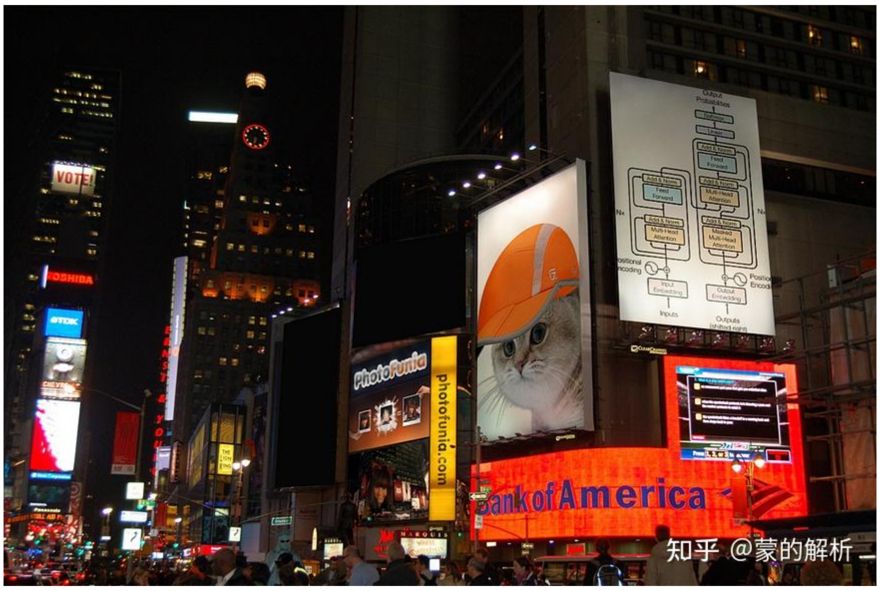
Solving Transformer by Hand: A Step-by-Step Math Example 手动求解 Transformer：分步数学示例 Part 2/2

蒙的解析
咨询行业 从业人员
关注他

来自专栏 · AI 大模型



Solving Transformer by Hand: A Step-by-Step Math Example
手动求解 Transformer：分步数学示例

Fareed Khan

I have already written a detailed blog on how transformers work using a very small sample of the dataset, which will be my best blog ever because it has elevated my profile and given me the motivation to write more. However, that blog is incomplete as it only covers **20% of the transformer architecture** and contains numerous calculation errors, as pointed out by readers. After a considerable amount of time has passed since that blog, I will be revisiting the topic in this new blog.
我已经写了一篇详细的博客，介绍变压器如何使用非常小的数据集样本工作，这将是我有史以来最好的博客，因为它提高了我的个人资料，并给了我写更多文章的动力。然而，正如读者指出的那样，该博客并不完整，因为它只涵盖了 Transformer 架构的 20%，并且包含大量计算错误。自该博客发布以来已经过去了相当长的时间，我将在这个新博客中重新审视该主题。

My previous blog on transformer architecture (**covers only 20%**):
我之前关于 Transformer 架构的博客（只涵盖了 20%）：

Understanding Transformers: A Step-by-Step Math Example — Part 1
了解 Transformer：分步数学示例 — 第 1 部分

I understand that the transformer architecture may seem scary, and you might have encountered various explanations on...
我知道变压器架构可能看起来很可怕，并且您可能遇到过关于......的各种解释。

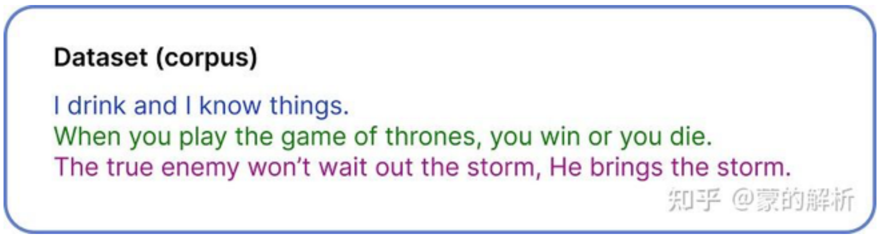
medium.com 媒体网站

I plan to explain the transformer again in the same manner as I did in my previous blog **(for both coders and non-coders)**, providing a complete guide with a step-by-step approach to understanding how they work.
我计划以与我在之前的博客中相同的方式再次解释变压器（针对编码员和非编码员），提供完整的指南，并逐步提供了解其工作原理的方法。

Table of Contents 目录

Step 1 — Defining our Dataset
第 1 步 - 定义我们的数据集

The dataset used for creating ChatGPT⁺ is **570 GB.** On the other hand, for our purposes, we will be using a very small dataset to perform numerical calculations visually.
用于创建 ChatGPT 的数据集为 570 GB。另一方面，出于我们的目的，我们将使用非常小的数据集来直观地执行数值计算。


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.

Our entire dataset containing only three sentences
我们的整个数据集仅包含三个句子

Our entire dataset contains only three sentences, all of which are dialogues taken from a TV show. Although our dataset is cleaned, in real-world scenarios like ChatGPT creation, cleaning a 570 GB dataset requires a significant amount of effort.
我们的整个数据集仅包含三个句子，所有这些句子都是取自电视节目的对话。尽管我们的数据集已清理，但在 ChatGPT 创建等现实场景中，清理 570 GB 的数据集需要大量工作。

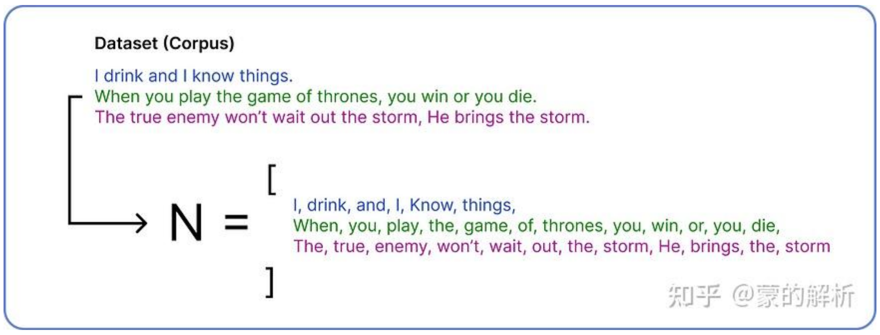
Step 2— Finding Vocab Size
第 2 步 — 查找词汇量

The vocabulary size determines the total number of **unique words** in our dataset. It can be calculated using the below formula, where **N** is the total number of words in our dataset.
词汇量大小决定了数据集中唯一单词的总数。可以使用以下公式计算，其中 N 是数据集中的单词总数。

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

vocab_size formula where N is total number of words
vocab_size 公式，其中 N 是单词总数

In order to find N, we need to break our dataset into individual words.
为了找到 N，我们需要将数据集分解为单独的单词。



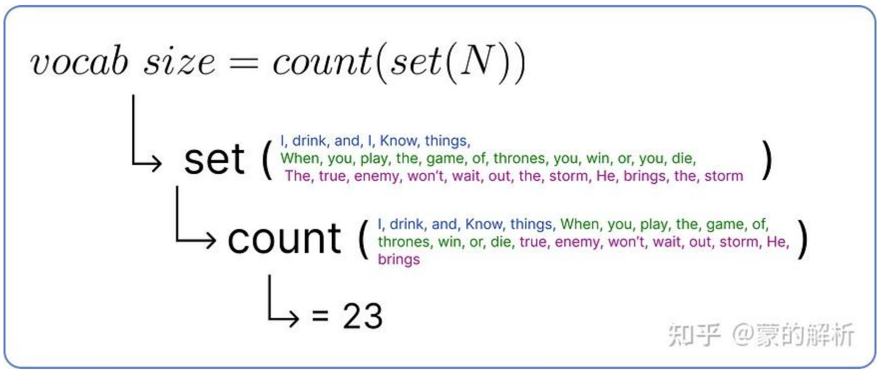
Dataset (Corpus)

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.

N = [
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
]

calculating variable **N** 计算变量N

After obtaining N, we perform a set operation to remove duplicates, and then we can count the unique words to determine the vocabulary size.
获得N后，我们执行集合操作来删除重复项，然后我们可以统计唯一单词来确定词汇量大小。



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

set (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)

count (I, drink, and, Know, things, When, you, play, the, game, of, thrones, win, or, die, true, enemy, won't, wait, out, storm, He, brings)

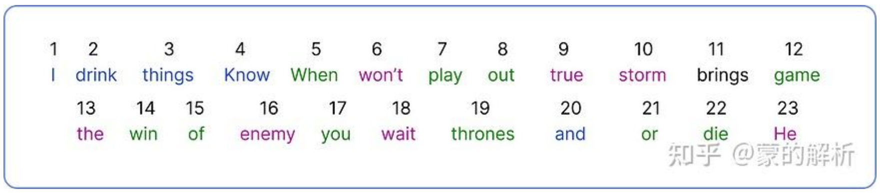
= 23

finding vocab size 寻找词汇量

Therefore, the vocabulary size is **23**, as there are **23** unique words in our dataset.
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。

Step 3 — Encoding 第 3 步 — 编码

Now, we need to assign a unique number to each unique word.
现在，我们需要为每个唯一的单词分配一个唯一的编号。



1	2	3	4	5	6	7	8	9	10	11	12
I	drink	things	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	

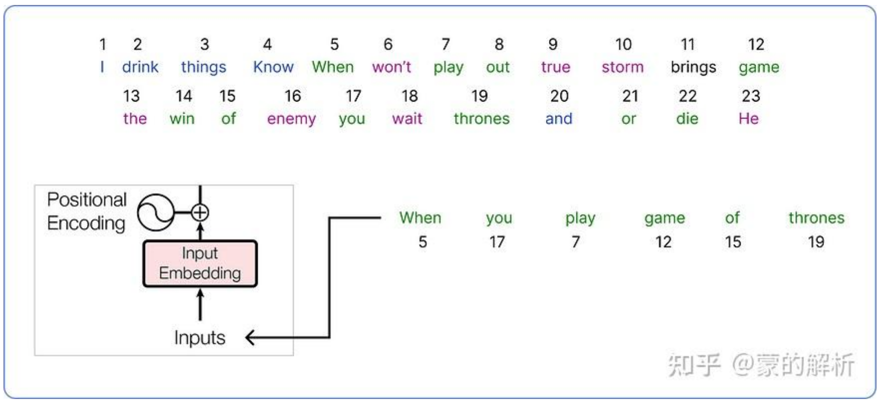
encoding our unique words
编码我们独特的词语

As we have considered a single token as a single word and assigned a number to it, ChatGPT has considered a portion of a word as a single token using this formula: 1 Token = 0.75 Word
由于我们将单个标记视为单个单词并为其分配了一个数字，ChatGPT 使用以下公式将单词的一部分视为单个标记： 1 Token = 0.75 Word

After encoding our entire dataset, it's time to select our input and start working with the transformer architecture.
对整个数据集进行编码后，是时候选择我们的输入并开始使用变压器架构了。

Step 4 — Calculating Embedding
第 4 步 — 计算嵌入

Let's select a sentence from our corpus that will be processed in our transformer architecture.
让我们从语料库中选择一个句子，它将在变压器架构中进行处理。

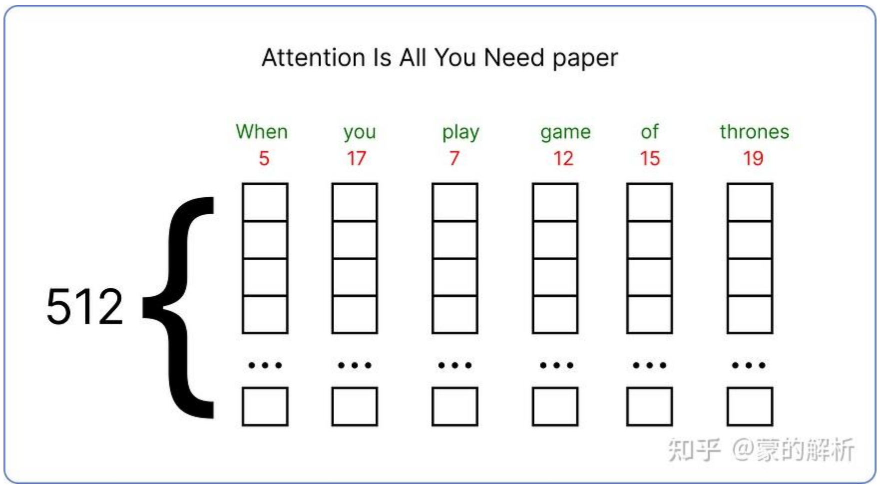


1	2	3	4	5	6	7	8	9	10	11	12
I	drink	things	Know	When	won't	play	out	true	storm	brings	game
13	14	15	16	17	18	19	20	21	22	23	
the	win	of	enemy	you	wait	thrones	and	or	die	He	

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

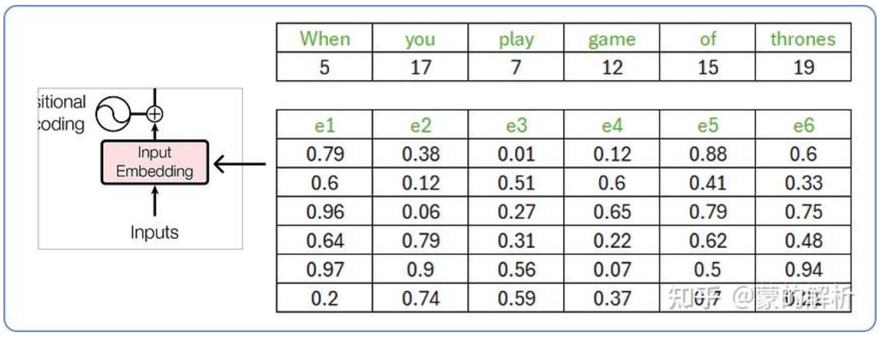
Input sentence for transformer
变压器输入语句

We have selected our input, and we need to find an embedding vector for it. The original paper uses a **512-dimensional embedding vector** for each input word.
我们已经选择了输入，我们需要为其找到一个嵌入向量。原始论文对每个输入单词使用 512 维的嵌入向量。



Original Paper uses 512 dimension vector
原论文使用512维向量

Since, for our case, we need to work with a smaller dimension of embedding vector to visualize how the calculation is taking place. So, we will be using a dimension of 6 for the embedding vector.
因为对于我们的例子，我们需要使用较小维度的嵌入向量来可视化计算是如何发生的。因此，我们将使用 6 维度作为嵌入向量。


sitional
coding
Input
Embedding
Inputs

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

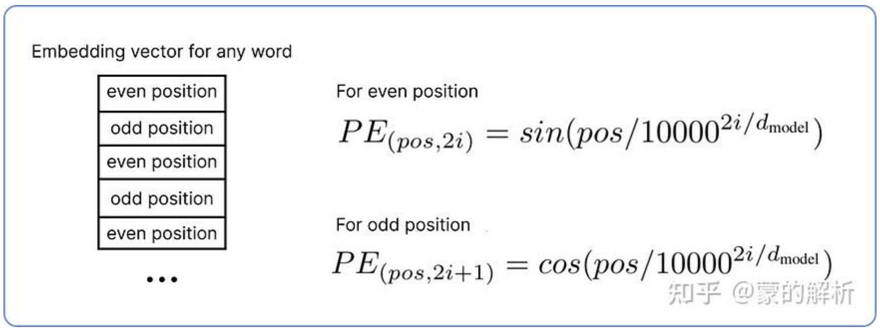
e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75
0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.42

Embedding vectors of our input
嵌入我们输入的向量

These values of the embedding vector are between 0 and 1 and are filled randomly in the beginning. They will later be updated as our transformer starts understanding the meanings among the words.
嵌入向量的这些值介于 0 和 1 之间，并且在开始时随机填充。当我们的变压器开始理解单词之间的含义时，它们稍后将被更新。

Step 5 — Calculating Positional Embedding
第 5 步 — 计算位置嵌入

Now we need to find positional embeddings for our input. There are two formulas for positional embedding depending on the position of the ith value of that embedding vector for each word.
现在我们需要为我们的输入找到位置嵌入。位置嵌入有两个公式，具体取决于每个单词的嵌入向量第 i 个值的位置。


Embedding vector for any word

even position
odd position
even position
odd position
even position
...

For even position

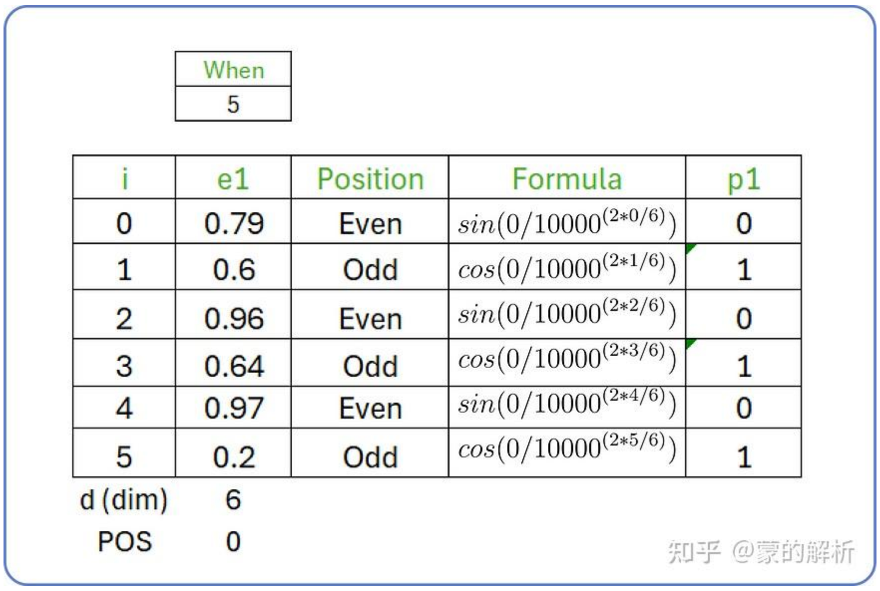
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

For odd position

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

Positional Embedding formula
位置嵌入公式

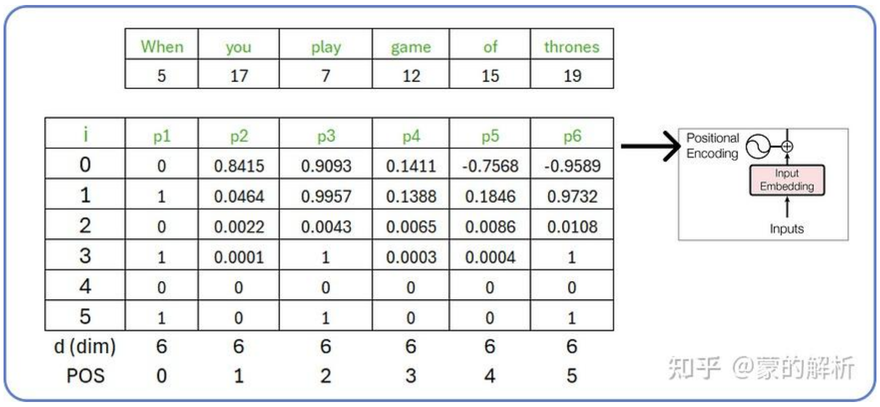
As you do know, our input sentence is **“when you play the game of thrones”** and the starting word is **“when”** with a starting index (POS) value is 0, having a dimension (d) of 6. For i from 0 to 5, we calculate the positional embedding for our first word of the input sentence.
如您所知，我们的输入句子是“when you play the game of Throws”，起始词是“when”，起始索引（POS）值为 0，维度（d）为 6 。对于 0 to 5 中的 i，我们计算输入句子的第一个单词的位置嵌入。


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			

Positional Embedding for word: **When**
单词的位置嵌入：何时

Similarly, we can calculate positional embedding for all the words in our input sentence.
类似地，我们可以计算输入句子中所有单词的位置嵌入。


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

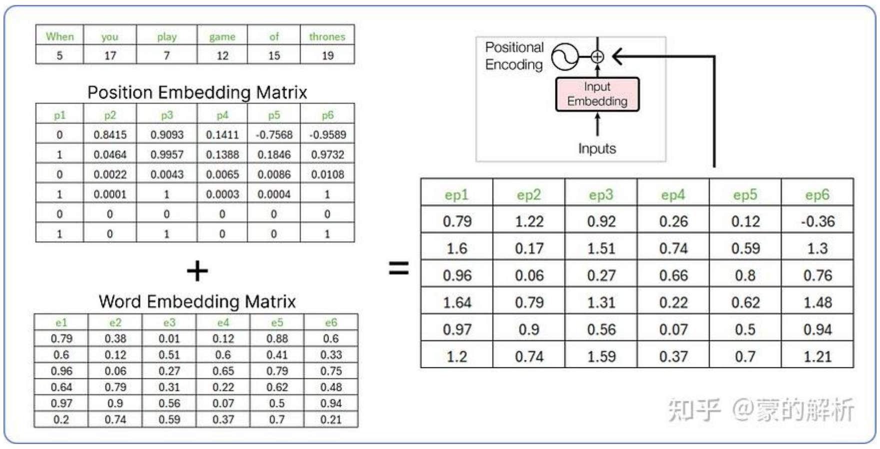
i	p1	p2	p3	p4	p5	p6
0	0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	1	0.0464	0.9957	0.1388	0.1846	0.9732
2	0	0.0022	0.0043	0.0065	0.0086	0.0108
3	1	0.0001	1	0.0003	0.0004	1
4	0	0	0	0	0	0
5	1	0	1	0	0	1
d (dim)	6	6	6	6	6	6
POS	0	1	2	3	4	5

Positional Encoding
Input Embedding
Inputs

Calculating Positional Embeddings of our input **(The calculated values are rounded)**
计算输入的位置嵌入（计算值四舍五入）

Step 6 — Concatenating Positional and Word Embeddings
第 6 步 — 连接位置嵌入和词嵌入

After calculating positional embedding, we need to add word embeddings and positional embeddings.
计算位置嵌入后，我们需要添加词嵌入和位置嵌入。



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
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

=

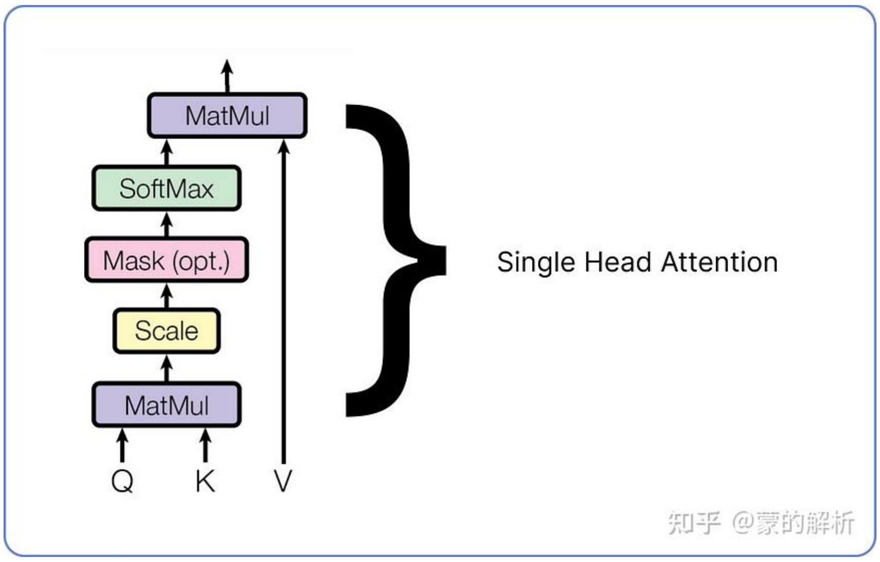
ep1	ep2	ep3	ep4	ep5	ep6
0.79	1.22	0.92	0.26	0.12	-0.36
1.6	0.17	1.51	0.74	0.59	1.3
0.96	0.06	0.27	0.66	0.8	0.76
1.64	0.79	1.31	0.22	0.62	1.48
0.97	0.9	0.56	0.07	0.5	0.94
1.2	0.74	1.59	0.37	0.7	1.21

concatenation step 连接步骤

This resultant matrix from combining both matrices (**Word embedding matrix** and **positional embedding matrix**) will be considered as an input to the encoder part.
组合两个矩阵（词嵌入矩阵和位置嵌入矩阵）所得的矩阵将被视为编码器部分的输入。

Step 7 — Multi Head Attention
第 7 步 — 多头注意力

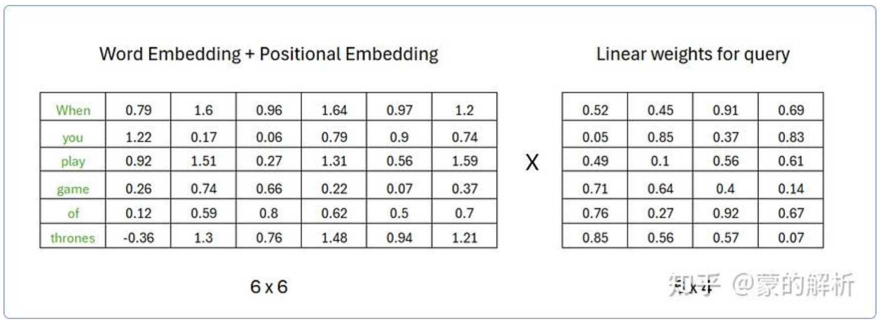
A multi-head attention is comprised of many single-head attentions. It is up to us how many single heads we need to combine. For example, LLaMA LLM⁺ from Meta has used 32 single heads in the encoder architecture. Below is the illustrated diagram of how a single-head attention looks like.
多头注意力由许多单头注意力组成。需要组合多少个单头取决于我们。例如，Meta 的 LLaMA LLM 在编码器架构中使用了 32 个单头。下面是单头注意力的示意图。



Single Head attention in Transformer
Transformer 中的单头注意力

There are three inputs: **query**, **key**, and **value**. Each of these matrices is obtained by multiplying a different set of weights matrix from the **Transpose** of same matrix that we computed earlier by adding the word embedding and positional embedding matrix.
共有三个输入：查询、键和值。这些矩阵中的每一个都是通过将一组不同的权重矩阵与我们之前通过添加单词嵌入和位置嵌入矩阵计算的相同矩阵的转置相乘而获得的。

Let’s say, for computing the query matrix, the set of weights matrix must have the number of rows the same as the number of columns of the transpose matrix, while the columns of the weights matrix can be any; for example, we suppose 4 columns in our weights matrix. The values in the weights matrix are between 0 and 1 randomly, which will later be updated when our transformer starts learning the meaning of these words.
比方说，为了计算查询矩阵，权重矩阵集合的行数必须与转置矩阵的列数相同，而权重矩阵的列可以是任意的；例如，我们假设权重矩阵中有 4 列。权重矩阵中的值随机位于 0 and 1 之间，稍后当我们的转换器开始学习这些单词的含义时，这些值将被更新。



Word Embedding + Positional Embedding

When	0.79	1.6	0.96	1.64	0.97	1.2
you	1.22	0.17	0.06	0.79	0.9	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.8	0.62	0.5	0.7
thrones	-0.36	1.3	0.76	1.48	0.94	1.21

6 x 6

X

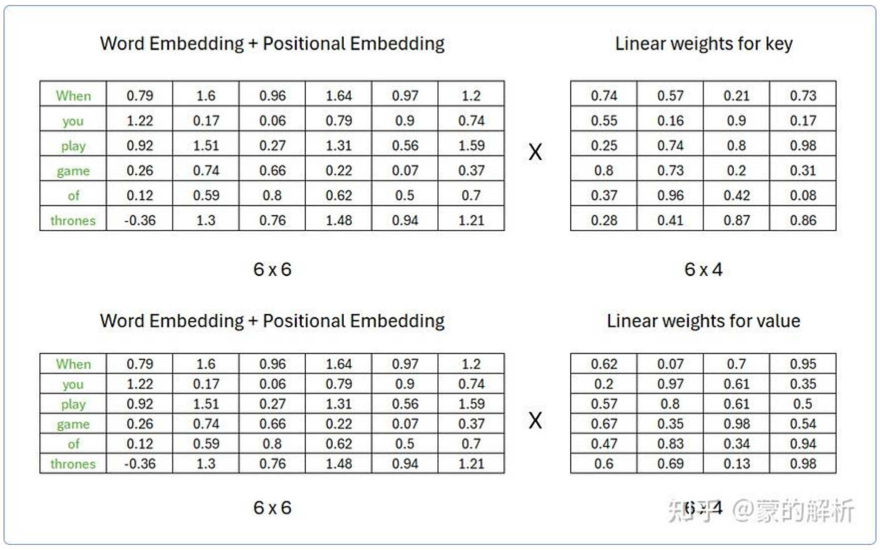
Linear weights for query

0.52	0.45	0.91	0.69
0.05	0.85	0.37	0.83
0.49	0.1	0.56	0.61
0.71	0.64	0.4	0.14
0.76	0.27	0.92	0.67
0.85	0.56	0.57	0.07

6 x 4

calculating Query matrix 计算查询矩阵

Similarly, we can compute the **key** and **value** matrices using the same procedure, but the values in the weights matrix must be different for both.
类似地，我们可以使用相同的过程计算键和值矩阵，但两者的权重矩阵中的值必须不同。



Word Embedding + Positional Embedding

When	0.79	1.6	0.96	1.64	0.97	1.2
you	1.22	0.17	0.06	0.79	0.9	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.8	0.62	0.5	0.7
thrones	-0.36	1.3	0.76	1.48	0.94	1.21

6 x 6

X

Linear weights for key

0.74	0.57	0.21	0.73
0.55	0.16	0.9	0.17
0.25	0.74	0.8	0.98
0.8	0.73	0.2	0.31
0.37	0.96	0.42	0.08
0.28	0.41	0.87	0.86

6 x 4

Word Embedding + Positional Embedding

When	0.79	1.6	0.96	1.64	0.97	1.2
you	1.22	0.17	0.06	0.79	0.9	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.8	0.62	0.5	0.7
thrones	-0.36	1.3	0.76	1.48	0.94	1.21

6 x 6

X

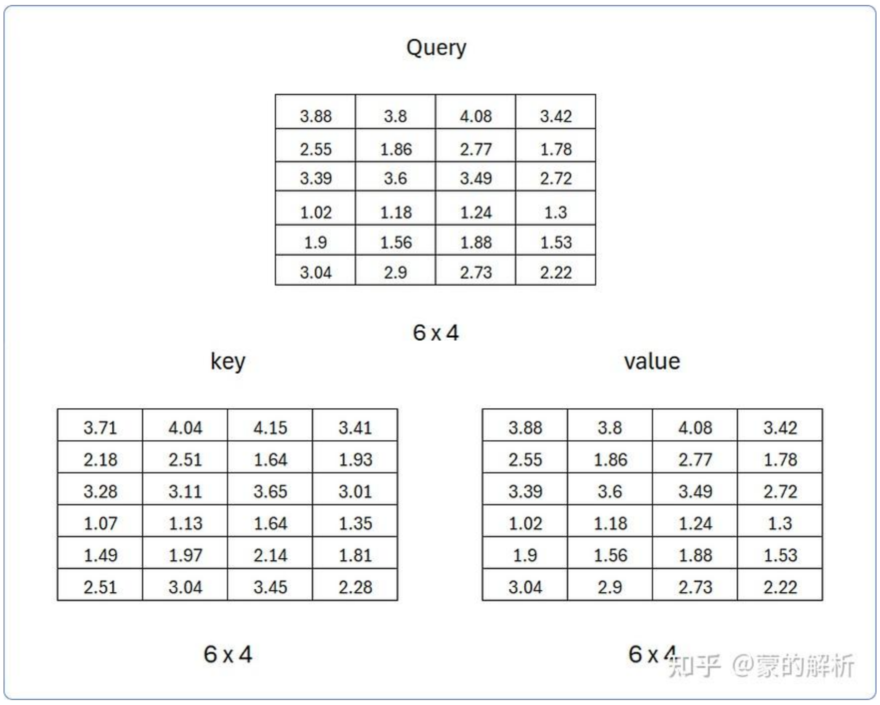
Linear weights for value

0.62	0.07	0.7	0.95
0.2	0.97	0.61	0.35
0.57	0.8	0.61	0.5
0.67	0.35	0.98	0.54
0.47	0.83	0.34	0.94
0.6	0.69	0.13	0.98

6 x 4

Calculating Key and Value Matrices
计算键和值矩阵

So, after multiplying matrices, the resultant **query**, **key**, and **values** are obtained:
因此，在矩阵相乘之后，得到结果查询、键和值：



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

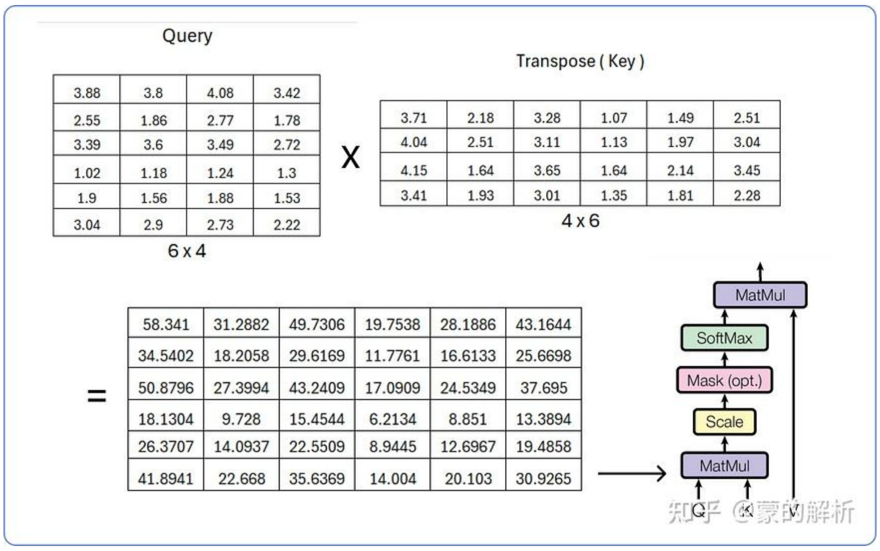
value

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

Query, Key, Value matrices
查询、键、值矩阵

Now that we have all three matrices, let’s start calculating single-head attention step by step.
现在我们已经有了所有三个矩阵，让我们开始逐步计算单头注意力。



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

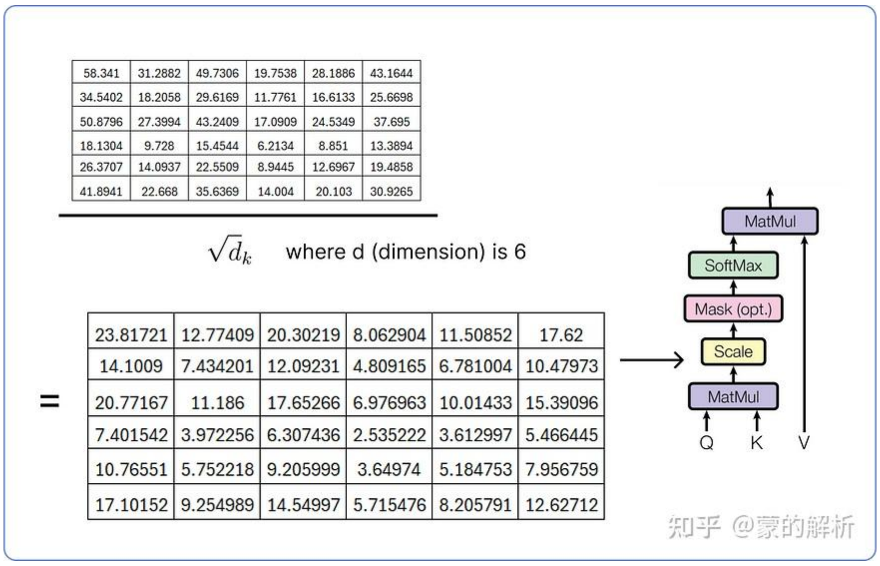
4 x 6

=

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

matrix multiplication between Query and Key
查询和密钥之间的矩阵乘法

For scaling the resultant matrix, we have to reuse the dimension of our embedding vector, which is 6.
为了缩放结果矩阵，我们必须重用嵌入向量的维度，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

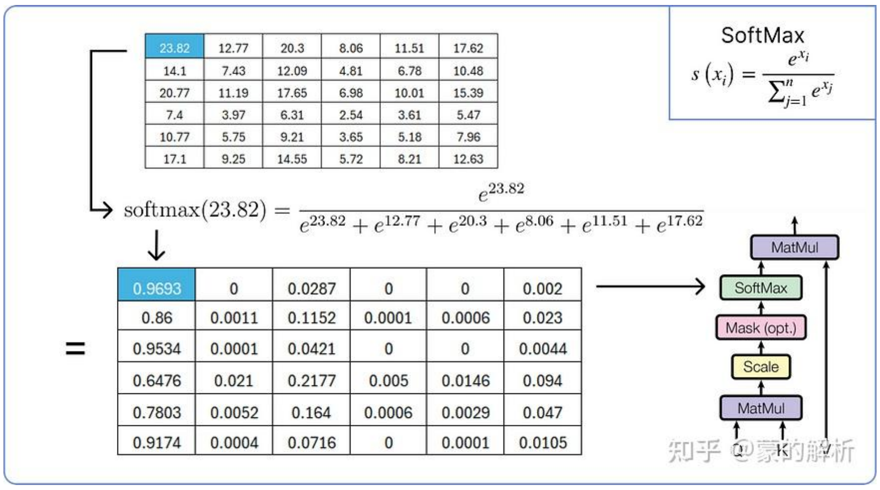
=

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

scaling the resultant matrix with dimension **5**
将结果矩阵缩放为维度 5

The next step of **masking is optional**, and we won't be calculating it. Masking is like telling the model to focus only on what's happened before a certain point and not peek into the future while figuring out the importance of different words in a sentence. It helps the model understand things in a step-by-step manner, without cheating by looking ahead.
下一步的屏蔽是可选的，我们不会计算它。屏蔽就像告诉模型在确定句子中不同单词的重要性时只关注某一点之前发生的事情，而不是展望未来。它帮助模型逐步理解事物，而不会因向前看而作弊。

So now we will be applying the **softmax** operation on our scaled resultant matrix.
所以现在我们将对缩放后的结果矩阵应用 softmax 运算。



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

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

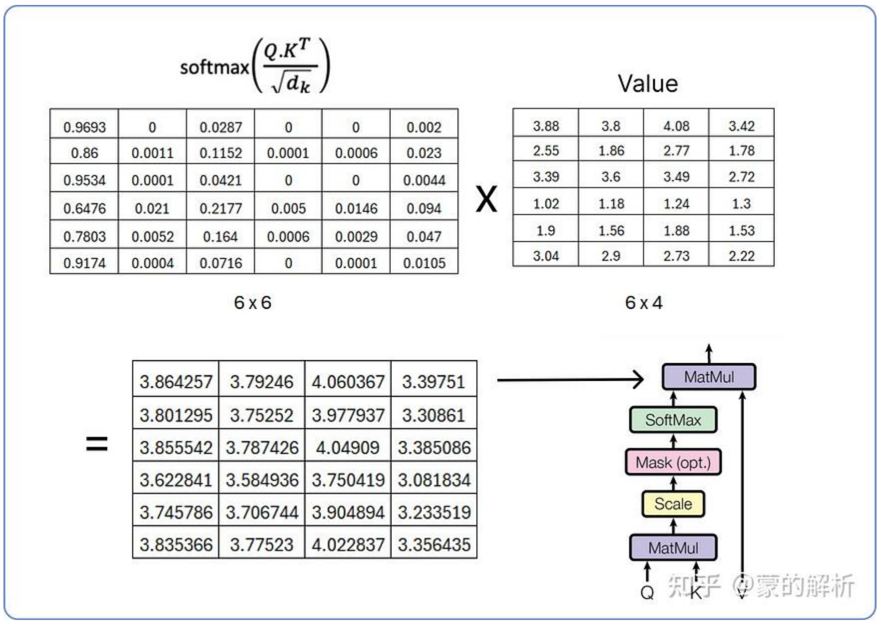
$$\text{softmax}(23.82) = \frac{e^{23.82}}{e^{23.82} + e^{12.77} + e^{20.3} + e^{8.06} + e^{11.51} + e^{17.62}}$$

=

0.9693	0	0.0287	0	0	0.002
0.86	0.0011	0.1152	0.0001	0.0006	0.023
0.9534	0.0001	0.0421	0	0	0.0044
0.6476	0.021	0.2177	0.005	0.0146	0.094
0.7803	0.0052	0.164	0.0006	0.0029	0.047
0.9174	0.0004	0.0716	0	0.0001	0.0105

Applying softmax on resultant matrix
对结果矩阵应用 softmax

Doing the final multiplication step to obtain the resultant matrix from single-head attention.
进行最后的乘法步骤以获得单头注意力的结果矩阵。



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

0.9693	0	0.0287	0	0	0.002
0.86	0.0011	0.1152	0.0001	0.0006	0.023
0.9534	0.0001	0.0421	0	0	0.0044
0.6476	0.021	0.2177	0.005	0.0146	0.094
0.7803	0.0052	0.164	0.0006	0.0029	0.047
0.9174	0.0004	0.0716	0	0.0001	0.0105

6 x 6

X

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

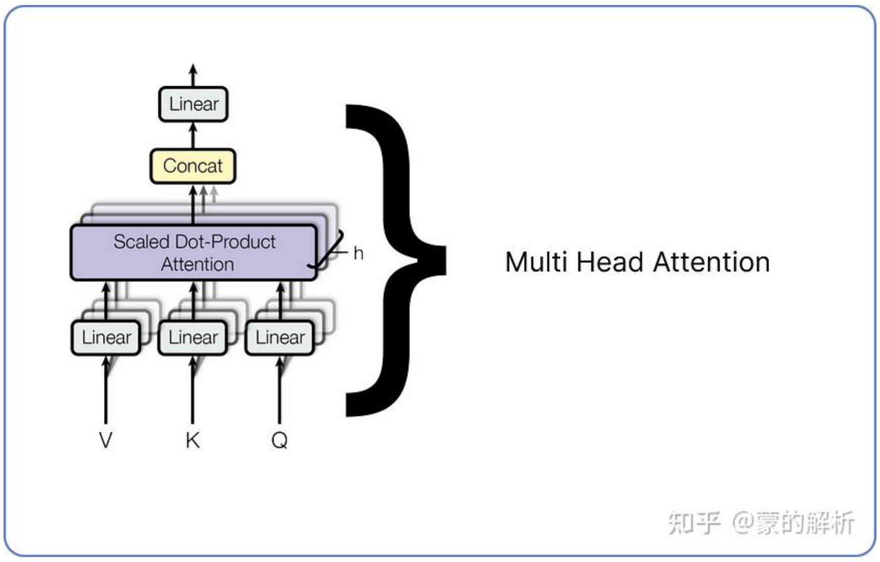
6 x 4

=

3.864257	3.79246	4.060367	3.39751
3.801295	3.75252	3.977937	3.30861
3.855542	3.787426	4.04909	3.385086
3.622841	3.584936	3.750419	3.081834
3.745786	3.706744	3.904894	3.233519
3.835366	3.77523	4.022837	3.356435

calculating the final matrix of single head attention
计算最终的单头注意力矩阵

We have calculated single-head attention, while multi-head attention comprises many single-head attentions, as I stated earlier. Below is a visual of how it looks like:
正如我之前所说，我们计算了单头注意力，而多头注意力由许多单头注意力组成。下面是它的外观：

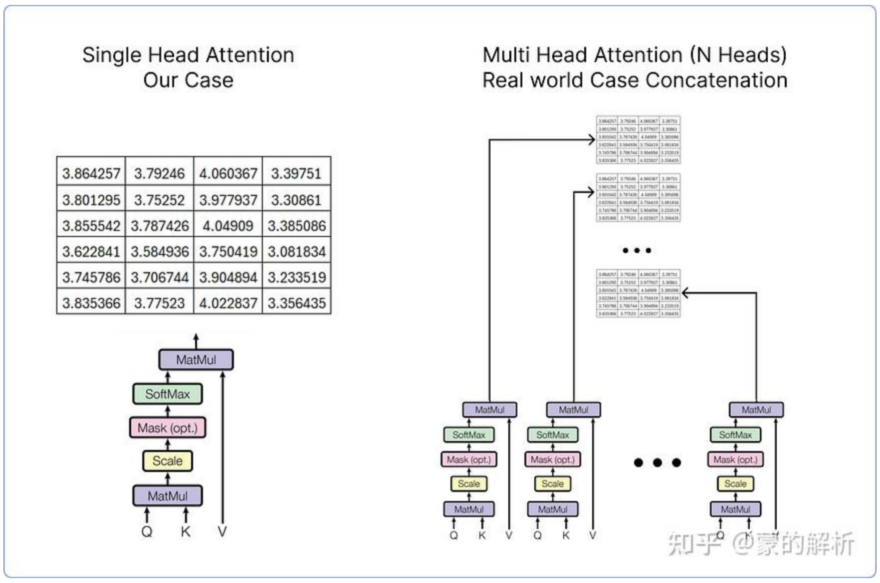


Multi Head attention in Transformer
Transformer 中的多头注意力

Each single-head attention has three inputs: **query**, **key**, and **value**, and each three have a different set of weights. Once all single-head attentions output their resultant matrices, they will all be concatenated, and the final concatenated matrix is once again transformed linearly by multiplying it with a set of weights matrix initialized with random values, which will later get updated when the transformer starts training.
每个单头注意力有三个输入：查询、键和值，并且每三个输入都有一组不同的权重。一旦所有单头注意力输出它们的结果矩阵，它们都会被连接起来，并且最终的连接矩阵通过与一组用随机值初始化的权重矩阵相乘再次被线性变换，权重矩阵稍后将在变压器启动时更新训练。

Since, in our case, we are considering a single-head attention, but this is how it looks if we are working with multi-head attention.

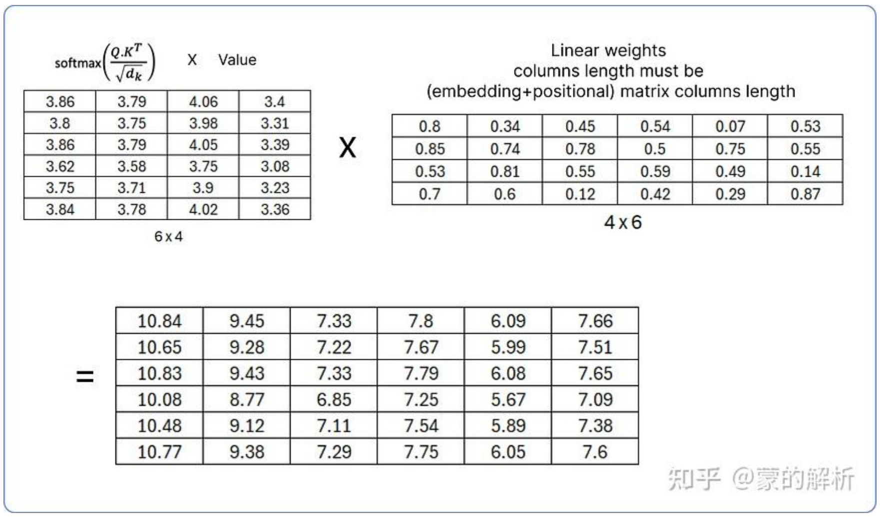
因为在我们的例子中，我们正在考虑单头注意力，但这就是我们使用多头注意力的情况。



3.864257	3.79246	4.060367	3.39751
3.801295	3.75252	3.977937	3.30861
3.855542	3.787426	4.04909	3.385086
3.622841	3.584936	3.750419	3.081834
3.745786	3.706744	3.904894	3.233519
3.835366	3.77523	4.022837	3.356435

Single Head attention vs Multi Head attention
单头注意力 vs 多头注意力

In either case, whether it’s single-head or multi-head attention, the resultant matrix needs to be once again transformed linearly by multiplying a set of weights matrix.
无论哪种情况，无论是单头注意力还是多头注意力，都需要通过乘以一组权重矩阵来再次对所得矩阵进行线性变换。


$\text{softmax}\left(\frac{Q.K^T}{\sqrt{d_k}}\right)$ X Value
Linear weights
columns length must be
(embedding+positional) matrix columns length
6 x 4
X
4 x 6
=
知乎 @蒙的解析

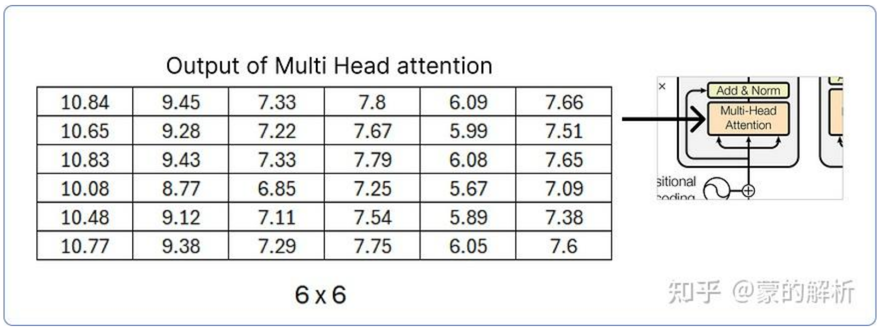
3.86	3.79	4.06	3.4
3.8	3.75	3.98	3.31
3.86	3.79	4.05	3.39
3.62	3.58	3.75	3.08
3.75	3.71	3.9	3.23
3.84	3.78	4.02	3.36

0.8	0.34	0.45	0.54	0.07	0.53
0.85	0.74	0.78	0.5	0.75	0.55
0.53	0.81	0.55	0.59	0.49	0.14
0.7	0.6	0.12	0.42	0.29	0.87

10.84	9.45	7.33	7.8	6.09	7.66
10.65	9.28	7.22	7.67	5.99	7.51
10.83	9.43	7.33	7.79	6.08	7.65
10.08	8.77	6.85	7.25	5.67	7.09
10.48	9.12	7.11	7.54	5.89	7.38
10.77	9.38	7.29	7.75	6.05	7.6

normalizing single head attention matrix
标准化单头注意力矩阵

Make sure the linear set of weights matrix number of columns must be equal to the matrix that we computed earlier (**word embedding + positional embedding**) matrix number of columns, because the next step, we will be adding the resultant normalized matrix with (**word embedding + positional embedding**) matrix.
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）矩阵的列数，因为下一步，我们将使用（词嵌入+位置嵌入）矩阵。


Add & Norm
Multi-Head Attention
sitional
coding

Output of Multi Head attention

10.84	9.45	7.33	7.8	6.09	7.66
10.65	9.28	7.22	7.67	5.99	7.51
10.83	9.43	7.33	7.79	6.08	7.65
10.08	8.77	6.85	7.25	5.67	7.09
10.48	9.12	7.11	7.54	5.89	7.38
10.77	9.38	7.29	7.75	6.05	7.6

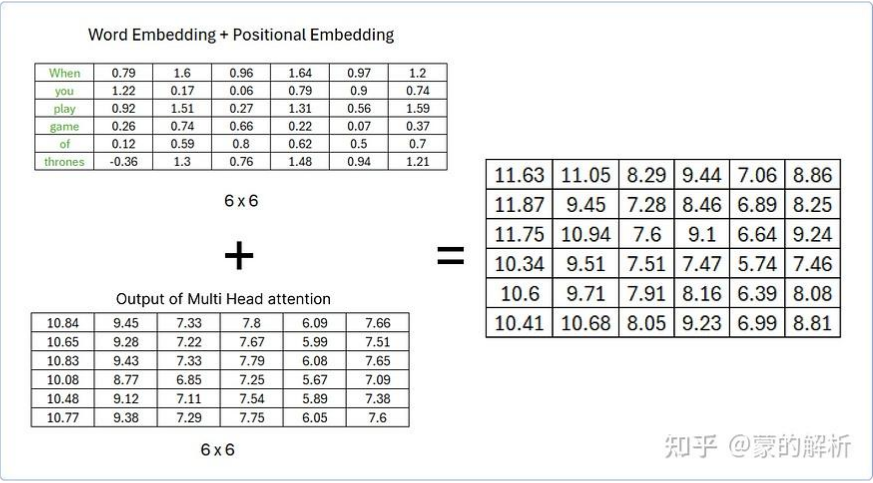
6 x 6

Output matrix of multi head attention
多头注意力的输出矩阵

As we have computed the resultant matrix for multi-head attention, next, we will be working on adding and normalizing step.
由于我们已经计算了多头注意力的结果矩阵，接下来我们将致力于添加和标准化步骤。

Step 8 — Adding and Normalizing
第 8 步 — 添加和规范化

Once we obtain the resultant matrix from multi-head attention, we have to add it to our original matrix. Let's do it first.
一旦我们从多头注意力中获得了结果矩阵，我们就必须将其添加到原始矩阵中。我们先来做吧。



Word Embedding + Positional Embedding

When	0.79	1.6	0.96	1.64	0.97	1.2
you	1.22	0.17	0.06	0.79	0.9	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.8	0.62	0.5	0.7
thrones	-0.36	1.3	0.76	1.48	0.94	1.21

6 x 6

+

Output of Multi Head attention

10.84	9.45	7.33	7.8	6.09	7.66
10.65	9.28	7.22	7.67	5.99	7.51
10.83	9.43	7.33	7.79	6.08	7.65
10.08	8.77	6.85	7.25	5.67	7.09
10.48	9.12	7.11	7.54	5.89	7.38
10.77	9.38	7.29	7.75	6.05	7.6

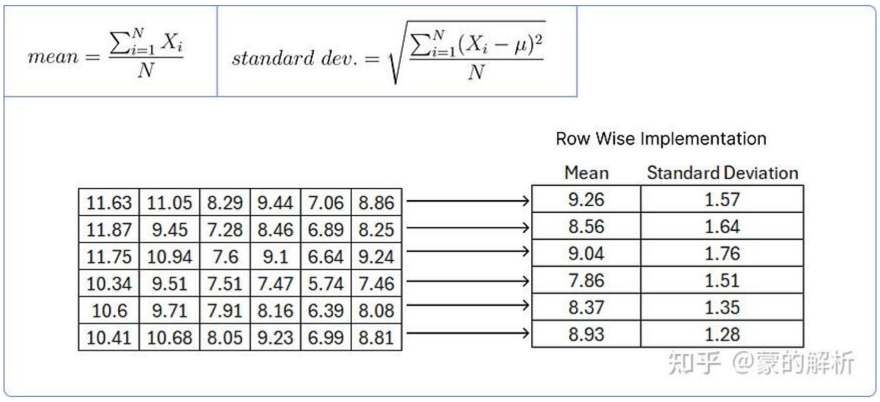
6 x 6

=

11.63	11.05	8.29	9.44	7.06	8.86
11.87	9.45	7.28	8.46	6.89	8.25
11.75	10.94	7.6	9.1	6.64	9.24
10.34	9.51	7.51	7.47	5.74	7.46
10.6	9.71	7.91	8.16	6.39	8.08
10.41	10.68	8.05	9.23	6.99	8.81

Adding matrices to perform add and norm step
添加矩阵以执行加法和标准化步骤

To normalize the above matrix, we need to compute the mean and standard deviation row-wise for each row.
为了标准化上述矩阵，我们需要逐行计算每行的平均值和标准差。



$$mean = \frac{\sum_{i=1}^{N} X_i}{N} \qquad standard\ dev. = \sqrt{\frac{\sum_{i=1}^{N}(X_i - \mu)^2}{N}}$$

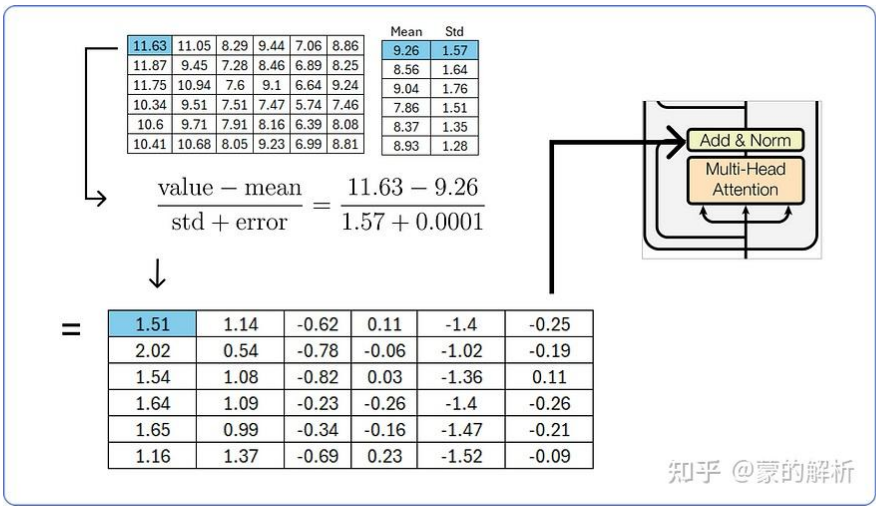
11.63	11.05	8.29	9.44	7.06	8.86
11.87	9.45	7.28	8.46	6.89	8.25
11.75	10.94	7.6	9.1	6.64	9.24
10.34	9.51	7.51	7.47	5.74	7.46
10.6	9.71	7.91	8.16	6.39	8.08
10.41	10.68	8.05	9.23	6.99	8.81

Row Wise Implementation

Mean	Standard Deviation
9.26	1.57
8.56	1.64
9.04	1.76
7.86	1.51
8.37	1.35
8.93	1.28

calculating meand and std.
计算平均值和标准差。

we subtract each value of the matrix by the corresponding row mean and divide it by the corresponding standard deviation.
我们将矩阵的每个值减去相应的行平均值，然后除以相应的标准差。



11.63	11.05	8.29	9.44	7.06	8.86
11.87	9.45	7.28	8.46	6.89	8.25
11.75	10.94	7.6	9.1	6.64	9.24
10.34	9.51	7.51	7.47	5.74	7.46
10.6	9.71	7.91	8.16	6.39	8.08
10.41	10.68	8.05	9.23	6.99	8.81

Mean	Std
9.26	1.57
8.56	1.64
9.04	1.76
7.86	1.51
8.37	1.35
8.93	1.28

$$\frac{\text{value} - \text{mean}}{\text{std} + \text{error}} = \frac{11.63 - 9.26}{1.57 + 0.0001}$$

=

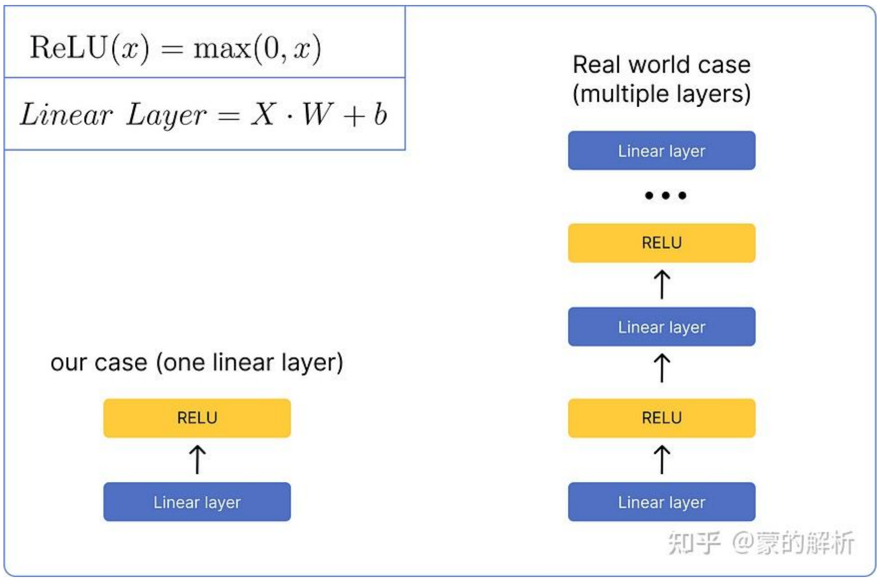
1.51	1.14	-0.62	0.11	-1.4	-0.25
2.02	0.54	-0.78	-0.06	-1.02	-0.19
1.54	1.08	-0.82	0.03	-1.36	0.11
1.64	1.09	-0.23	-0.26	-1.4	-0.26
1.65	0.99	-0.34	-0.16	-1.47	-0.21
1.16	1.37	-0.69	0.23	-1.52	-0.09

normalizing the resultant matrix
标准化结果矩阵

Adding a small value of error prevents the denominator from being zero and avoids making the entire term infinity.
添加一个小的误差值可以防止分母为零并避免使整个项无穷大。

Step 9 — Feed Forward Network
第 9 步——前馈网络

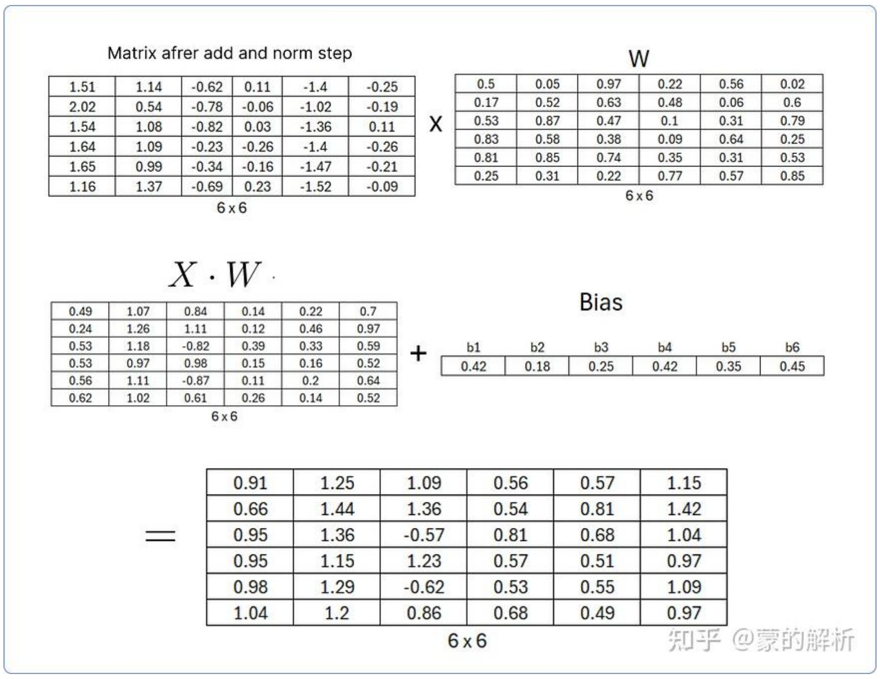
After normalizing the matrix, it will be processed through a feedforward network. We will be using a very basic network that contains only one linear layer and one ReLU activation function layer. This is how it looks like visually:
矩阵归一化后，将通过前馈网络进行处理。我们将使用一个非常基本的网络，仅包含一个线性层和一个 ReLU 激活函数层。这是它在视觉上的样子：



$\mathrm{ReLU}(x) = \max(0, x)$
$Linear\ Layer = X \cdot W + b$
Real world case (multiple layers)
Linear layer
RELU
Linear layer
RELU
Linear layer
our case (one linear layer)
RELU
Linear layer
知乎 @蒙的解析

Feed Forward network comparison
前馈网络比较

First, we need to calculate the linear layer by multiplying our last calculated matrix with a random set of weights matrix, which will be updated when the transformer starts learning, and adding the resultant matrix to a bias matrix that also contains random values.
首先，我们需要通过将最后计算的矩阵与一组随机权重矩阵相乘来计算线性层，该权重矩阵将在变压器开始学习时更新，并将所得矩阵添加到也包含随机值的偏差矩阵中。


Matrix afrer add and norm step

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

6 x 6

X

W

0.5	0.05	0.97	0.22	0.56	0.02
0.17	0.52	0.63	0.48	0.06	0.6
0.53	0.87	0.47	0.1	0.31	0.79
0.83	0.58	0.38	0.09	0.64	0.25
0.81	0.85	0.74	0.35	0.31	0.53
0.25	0.31	0.22	0.77	0.57	0.85

6 x 6

$X \cdot W$

0.49	1.07	0.84	0.14	0.22	0.7
0.24	1.26	1.11	0.12	0.46	0.97
0.53	1.18	-0.82	0.39	0.33	0.59
0.53	0.97	0.98	0.15	0.16	0.52
0.56	1.11	-0.87	0.11	0.2	0.64
0.62	1.02	0.61	0.26	0.14	0.52

6 x 6

+

Bias

b1	b2	b3	b4	b5	b6
0.42	0.18	0.25	0.42	0.35	0.45

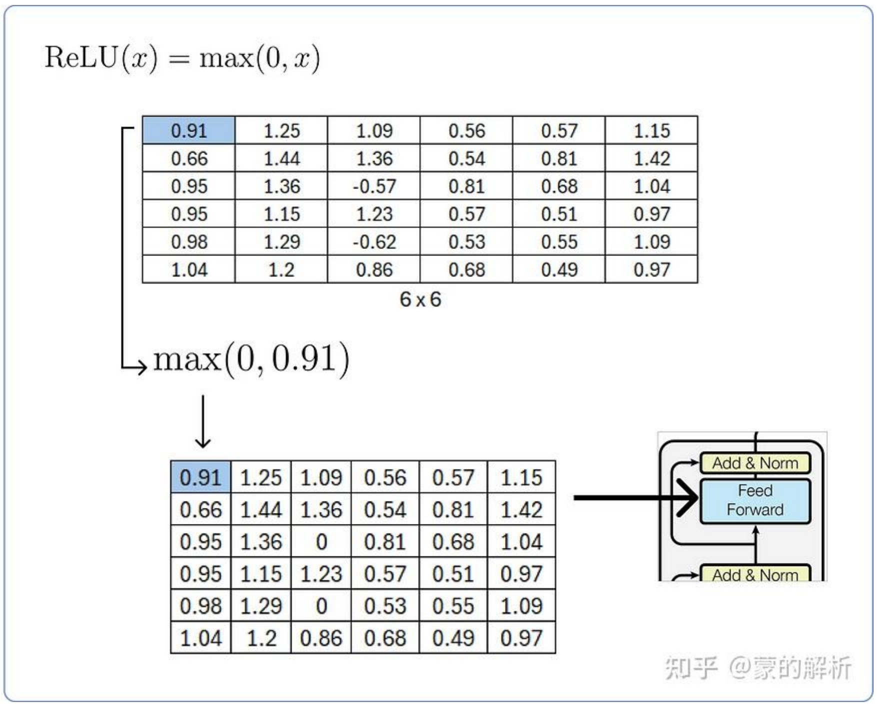
=

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	-0.57	0.81	0.68	1.04
0.95	1.15	1.23	0.57	0.51	0.97
0.98	1.29	-0.62	0.53	0.55	1.09
1.04	1.2	0.86	0.68	0.49	0.97

6 x 6

Calculating Linear Layer 计算线性层

After calculating the linear layer, we need to pass it through the ReLU layer and use its formula.
计算完线性层后，我们需要将其传递到ReLU层并使用其公式。



$$\mathrm{ReLU}(x) = \max(0, x)$$

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	-0.57	0.81	0.68	1.04
0.95	1.15	1.23	0.57	0.51	0.97
0.98	1.29	-0.62	0.53	0.55	1.09
1.04	1.2	0.86	0.68	0.49	0.97

6 x 6

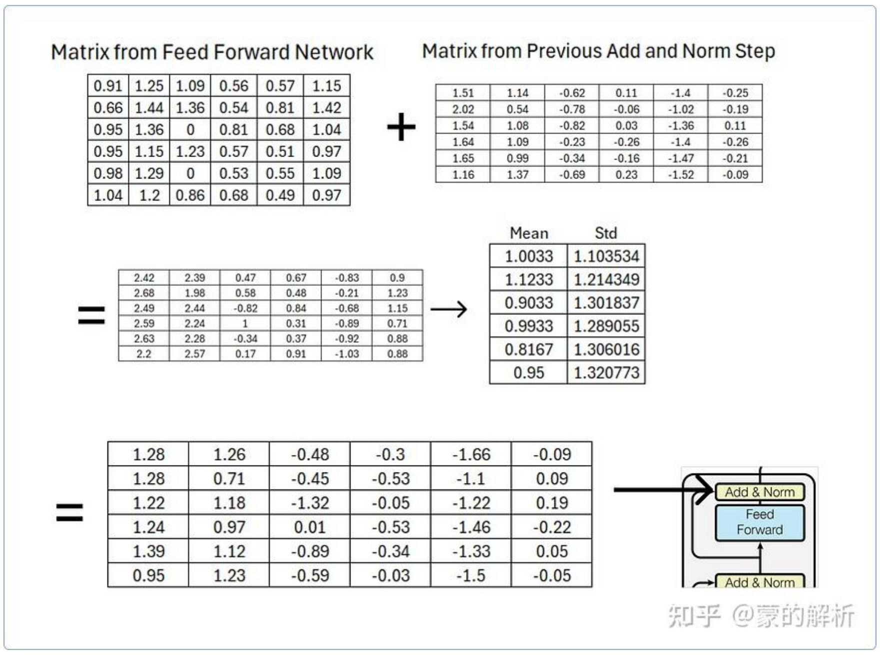
$$\max(0, 0.91)$$

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

Calculating ReLU Layer 计算 ReLU 层

Step 10 — Adding and Normalizing Again
第 10 步 — 再次添加并标准化

Once we obtain the resultant matrix from feed forward network, we have to add it to the matrix that is obtained from previous add and norm step, and then normalizing it using the row wise mean and standard deviation.
一旦我们从前馈网络获得结果矩阵，我们就必须将其添加到从之前的添加和归一步骤获得的矩阵中，然后使用行均值和标准差对其进行归一化。



Matrix from Feed Forward Network

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

+

Matrix from Previous Add and Norm Step

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

$\rightarrow$

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

Add and Norm after Feed Forward Network
前馈网络后添加并归一化

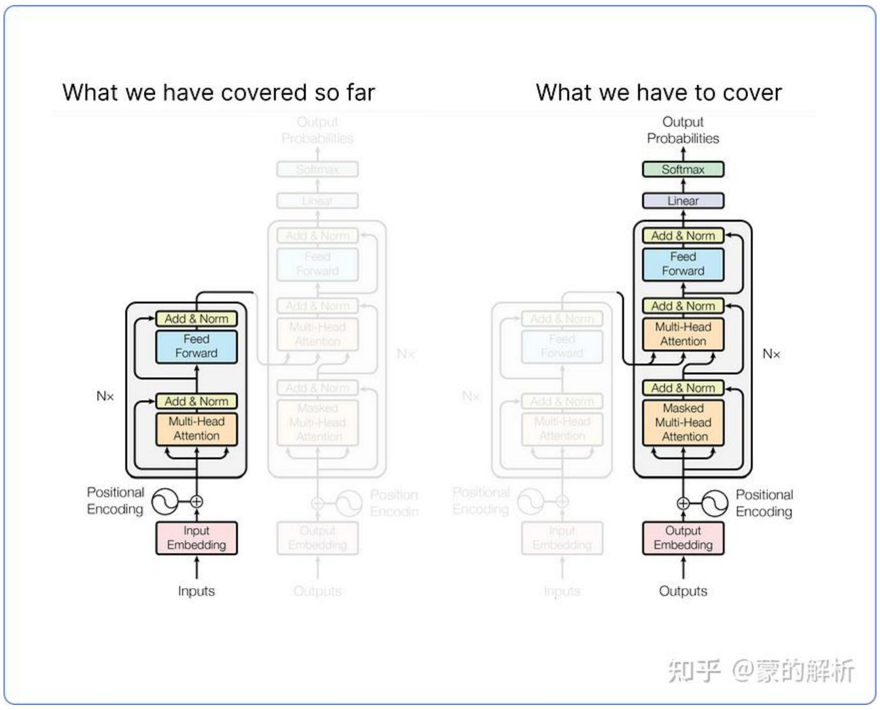
The output matrix of this add and norm step will serve as the query and key matrix in one of the multi-head attention mechanisms present in the decoder part, which you can easily

understand by tracing outward from the add and norm to the decoder section.
此加法和范数步骤的输出矩阵将充当解码器部分中存在的多头注意机制之一中的查询和关键矩阵，您可以通过从加法和范数向外追踪到解码器部分来轻松理解它。

Step 11 — Decoder Part
第11步——解码器部分

The good news is that up until now, we have calculated **Encoder part**, all the steps that we have performed, from encoding our dataset to passing our matrix through the feedforward network, are unique. It means we haven't calculated them before. But from now on, all the upcoming steps that is the remaining architecture of the transformer (**Decoder part**) are going to involve similar kinds of matrix multiplications.
好消息是，到目前为止，我们已经计算了编码器部分，我们执行的所有步骤，从编码数据集到通过前馈网络传递矩阵，都是唯一的。这意味着我们之前没有计算过它们。但从现在开始，接下来的所有步骤，即变压器的剩余架构（解码器部分）将涉及类似类型的矩阵乘法。

Take a look at our transformer architecture. What we have covered so far and what we have to cover yet:
看看我们的变压器架构。到目前为止我们已经涵盖的内容以及我们还需要涵盖的内容：

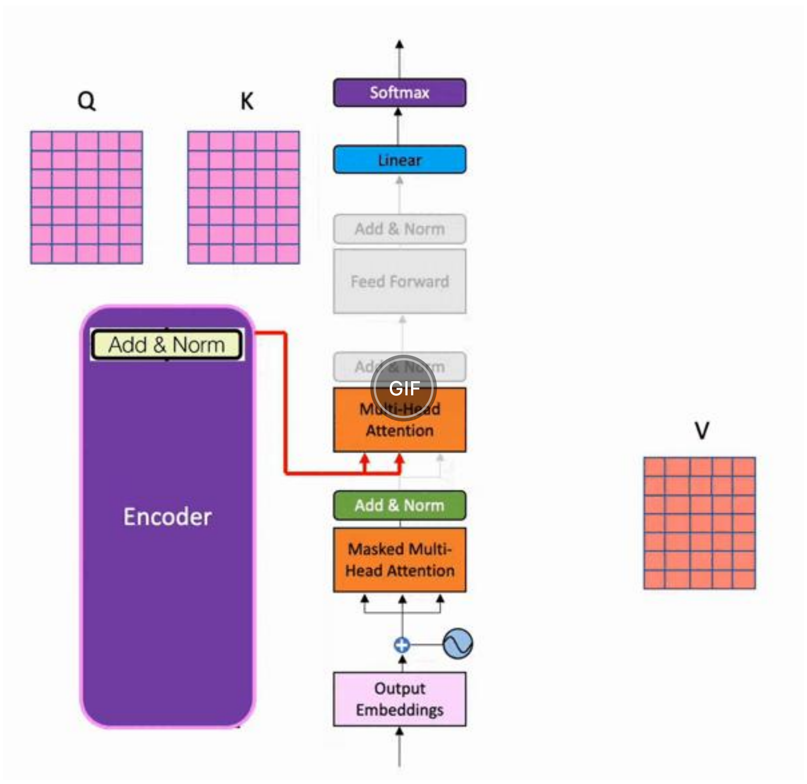


Upcoming steps illustration
后续步骤说明

We won't be calculating the entire decoder because most of its portion contains similar calculations to what we have already done in the encoder. Calculating the decoder in detail would only make the blog lengthy due to repetitive steps. Instead, we only need to focus on the calculations of the input and output of the decoder.
我们不会计算整个解码器，因为它的大部分部分包含与我们在编码器中已经完成的类似的计算。详细计算解码器只会因重复步骤而使博客变得冗长。相反，我们只需要关注解码器的输入和输出的计算。

When training, there are two inputs to the decoder. One is from the encoder, where the output matrix of the last add and norm layer serves as the **query** and **key** for the second multi-head attention layer in the decoder part. Below is the visualization of it (from batool haider):
训练时，解码器有两个输入。一种来自编码器，其中最后一个加法和规范层的输出矩阵充当解码

器部分中第二个多头注意层的查询和密钥。下面是它的可视化（来自batool Haider）：

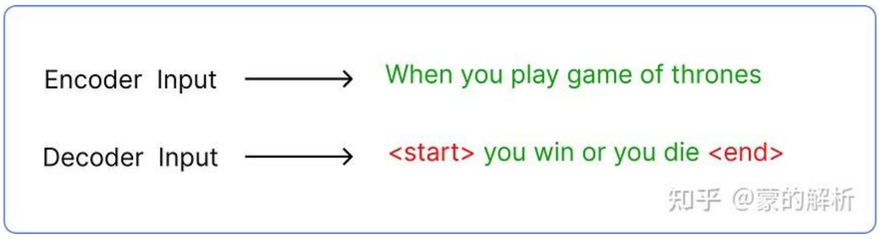


Visualization is from Batool Haider
可视化来自 Batool Haider

While the value matrix comes from the decoder after the first **add and norm** step.
而值矩阵来自解码器在第一个加法和归一步骤之后。

The second input to the decoder is the predicted text. If you remember, our input to the encoder is when you play game of thrones so the input to the decoder is the predicted text, which in our case is you win or you die .
解码器的第二个输入是预测文本。如果您还记得，我们对编码器的输入是 when you play game of thrones ，因此解码器的输入是预测文本，在我们的例子中是 you win or you die 。

But the predicted input text needs to follow a standard wrapping of tokens that make the transformer aware of where to start and where to end.
但是预测的输入文本需要遵循标记的标准包装，使转换器知道从哪里开始和在哪里结束。

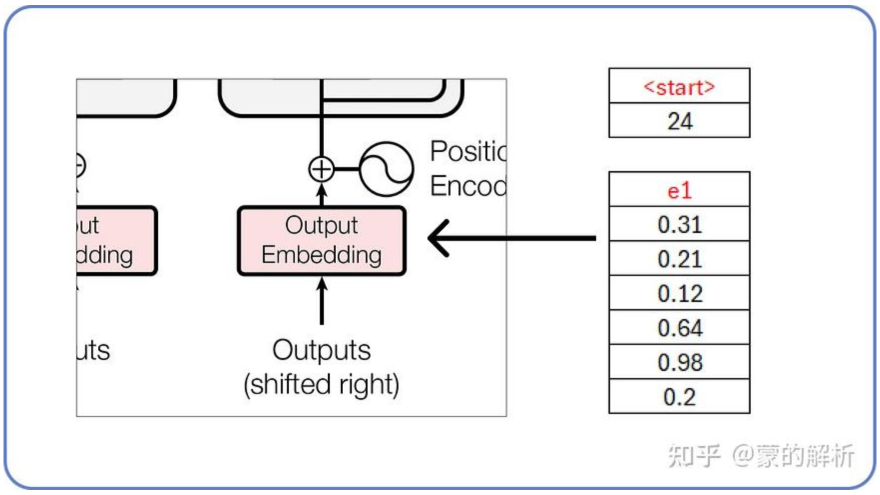


Encoder Input $\longrightarrow$ When you play game of thrones

Decoder Input $\longrightarrow$ <start> you win or you die <end>

input comparison of encoder and decoder
编码器和解码器的输入比较

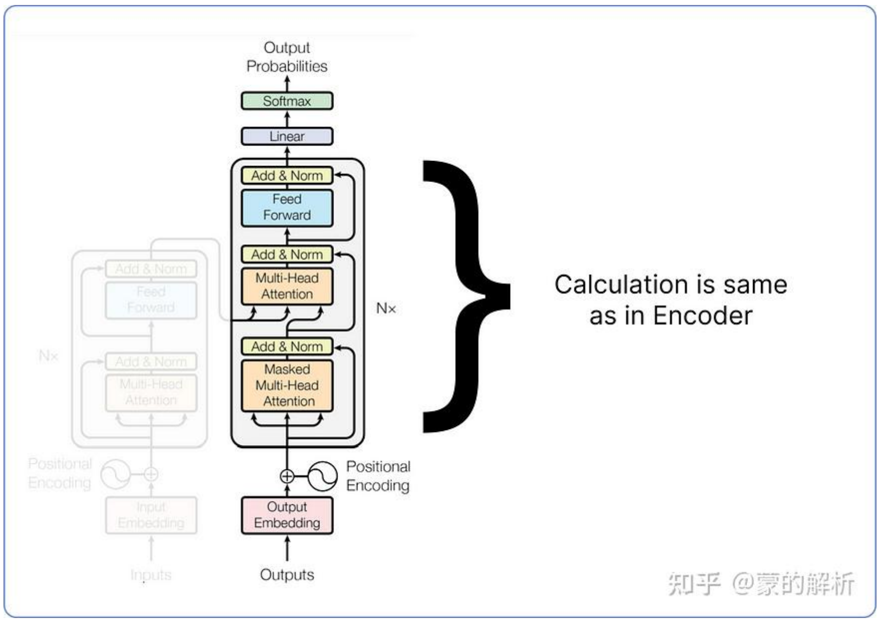
Where <start> and <end> are two new tokens being introduced. Moreover, the decoder takes one token as an input at a time. It means that <start> will be served as an input, and you must be the predicted text for it.
其中 <start> 和 <end> 是引入的两个新令牌。此外，解码器一次将一个令牌作为输入。这意味着 <start> 将作为输入，而 you 必须是它的预测文本。



Decoder input **<start>** word
解码器输入<start>字

As we already know, these embeddings are filled with random values, which will later be updated during the training process.
正如我们所知，这些嵌入填充有随机值，这些值稍后将在训练过程中更新。

Compute rest of the blocks in the same way that we computed earlier in the encoder part.
按照我们之前在编码器部分计算的相同方式计算其余块。



Calculating Decoder 计算解码器

Before diving into any further details, we need to understand what masked multi-head attention is, using a simple mathematical example.
在深入研究任何进一步的细节之前，我们需要使用一个简单的数学示例来了解什么是屏蔽多头注意力。

Step 12 — Understanding Mask Multi Head Attention
第 12 步 — 了解 Mask Multi Head Attention

In a Transformer, the masked multi-head attention is like a spotlight that a model uses to focus on different parts of a sentence. It's special because it doesn't let the model cheat by looking at words that come later in the sentence. This helps the model understand and

generate sentences step by step, which is important in tasks like talking or translating words into another language.
在 Transformer 中，屏蔽的多头注意力就像聚光灯，模型用它来关注句子的不同部分。它很特别，因为它不会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，这对于说话或将单词翻译成另一种语言等任务非常重要。

Suppose we have the following input matrix, where each row represents a position in the sequence, and each column represents a feature:
假设我们有以下输入矩阵，其中每行代表序列中的一个位置，每列代表一个特征：

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

inpur matrix for masked multi head attentions
用于屏蔽多头注意力的 inpur 矩阵

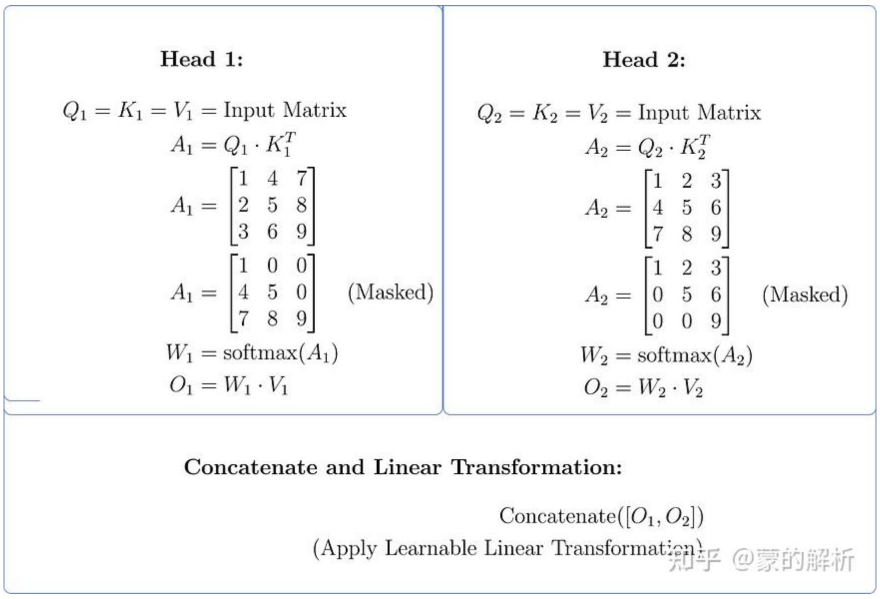
Now, let's understand the masked multi-head attention components having two heads:
现在，让我们了解具有两个头的屏蔽多头注意力组件：

1. **Linear Projections (Query, Key, Value):** Assume the linear projections for each head: **Head 1:** $Wq1, Wk1, Wv1$ and **Head 2:** $Wq2, Wk2, Wv2$
   线性投影（查询、键、值）：假设每个头的线性投影：头 1：Wq1、Wk1、Wv1 和头 2：Wq2、Wk2、Wv2
2. **Calculate Attention Scores:** For each head, calculate attention scores using the dot product of Query and Key, and apply the mask to prevent attending to future positions.
   计算注意力分数：对于每个头，使用查询和密钥的点积计算注意力分数，并应用掩码以防止关注未来的位置。
3. **Apply Softmax:** Apply the softmax function to obtain attention weights.
   应用Softmax：应用softmax函数来获取注意力权重。
4. **Weighted Summation (Value):** Multiply the attention weights by the Value to get the weighted sum for each head.
   加权求和（值）：将注意力权重乘以值，得到每个头的加权和。
5. **Concatenate and Linear Transformation:** Concatenate the outputs from both heads and apply a linear transformation.
   连接和线性变换：连接两个头的输出并应用线性变换。

Let's do a simplified calculation:
我们来做一个简单的计算：

Assuming two conditions 假设两个条件

- $Wq1 = Wk1 = Wv1 = Wq2 = Wk2 = Wv2 = I$, the identity matrix.
  Wq1 = Wk1 = Wv1 = Wq2 = Wk2 = Wv2 = I，单位矩阵。
- $Q = K = V = $**Input Matrix 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:

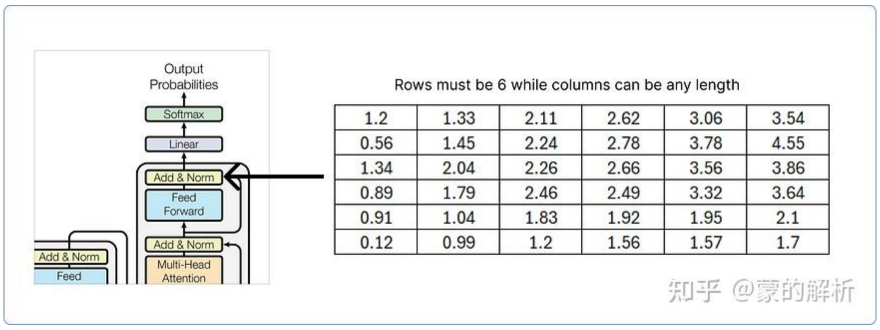
$$\text{Concatenate}([O_1, O_2])$$
(Apply Learnable Linear Transformation)

Mask Multi Head Attention (**Two Heads**)
面膜多头注意（两个头）

The concatenation step combines the outputs from the two attention heads into a single set of information. Imagine you have two friends who each give you advice on a problem. Concatenating their advice means putting both pieces of advice together so that you have a more complete view of what they suggest. In the context of the transformer model, this step helps capture different aspects of the input data from multiple perspectives, contributing to a richer representation that the model can use for further processing.
连接步骤将两个注意力头的输出组合成一组信息。想象一下，您有两个朋友，他们每个人都为您提供有关问题的建议。连接他们的建议意味着将两条建议放在一起，以便您可以更全面地了解他们的建议。在变压器模型的上下文中，此步骤有助于从多个角度捕获输入数据的不同方面，从而有助于模型可用于进一步处理的更丰富的表示。

Step 13 — Calculating the Predicted Word
第 13 步 — 计算预测词

The output matrix of the last add and norm block of the decoder must contain the same number of rows as the input matrix, while the number of columns can be any. Here, we work with 6.
解码器最后一个加法和范数块的输出矩阵必须包含与输入矩阵相同的行数，而列数可以是任意的。在这里，我们使用 6 。



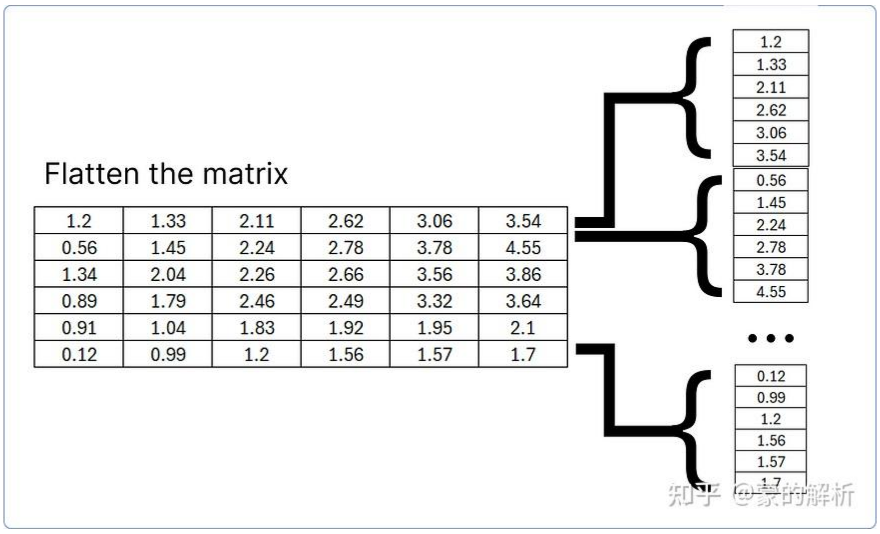
Rows must be 6 while columns can be any length

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

Add and Norm output of decoder
解码器输出的加法和归一化

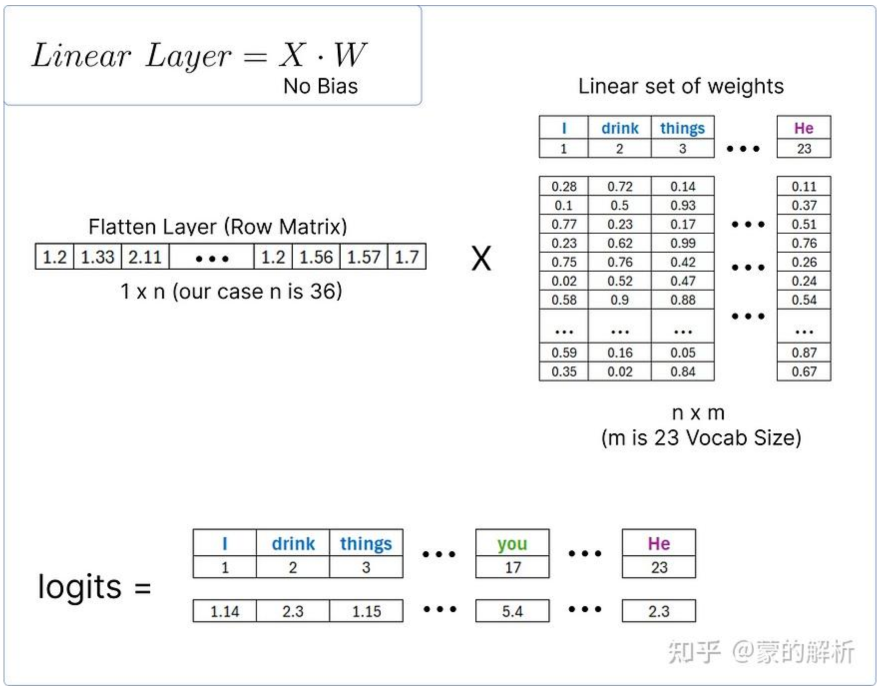
The last **add and norm block** resultant matrix of the decoder must be flattened in order to match it with a linear layer to find the predicted probability of each unique word in our dataset (corpus).

解码器的最后一个加法和范数块结果矩阵必须展平，以便将其与线性层匹配，以找到数据集（语料库）中每个唯一单词的预测概率。



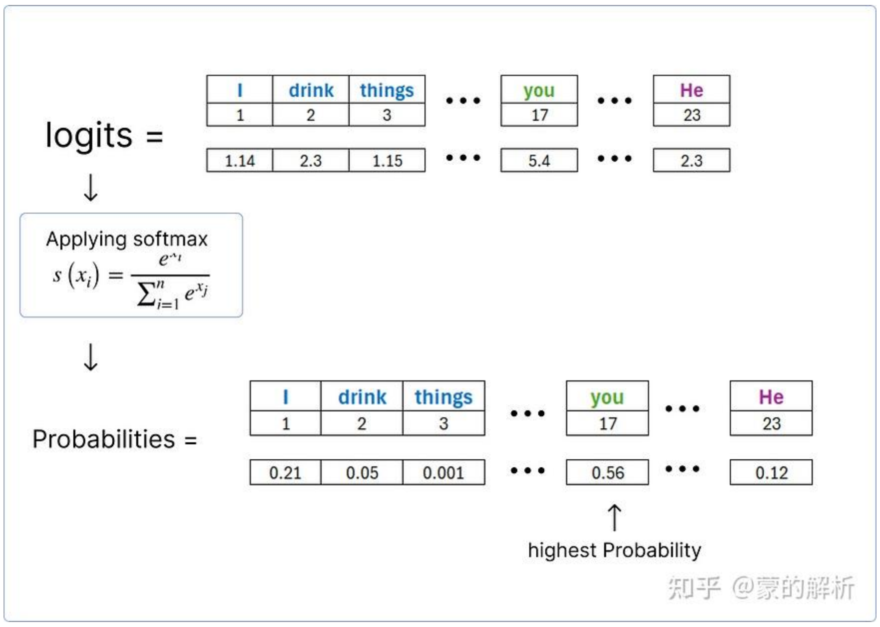
flattened the last add and norm block matrix
展平最后一个加法和范数块矩阵

This flattened layer will be passed through a linear layer to compute the **logits** (scores) of each unique word in our dataset.
这个扁平层将通过线性层来计算数据集中每个唯一单词的 logits（分数）。



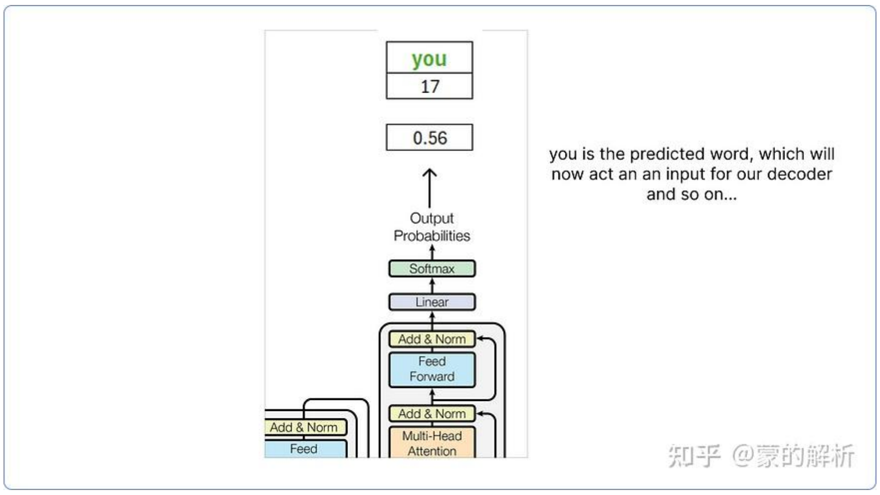
Calculating Logits 计算逻辑

Once we obtain the logits, we can use the **softmax** function to normalize them and find the word that contains the highest probability.
一旦我们获得了 logits，我们就可以使用 softmax 函数对它们进行归一化并找到包含最高概率的单词。



Finding the Predicted word
寻找预测的单词

So based on our calculations, the predicted word from the decoder is you.
因此，根据我们的计算，解码器预测的单词是 you 。



Final output of decoder 解码器的最终输出

This predicted word you, will be treated as the input word for the decoder, and this process continues until the <end> token is predicted.
该预测的单词 you 将被视为解码器的输入单词，并且此过程将继续，直到预测出 <end> 标记。

Important Points 要点

1. The above example is very simple, as it does not involve epochs or any other important parameters that can only be visualized using a programming language like Python.
   上面的例子非常简单，因为它不涉及纪元或任何其他重要参数，这些参数只能使用Python等编程语言可视化。
2. It has shown the process only until training, while evaluation or testing cannot be visually seen using this matrix approach.
   它只显示了训练之前的过程，而使用这种矩阵方法无法直观地看到评估或测试。
3. Masked multi-head attentions can be used to prevent the transformer from looking at the future, helping to avoid overfitting your model.

屏蔽多头注意力机制可用于防止 Transformer 着眼于未来，从而有助于避免模型过度拟合。

Conclusion 结论

In this blog, I have shown you a very basic way of how transformers mathematically work using matrix approaches. We have applied positional encoding, softmax, feedforward network, and most importantly, multi-head attention.
在这篇博客中，我向您展示了变压器如何使用矩阵方法进行数学工作的非常基本的方法。我们应用了位置编码、softmax、前馈网络，以及最重要的多头注意力。

所属专栏 · 2024-09-11 14:44 更新

AI 大模型
蒙的解析 咨询行业 从业人员
49 篇内容 · 37 赞同

订阅

最热内容 · Building “Auto-Analyst” — A data analytics AI agentic system 构建“自动分析师”——数据分析人工智能代理系统

发布于 2024-05-09 07:05 · 河北

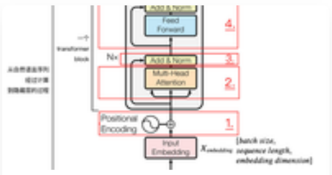
LLM（大型语言模型） Transformer 大模型

理性发言，友善互动

还没有评论，发表第一个评论吧

推荐阅读

我寻思Transformer也没有那么难［究极缝合怪版］ - Attentio...
Yuki



Bert前篇：手把手带你详解Transformer原理
数据分析Dogs

香侬读 | Transforme
up和LayerNorm的
香侬科技