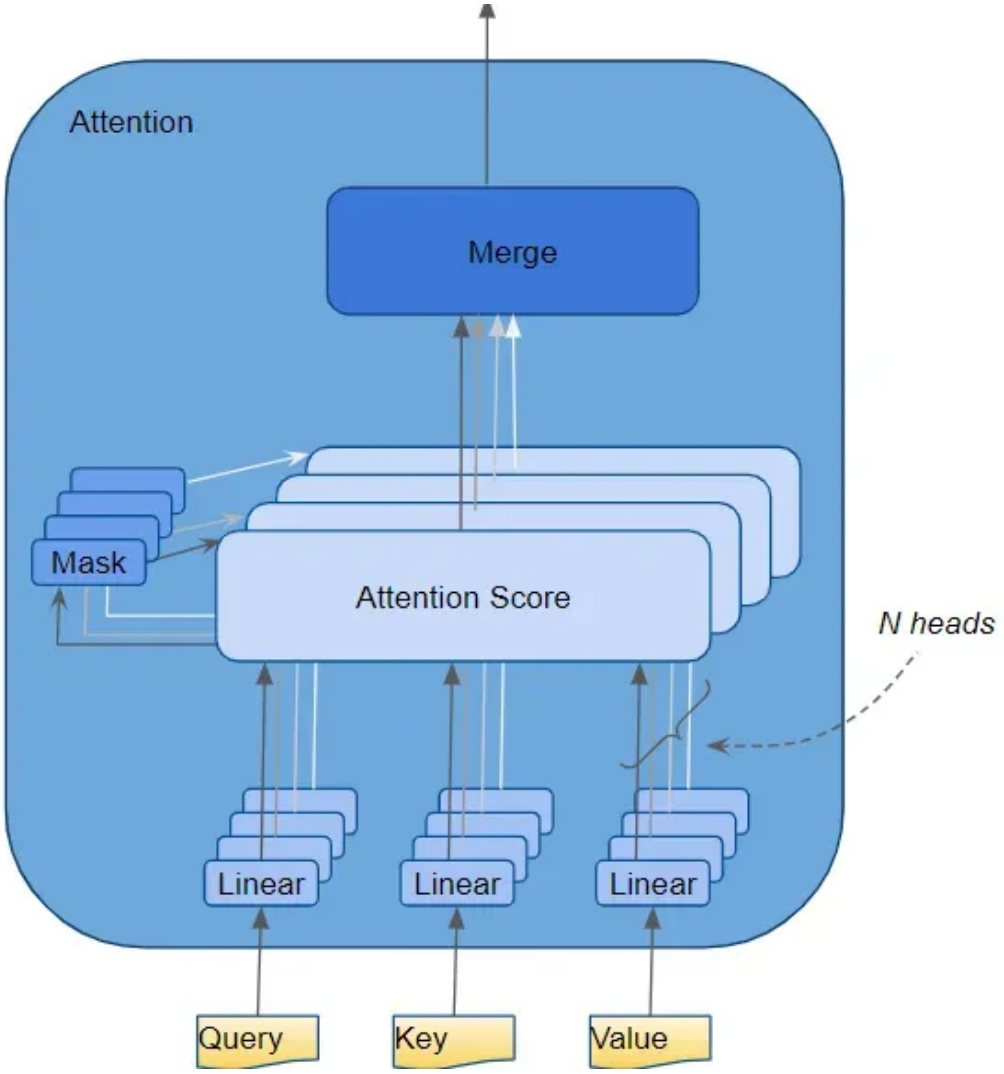
- 在解码器的自注意力过程中，解码器的输入会同时作为查询（Query）、键（Key）和值（Value）这三个关键参数的输入。

- 而在解码器的编码器 - 解码器注意力机制中，整个模型堆叠中最后一个编码器的输出则被用作值（Value）和键（Key）参数的输入。与此同时，紧接其下的自注意力模块（以及 Layer Norm 模块）的输出则作为查询（Query）参数的输入。这一过程实现了解码器对编码器输出信息的细致分析与利用。

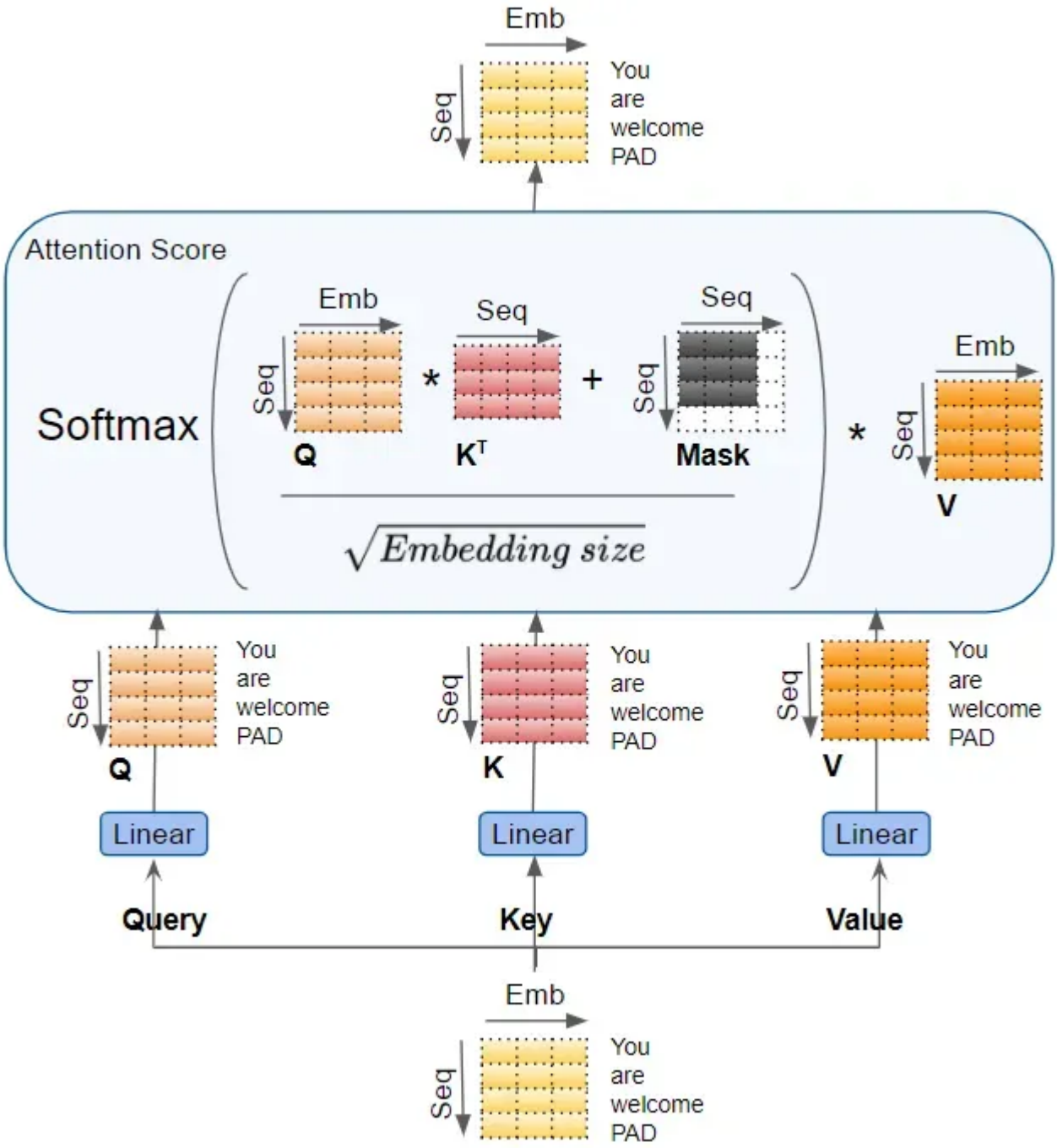


#07
多头注意力机制

在 Transformer 模型中，每个专注于处理特定信息的注意力单元被称作一个“注意力头”。模型通过并行地使用多个这样的注意力头，形成了所谓的多头注意力机制。这种机制通过结合多个执行类似任务的注意力计算，显著增强了模型对信息的辨别和处理能力，使得其在处理复杂数据时更加高效和精确。



在 Transformer 中，查询（Query）、键（Key）和值（Value）每一个都通过各自独立的线性层进行处理，每层都配备了专门的权重。这个过程分别产生了称为 Q、K 和 V 的三个结果。随后，这些结果根据注意力公式结合起来，形成了所谓的注意力得分。



这里需要特别理解的是，Q、K 和 V 值实际上是序列中每个词的编码表征。通过注意力计算，模型将序列中的每个词与序列中的其他所有词进行综合考虑，从而使得注意力得分为序列中每个词赋予了一个独特的评分。

之前我们讨论解码器时，简要提到了遮蔽（masking）机制。上面的注意力机制图中也显示了遮蔽机制，让我们看看它是如何工作的。

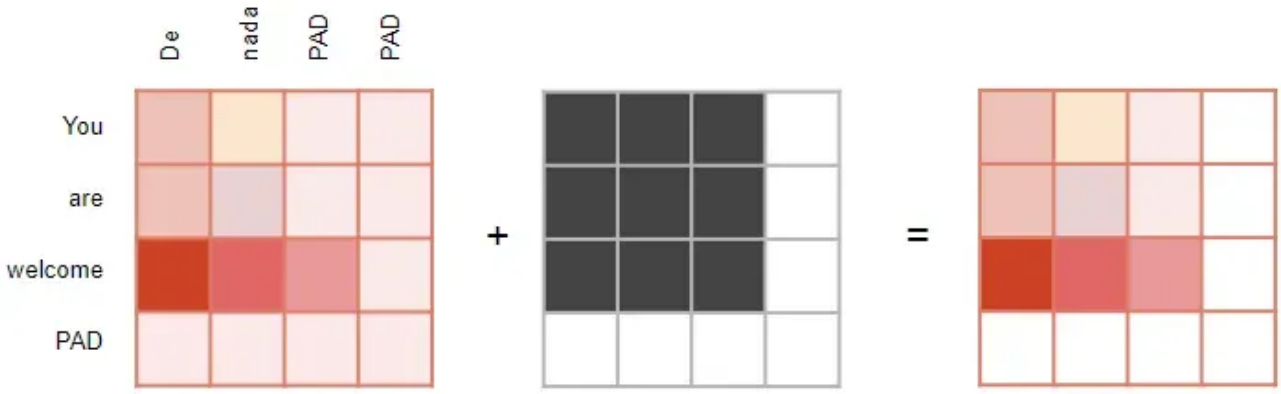
#08
注意力遮蔽机制

在计算注意力得分的过程中，注意力模块采用了一个关键的步骤：遮蔽机制。这种遮蔽主要有两个作用：

在编码器自注意力和编码器 - 解码器注意力中：遮蔽确保了输入句子中的填充部分（padding）在注意力计算中被忽略，从而避免这些非实际内容的干扰。（需要注意的是，由于输入序列长度可能不同，通常会通过添加填充令牌来使序列长度统一，以便于 Transformer 的处理。）

Encoder Self-Attention Scores

编码器 - 解码器注意力也采用了同样的遮蔽机制。



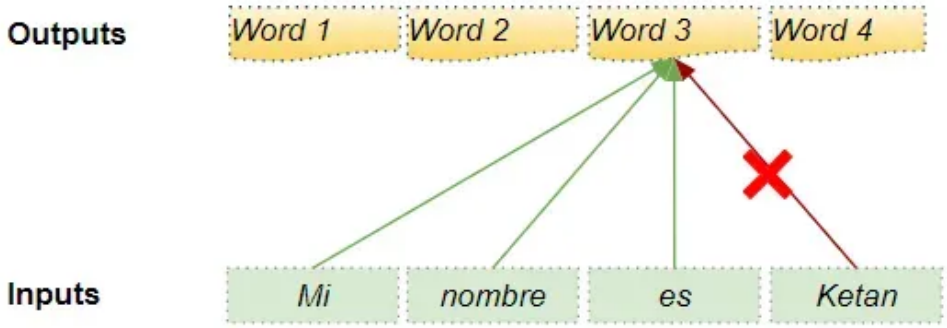
Encoder-Decoder Attention Scores

在解码器自注意力中：掩码的作用是防止解码器在预测下一个单词时“窥视”目标句子的后续部分。

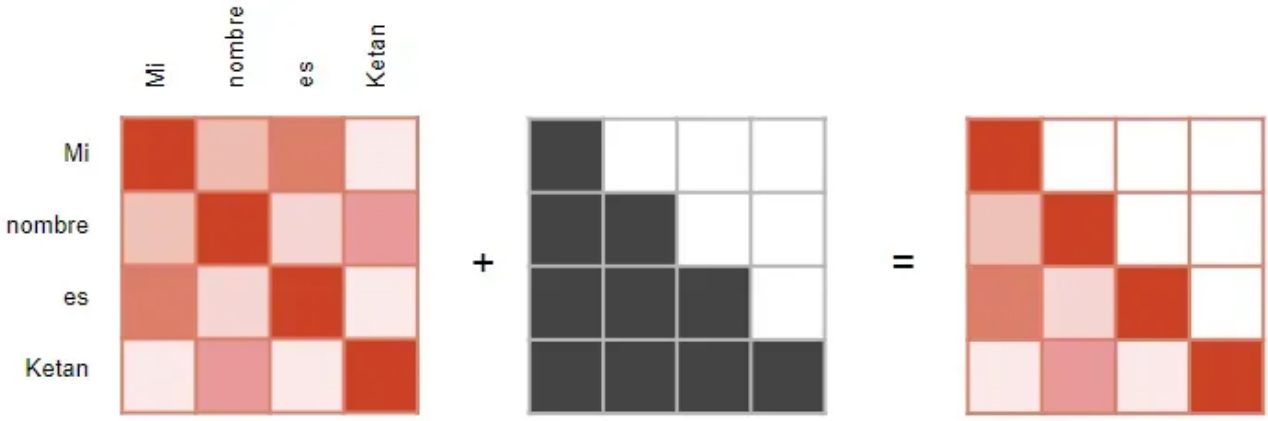
解码器处理源序列中的单词，并利用它们预测目标序列中的单词。在训练过程中，这通常是通过教师强制方法完成的，即将完整的目标序列作为解码器的输入。因此，在预测特定位置的单词

时，解码器可以同时参考该位置前后的目标单词。这可能导致解码器“作弊”，利用未来的目标单词进行预测。

例如，在预测“第 3 个单词”时，解码器应当仅参考目标句子中的前三个单词，而不是第四个单词“Ketan”。



因此，解码器会屏蔽掉序列中后面出现的输入单词。



Decoder Self-Attention Scores

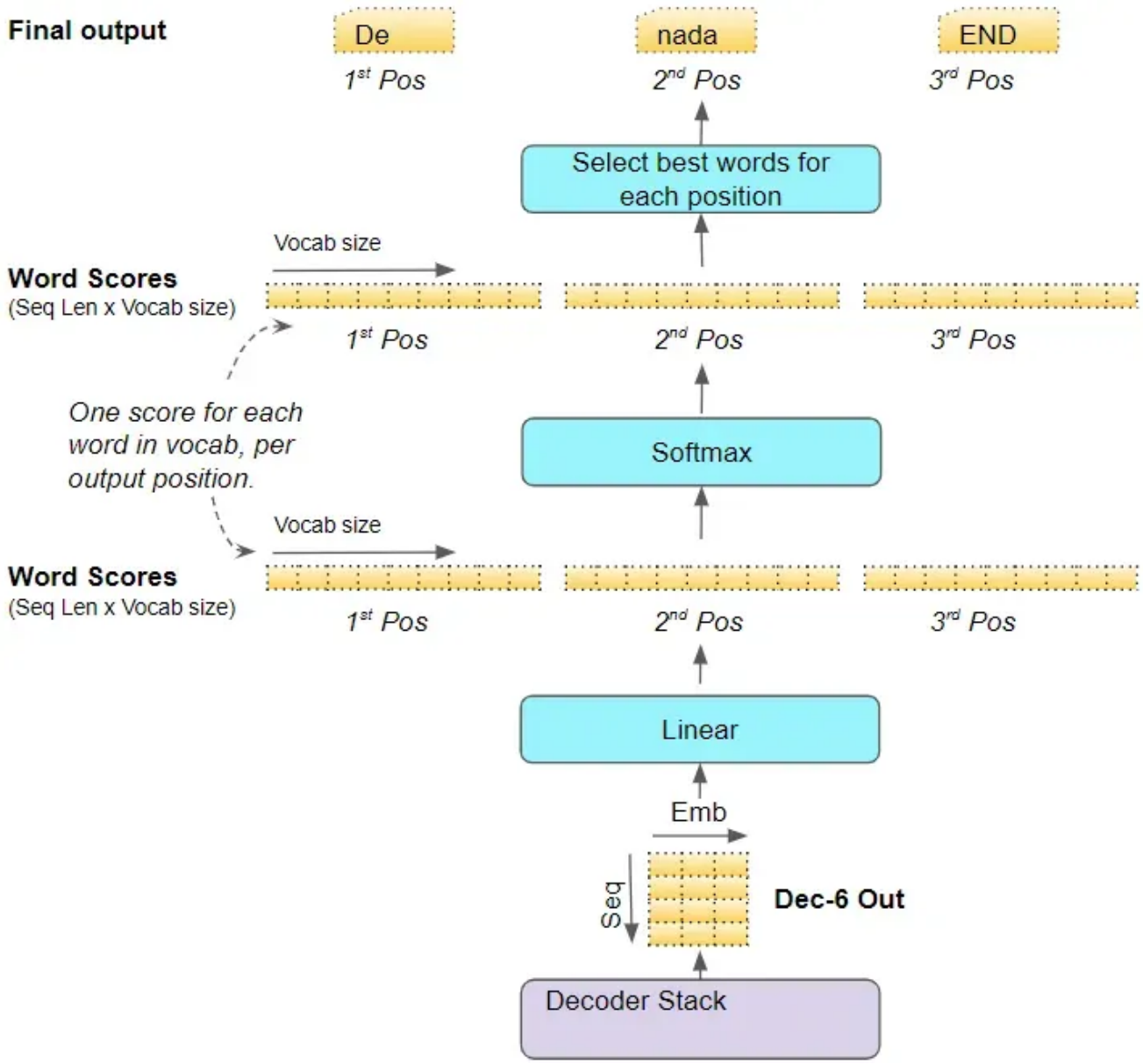
在计算注意力得分的过程中（请参考之前展示的计算图），遮蔽被用于 Softmax 操作之前的分子部分。这些被遮蔽的元素（表示为白色方块）被设置为负无穷大值，因此经过 Softmax 处理后，这些值就变为零。

#09
生成输出

在 Transformer 的堆栈结构中，最后一个解码器的输出被传递给输出组件，它负责将这些输出转换为最终的输出句子。

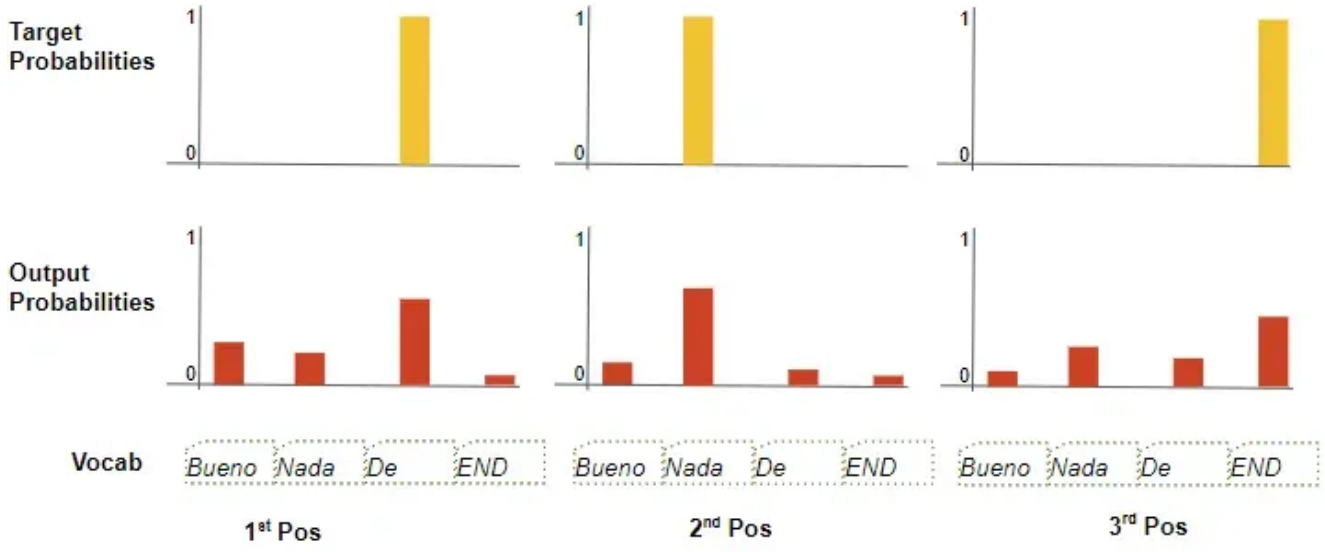
此过程中，线性层的作用是将解码器的向量映射到单词得分上。这意味着模型会为句子中的每个位置的每个唯一单词生成一个得分，这些得分反映了目标词汇表中每个单词在句子相应位置出现的可能性。例如，如果我们的最终输出句子包含 7 个单词，而目标西班牙语词汇表有 10000 个唯一单词，那么我们就会为这 7 个位置的每一个生成 10000 个得分值。

接下来，Softmax 层将这些得分转换成概率值（总和为 1。0）。在每个位置，模型都会找到具有最高概率的单词对应的索引，然后将这个索引映射到词汇表中的对应单词。最终，这些单词组成了 Transformer 的输出序列，完成了从输入到输出的整个转换过程。



#10
训练和损失函数

在训练过程中，我们使用诸如交叉熵损失之类的损失函数，将生成的输出概率分布与目标序列进行比较。概率分布显示了每个单词在该位置出现的概率。



让我们假设目标词汇表只包含四个词。我们的目标是生成一个与我们预期的目标序列“De nada END”相匹配的概率分布。

这就意味着，对于句子中的第一个词位置，概率分布应该完全偏向“De”（概率为 1），而词汇表中其他所有词的概率都应为 0。同理，对于第二和第三个词位置，“nada”和“END”分别应有概率为 1。

通常，模型会通过损失函数来计算梯度，并通过反向传播的方式来训练 Transformer。

#11
结论

希望通过这篇文章，你能对 Transformer 在训练过程中的内部运作有所了解。正如在之前的文章中所讨论的，虽然 Transformer 在推理阶段以循环的方式运行，但其大部分的处理流程是相同的。

正是多头注意力模块赋予了 Transformer 强大的能力。在下一篇文章中，我们将继续深入探索，更详细地了解注意力是如何计算的。

正是多头注意力模块赋予了 Transformer 强大的处理能力。在下一篇文章中，我们将继续深入探索，更详细地了解注意力机制的计算细节。

原文链接：https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34

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。