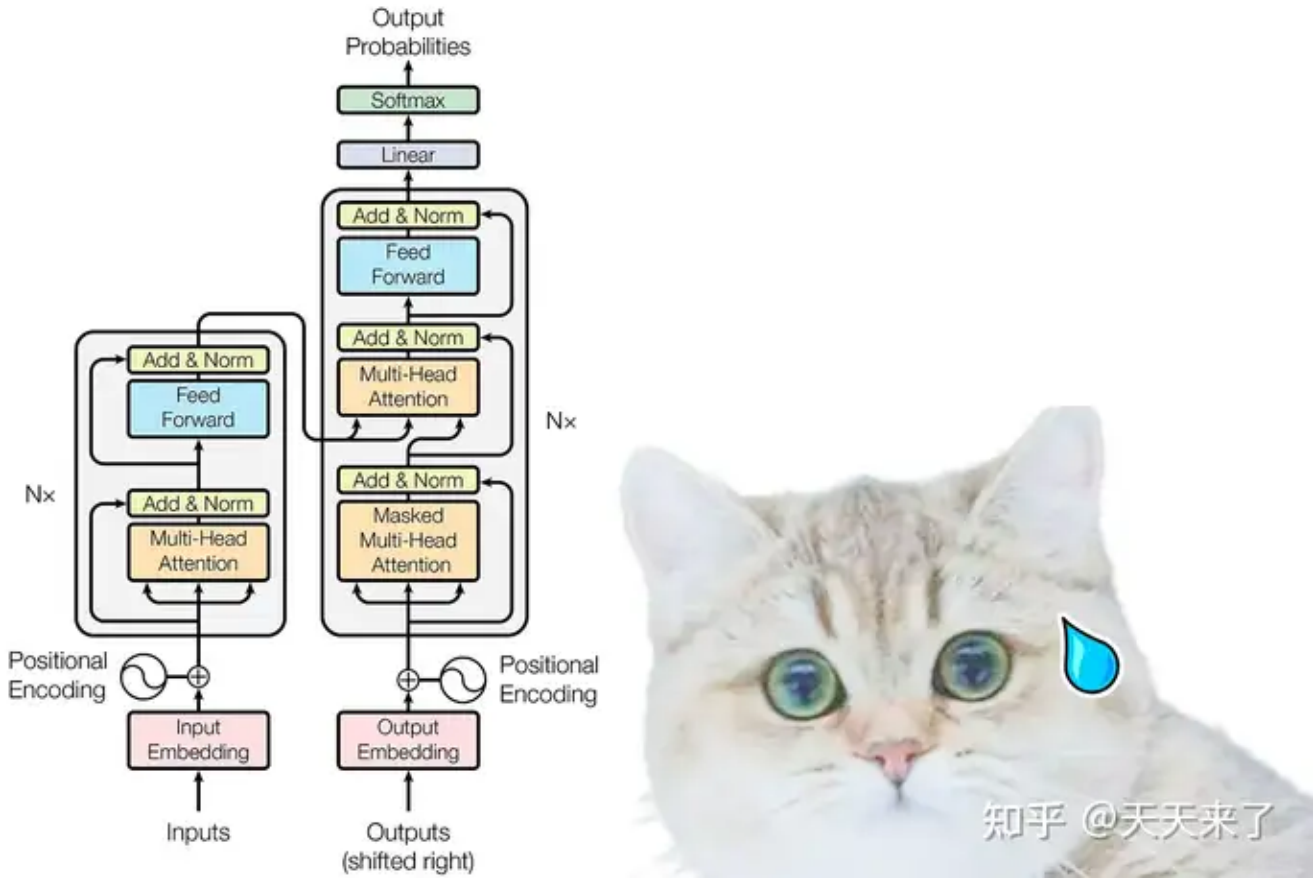
手把手教你了解Transformer —Part 1

Tim在路上
北京邮电大学 计算机学硕士

48 人赞同了该文章

> 本文引用和翻译自Fareed Khan的文章。

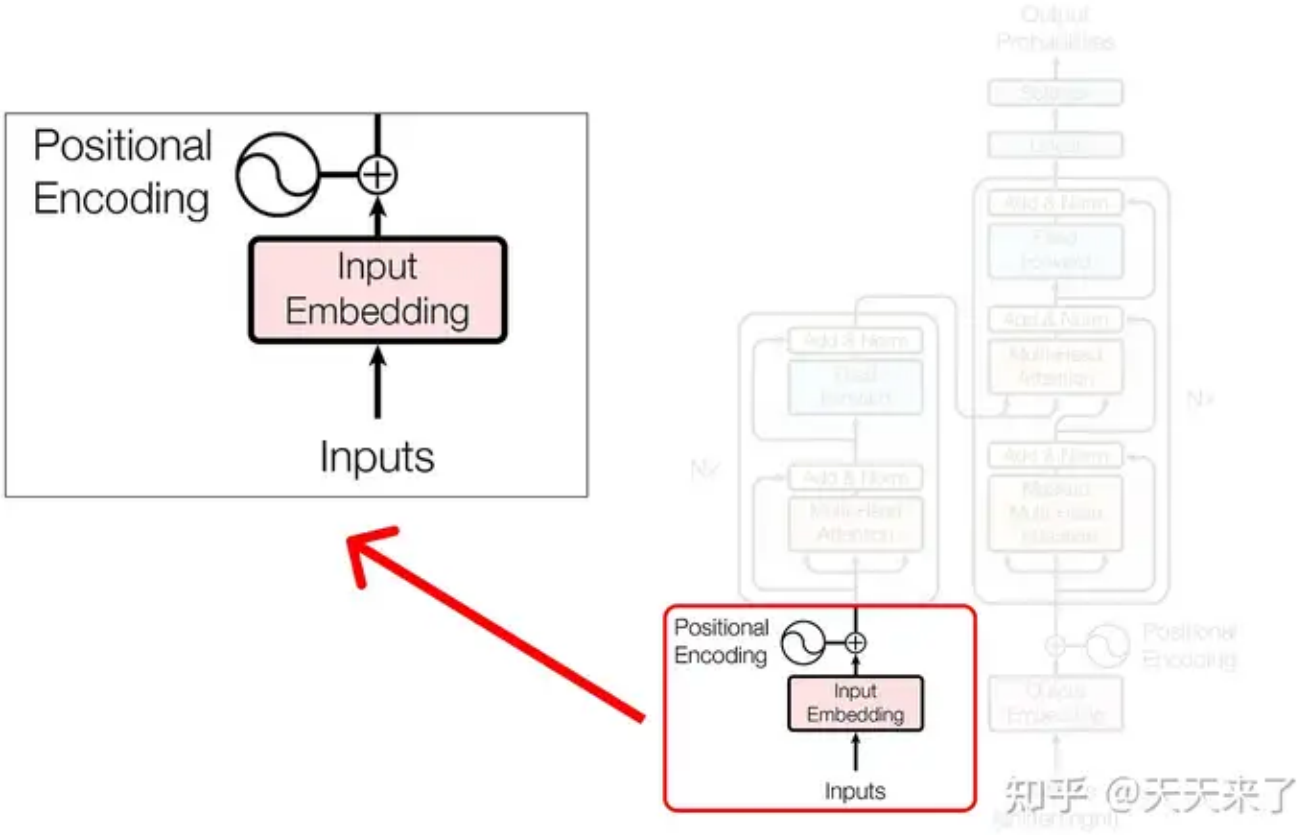
作为计算机的方面的从事人员，每个人应该或多或少的了解下Transformer。



我知道Transformer 架构可能看起来很可怕，您也可能已经在视频网站或博客上看到过各种各样的解释。然而，在我这里，我将尽可能的通过最简单的示例，一点点的来阐明Transformer的原理。通过这样做，希望能够简化我们对 Transformer 架构的理解。

让我们开始吧！

1. 输入和位置编码 （ Inputs and Positional Encoding ）



我们首先来解决最开始的部分，即确定Inputs和positional encoding。

步骤 1 （定义数据）

第一步是定义我们的**数据集（即语料库）。**

Dataset (Courpus)

1. I drink and I know things.
2. When you play the game of thrones, you win or you die.
3. The true enemy won't wait out the storm, He brings the storm.

这里我们取自**《权力的游戏》中的三段对话来作为我们的数据集。

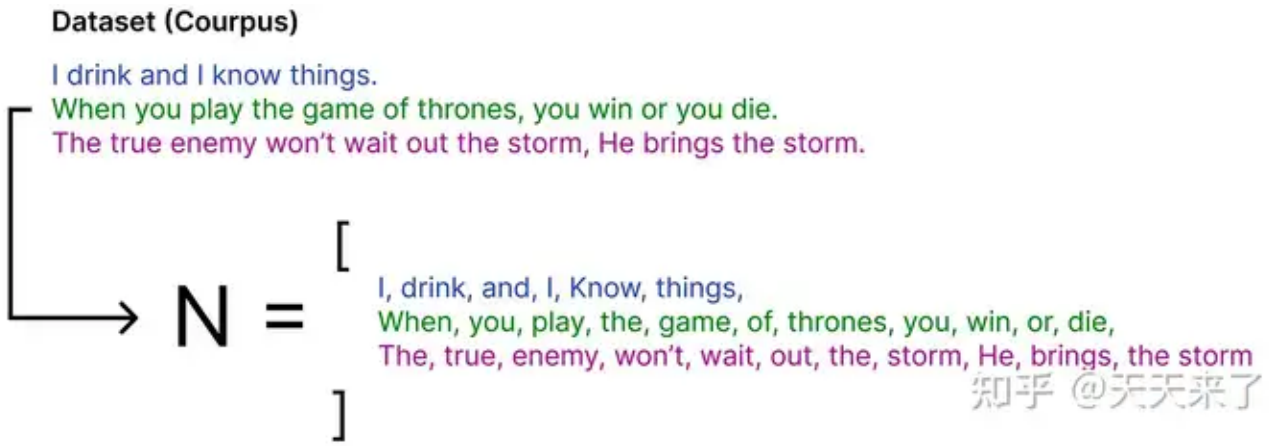
尽管这个数据集看起来很小，但较小的数据集有助于我们更好的一步步通过数学方程推导出我们的结果。

步骤 2（计算词汇量）

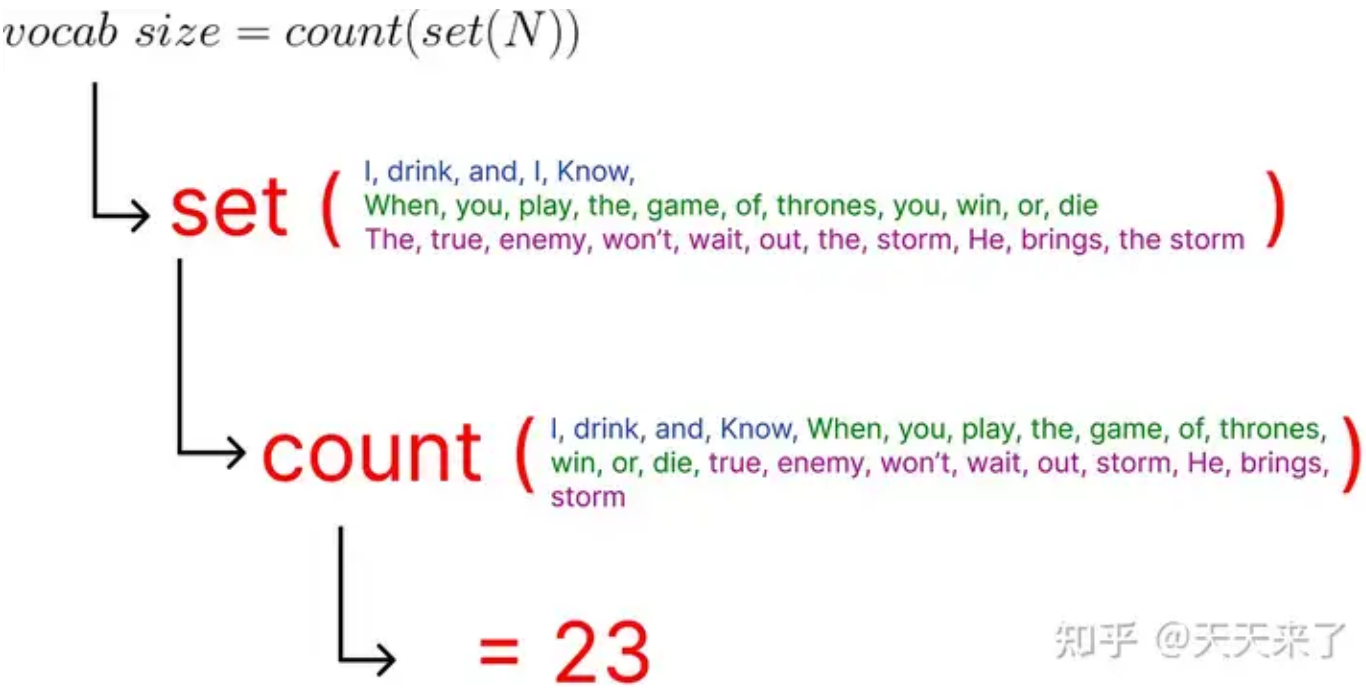
为了确定词汇量大小，我们需要确定数据集中唯一单词的总数。

$$vocab\ size = count(set(N))$$

其中 N 是所有单词的列表，每个单词都是一个标记，我们将把数据集分成一个标记列表，即找到 **N。**



获得标记列表（记为 N）后，我们可以应用公式来计算词汇表大小。



这里我们先将数据集进行了单词切分，然后对其进行去重并计数。最后得出，词汇量大小为 23，即给定列表中有 23 个唯一单词。

步骤 3（编码和嵌入）

我们为数据集的每个唯一单词分配一个整数，如下所示：

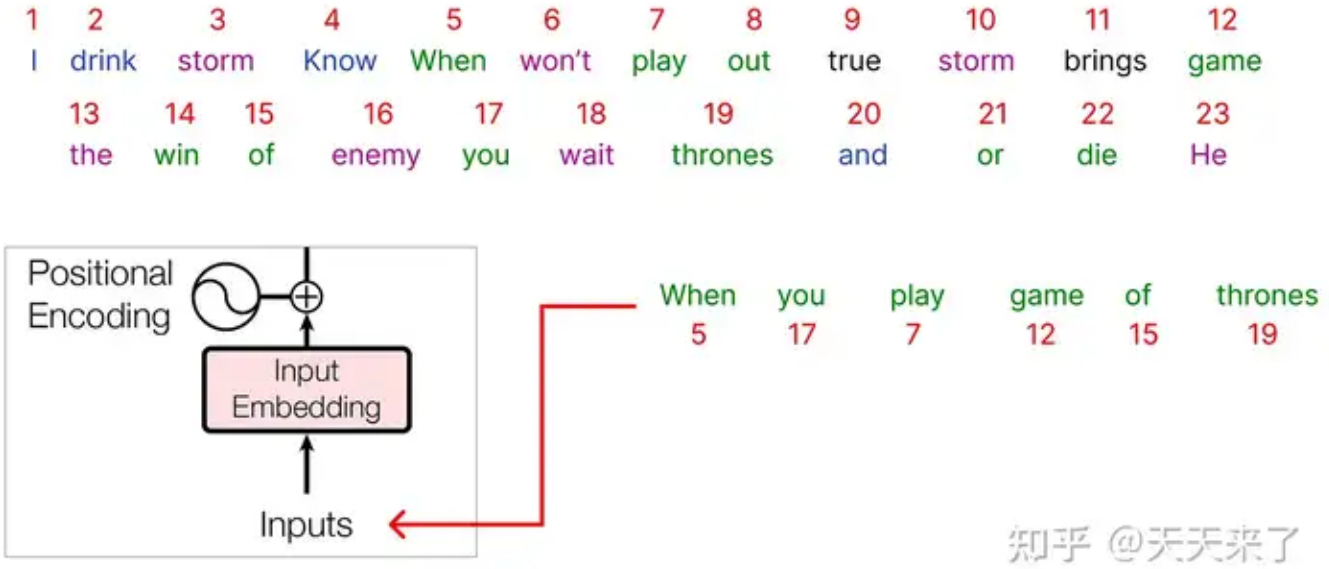
1	2	3	4	5	6	7	8	9	10	11	12
I	drink	storm	Know	When	won't	play	out	true	storm	brings	game

13	14	15	16	17	18	19	20	21	22	23
the	win	of	enemy	you	wait	thrones	and	or	die	He

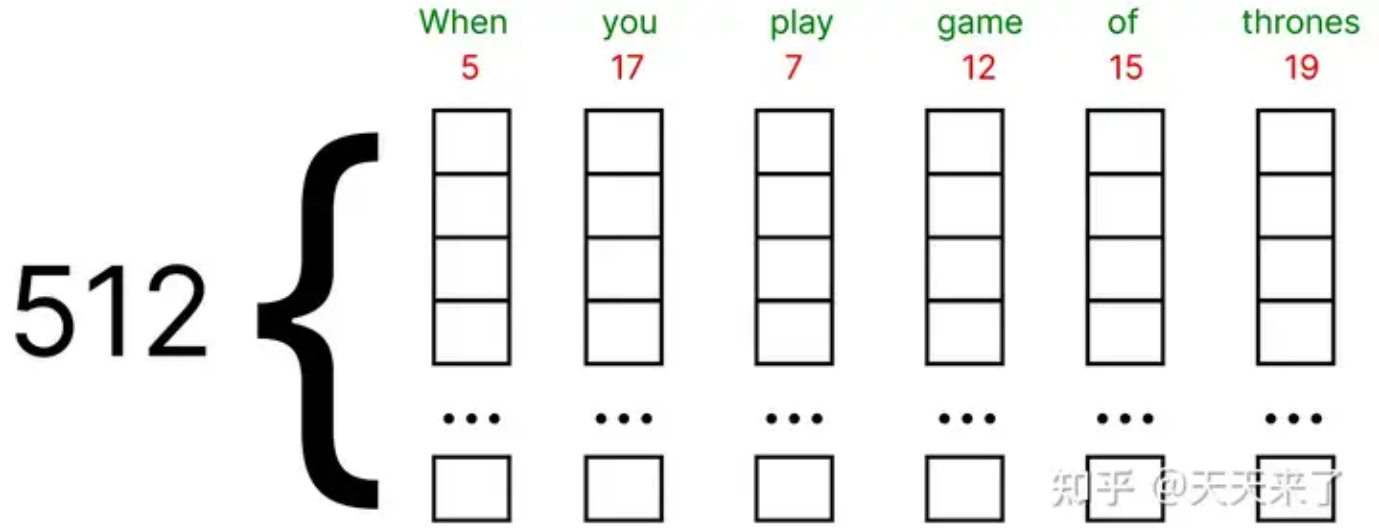
这个分配的数字，可以看做我们对数据集进行了编码。在编码后，是时候选择我们的输入了。

我们将从语料库中选择一个句子作为开头：

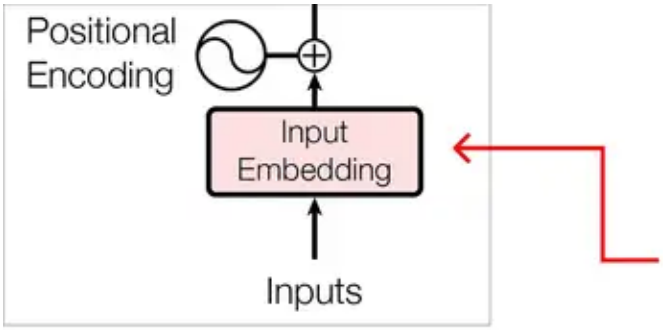
“When you play game of thrones”



将这个句子中的每个单词转换为编码的整数，并且每个相应的整数值都关联一个的Embedding(嵌入向量)。



这些Embedding嵌入可以使用Google Word2vec生成。在我们的例子中，我们假设每个单词所关联的的嵌入向量Embedding的默认值是，由随机的0 和 1之间浮点数填充。
原始论文使用**512**维的Embedding嵌入向量，这里为了举例，我们使用**5维**作为示例。



When	you	play	game	of	thrones
5	17	7	12	15	19
e1	e2	e3	e4	e5	e6
0.19	0.07	0.94	0.43	0.8	0.34
0.6	0.7	0.73	0.49	0.65	0.72
0.45	0.96	0.02	0.09	0.57	0.79
0.56	0.97	0.85	0.2	0.34	0.15
0.13	0.58	0.16	0.26	0.05	0.75

现在，每个单词都由维度为5的Embedding嵌入向量来表示，其中的数值是由**RAND()随机器生成的。**

步骤 4（位置嵌入）

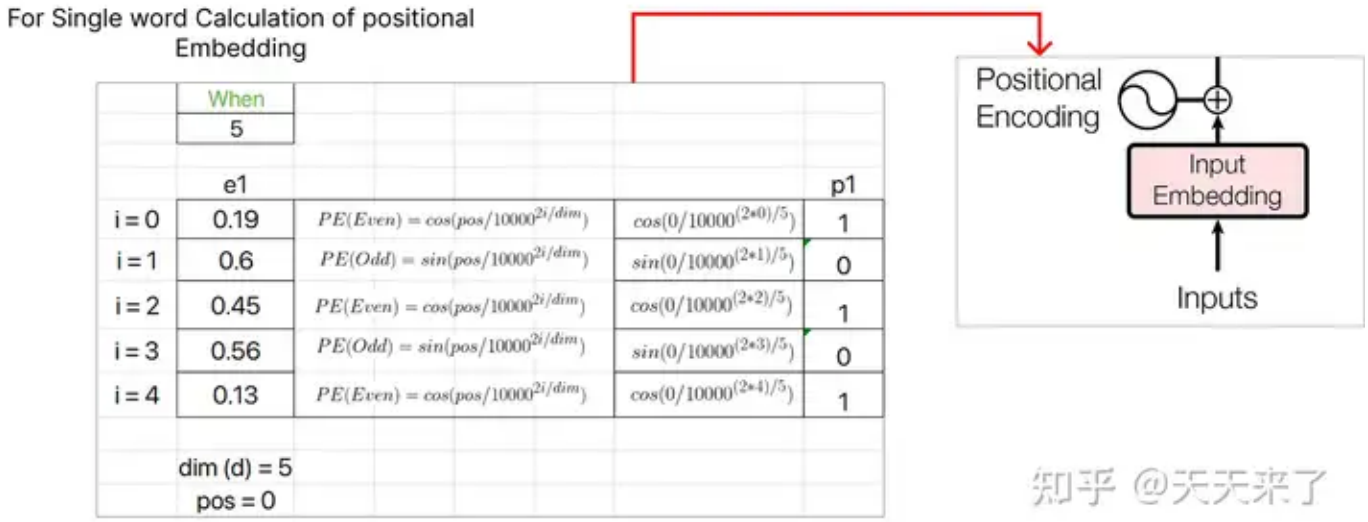
让我们考虑第一个单词，即**“When”**，并计算它的位置嵌入向量（Positional Encoding）。

位置嵌入(Positional Encoding)有两个公式：

$$PE(Odd) = sin(pos/10000^{2i/dim})$$
$$PE(Even) = cos(pos/10000^{2i/dim})$$

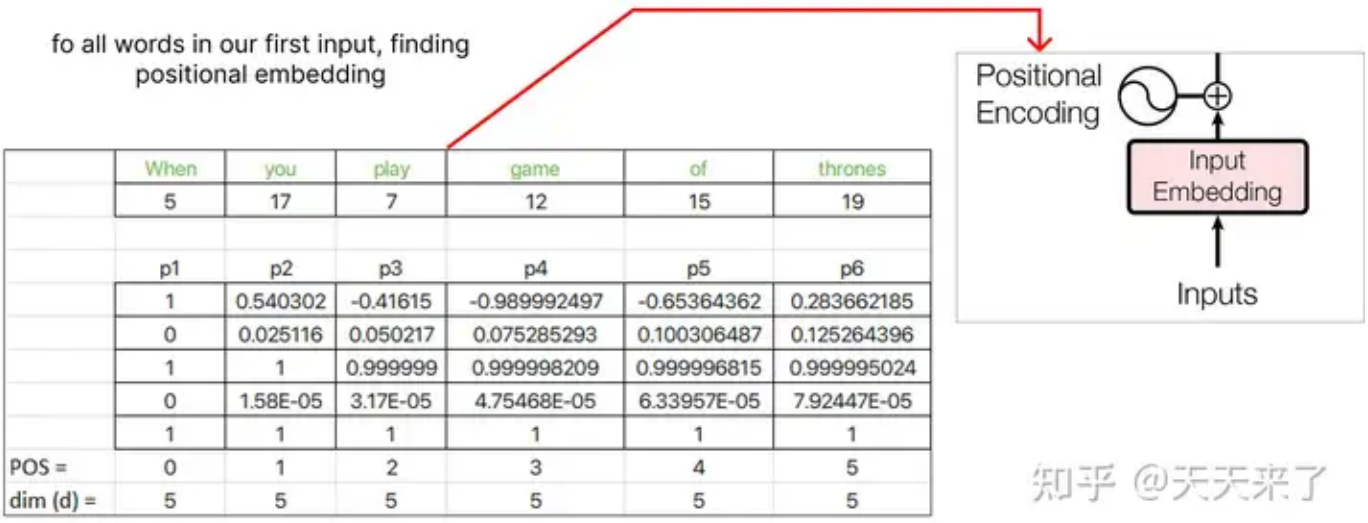
其中的pos为单词的位置，从0开始； i为Embedding数组中的索引下标，i的奇偶决定了使用哪个公式来计算PE值；dim为嵌入向量Embedding的维度。

第一个单词“**When**”**的POS值将为零，因为它对应于序列的起始索引**。维度值表示嵌入向量的维数，在本例中为 5。下面我们来看下如何进行计算：

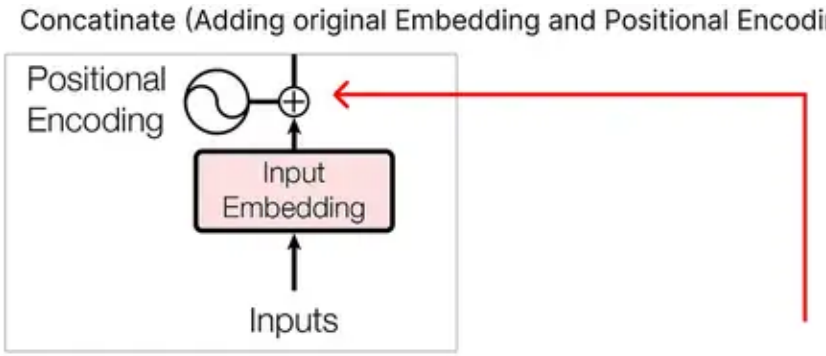


通过Positional Encoding的公式，我们得出了“**When**”的位置Embedding，即10101。

接下来我们继续计算“**you**”这个单词，此时pos 值 1；后续的每个单词依次增加每个 pos的值。

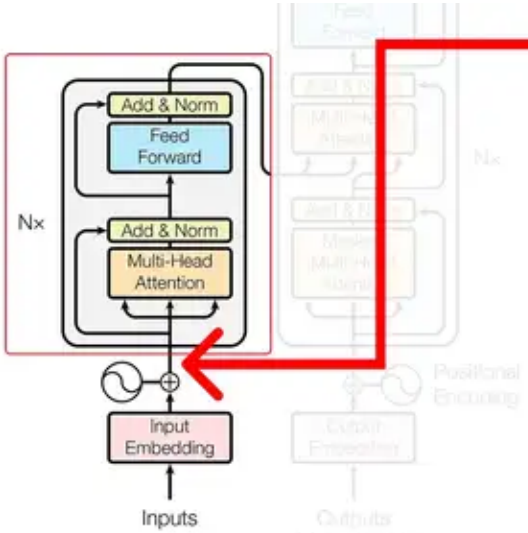


生成位置嵌入Embedding后，我们可以将其与原始单词嵌入连接起来。



When	you	play	game	of	thrones
5	17	7	12	15	19
e1	e2	e3	e4	e5	e6
0.19	0.07	0.94	0.43	0.8	0.34
0.6	0.7	0.73	0.49	0.65	0.72
0.45	0.96	0.02	0.09	0.57	0.79
0.56	0.97	0.85	0.21	0.84	0.75
0.13	0.58	0.16	0.26	0.05	0.75
+	+	+	+	+	+
p1	p2	p3	p4	p5	p6
1	0.540302306	-0.416146837	-0.989992497	-0.653643621	0.283662185
0	0.025116223	0.050216599	0.075285293	0.100306487	0.125264396
1	0.999999801	0.999999204	0.999998209	[illegible]	[illegible]
0	1.58489E-05	3.16979E-05	4.75468E-05	6.33957E-05	7.92447E-05
1	1	1	1	1	1

我们得到的结果向量是 e1+p1、e2+p2、…. 、e6+p6等的总和。

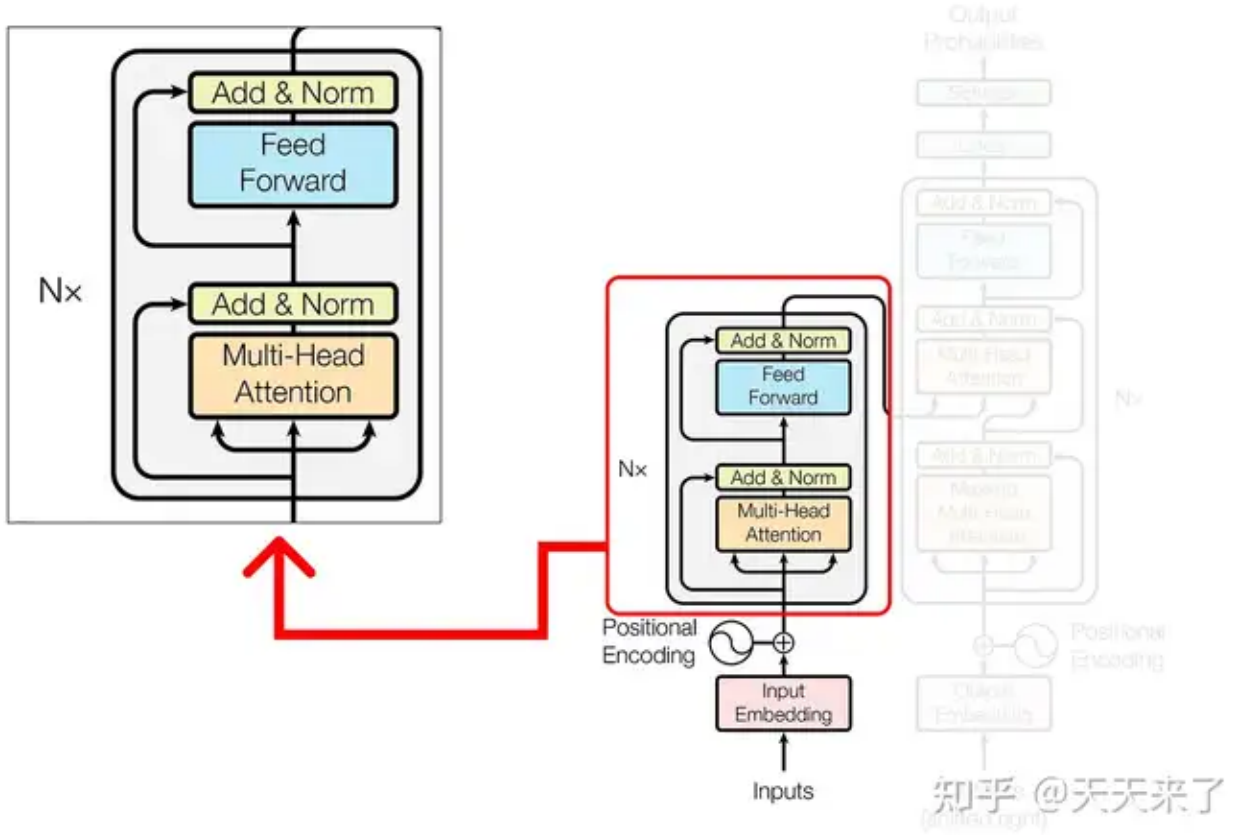


When	you	play	game	of	thrones
5	17	7	12	15	19
ep1	ep2	ep3	ep4	ep5	ep6
1.19	0.6103	0.5239	-0.56	0.1464	0.6237
0.6	0.7251	0.7802	0.5653	0.7503	0.8453
1.45	1.96	1.02	1.09	1.57	1.79
0.56	0.97	0.85	0.21	0.8401	0.7501
1.13	1.58	1.16	1.26	1.05	1.75

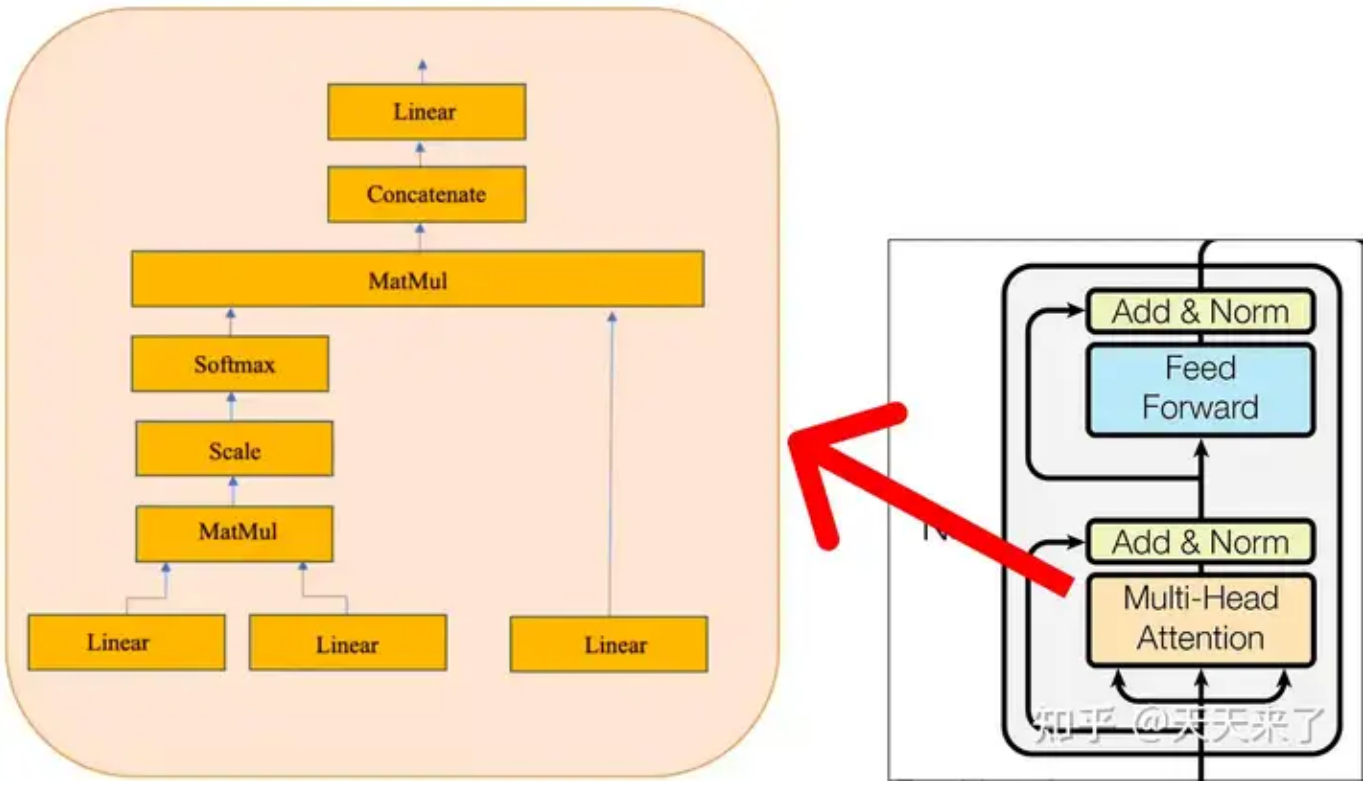
最后，我们将Transformer初始化部分的输出作为编码器的输入。

2. 编码器（Encoder）

在编码器中，我们将执行复杂的操作，其涉及queries, keys, and values这三个数值矩阵，我们简称Q、K和V。这些操作对于输入数据的转换和提取是非常有意义的。



如上图所示，在**Encoder内部有**multi-head attention，我们先来解释下attention机制。在单个attention layer由以下几个关键部分组成：

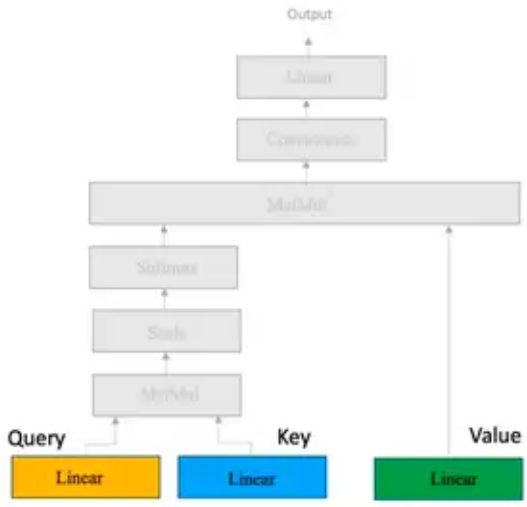


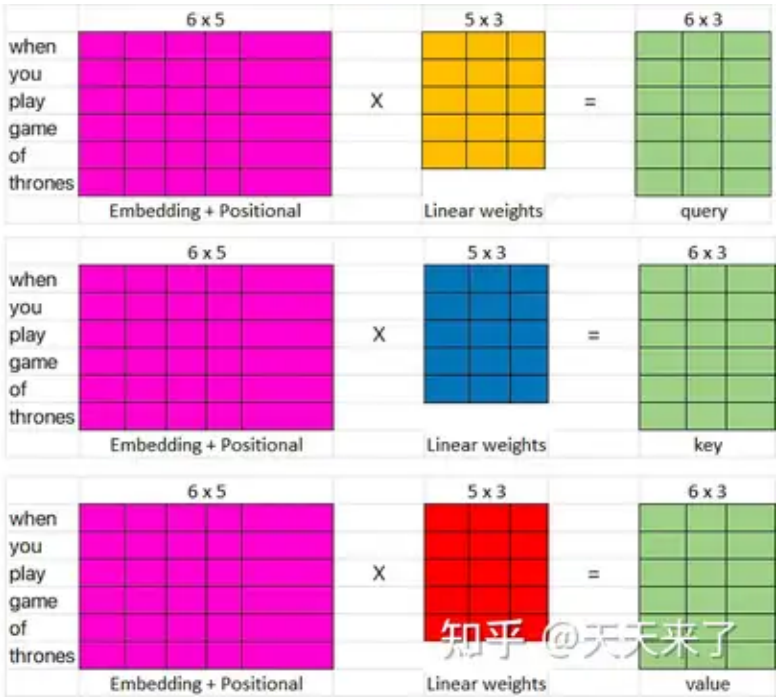
请注意，黄色框代表的模型机制是单个attention机制。multi-head attention 代表的是多个黄色框存在。这里我们为了这个示例简单，我们将仅考虑一个（即single-head attention）。

步骤 1 （执行single-head attention）

attention注意力层有3个输入矩阵：

Query
Key
Value





如上图所示，三个输入矩阵中，**粉色矩阵**表示从上一步将位置嵌入Embedding添加到词嵌入Embedding矩阵中获得的转置输出。

另一方面，线性权重矩阵，**黄色、蓝色和红色**代表attention注意力机制中使用的权重参数矩阵。这些矩阵可以具有任意数量的列维数，但行数必须与输入矩阵中的列数相同，因为他们之间要做乘法。

在我们的例子中，我们假设线性矩阵（黄色、蓝色和红色）包含是随机权重。这些权重通常是随机初始化的，然后在训练过程中通过反向传播和梯度下降等技术进行调整。

下面让我们计算一下（Q、K和V的值）：

	Embedding + Positional						Linear weights				Query		
when	1.19	0.6	1.5	0.56	1.13		0.552	0.26	0.208		2.8	2.3	2.44
you	0.61	0.73	2	0.97	1.58		0.885	0.21	0.262		3.3	3	3.33
play	0.52	0.78	1	0.85	1.16	X	0.178	0.89	0.297	=	2.7	2.1	2.54
game	-0.56	0.57	1.1	0.21	1.26		0.605	0.9	0.994		1.7	1.2	1.74
of	0.15	0.75	1.6	0.84	1.05		0.906	0.07	0.931		2.5	2.4	2.51
thrones	0.62	0.85	1.8	0.75	1.75						3.5	2.7	3.26

	Embedding + Positional						Linear weights				Key		
when	1.19	0.6	1.5	0.56	1.13		0.996	0.22	0.237		2.9	1.6	1.9
you	0.61	0.73	2	0.97	1.58		0.667	0.87	0.785		3	2	2.45
play	0.52	0.78	1	0.85	1.16	X	0.587	0.1	0.15	=	2.3	1.6	2.06
game	-0.56	0.57	1.1	0.21	1.26		0.565	0.17	0.793		0.7	1.2	1.18
of	0.15	0.75	1.6	0.84	1.05		0.122	0.52	0.427		2.2	1.5	1.97
thrones	0.62	0.85	1.8	0.75	1.75						2.9	2.1	2.42

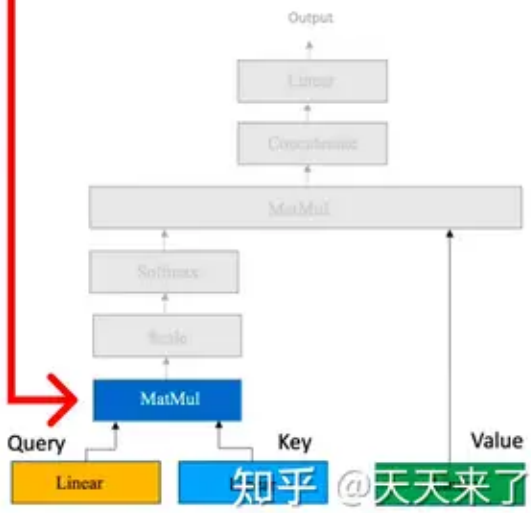
	Embedding + Positional						Linear weights				Value		
when	1.19	0.6	1.5	0.56	1.13		0.896	0.58	0.019		3	2.8	1.98
you	0.61	0.73	2	0.97	1.58		0.045	0.43	0.594		3.3	3.3	2.7
play	0.52	0.78	1	0.85	1.16	X	0.717	0.52	0.495	=	2.3	2.4	1.97
game	-0.56	0.57	1.1	0.21	1.26		0.658	0.27	0.288		1	1.6	1.73
of	0.15	0.75	1.6	0.84	1.05		0.443	0.87	0.64		2.3	2.4	2.14
thrones	0.62	0.85	1.8	0.75	1.75						3.2	3.4	2.74

当我们在attention注意力机制中获得了**Q、K和V**矩阵，我们就可以进行下一步的的矩阵乘法计算。

$$query * transpose(key)$$

Query				Transpose(Key)					
2.8	2.31	2.44							
3.35	3.04	3.33		2.89	2.98	2.26	0.731047	2.170039	2.872435
2.73	2.05	2.54	X	1.62	1.95	1.64	1.166483	1.531395	2.091971
1.65	1.22	1.74		1.9	2.45	2.06	1.179864	1.973794	2.422168
2.48	2.42	2.51							
3.45	2.73	3.26							

16.46063	18.84472	15.16177	7.622069	14.4337	18.78968
20.91132	24.08375	19.43355	9.924581	18.49635	24.04522
16.02201	18.36819	14.7777	7.386308	14.07696	18.28246
10.05137	11.57518	9.327719	4.683798	8.888656	11.51414
15.85299	18.28119	14.76566	7.59949	14.04828	18.27532
20.56783	23.60403	19.00353	9.550073	18.09826	23.51283

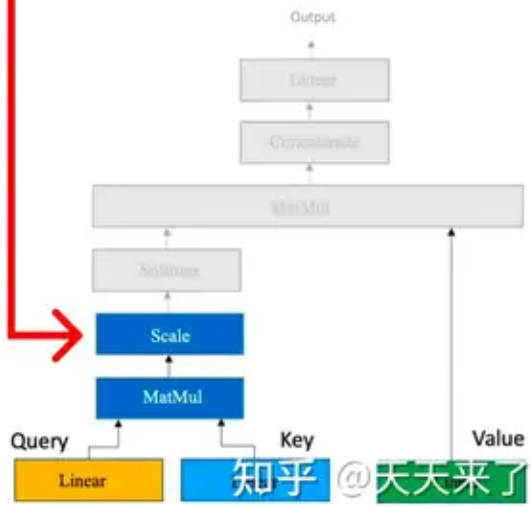


16.46063	18.84472	15.16177	7.622069	14.4337	18.78968
20.91132	24.08375	19.43355	9.924581	18.49635	24.04522
16.02201	18.36819	14.7777	7.386308	14.07696	18.28246
10.05137	11.57518	9.327719	4.683798	8.888656	11.51414
15.85299	18.28119	14.76566	7.59949	14.04828	18.27532
20.56783	23.60403	19.00353	9.550073	18.09826	23.51283

$$\frac{}{\sqrt{d_k}}$$ where d is 5 (dimension of embedding vector)

=

3.3	3.77	3.03	1.52	2.89	3.76
4.2	4.82	3.89	1.98	3.7	4.81
3.2	3.67	2.96	1.48	2.82	3.66
2	2.32	1.87	0.94	1.78	2.3
3.2	3.66	2.95	1.52	2.81	3.66
4.1	4.72	3.8	1.91	3.62	4.7



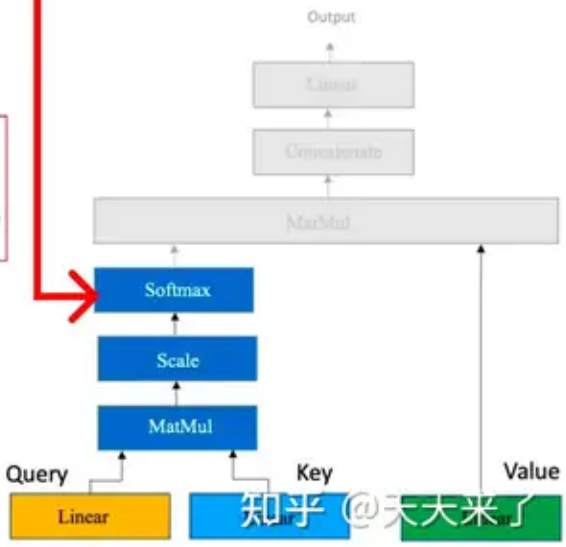
3.3	3.77	3.03	1.52	2.89	3.76
4.2	4.82	3.89	1.98	3.7	4.81
3.2	3.67	2.96	1.48	2.82	3.66
2	2.32	1.87	0.94	1.78	2.3
3.2	3.66	2.95	1.52	2.81	3.66
4.1	4.72	3.8	1.91	3.62	4.7

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_i)}$$

$$\frac{\exp(3.3)}{\exp(3.3) + \exp(3.77) + \exp(3.03) + \exp(1.52) + \exp(2.89) + \exp(3.76)}$$

=

0.17	0.33	0.235	0.07	0.27	0.88
0.16	0.36	0.22	0.04	0.24	0.95
0.17	0.33	0.237	0.07	0.28	0.87
0.18	0.28	0.238	0.12	0.29	0.63
0.17	0.33	0.236	0.07	0.28	0.87
0.16	0.36	0.222	0.04	0.24	0.95



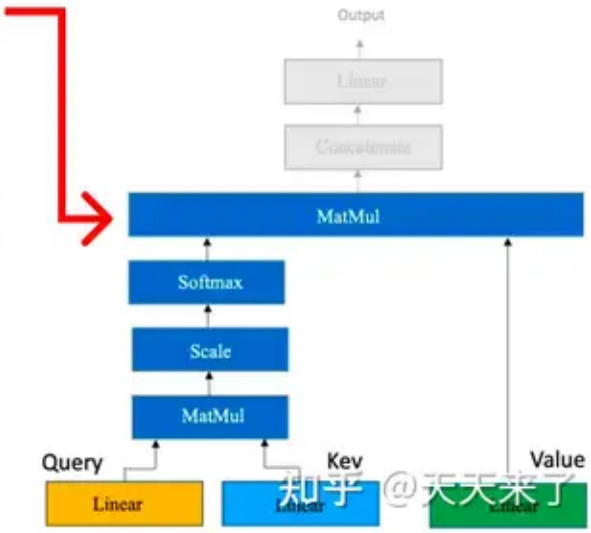
现在我们将结果矩阵与我们之前计算的值矩阵相乘：

$$\text{softmax}\left(\frac{Q.K^T}{\sqrt{d_k}}\right)$$

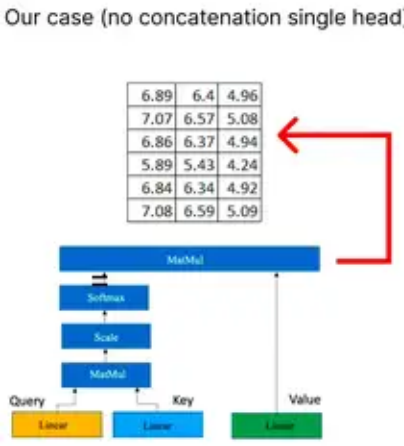
resultant matrix							Value		
0.17	0.33	0.23	0.07	0.27	0.88		3	3.4	2.06
0.16	0.36	0.22	0.04	0.24	0.95		3.97	3.7	2.84
0.17	0.33	0.24	0.07	0.28	0.87	X	2.64	2.6	2.24
0.18	0.28	0.24	0.12	0.29	0.63		2.46	1.3	1.49
0.17	0.33	0.24	0.07	0.28	0.87		3.02	2.7	2.19
0.16	0.36	0.22	0.04	0.24	0.95		3.93	3.6	2.78

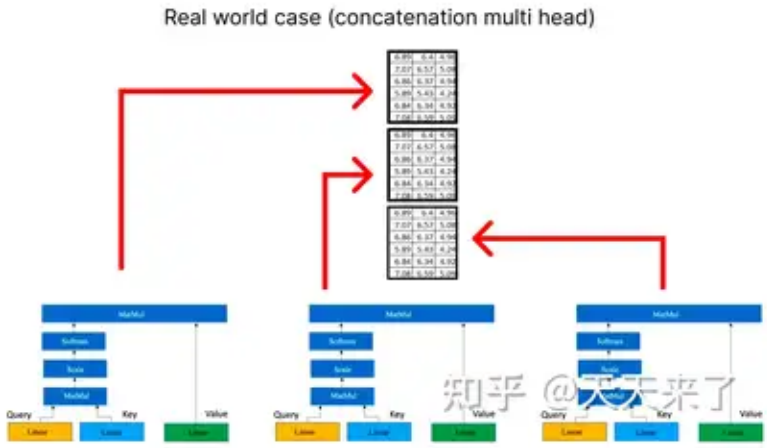
=

6.89	6.4	4.96
7.07	6.57	5.08
6.86	6.37	4.94
5.89	5.43	4.24
6.84	6.34	4.92
7.08	6.59	5.09

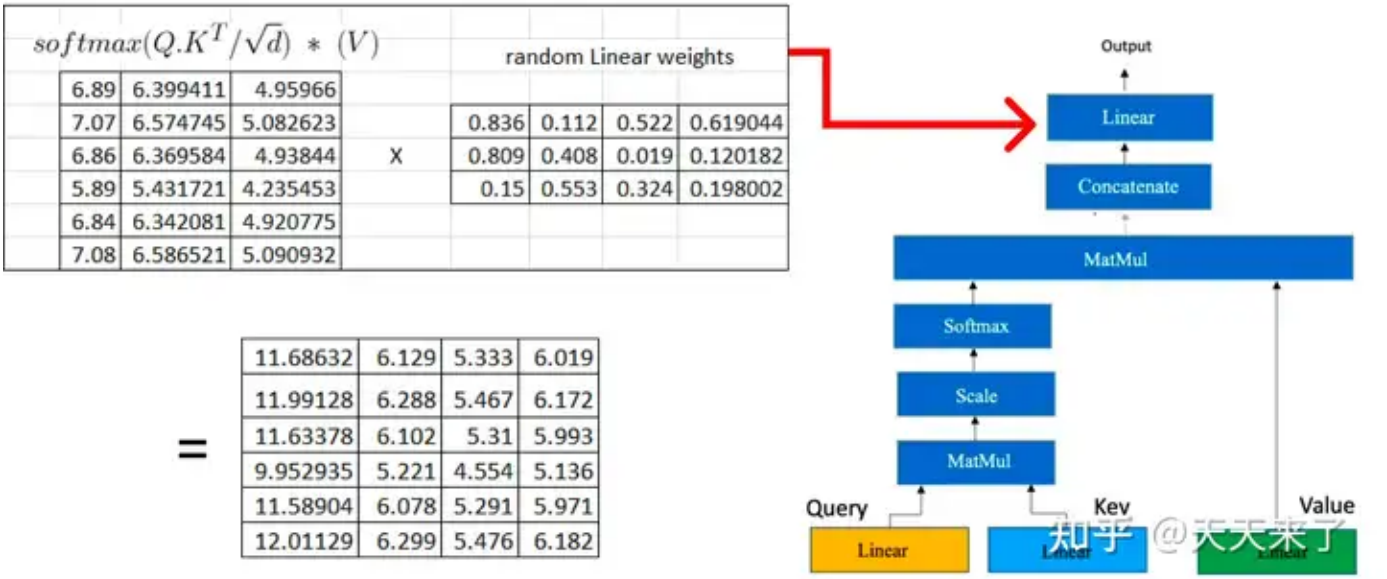


如果我们有multi-head attention，每个注意力attention都会产生一个维度 (6x3) 的矩阵，下一步会将这些矩阵连接concat在一起。

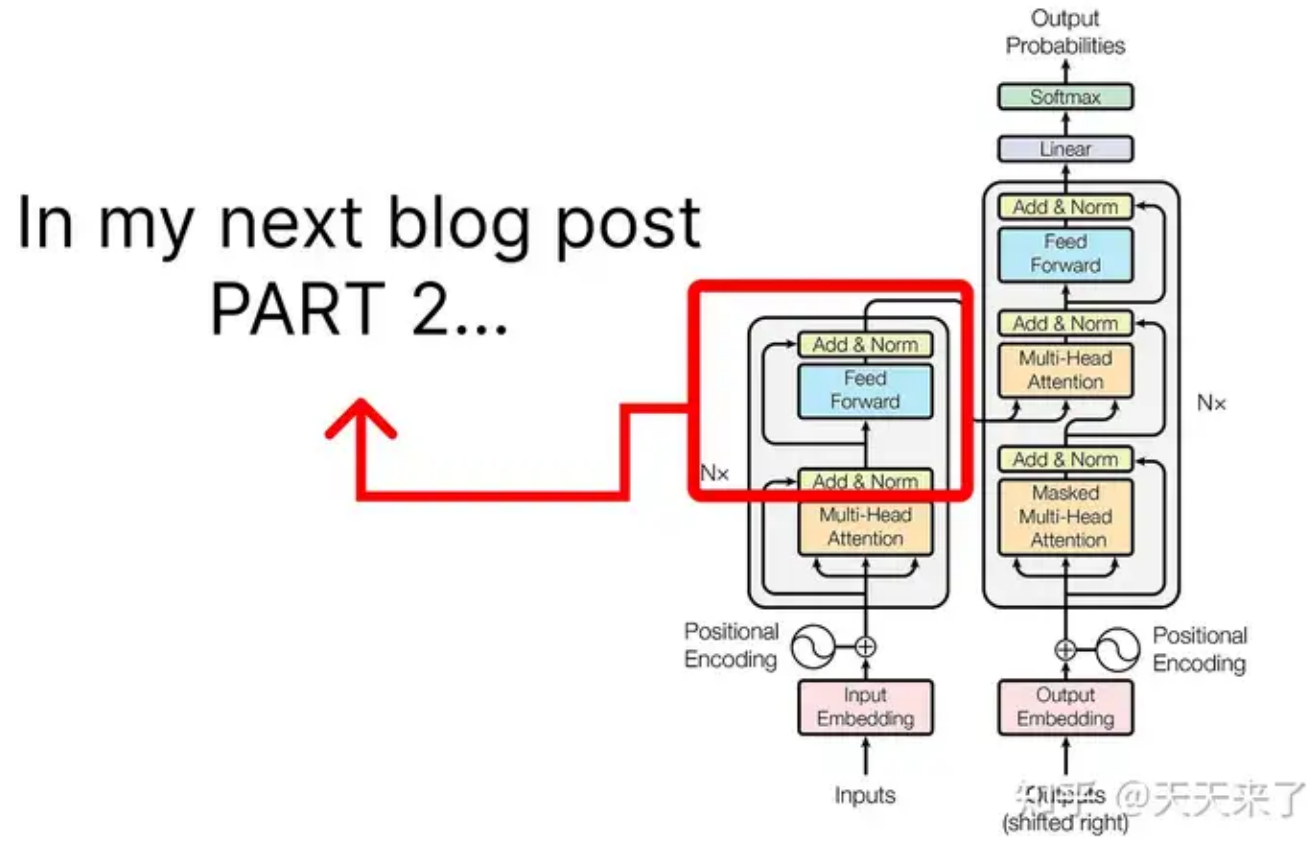




在下一步中，我们将再次执行类似于获取Q、K和V矩阵的过程的线性变换。该线性变换应用于从multi-head attention获得的级联矩阵。



由于文章已经变得很长，在下一部分中，我们将把重点转移到讨论编码器**Encoder**架构中下面部分的。



发布于 2023-07-15 08:16 · IP 属地北京