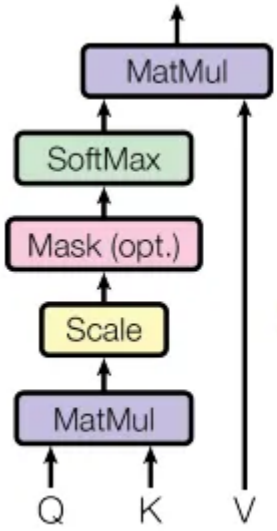
漫画 Transformer: 手把手用数学公式推导

Original Khan AI大模型世界 2024年12月13日 20:21 美国 标题已修改

我学习的时候总有个执念：这个背后的底层原理是什么？

这个执念经常会让我在理解新的知识的时候，造成很大的障碍。如果我不能理解它的底层原理，我就很难去理解在它基础上构建的知识。

GPT正属于这类型。

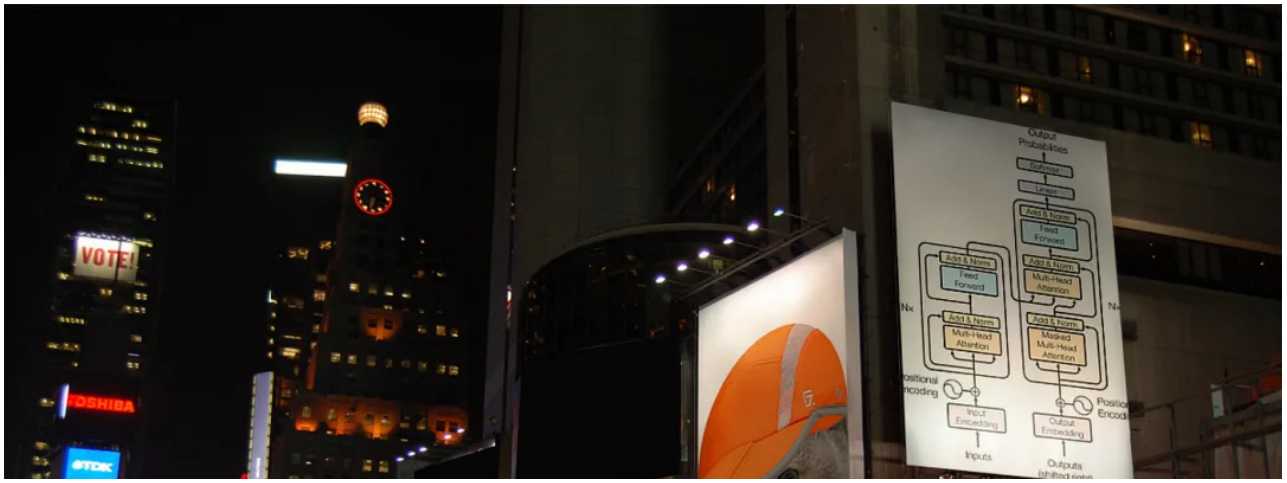


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

我曾经看了不下于几十篇关于Tranformer的视频、教程，但是最后特别是对于Q、K、V非常迷惑。

这篇文章完全解开了我之前的困惑。**所以希望大家一定耐心看完。**



纽约的 Transformer（由**PhotoFunia 创建**）

第一步 - 定义数据集

用于创建 ChatGPT 的数据集为 570 GB。**但是，我们只是为了说明问题，所以我们使用一个非常小的数据集来执行可视化的数值计算。**

Dataset (corpus)

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

我们的整个数据集仅包含三个句子

我们的整个数据集只包含三个句子，这些句子都是来自电视剧的对话。尽管我们的数据集已经清理过，但在 ChatGPT 创建等现实场景中，清理一个 570GB 的数据集需要大量的努力。

第二步——计算词汇量

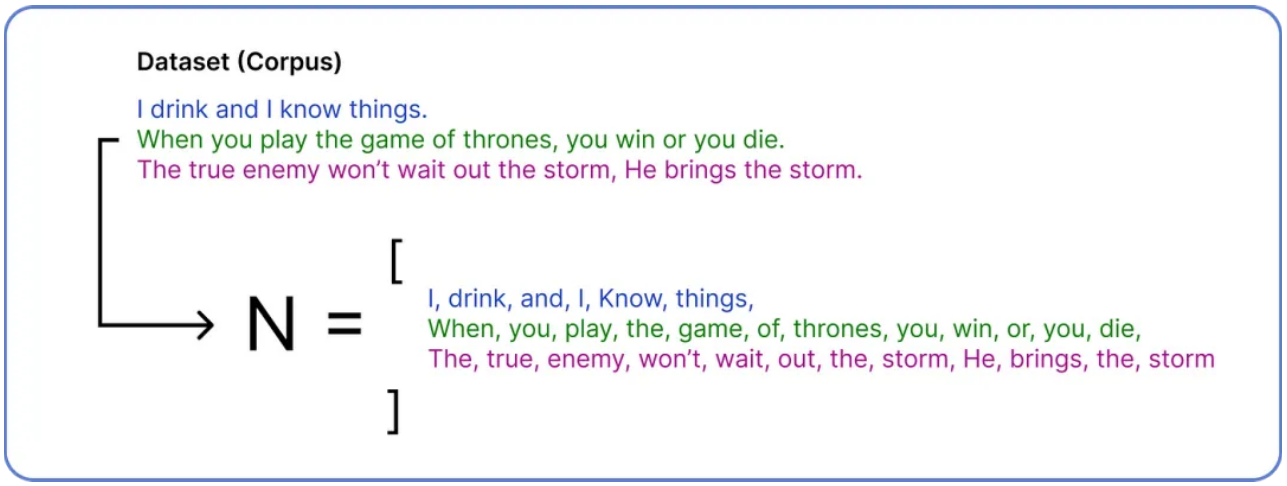
词汇量也就是我们**数据集**中独特单词的总数。它可以通过以下公式计算，其中**独特单词** 的总数为**N**。

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

词汇量公式，其中 N 为总词数

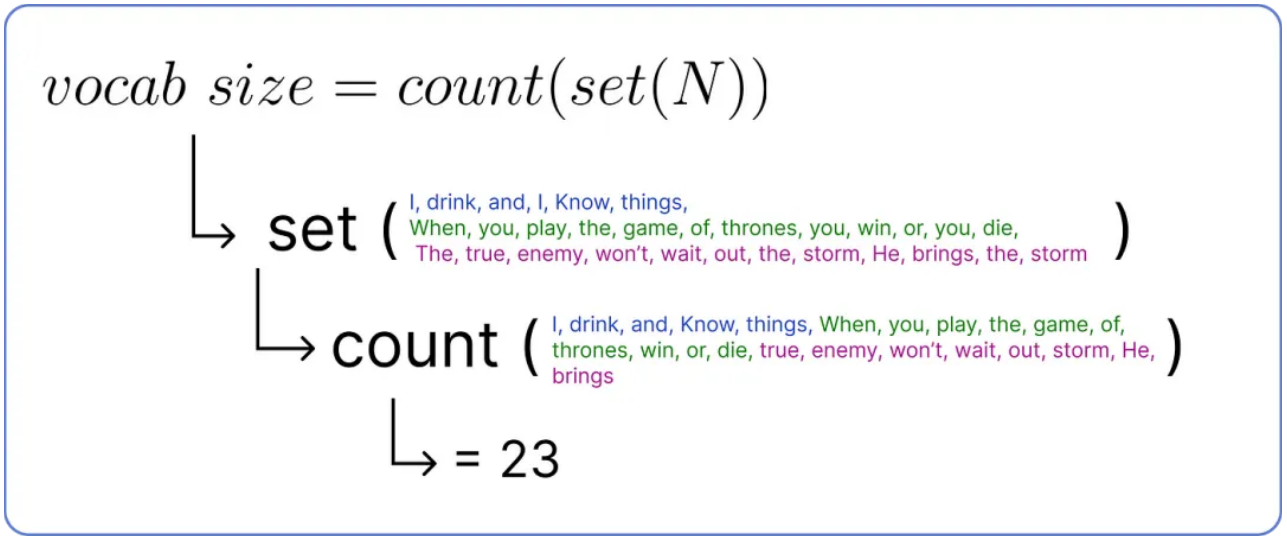
为了找到 N，我们需要将我们的数据集分解成单个单词。

这里用到了一点点集合的知识哦～



> 计算变量 **N**

在获得 N 之后，我们执行集合操作以删除重复项，然后我们可以计算独特单词的数量以确定词汇量。

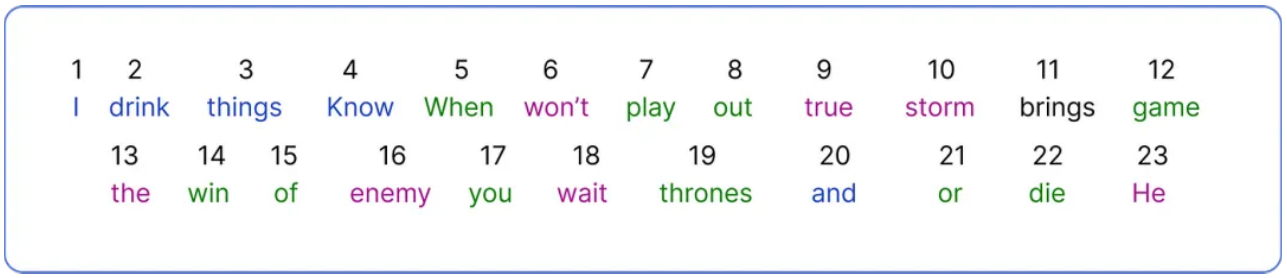


> 查找词汇量大小

因此，词汇量大小为**23**，因为我们的数据集中有**23** 个独特的单词。

步骤 3 — Encoding 编码

现在，我们需要为每个单独的单词分配一个唯一的数字。



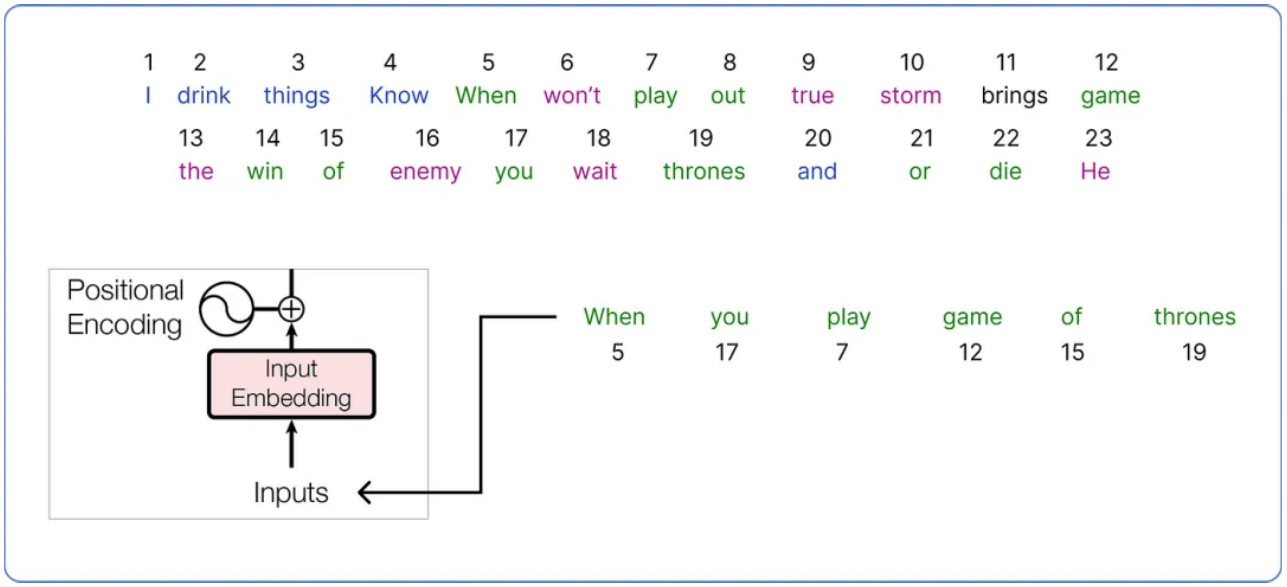
> 为每个单词Encoding 编码

我们已将单个 Token 视为一个单词并为其分配一个数字，ChatGPT 则使用此公式将单词的一部分视为单个Token： `1 个Token = 0.75 个单词`

Encoding 完整个数据集后，现在是时候选择我们的输入并开始使用 Transformer 架构工作了。

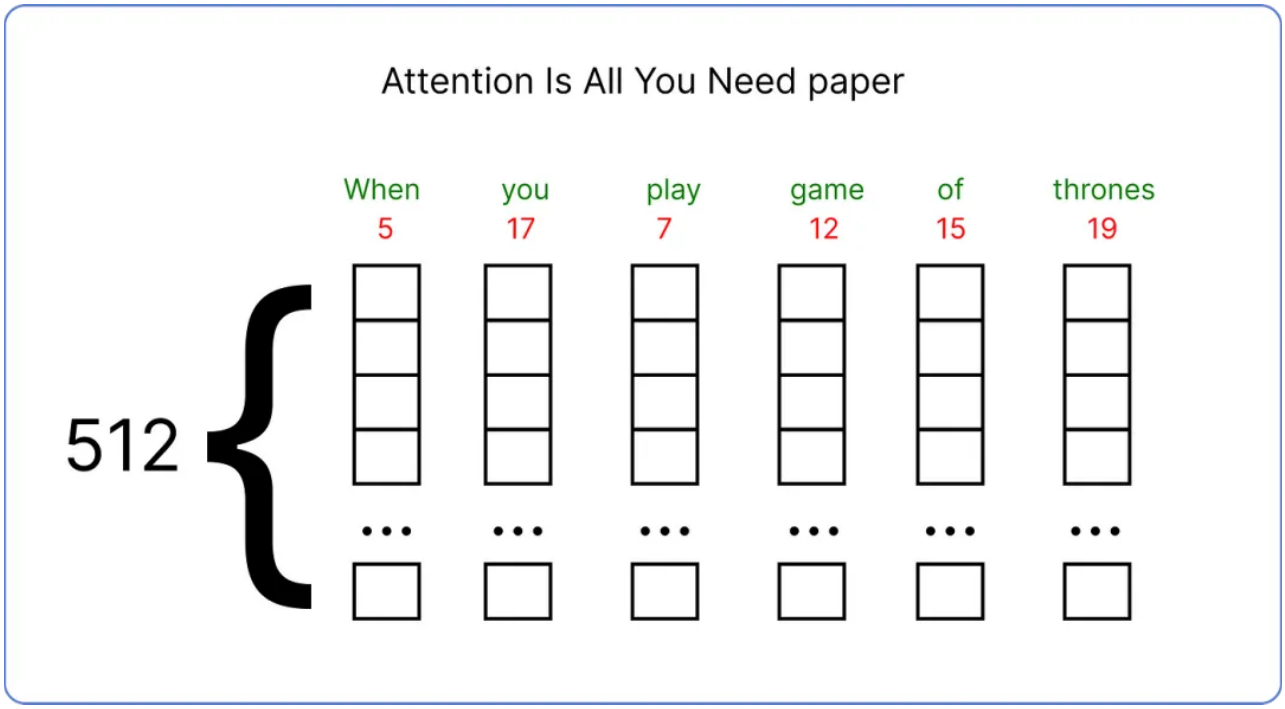
步骤 4 — 计算嵌入Embedding

让我们从我们的语料库中选取一个将在我们的 Transformer 架构中处理的句子。



输入（input）以供 Transformer 使用

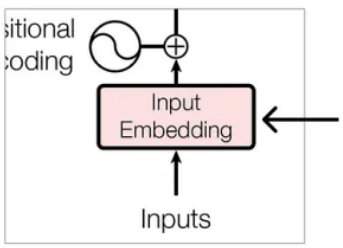
我们已经选择了我们的输入，并需要为其找到一个嵌入向量。原始论文为每个输入词使用了一个 512 维的嵌入向量。



> 原始论文使用 512 维向量

由于在我们的情况下，我们需要使用较小的嵌入向量维度来可视化计算的进行过程。因此，我们将使用嵌入向量的维度 6 。

> 大小 6 其实是靠经验得来的。不过也有大拿给出了一个公式：
>
> **n > 8.33logN**



When	you	play	game	of	thrones
5	17	7	12	15	19

e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75
0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.21

嵌入输入向量

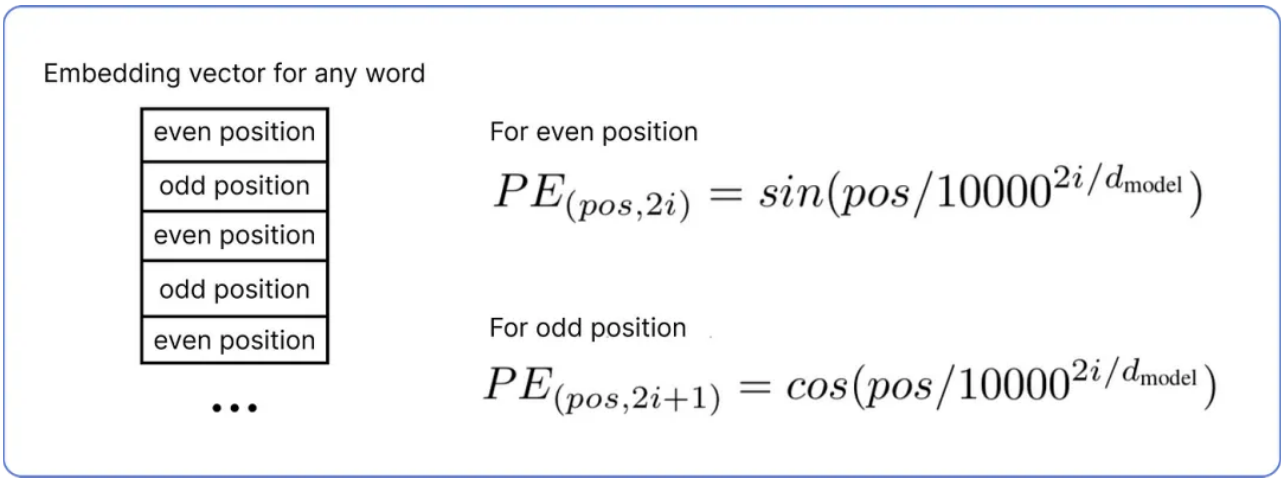
这些嵌入向量的值介于 0 和 1 之间，我们先用随机数来填充一下矩阵。

随着我们的 transformer 开始理解词语之间的含义，这些值随后会通过计算了更新（学习）。

步骤 5 — 计算位置嵌入Positional Embedding

现在我们需要为我们的输入Input 计算位置嵌入Positional Embedding。

根据嵌入向量中第 i 个值的每个词的位置，有两种计算位置嵌入Positional Embedding的公式：



位置嵌入公式

正如您所知，我们的输入句子是“when you play the game of thrones”，起始词是“when”，起始索引（POS）值为 0，维度（d）为 6。

对于 i 从 0 to 5，我们计算输入句子第一个词的位置嵌入Positional Embedding：

When
5

i	e1	Position	Formula	p1
0	0.79	Even	$sin(0/10000^{(2*0/6)})$	0
1	0.6	Odd	$cos(0/10000^{(2*1/6)})$	1
2	0.96	Even	$sin(0/10000^{(2*2/6)})$	0
3	0.64	Odd	$cos(0/10000^{(2*3/6)})$	1
4	0.97	Even	$sin(0/10000^{(2*4/6)})$	0
5	0.2	Odd	$cos(0/10000^{(2*5/6)})$	1

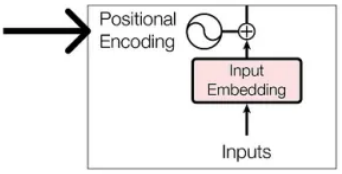
d (dim)	6
POS	0

位置嵌入：单词：**when**

同样，我们可以为我们输入句子（input）中的所有单词计算位置嵌入Positional Embedding。

When	you	play	game	of	thrones
5	17	7	12	15	19

i	p1	p2	p3	p4	p5	p6
0	0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	1	0.0464	0.9957	0.1388	0.1846	0.9732
2	0	0.0022	0.0043	0.0065	0.0086	0.0108
3	1	0.0001	1	0.0003	0.0004	1
4	0	0	0	0	0	0
5	1	0	1	0	0	1
d (dim)	6	6	6	6	6	6
POS	0	1	2	3	4	5



计算输入的位置嵌入**（计算出的值已四舍五入）**

步骤 6 — 连接位置和词嵌入Concatenating Positional and Word Embeddings

在计算位置嵌入后，我们需要添加词嵌入Word Embedding和位置嵌入Positional Embedding。

When	you	play	game	of	thrones
5	17	7	12	15	19

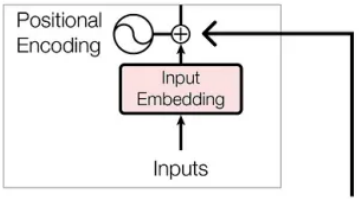
Position Embedding Matrix

p1	p2	p3	p4	p5	p6
0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	0.0464	0.9957	0.1388	0.1846	0.9732
0	0.0022	0.0043	0.0065	0.0086	0.0108
1	0.0001	1	0.0003	0.0004	1
0	0	0	0	0	0
1	0	1	0	0	1

+

Word Embedding Matrix

e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75



=

ep1	ep2	ep3	ep4	ep5	ep6
0.79	1.22	0.92	0.26	0.12	-0.36
1.6	0.17	1.51	0.74	0.59	1.3
0.96	0.06	0.27	0.66	0.8	0.76
1.64	0.79	1.31	0.22	0.62	1.48
0.97	0.9	0.56	0.07	0.5	0.94
1.2	0.74	1.59	0.37	0.7	1.21

0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.21

连接步骤

这个由两个矩阵（**词嵌入矩阵**和**位置嵌入矩阵**）组合而成的结果矩阵将被视为编码部分的输入。

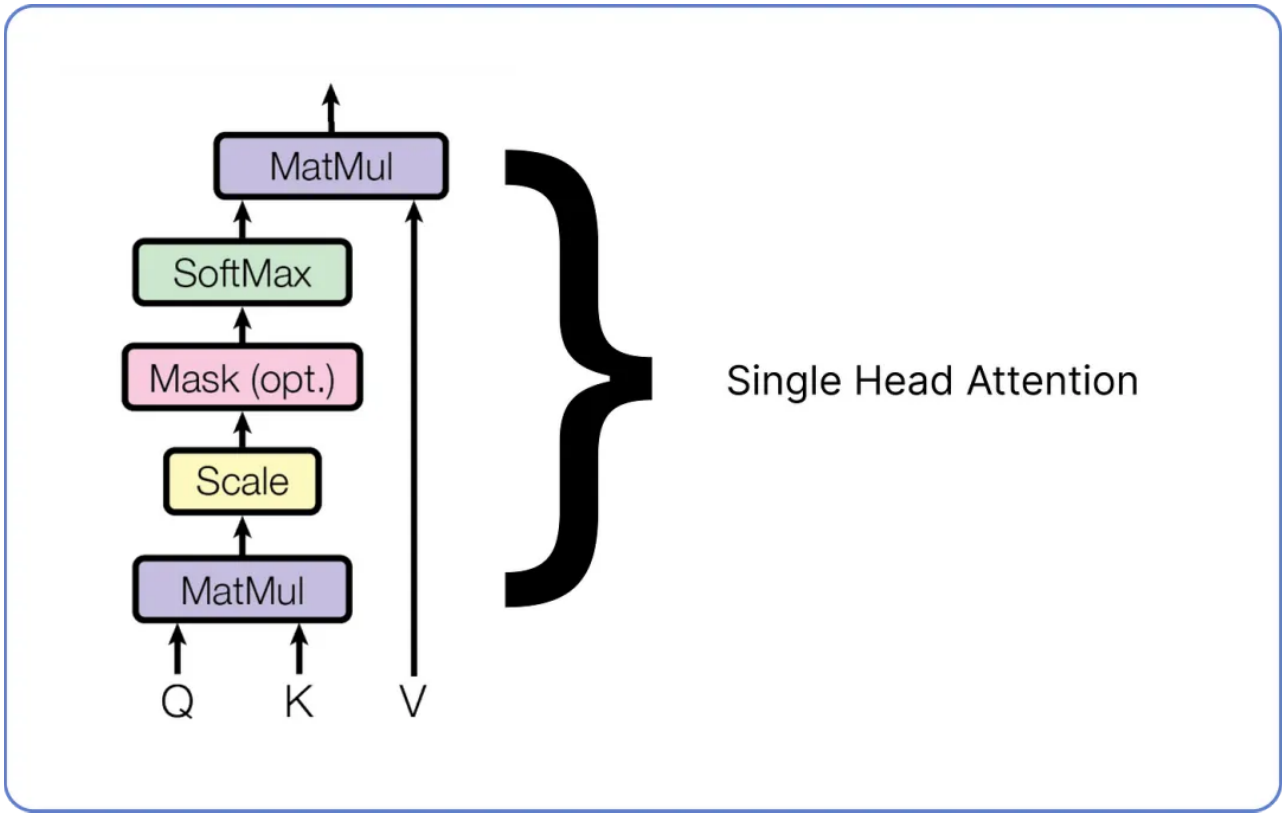
步骤 7 — 多头注意力Multi Head Attention

多头注意力由许多单头注意力组成。

我们需要组合多少个单头注意力取决于我们。

例如，Meta 的 LLaMA LLM 在编码器架构中使用了 32 个单头注意力。

以下是单头注意力外观的示意图。

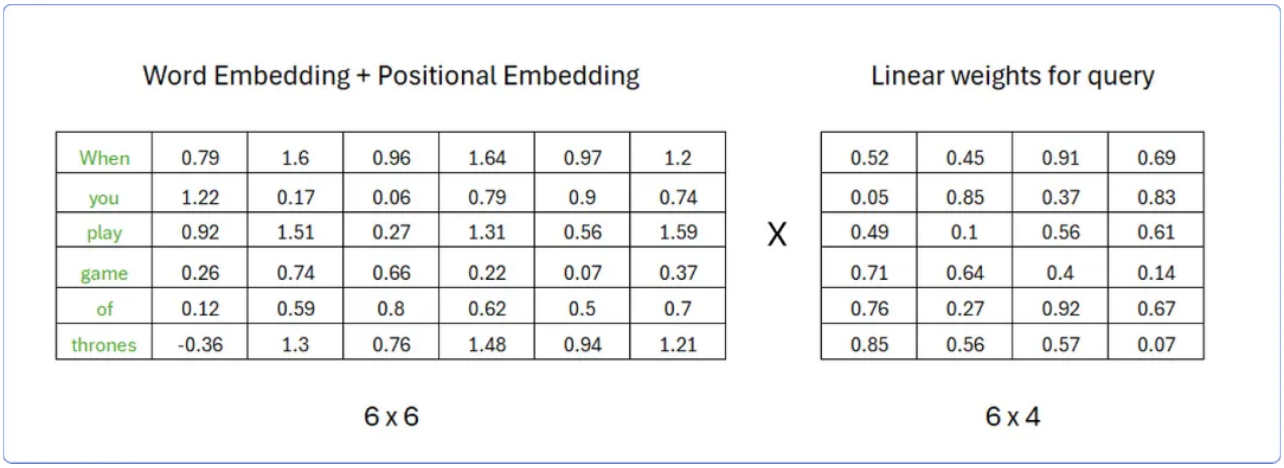


单头注意力机制在 Transformer 中有三个输入：**查询Query**、**键Key**和**值Value**。

这些矩阵是通过将之前计算的**相同矩阵**的转置与**词嵌入矩阵**和**位置嵌入矩阵**相加，**乘以不同的权重矩阵**获得的。

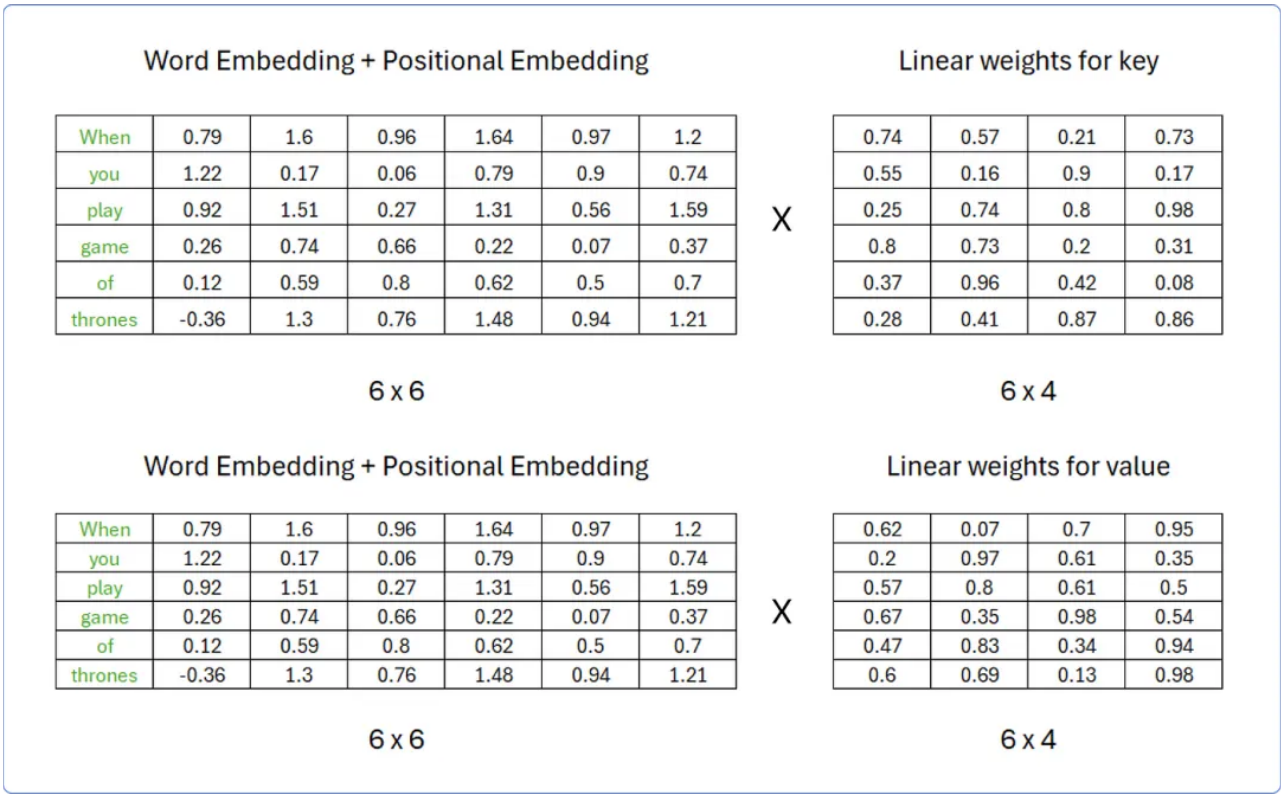
假设，为了计算**查询矩阵Query**，**权重矩阵**的行数必须与**转置矩阵**的列数相同，而**权重矩阵**的列数可以是任意的；例如，我们假设权重矩阵中有 `4` 列。

权重矩阵中的值是 `0` 和 `1` 之间随机数，当我们的转换器开始学习这些词的意义时，这些值将随后被更新。



计算查询矩阵Query

同样，我们可以使用相同的程序来计算**键** 和**值** 矩阵，但权重矩阵中的值必须对两者都不同。



计算键Key和值Value矩阵

因此，在矩阵相乘后，得到的结果**查询Query**、**键Key** 和**值Value** 如下：

Query

3.88	3.8	4.08	3.42
2.55	1.86	2.77	1.78
3.39	3.6	3.49	2.72
1.02	1.18	1.24	1.3
1.9	1.56	1.88	1.53
3.04	2.9	2.73	2.22

6 x 4

key

3.71	4.04	4.15	3.41
2.18	2.51	1.64	1.93
3.28	3.11	3.65	3.01
1.07	1.13	1.64	1.35
1.49	1.97	2.14	1.81
2.51	3.04	3.45	2.28

6 x 4

value

3.88	3.8	4.08	3.42
2.55	1.86	2.77	1.78
3.39	3.6	3.49	2.72
1.02	1.18	1.24	1.3
1.9	1.56	1.88	1.53
3.04	2.9	2.73	2.22

6 x 4

查询、键、值矩阵

现在我们已经有了这三个矩阵，让我们一步一步地开始计算单头注意力。

Query

3.88	3.8	4.08	3.42
2.55	1.86	2.77	1.78
3.39	3.6	3.49	2.72
1.02	1.18	1.24	1.3
1.9	1.56	1.88	1.53
3.04	2.9	2.73	2.22

6 x 4

X

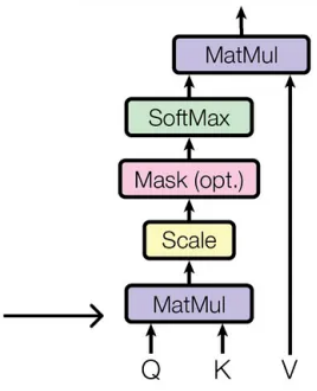
Transpose (Key)

3.71	2.18	3.28	1.07	1.49	2.51
4.04	2.51	3.11	1.13	1.97	3.04
4.15	1.64	3.65	1.64	2.14	3.45
3.41	1.93	3.01	1.35	1.81	2.28

4 x 6

=

58.341	31.2882	49.7306	19.7538	28.1886	43.1644
34.5402	18.2058	29.6169	11.7761	16.6133	25.6698
50.8796	27.3994	43.2409	17.0909	24.5349	37.695
18.1304	9.728	15.4544	6.2134	8.851	13.3894
26.3707	14.0937	22.5509	8.9445	12.6967	19.4858
41.8941	22.668	35.6369	14.004	20.103	30.9265



查询与键的矩阵乘法

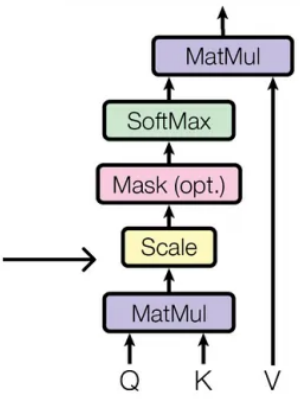
为了缩放结果矩阵，我们必须重复使用我们的嵌入向量（embedding vector）的维度，即 6 。

58.341	31.2882	49.7306	19.7538	28.1886	43.1644
34.5402	18.2058	29.6169	11.7761	16.6133	25.6698
50.8796	27.3994	43.2409	17.0909	24.5349	37.695
18.1304	9.728	15.4544	6.2134	8.851	13.3894
26.3707	14.0937	22.5509	8.9445	12.6967	19.4858
41.8941	22.668	35.6369	14.004	20.103	30.9265

$\sqrt{d_k}$ where d (dimension) is 6

=

23.81721	12.77409	20.30219	8.062904	11.50852	17.62
14.1009	7.434201	12.09231	4.809165	6.781004	10.47973
20.77167	11.186	17.65266	6.976963	10.01433	15.39096
7.401542	3.972256	6.307436	2.535222	3.612997	5.466445
10.76551	5.752218	9.205999	3.64974	5.184753	7.956759
17.10152	9.254989	14.54997	5.715476	8.205791	12.62712



缩放结果矩阵，维度为**5**

下一步是**掩码**是可选的，这里我们不计算。

> 掩码就像告诉模型只关注某个点之前发生的事情，在确定句子中不同单词的重要性时不要窥视未来。它帮助模型以逐步的方式理解事物，而不会通过提前查看来作弊。

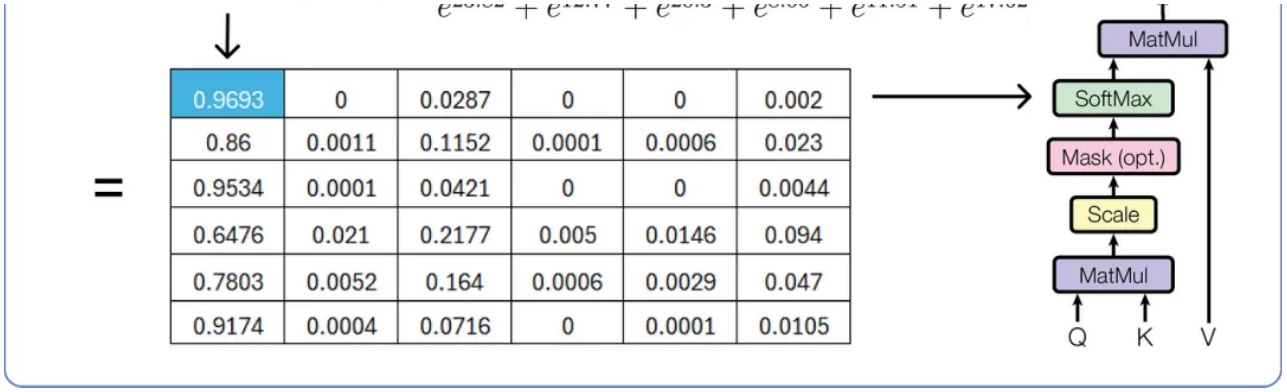
因此，我们现在将对缩放后的结果矩阵应用**softmax** 操作。

23.82	12.77	20.3	8.06	11.51	17.62
14.1	7.43	12.09	4.81	6.78	10.48
20.77	11.19	17.65	6.98	10.01	15.39
7.4	3.97	6.31	2.54	3.61	5.47
10.77	5.75	9.21	3.65	5.18	7.96
17.1	9.25	14.55	5.72	8.21	12.63

SoftMax

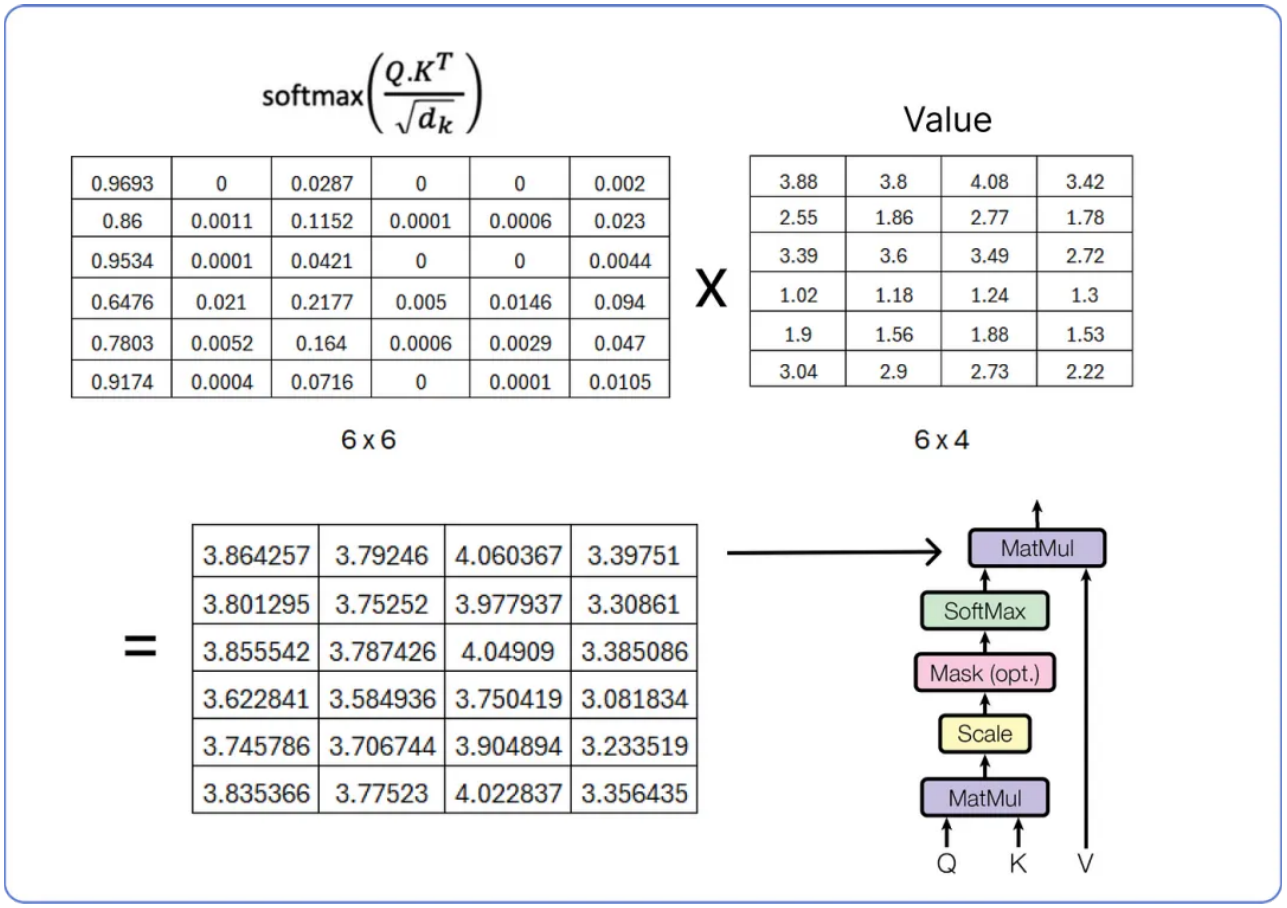
$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$$\text{softmax}(23.82) = \frac{e^{23.82}}{e^{23.82} + e^{12.77} + e^{20.3} + e^{8.06} + e^{11.51} + e^{17.62}}$$



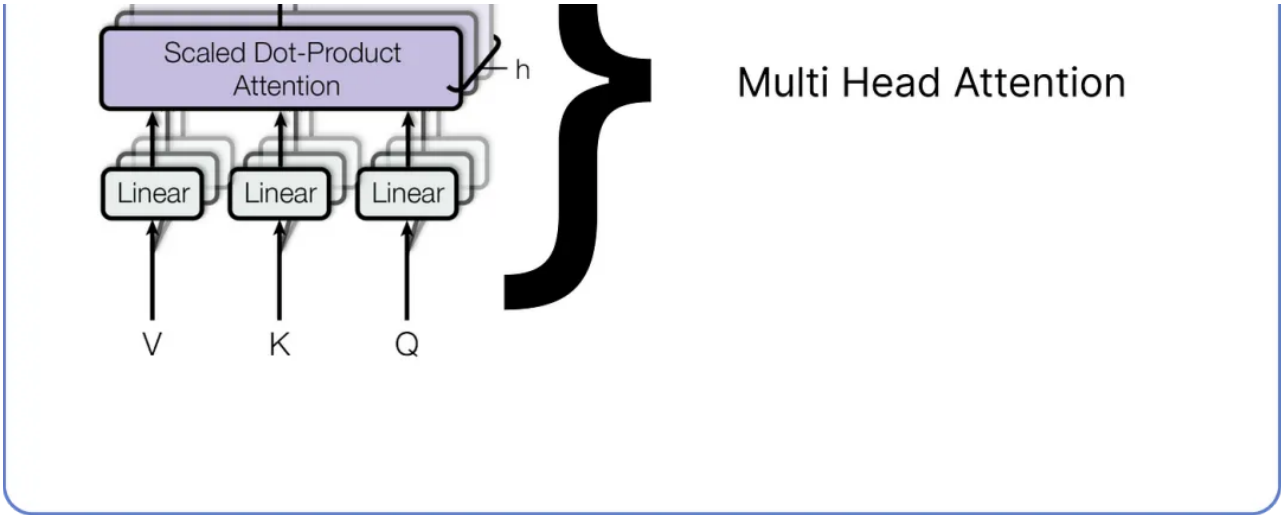
应用 softmax 到结果矩阵

执行最终的乘法步骤以从单头注意力中获取结果矩阵。



计算单头注意力的最终矩阵

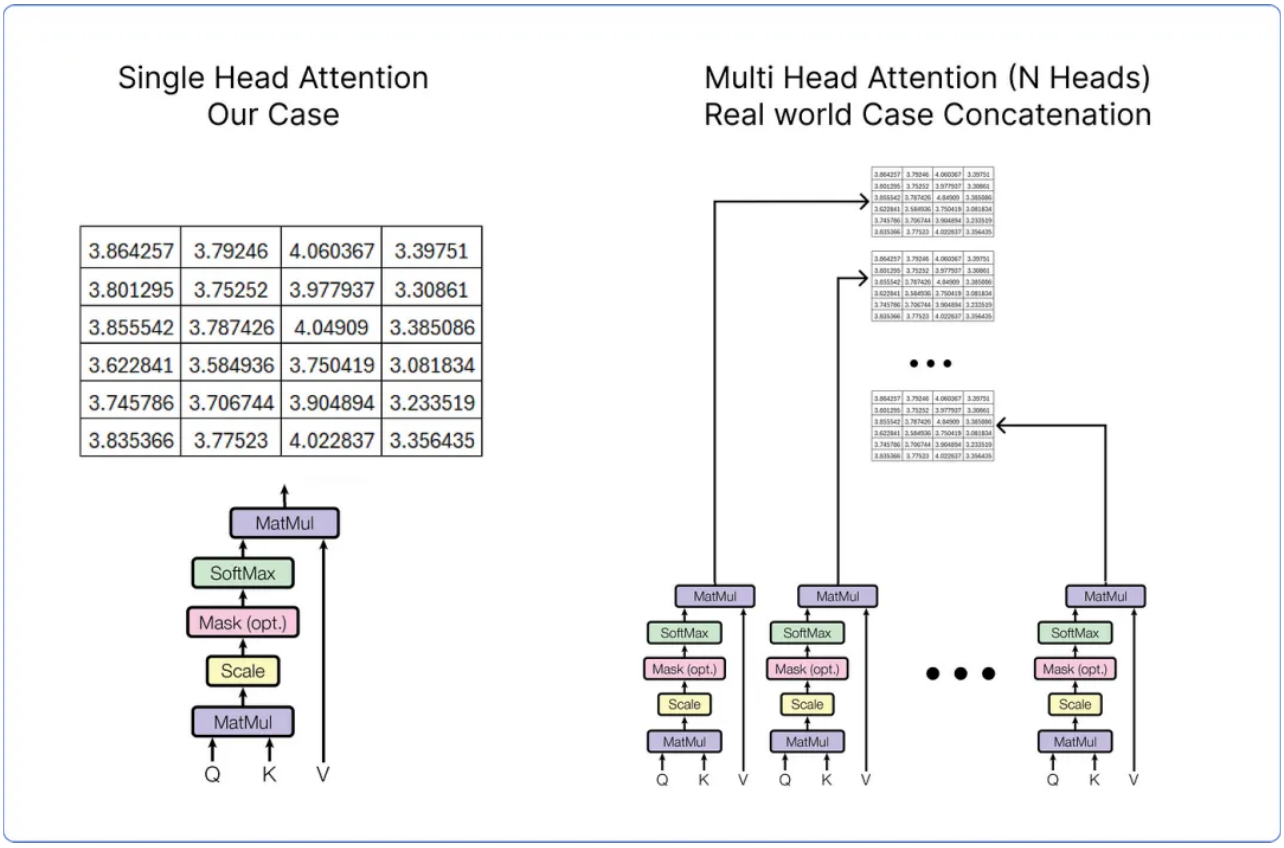
我们已经计算了单头注意力，而多头注意力由多个单头注意力组成，正如我之前所述。下面是它的可视化效果：



多头注意力机制在 Transformer 中

每个单头注意力有三个输入：**查询**、**键**和**值**，每个都有不同的权重集。一旦所有单头注意力输出它们的结果矩阵，它们将被连接起来，最终的连接矩阵再次通过乘以一组随机初始化的权重矩阵进行线性变换，这些权重矩阵将在 transformer 开始训练时进行更新。

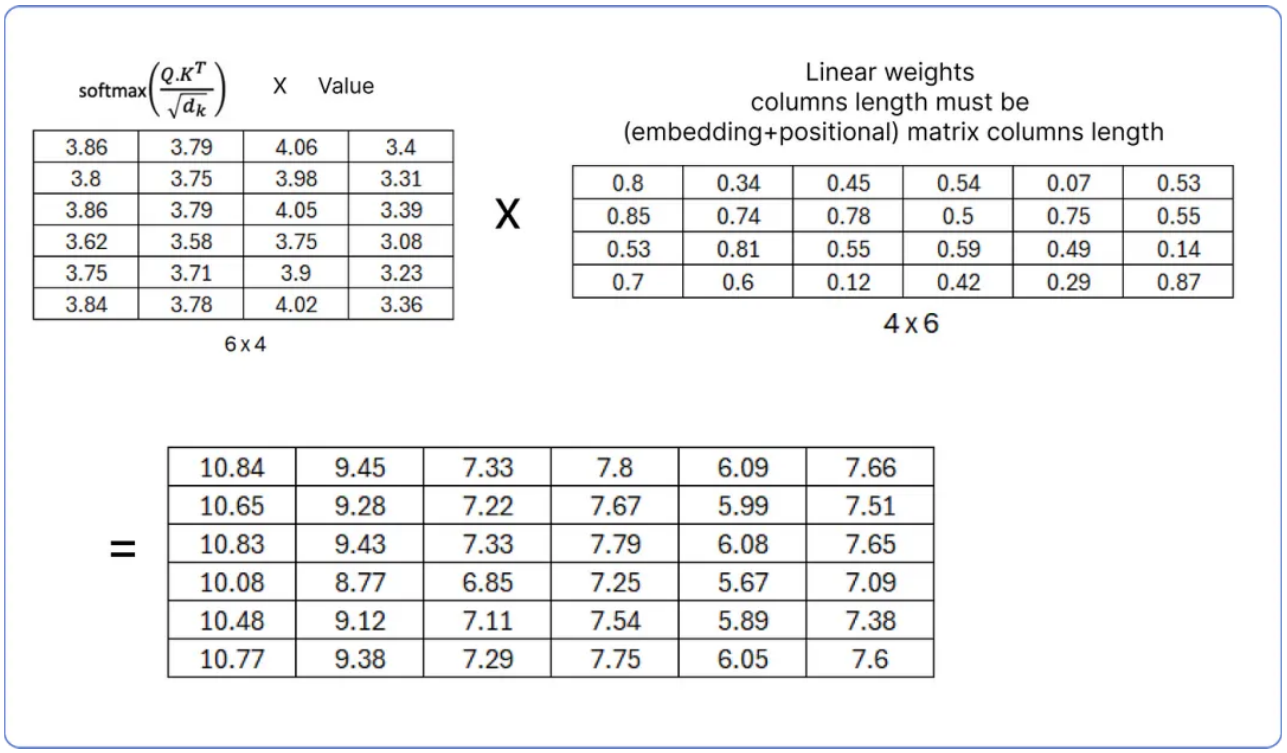
由于在我们的情况下，我们考虑的是单头注意力，但如果我们在处理多头注意力，它看起来是这样的。



单头注意力与多头注意力

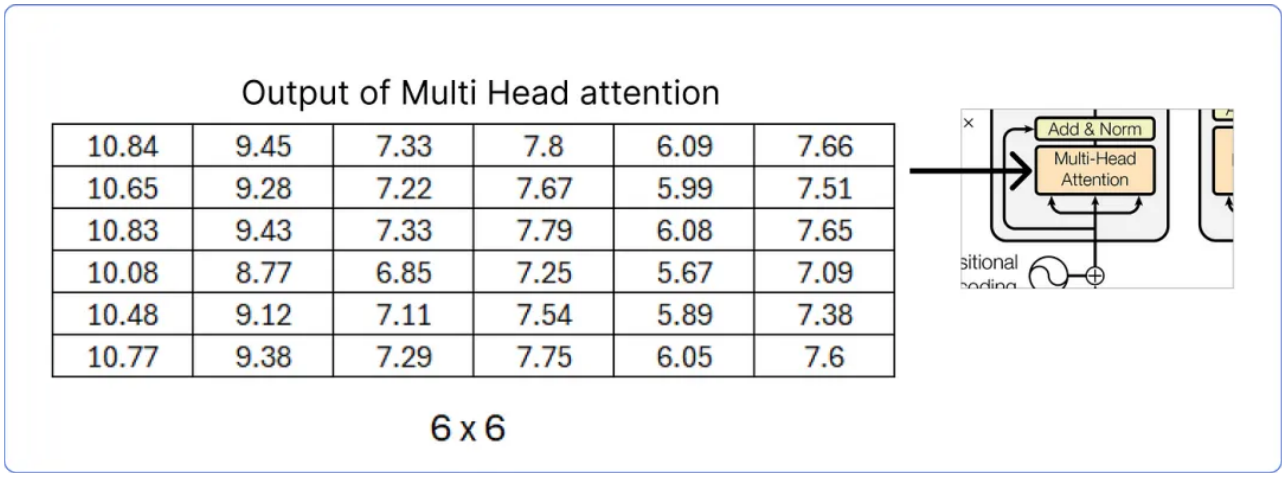
在任何情况下，无论是单头注意力还是多头注意力，结果矩阵都需要再次通过乘以一组权

重矩阵进行线性变换。



标准化单头注意力矩阵

确保线性权重矩阵的列数必须等于我们之前计算的矩阵（**词嵌入 + 位置嵌入**）的列数，因为在下一步，我们将把结果归一化矩阵与（**词嵌入 + 位置嵌入**）矩阵相加。



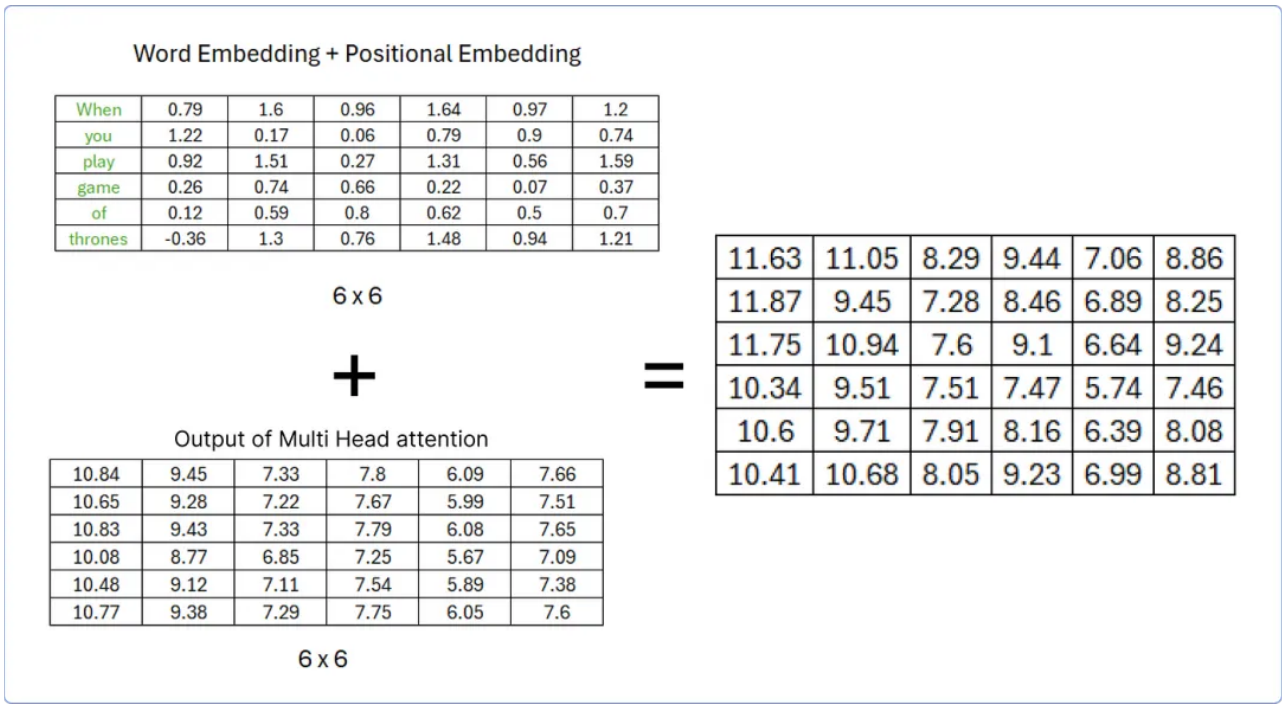
输出多头注意力矩阵

我们已计算出多头注意力的结果矩阵，接下来，我们将进行添加和归一化步骤。

步骤 8 — 添加和归一化

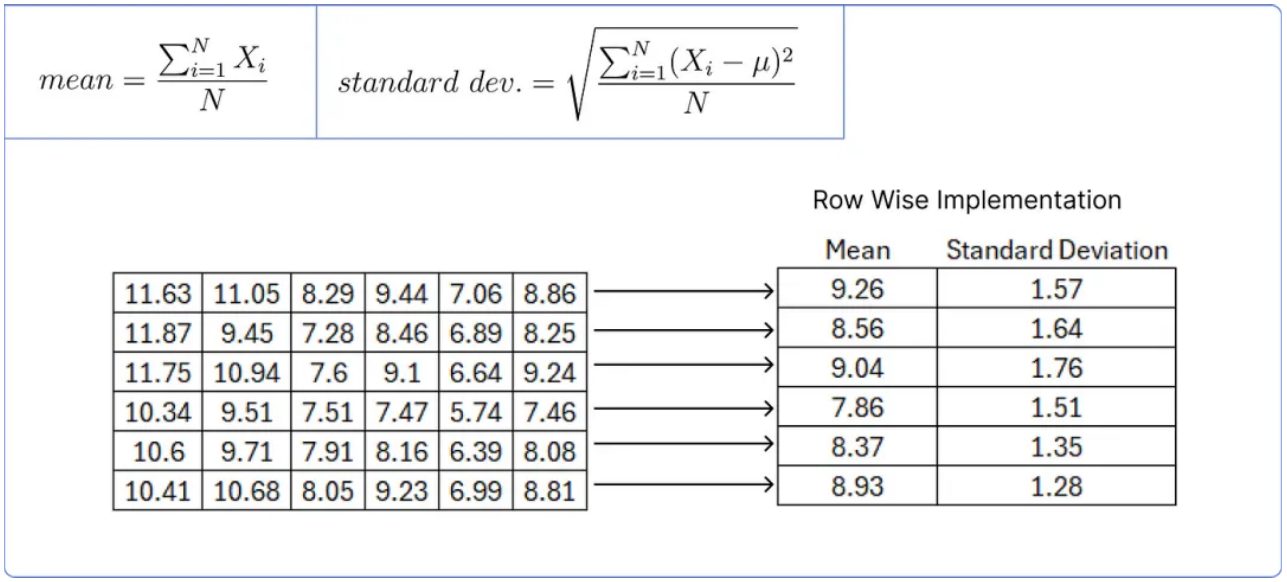
一旦我们从多头注意力中获取到结果矩阵，我们必须将其添加到我们的原始矩阵中。我们

先来做这个。



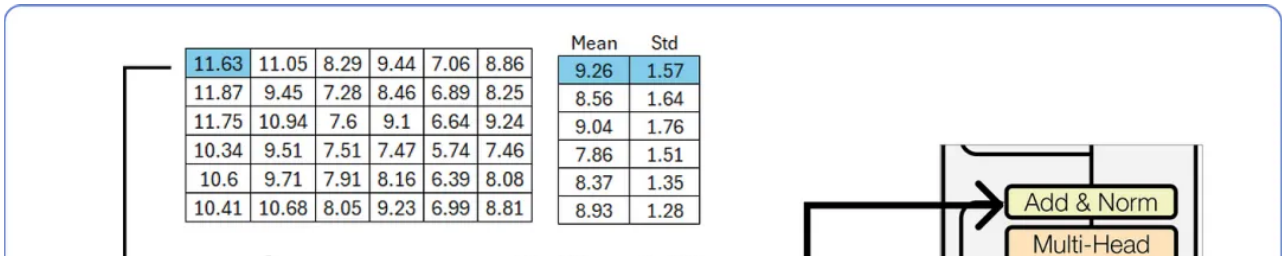
添加矩阵以执行加法和范数步骤

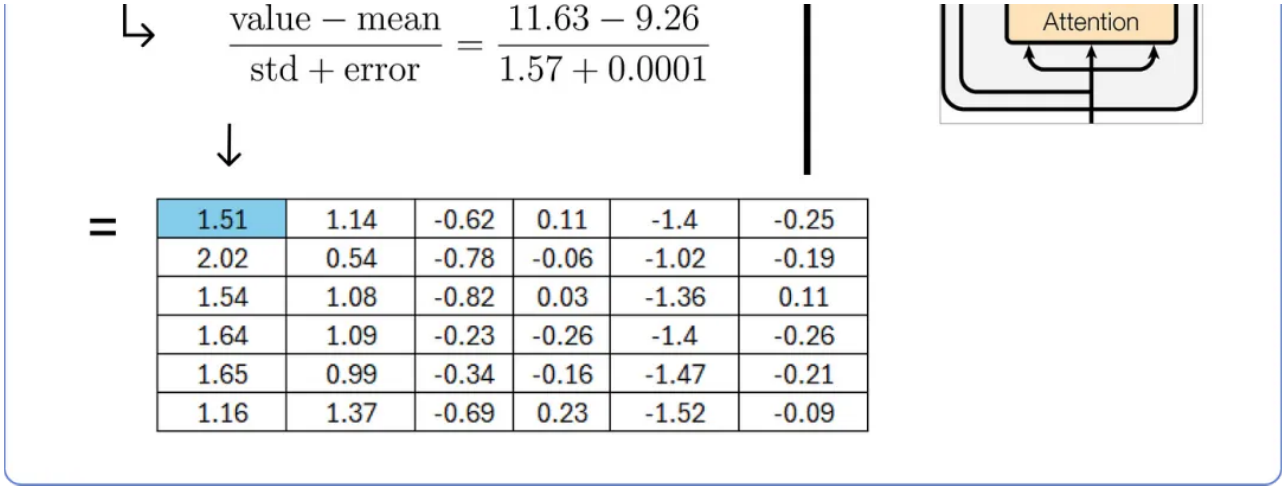
为了规范化上述矩阵，我们需要计算每行的均值和标准差。



计算均值和标准差

我们用矩阵中每个值减去对应行的平均值，然后除以对应的标准差。



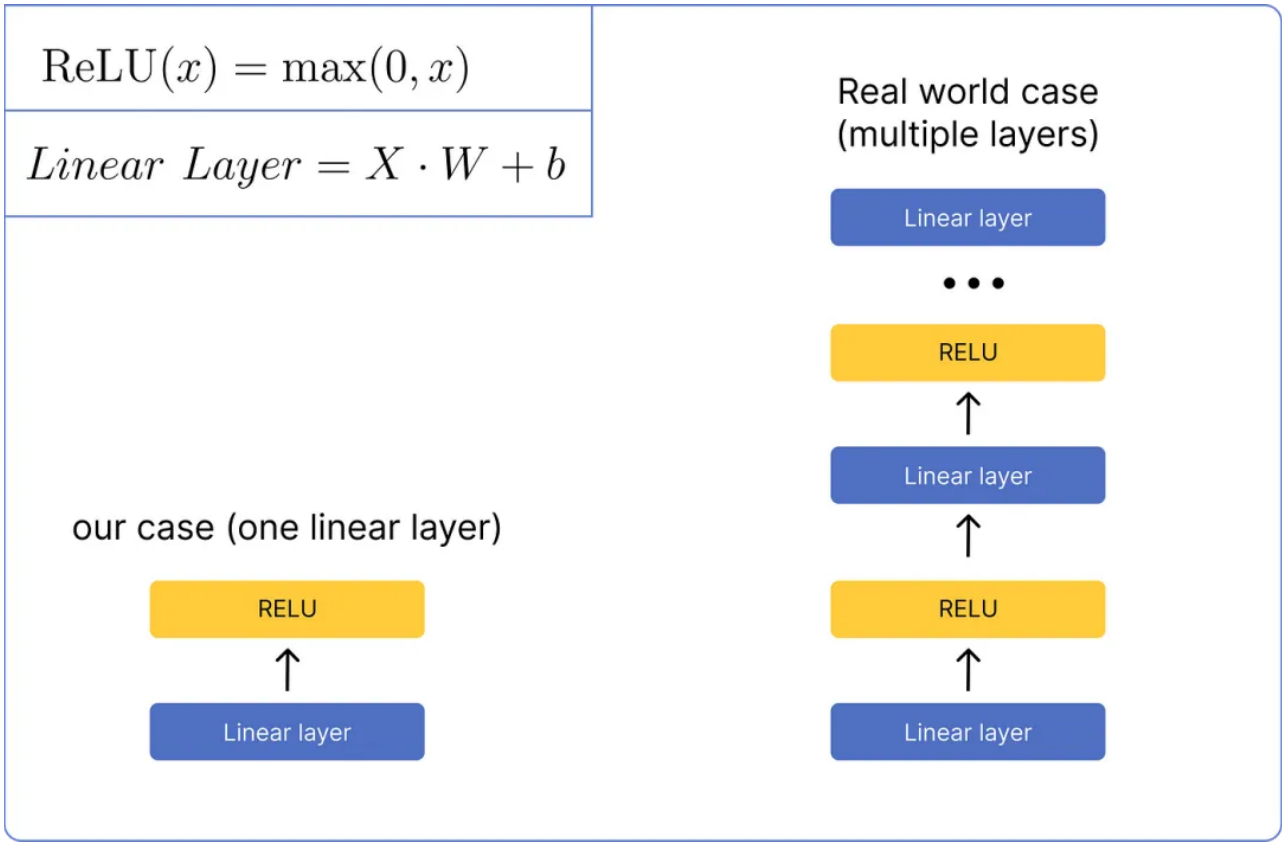


标准化结果矩阵

添加一个小的误差值可以防止分母为零，从而避免使整个项趋于无穷大。

步骤 9 — 前馈网络

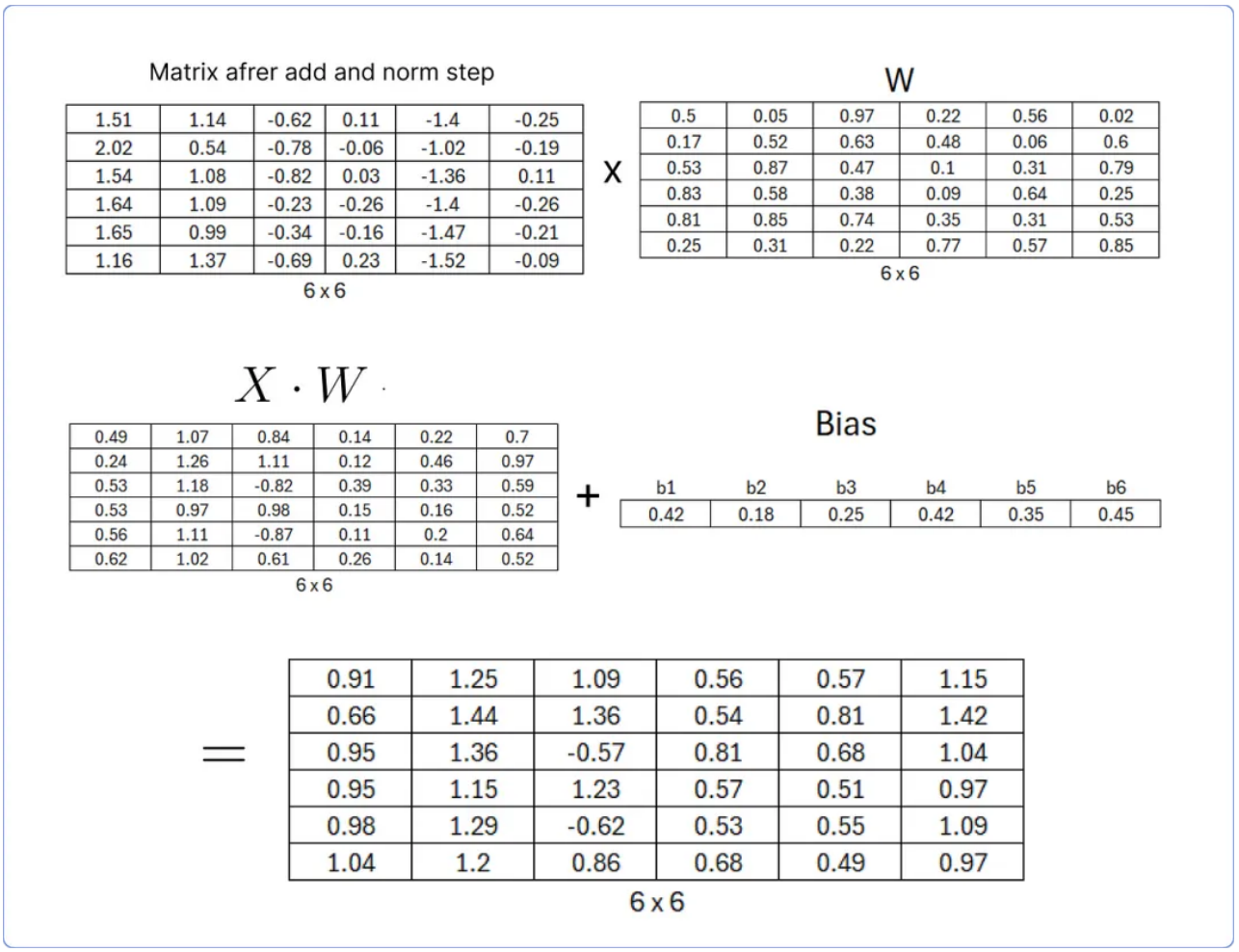
在将矩阵归一化后，它将通过前馈网络进行处理。我们将使用一个非常基本的网络，该网络只包含一个线性层和一个 ReLU 激活函数层。这是它的视觉外观：



前馈网络比较

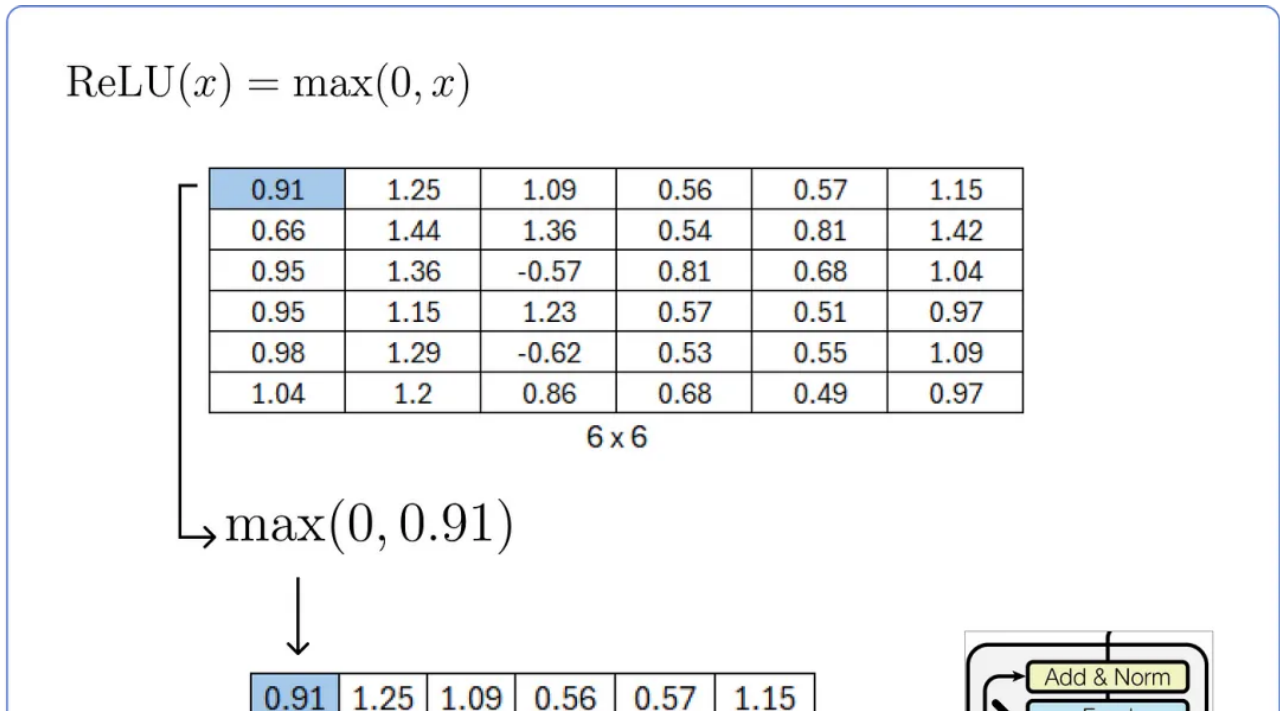
首先，我们需要通过将我们最后计算的矩阵与一组随机的权重矩阵相乘来计算线性层，该

权重矩阵在 transformer 开始学习时将更新，并将结果矩阵添加到一个也包含随机值的偏置矩阵中。

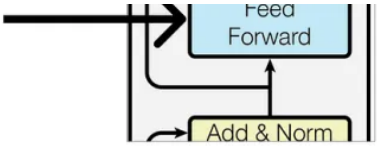


计算线性层

在计算线性层之后，我们需要将其通过 ReLU 层并使用其公式。



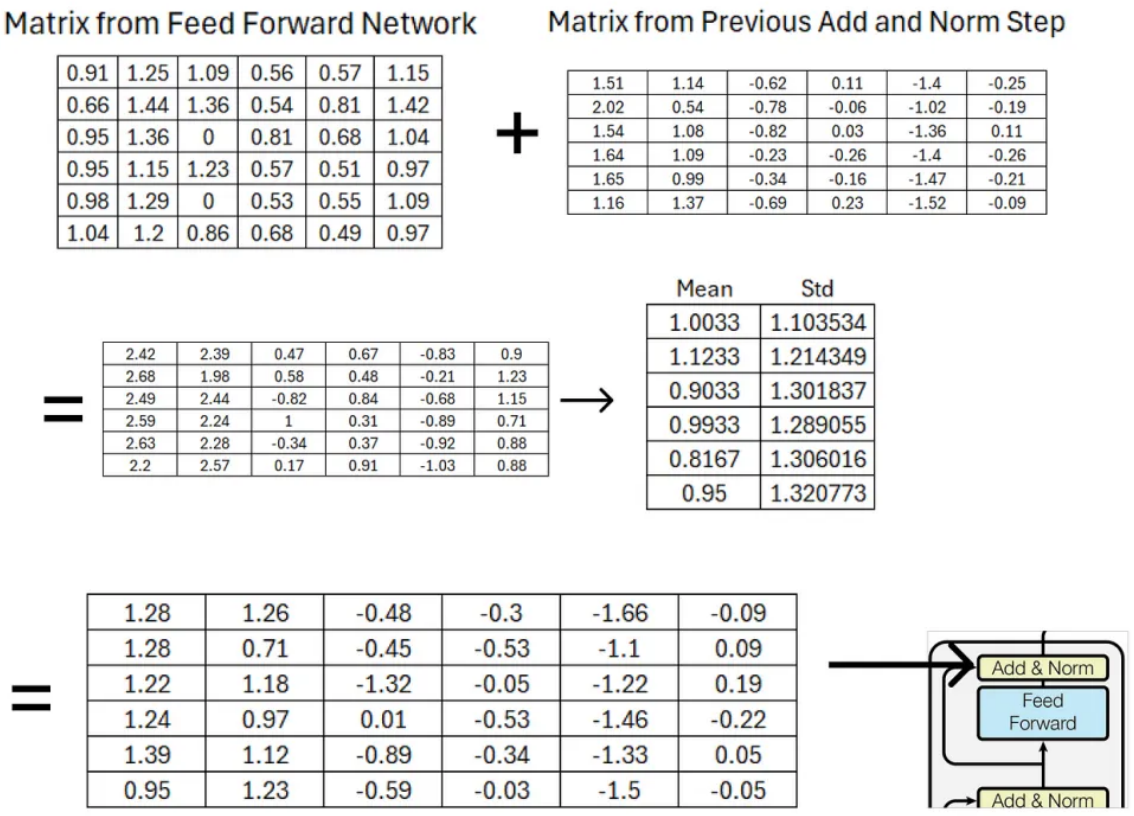
0.66	1.44	1.36	0.54	0.81	1.42
0.95	1.36	0	0.81	0.68	1.04
0.95	1.15	1.23	0.57	0.51	0.97
0.98	1.29	0	0.53	0.55	1.09
1.04	1.2	0.86	0.68	0.49	0.97



计算 ReLU 层

第 10 步 — 再次添加和归一化

一旦我们从前馈网络获得结果矩阵，我们必须将其添加到从先前添加和归一化步骤获得的矩阵中，然后使用行均值和标准差对其进行归一化。



0.91	1.25	1.09	0.56	0.57	1.15
0.66	1.44	1.36	0.54	0.81	1.42
0.95	1.36	0	0.81	0.68	1.04
0.95	1.15	1.23	0.57	0.51	0.97
0.98	1.29	0	0.53	0.55	1.09
1.04	1.2	0.86	0.68	0.49	0.97

1.51	1.14	-0.62	0.11	-1.4	-0.25
2.02	0.54	-0.78	-0.06	-1.02	-0.19
1.54	1.08	-0.82	0.03	-1.36	0.11
1.64	1.09	-0.23	-0.26	-1.4	-0.26
1.65	0.99	-0.34	-0.16	-1.47	-0.21
1.16	1.37	-0.69	0.23	-1.52	-0.09

2.42	2.39	0.47	0.67	-0.83	0.9
2.68	1.98	0.58	0.48	-0.21	1.23
2.49	2.44	-0.82	0.84	-0.68	1.15
2.59	2.24	1	0.31	-0.89	0.71
2.63	2.28	-0.34	0.37	-0.92	0.88
2.2	2.57	0.17	0.91	-1.03	0.88

Mean	Std
1.0033	1.103534
1.1233	1.214349
0.9033	1.301837
0.9933	1.289055
0.8167	1.306016
0.95	1.320773

1.28	1.26	-0.48	-0.3	-1.66	-0.09
1.28	0.71	-0.45	-0.53	-1.1	0.09
1.22	1.18	-1.32	-0.05	-1.22	0.19
1.24	0.97	0.01	-0.53	-1.46	-0.22
1.39	1.12	-0.89	-0.34	-1.33	0.05
0.95	1.23	-0.59	-0.03	-1.5	-0.05

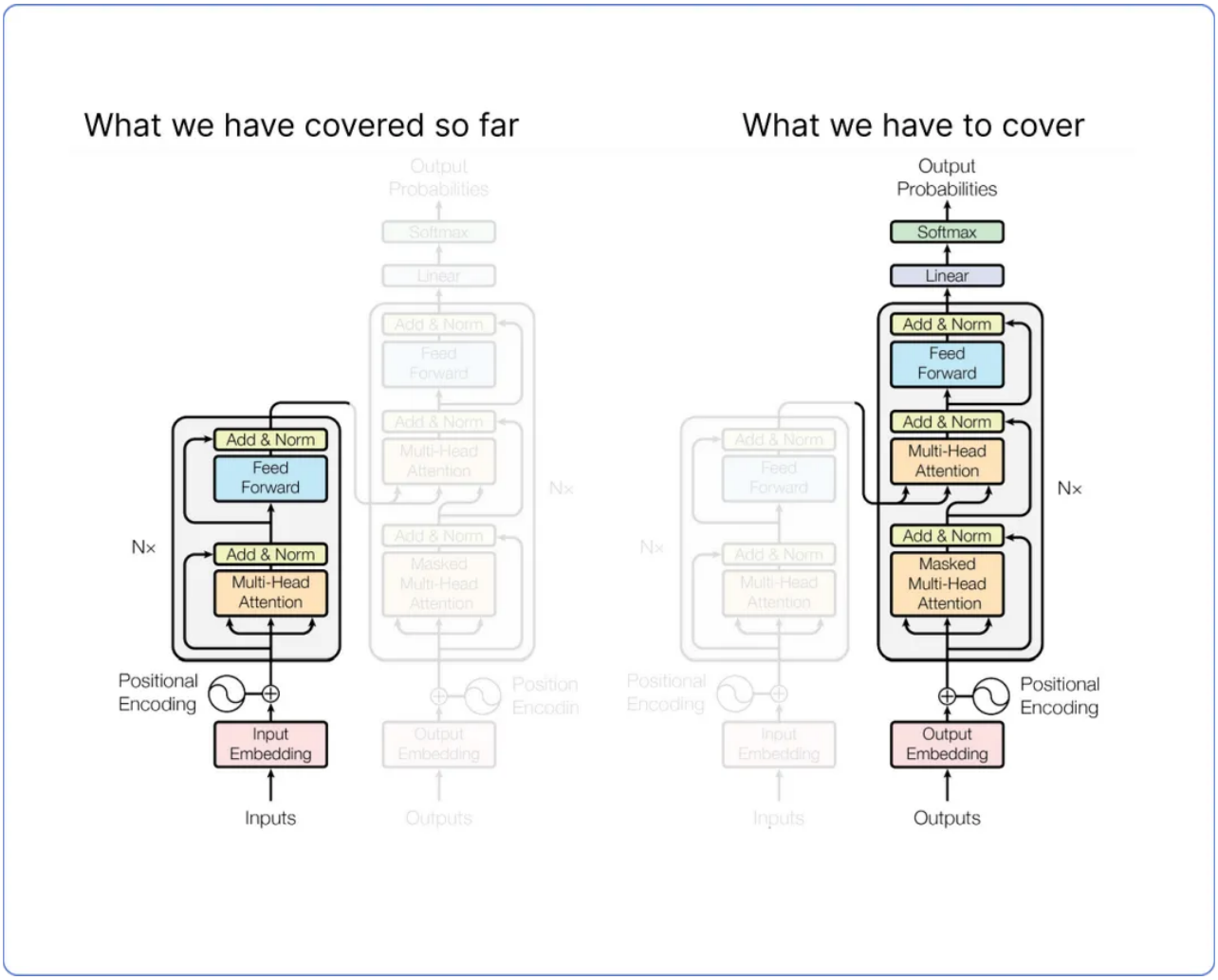
添加和归一化在前馈网络之后

该加法和归一化步骤的输出矩阵将作为解码器部分中存在的多头注意力机制之一的查询和键矩阵，您可以通过从加法和归一化追踪到解码器部分来轻松理解。

步骤 11 — 解码器部分

好消息是，到目前为止，我们已经计算了**编码器部分**，我们所执行的每一个步骤，从编码我们的数据集到将我们的矩阵通过前馈网络传递，都是独特的。这意味着我们之前没有计算过它们。但从现在开始，所有即将到来的步骤，即变换器（**解码器部分**）的剩余架构，都将涉及类似类型的矩阵乘法。

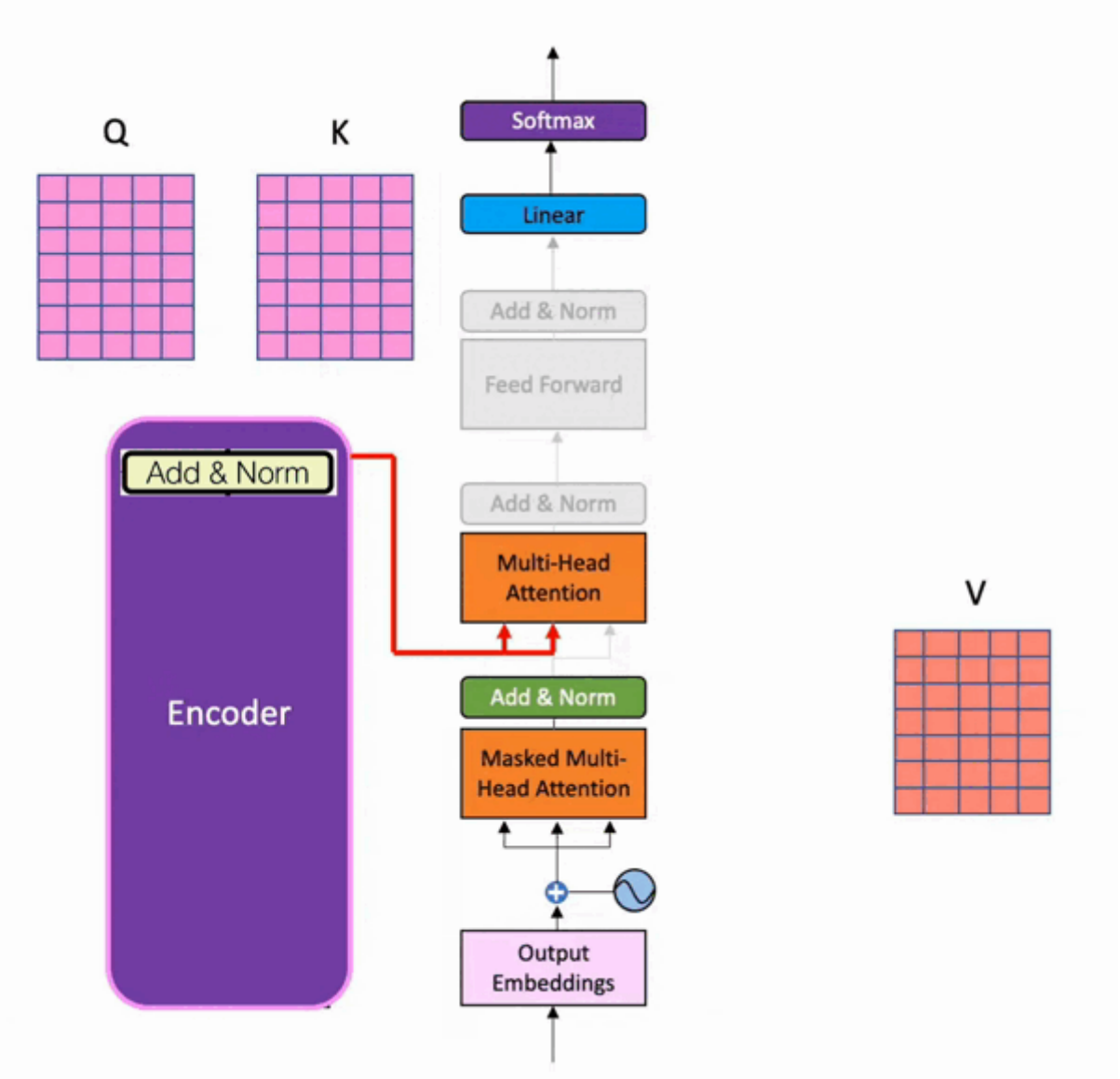
查看我们的 Transformer 架构。到目前为止我们已经覆盖的内容以及我们还需要覆盖的内容：



即将进行的步骤插图

我们不会计算整个解码器，因为其中大部分部分包含与我们已经在编码器中完成的类似计算。详细计算解码器只会因为重复步骤而使博客变长。相反，我们只需要关注解码器的输入和输出计算。

在训练时，解码器有两个输入。一个是来自编码器，其中最后一个加和归一化层的输出矩阵作为**查询**和**键**，用于解码器部分的第二个多头注意力层。以下是它的可视化（来自 **batool haider**）：



可视化来自**巴图尔·海德**

当值矩阵来自解码器在第一次**添加和归一化** 步骤之后。

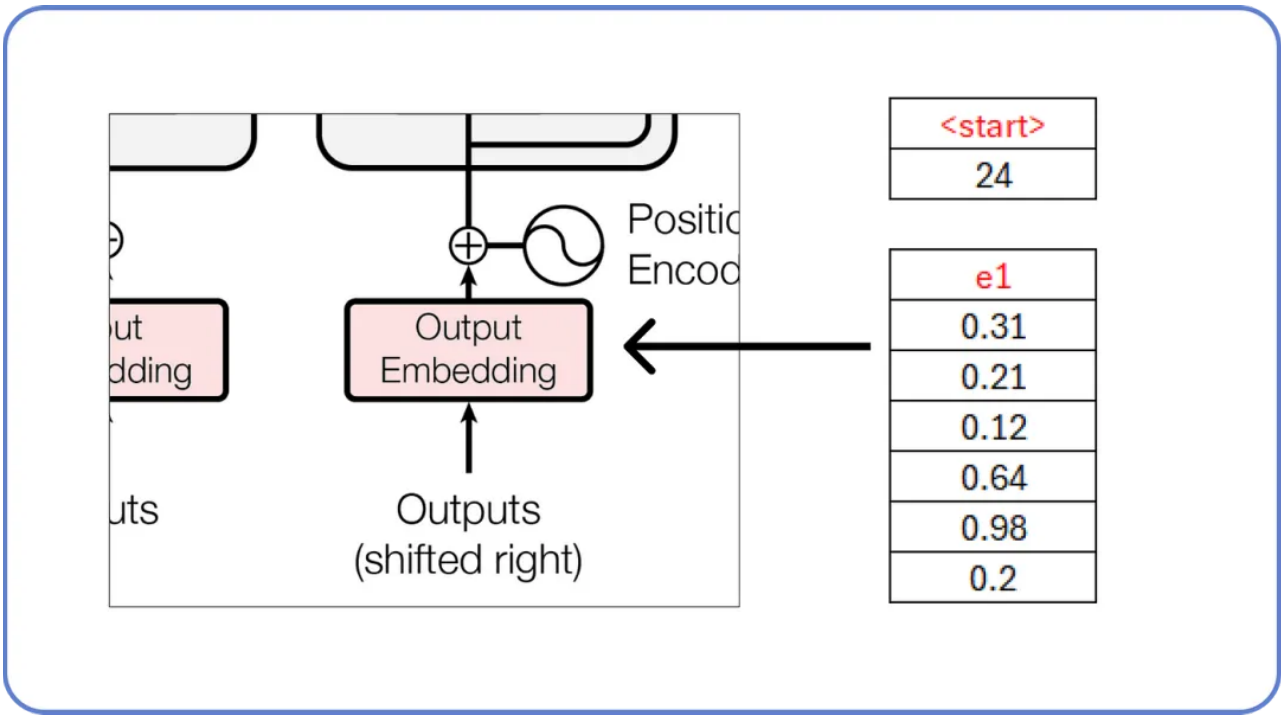
解码器的第二个输入是预测的文本。如果你还记得，我们输入到编码器的是 当你玩权力的游戏 ，所以解码器的输入是预测的文本，在我们的例子中是 你赢或你死 。

但是预测输入文本需要遵循一个标准的令牌包装，使 transformer 知道从哪里开始和在哪里结束。

Encoder Input ⟶ When you play game of thrones

输入：编码器与解码器的比较

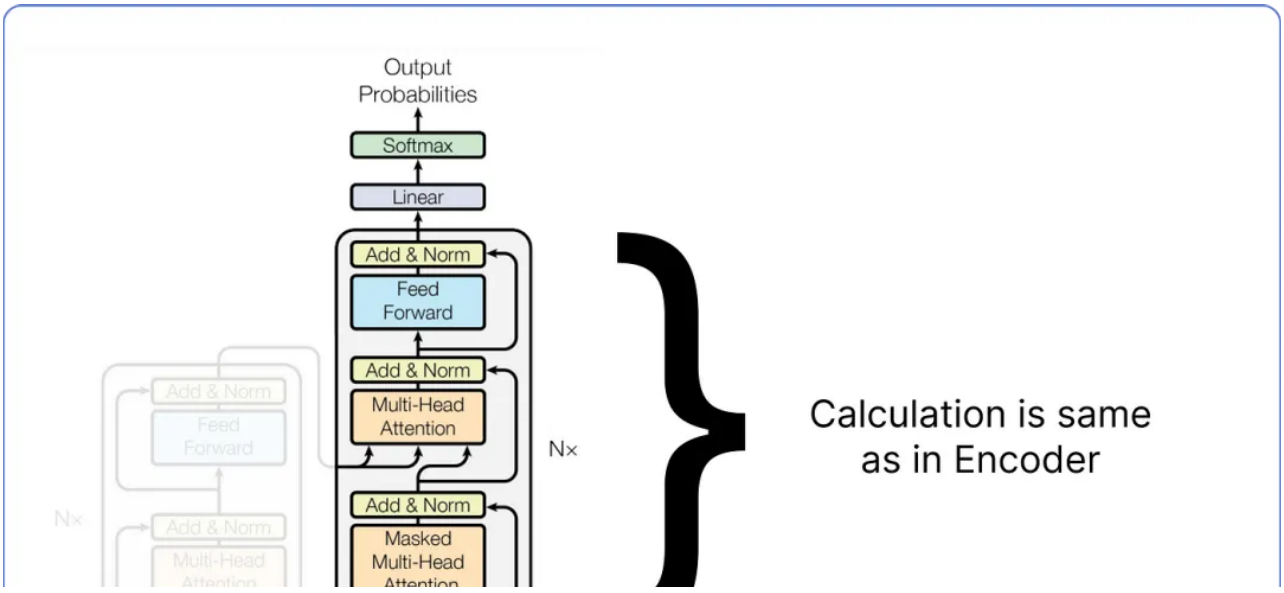
在哪里 `<start>` 和 `<end>` 是两个新标记被引入。此外，解码器每次只接受一个标记作为输入。这意味着 `<start>` 将作为输入，而 `你` 必须是它的预测文本。

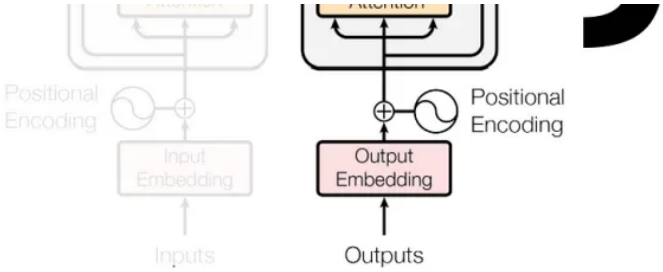


解码器输入 单词

正如我们已知，这些嵌入值充满了随机值，这些值将在训练过程中更新。

计算剩余的块，方法与我们之前在编码器部分计算的方法相同。





计算解码器

在深入任何更详细的内容之前，我们需要理解什么是掩码多头注意力，通过一个简单的数学例子来说明。

步骤 12 — 理解掩码多头注意力

在 Transformer 中，掩码多头注意力就像模型用来关注句子不同部分的聚光灯。它很特别，因为它不让模型通过查看句子后面的单词来作弊。这有助于模型逐步理解和生成句子，这对于像说话或把单词翻译成另一种语言这样的任务很重要。

假设我们有一个以下输入矩阵，其中每一行代表序列中的一个位置，每一列代表一个特征：

$$\text{Input Matrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

输入矩阵，用于掩码多头注意力

现在，让我们了解具有两个头的掩码多头注意力组件

1. 线性投影（查询、键、值）：假设每个头的线性投影：**线性投影（查询、键、值）：** 假设每个头的线性投影：**头 1：_Wq_1,_Wk_1,_Wv_1** 和**头 2：_Wq_2,_Wk_2,_Wv_2**
2. 计算注意力分数：对于每个头，使用查询和键的点积来计算注意力分数，并应用掩码以防止关注未来的位置。

3. 应用 Softmax：应用 softmax 函数以获得注意力权重。
4. < strong id=0 > 加权求和（值）： 将注意力权重乘以值以获得每个头的加权求和。
5. 将两个头的输出连接起来并应用线性变换。

让我们做一个简化的计算：

假设两个条件

- 强 _Wq_1 = _Wk_1 = _Wv_1 = _Wq_2 = _Wk_2 = _Wv_2 = _I_，单位矩阵。
- **Q=*K=V=输入矩阵***

Head 1:

$$Q_1 = K_1 = V_1 = \text{Input Matrix}$$
$$A_1 = Q_1 \cdot K_1^T$$
$$A_1 = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$
$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{(Masked)}$$
$$W_1 = \text{softmax}(A_1)$$
$$O_1 = W_1 \cdot V_1$$

Head 2:

$$Q_2 = K_2 = V_2 = \text{Input Matrix}$$
$$A_2 = Q_2 \cdot K_2^T$$
$$A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
$$A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix} \quad \text{(Masked)}$$
$$W_2 = \text{softmax}(A_2)$$
$$O_2 = W_2 \cdot V_2$$

Concatenate and Linear Transformation:

$$\text{Concatenate}([O_1, O_2])$$
(Apply Learnable Linear Transformation)

掩码多头注意力（**两个头**）

步骤将两个注意力头的输出合并成一个单一的信息集。

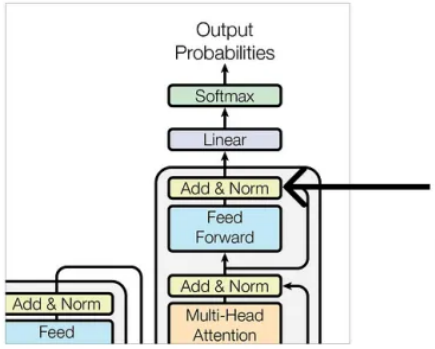
想象你有两个朋友，他们各自给你提供关于问题的建议。合并他们的建议意味着将这两条建议放在一起，以便你能够更全面地了解他们的建议。

在 Transformer 模型中，这一步骤有助于从多个角度捕捉输入数据的各个方面，有助于模型在进一步处理中使用更丰富的表示。

步骤 13 — 计算预测单词

输出矩阵必须包含与输入矩阵相同的行数，而列数可以是任何数量。在这里，我们处理 6

。

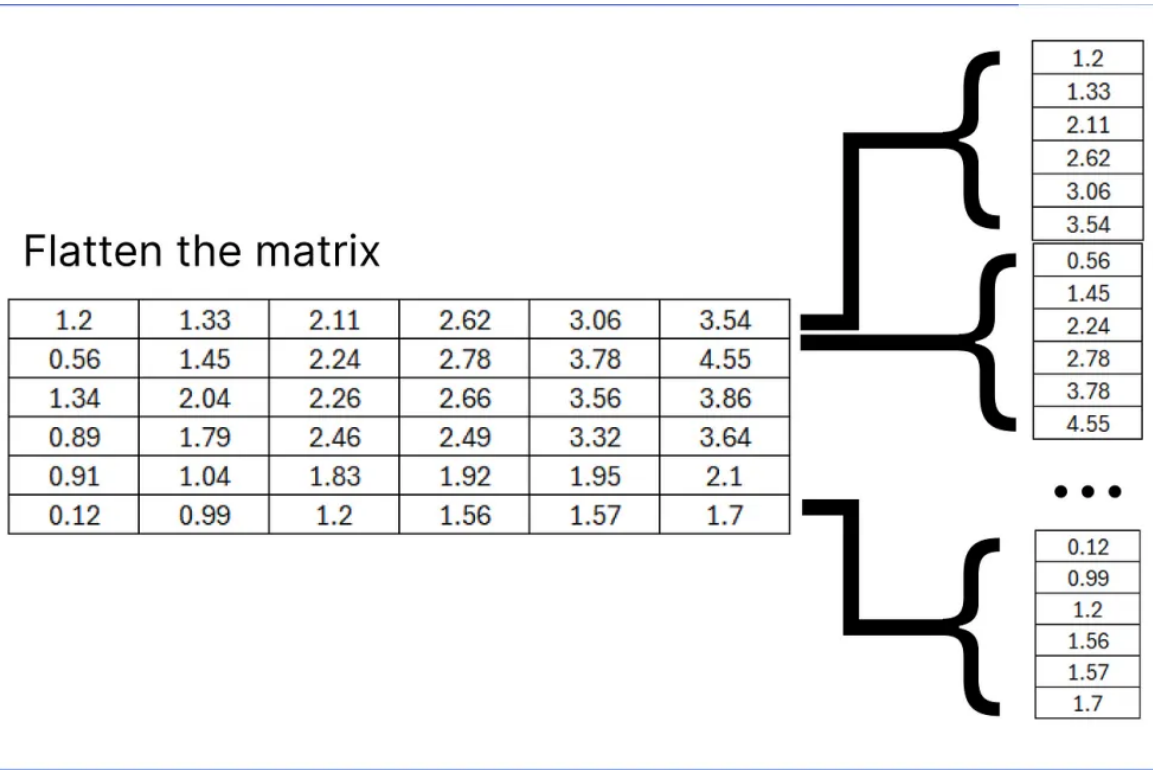


Rows must be 6 while columns can be any length

1.2	1.33	2.11	2.62	3.06	3.54
0.56	1.45	2.24	2.78	3.78	4.55
1.34	2.04	2.26	2.66	3.56	3.86
0.89	1.79	2.46	2.49	3.32	3.64
0.91	1.04	1.83	1.92	1.95	2.1
0.12	0.99	1.2	1.56	1.57	1.7

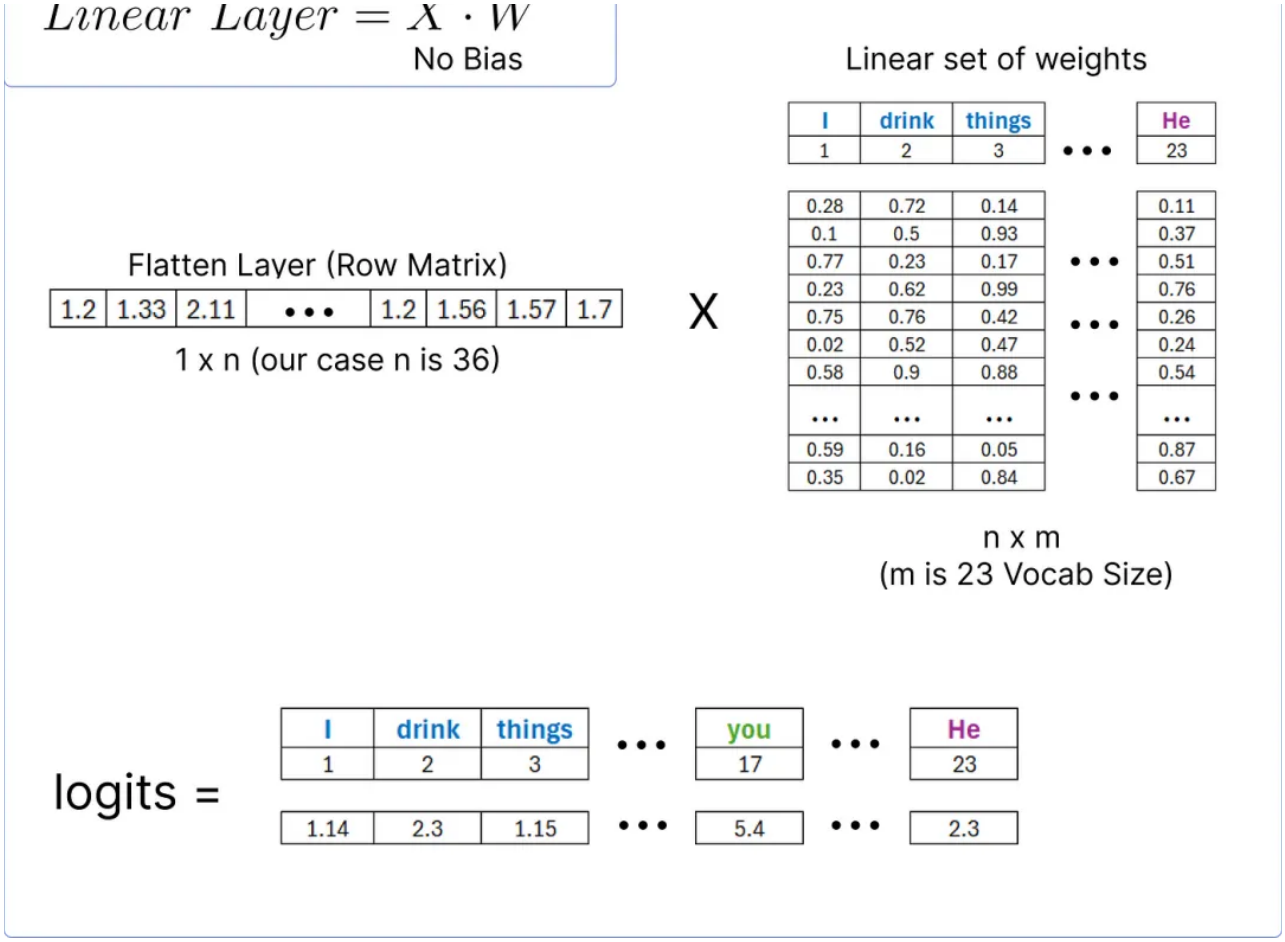
解码器输出添加和归一化

解码器的最后一个**添加和归一化块**的结果矩阵必须展平，以便与线性层匹配，以找到我们数据集（语料库）中每个独特单词的预测概率。



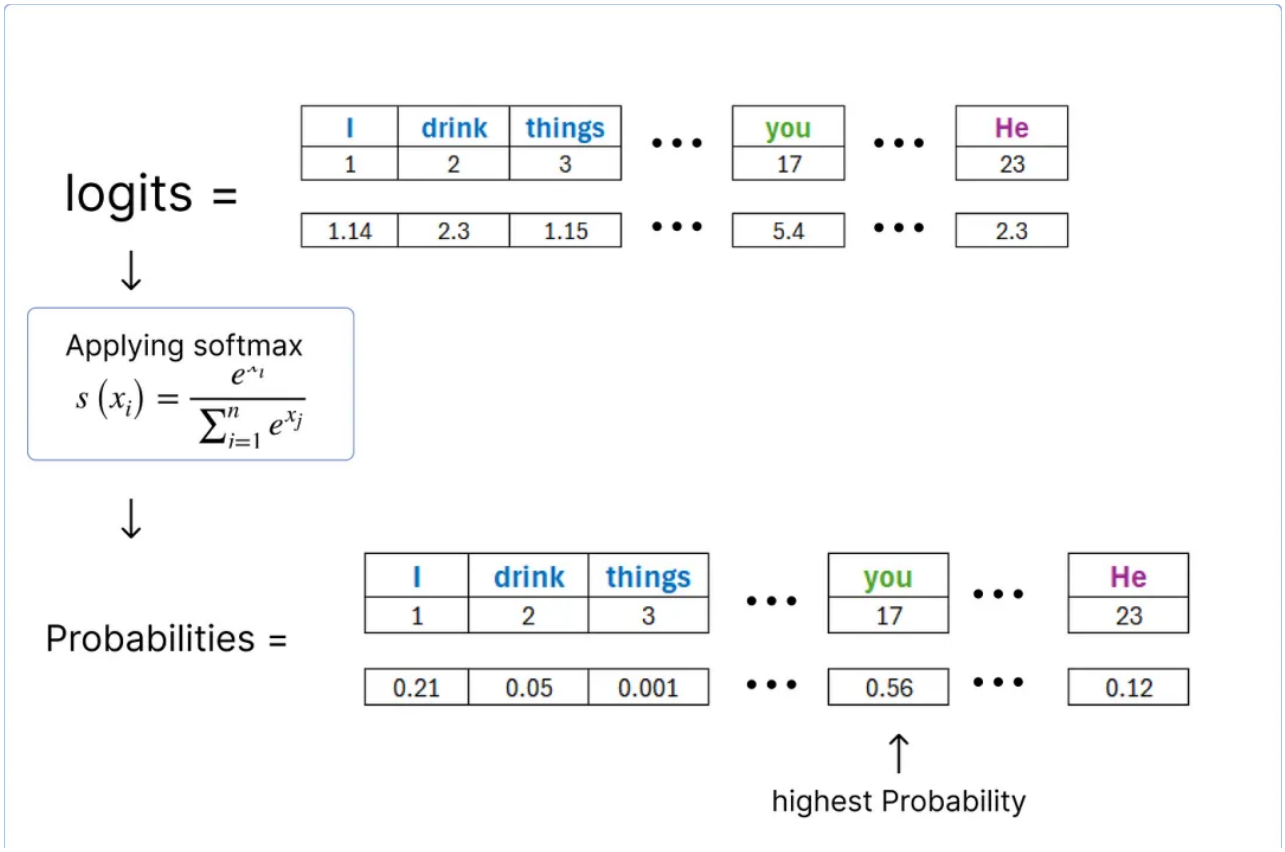
将最后一个加和范数块矩阵展平

这个展平层将通过一个线性层传递，以计算我们数据集中每个独特单词的**logits**（得分）。



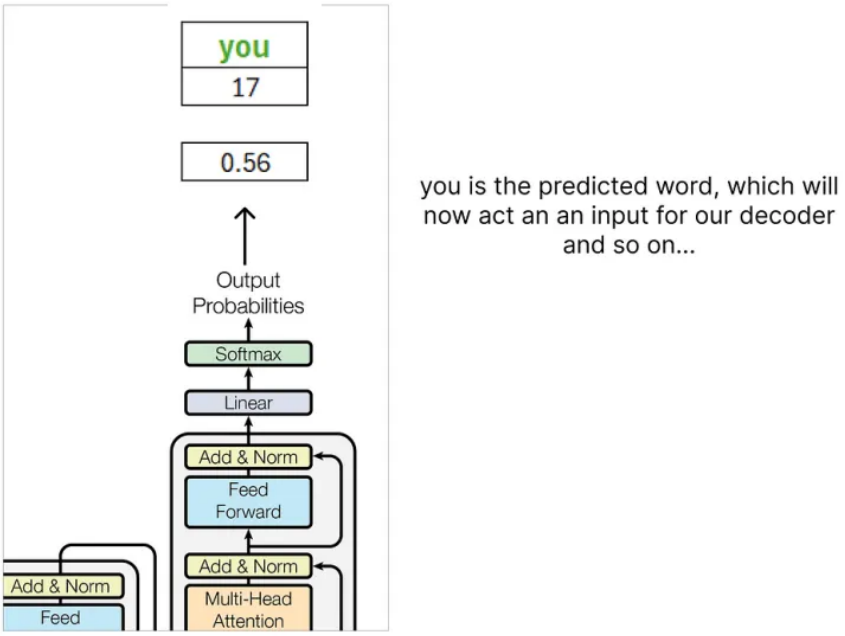
计算对数几率

一旦我们获得 logits，就可以使用**softmax** 函数对它们进行归一化，并找到包含最高概率的单词。



寻找预测的词

因此，根据我们的计算，解码器预测的词是 你 。



解码器的最终输出

这个预测的单词 你 将被视为解码器的输入单词，这个过程会一直持续到预测到 <end> 标记为止。

重要要点

1. 上述示例非常简单，因为它不涉及 epoch 或其他只能使用 Python 等编程语言可视化的重要参数。
2. 它只展示了训练过程，而使用这种方法无法直观地看到评估或测试。
3. 掩码多头注意力可以用来防止 transformer 查看未来，有助于避免模型过拟合。

结论

在这篇博客中，我向您展示了一种非常基础的通过矩阵方法来理解 transformers 数学工作原理的方式。我们应用了位置编码、softmax、前馈网络，最重要的是多头注意力。

未来，我将发布更多关于变压器和LLM的博客，因为我的核心关注点是自然语言处理。更

重要的是，如果你想从头开始使用 Python 构建自己的百万参数LLM，我已经写了一篇关于这个的博客，它在 Medium 上受到了很多赞赏。你可以在这里阅读：

度过一个愉快的阅读时光！