

Repeat timeline

done: 24.08.28

24.08.29

Transformer // Friday 240823

2017 <Attention is all you need> seq2seq

Bert 从 Transformer 衍生出的预训练语言模型

content

(1) 直观认识

LSTM 迭代，串行 当前层处理完，下一个字

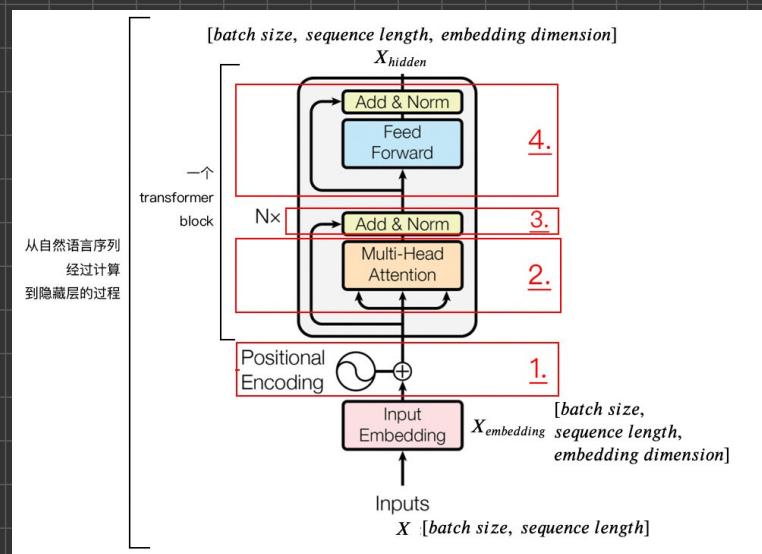
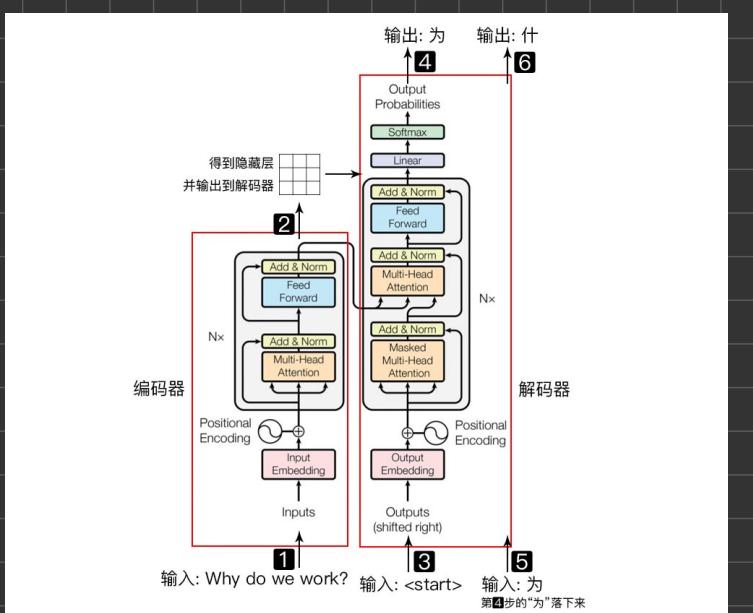
Transformer 并行 所有层同时训练

{ 计算效率 up up
positional encoding
self-attention
全连接层

position encoding 理解为词向量

Transformer { encoder input $\xrightarrow{\text{map}}$ 隐藏层
decoder 隐藏层 $\xrightarrow{\text{map}}$ 自然语言序列

Encoder 自然语言序列 $\xrightarrow{\text{Map}}$ hidden layer



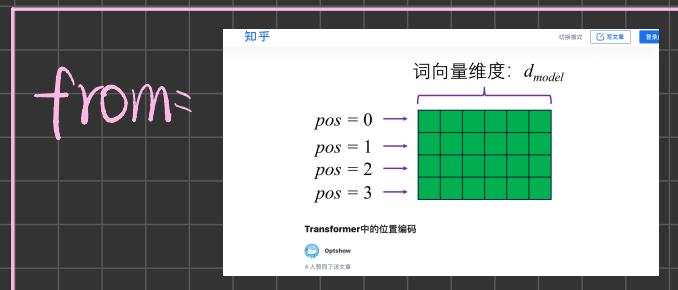
STAR key points:

□ $X \in [\text{batch size}, \text{sequence length}]$

X embedding

20240826

Attention Positional Encoding



$$PE(\text{pos}, 2i) = \sin \frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}} = \sin(w_i \cdot \text{pos})$$

$$PE(\text{pos}, 2i+1) = \cos \frac{\text{pos}}{10000^{\frac{2i+1}{d_{\text{model}}}}} = \cos(w_i \cdot \text{pos})$$

$$w_i = \frac{1}{10000^{\frac{2i}{d_{\text{model}}}}}$$

词向量维度 d_{model}

pos=0		...	
pos=1		...	
pos=2		...	
pos=3		...	

(1) pos 是什么?

如 $\text{pos}=0$ } 一个词向量
 } 数学向量未表示一个词语
 } eq. one-hot, word2vec

(2) i 怎么理解?

i 每一个词向量的元素位置

} 2i 偶数位置
 } 2i+1 奇数位置

eg $\text{pos}=0$

$i=0$ 时 $2i$ 和 $2i+1$ 表示该词向量的第一个和第二个元素

$i=1$ 时 $2i$ 和 $2i+1$ ~ 第 3 个和第 4 个 ~

$i=2$ 时 $2i$ 和 $2i+1$ ~ 第 5 个和第 6 个 ~

NOTE: 人的计数从 1 开始 计算机从 0 开始

(3) 位置向量的编码长度

$$\begin{cases} \text{PE}(pos, 2i) \\ \text{PE}(pos, 2i+1) \end{cases} \quad \text{eq } pos=0 \quad \begin{cases} \text{PE}(pos=0, 2i) \\ \text{PE}(pos=0, 2i+1) \end{cases}$$

$d_{\text{PE}} = d_{\text{model}}$

每个位置编码向量长度 $d_{\text{PE}} = \text{词向量维度 } d_{\text{model}}$

(4) 具体的应用?

eg1: 若 $pos=0$ $d_{\text{model}}=4$, 则该词向量表示成:

$$x_0 = [x_{00}, x_{01}, x_{02}, x_{03}]$$

现对该词进行位置编码

$$\text{PE}_0 = [\text{PE}(0,0), \text{PE}(0,1), \text{PE}(0,2), \text{PE}(0,3)]$$

NOTE: $d_{\text{PE}_0} = d_{x_0} = 4 = d_{\text{model}}$ ($\text{词向量长度} = \text{位置编码长度}$)

当 $i=0$ 时, (一定要区分, i 不是位置, $i=0, 1$ 那可对该词进行位置编码)

$$\text{PE}(0,0) = \sin(w_0 \cdot 0) = \sin 0 = 0$$

$$\text{PE}(pos, 2i) = \sin \frac{pos}{10000 \frac{2i}{d_{\text{model}}}} = \sin(w_i \cdot pos)$$

$$\text{PE}(pos, 2i+1) = \cos \frac{pos}{10000 \frac{2i+1}{d_{\text{model}}}} = \cos(w_i \cdot pos)$$

$$w_i = \frac{1}{10000 \frac{2i}{d_{\text{model}}}}$$

$$\text{PE}(0,1) = \cos(w_0 \cdot pos) = \cos(w_0 \cdot 0) = 1$$

当*i*=1时

$$PE(0,2) = \sin(w_1 \cdot 0) = \sin\left(\frac{1}{10000^{\frac{2}{7}}} \cdot 0\right) = 0$$

$$PE(0,3) = \cos(w_1 \cdot 0) = 1$$

∴ x_0 的位置编码 $PE_0 = [\sin(w_0 \cdot 0), \cos(w_0 \cdot 0), \sin(w_1 \cdot 0), \cos(w_1 \cdot 0)]$
 $= [0, 1, 0, 1]$

★ 位置编码后的词向量

$$\hat{x}_0 = x_0 + PE_0$$

$$= [x_{00} + \sin(w_0 \cdot 0), x_{01} + \cos(w_0 \cdot 0), x_{02} + \sin(w_1 \cdot 0), x_{03} + \cos(w_1 \cdot 0)]$$

用向量表示叠加过程

$$pos=0 \quad \boxed{\quad \quad \quad} \quad + \quad \boxed{\quad \quad \quad} \quad = \quad \boxed{\quad \quad \quad} \quad \hat{x}_0$$

eq2.

例子2：

假设 $pos = 1$ 位置的词向量长度为8，即 $d_{model} = 8$ ，这个词向量可以表示为

$$x_1 = [x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}], \quad (7)$$

那么对应这个词向量 x_1 的位置编码向量长度 $d_{PE1} = 8$ ，可以表示为，

$$PE1 = [PE_{(1,0)}, PE_{(1,1)}, PE_{(1,2)}, PE_{(1,3)}, PE_{(1,4)}, PE_{(1,5)}, PE_{(1,6)}, PE_{(1,7)}], \quad (8)$$

由于长度为8，所以根据上面公式可知 $i \in [0, 3]$ ，即

当 $i = 0$ 时，

$$PE1_{(1,0)} = \sin(w_0 \cdot 1) = \sin\left(\frac{1}{10000^{2 \times 0/7}} \cdot 1\right),$$

$$PE1_{(1,1)} = \cos(w_0 \cdot 1) = \cos\left(\frac{1}{10000^{2 \times 0/7}} \cdot 1\right)$$

当 $i = 1$ 时，

$$PE1_{(1,2)} = \sin(w_1 \cdot 1) = \sin\left(\frac{1}{10000^{2 \times 1/7}} \cdot 1\right),$$

$$PE1_{(1,3)} = \cos(w_1 \cdot 1) = \cos\left(\frac{1}{10000^{2 \times 1/7}} \cdot 1\right)$$

当 $i = 2$ 时，

$$PE1_{(1,4)} = \sin(w_2 \cdot 1) = \sin\left(\frac{1}{10000^{2 \times 2/7}} \cdot 1\right),$$

$$PE1_{(1,5)} = \cos(w_2 \cdot 1) = \cos\left(\frac{1}{10000^{2 \times 2/7}} \cdot 1\right)$$

当 $i = 3$ 时，

$$PE1_{(1,6)} = \sin(w_3 \cdot 1) = \sin\left(\frac{1}{10000^{2 \times 3/7}} \cdot 1\right),$$

$$PE1_{(1,7)} = \cos(w_3 \cdot 1) = \cos\left(\frac{1}{10000^{2 \times 3/7}} \cdot 1\right)$$

那么位置编码后的词向量就是

$$\hat{x}_1 = x_1 + PE1 \quad (9)$$

24 0826 Transformer

from: <https://wmathor.com/index.php/archives/1438/>

(1) positional encoding

① 位置嵌入维度 = 词向量的维度

② PE 的 shape ?

for 整个词表 位置嵌入是一个矩阵

[max-sequence-length, embedding-dimension]

口 这2个都是什么 meaning?

max-sequence-length: 限定每个句子最长由多少个词构成

embedding-dim: 嵌入维度 = 词向量维度

D _____? (为什么会说这句话，带着思考学习)

以字为单位训练 Transformer

初始化字典大小 [vocab-size, embedding-dimension]

vocab_size 字典中所有字的数量

embedding-dimension 字向量的维度

pytorch: nn.Embedding(vocab-size, embedding-dimension)

字和词

? 这段话目的是什么?

口 positional encoding 公式

$$PE(pos, 2t) = \sin(pos / 10000^{2t/d_{model}})$$

$$PE(pos, 2t+1) = \cos(pos / 10000^{2t/d_{model}})$$

| pos: 一句话中某个字的位置 (这个字的定义很像词)

| t: 字向量的维度

| d_model: = embedding-dimension

pos: [0, max_sequence_length]

i: [0, embedding_dimension/2)

dmodel = embedding_dimension

周期性变化怎么理解？

sin, cos 对应着

embedding 维度上一组奇数 & 偶数的序号的维度

如 0, 1 一组, 2, 3 一组

分别用 sin & cos 处理 从而产生周期性变化

位置嵌入在 embedding_dimension 维度上 随着维度增大 周期变化会越来越慢

最终产生包含位置信息的纹理

embedding 维度上 维度应当

老 pos=0 1 1 1 1 1 1 embedding_dimension = 256

鹰 pos=1

报 pos=2

小 pos=3

鸡 pos=4

$$PE(pos, 2i) = \sin(pos \cdot w_i) \quad \uparrow \quad PE(pos, 2i+1) \downarrow \quad (pos \text{ 不变情况下})$$

$$PE(pos, 2i+1) = \cos(pos \cdot w_i)$$

$$w_i = 1/10000^{2i/dmodel}$$

为什么这个公式能够反应位置信息？

位置嵌入函数从 2π 到 $10000 * 2\pi$ 变化

而每一个位置在 positional_dimension 维度上都会得到不同周期的 sin 和 cos 函数的取值组合，从而产生唯一的纹理信息

⇒ 使模型能够学习位置之间的依赖关系和自然语言的时序特性

Add. positional encoding 再学再探

(1) Transformer 以字作为输入

<https://wmathor.com/index.php/archives/1453/>

{ 字 → 嵌入 > 对应位置相加 (数值上进行加和)
位置嵌入

(2) 若我们设计一个 position_encoding

T0, T1 之间数分配给每个字 { 0 给第 1 个
| 给最后一个
 $\Rightarrow PE = \frac{pos}{T-1}$ T = 序列长度 { pos 从 0 开始计数
| 长度 T 从 1 开始计数

(3) 可视化 50 个词 128 维

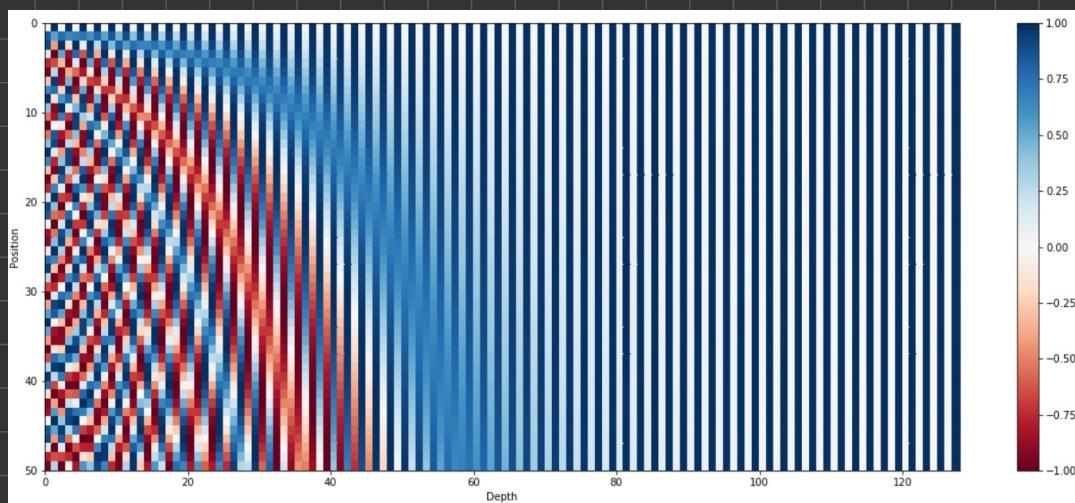


Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector \vec{p}_t

你可以想象下 t 时刻字的位置编码 \vec{p}_t 是一个包含 \sin 和 \cos 函数的向量 (假设 d 可以被 2 整除)

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_0 \cdot t) \\ \cos(\omega_0 \cdot t) \\ \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \vdots \\ \sin(\omega_{\frac{d}{2}-1} \cdot t) \\ \cos(\omega_{\frac{d}{2}-1} \cdot t) \end{bmatrix}_{d \times 1}$$

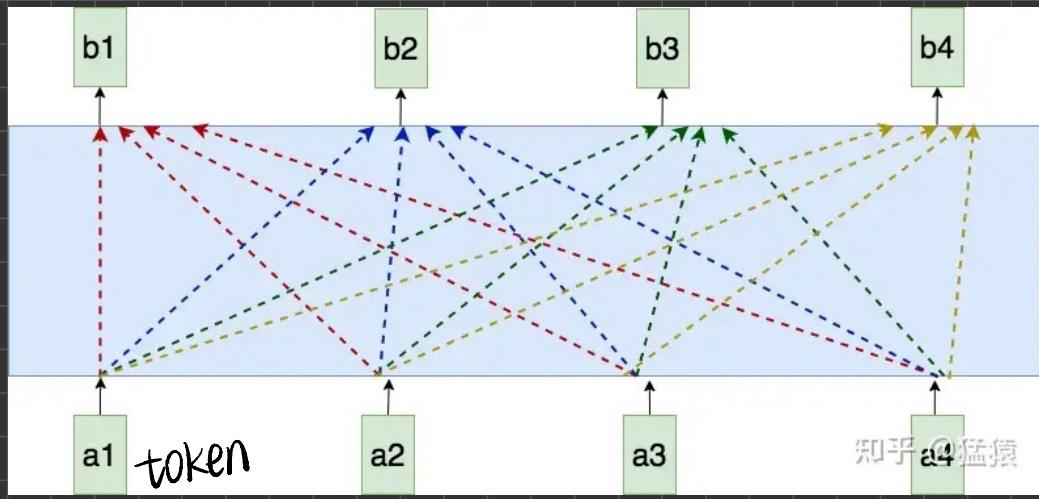
Add2.

如何理解Transformer论文中的positional encoding，和三角函数有什么关系？ - 猛猿的回答 - 知乎

<https://www.zhihu.com/question/347678607/answer/2301693596>

✓ Input = Input_embedding + positional_encoding

(2)



在self-attention模型中，输入是一整排的tokens，对于人来说，我们很容易知道tokens的位置信息，比如：

- (1) 绝对位置信息。a1是第一个token，a2是第二个token.....
- (2) 相对位置信息。a2在a1的后面一位，a4在a2的后面两位.....
- (3) 不同位置间的距离。a1和a3差两个位置，a1和a4差三个位置....

但是这些对于self-attention来说，是无法分辨的信息，因为self-attention的运算是无向的。因为，我们要想办法，把tokens的位置信息，喂给模型。

③ 假如我(小白)设计位置编码

二、构造位置编码的方法 / 演变历程

✓ 2.1 用整型值标记位置 \Rightarrow 无界的不行

一种自然而然的想法是，给第一个token标记1，给第二个token标记2...，以此类推。
这种方法产生了以下几个主要问题：

- (1) 模型可能遇见比训练时所用的序列更长的序列。不利于模型的泛化。
- (2) 模型的位置表示是无界的。随着序列长度的增加，位置值会越来越大。

✓ 2.2 用[0,1]范围标记位置 \Rightarrow 不同的向量长度相对位置要相同

为了解决整型值带来的问题，可以考虑将位置值的范围限制在[0, 1]之内，其中，0表示第一个token，1表示最后一个token。比如有3个token，那么位置信息就表示成[0, 0.5, 1]；若有四个token，位置信息就表示成[0, 0.33, 0.69, 1]。

但这样产生的问题是，当序列长度不同时，token间的相对距离是不一样的。例如在序列长度为3时，token间的相对距离为0.5；在序列长度为4时，token间的相对距离就变为0.33。

因此，我们需要这样一种位置表示方式，满足于：

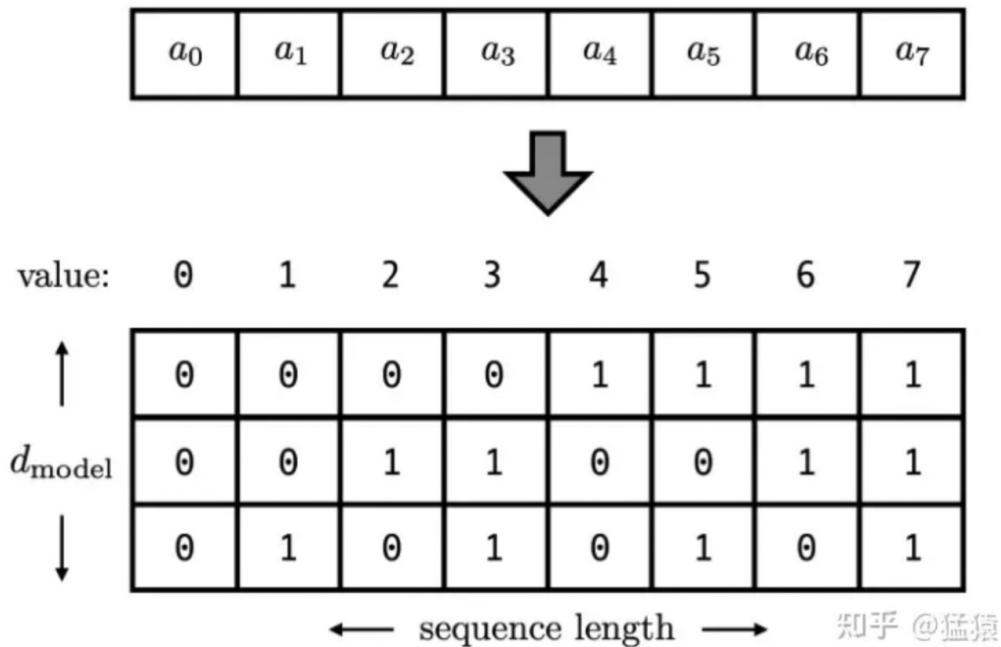
- (1) 它能用来表示一个token在序列中的绝对位置
- (2) 在序列长度不同的情况下，不同序列中token的相对位置/距离也要保持一致
- (3) 可以用来表示模型在训练过程中从来没有看到过的句子长度。

2.3 用二进制向量标记位置

离散的不行

考虑到位置信息作用在input embedding上，因此比起用单一的值，更好的方案是用一个和input embedding维度一样的向量来表示位置。这时我们就很容易想到二进制编码。如下图，假设 $d_{model} = 3$ ，那么我们的位置向量可以表示成：

输入维度 $2^3 = 8$ 可以表示最多8个token

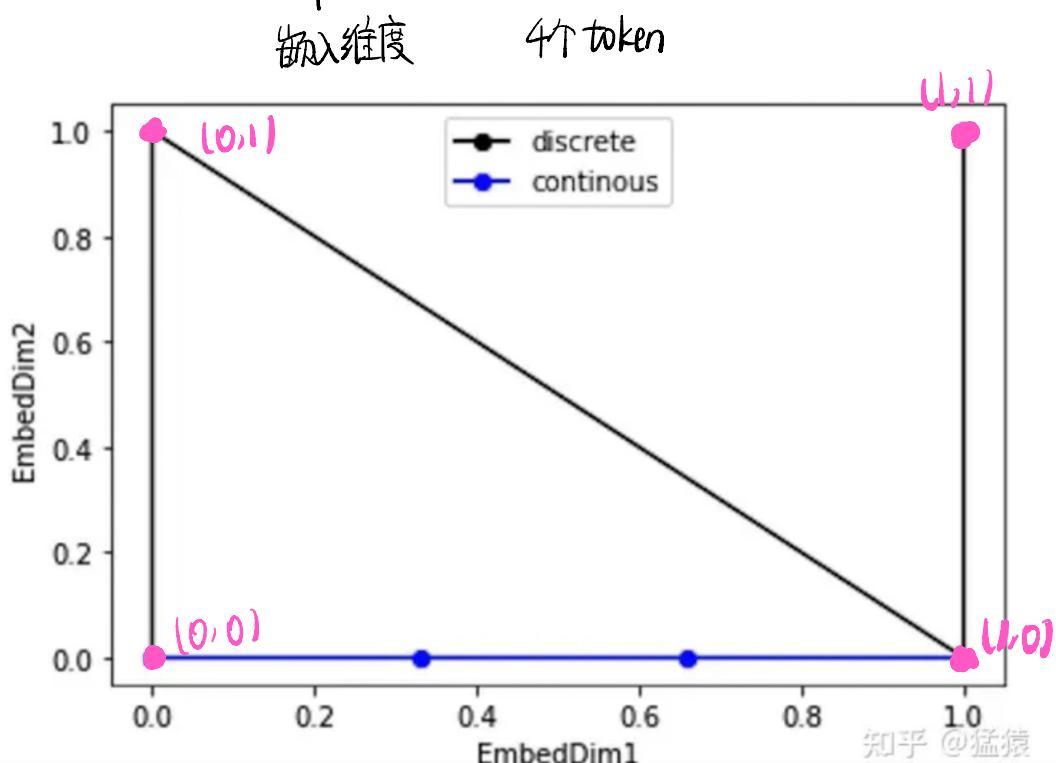


这下所有的值都是有界的（位于0, 1之间），且transformer中的 d_{model} 本来就已经足够大，基本可以把我们要的每一个位置都编码出来了。

$$2^{50} > 1412$$

但是这种编码方式也存在问题：这样编码出来的位置向量，处在一个离散的空间中，不同位置间的变化是不连续的。假设 $d_{model} = 2$ ，我们有4个位置需要编码，这四个位置向量可以表示成 $[0,0], [0,1], [1,0], [1,1]$ 。我们把这4个位置向量空间做出来：

而原文中 $d_{model} = 512$



PyTorch reference

知乎

如何理解Transformer论文中的positional encoding，和三角函数有什么关系？ - 猛猿的回答 - 知乎

<https://www.zhihu.com/question/347678607/answer/2301693596>

wmathor

<https://wmathor.com/index.php/archives/1438/>

一文教你彻底理解Transformer中Positional Encoding

- 月球上的人的文章 - 知乎

<https://zhuanlan.zhihu.com/p/338592312>

Transformer 位置编码框架归纳

(1) positional encoding & positional embedding

(2) 是什么? | def (1个)
eg (2个)

(3) 为什么? (positional encoding 的演化过程) (5步)

(1) positional encoding 不需要训练参数 通过 位置编码
positional embedding 位置嵌入 是训练出来的

(2) 是什么?

note: 彻底明白了, 学习逻辑给出 来 对着写几块吧。

2.5 用 \sin & \cos 表示位置

不同的位置向量通过线性变换得到

↓ 绝对位置
↓ 相对位置

$$\text{即 } PE_{t+\Delta t} = T_{\Delta t} * PE_t$$

T : 线性变换矩阵

向量在线性空间: 旋转

若 t 为一个角度 Δt : 旋转的角度

写成

$$\begin{pmatrix} \sin(t + \Delta t) \\ \cos(t + \Delta t) \end{pmatrix} = \begin{pmatrix} \cos \Delta t & \sin \Delta t \\ -\sin \Delta t & \cos \Delta t \end{pmatrix} \begin{pmatrix} \sin t \\ \cos t \end{pmatrix}$$

$$PE_t = [\sin(\omega t), \cos(\omega t), \sin(\omega t), \cos(\omega t), \dots, \sin(\omega_{\text{model}} t), \cos(\omega_{\text{model}} t)]$$

t

$$PE_t \in \mathbb{R}^d \quad PE_t^{(i)}$$

$$d_{\text{model}} = 512$$

$$PE_t^{(i)} = \begin{cases} \sin \omega_i t & k=2i \text{ even} \\ \cos \omega_i t & k=2i+1 \end{cases}$$

$$\omega_i = \frac{1}{10000^{2i}/d_{\text{model}}}$$

$$i = 0, 1, 2, \dots, \frac{d_{\text{model}}}{2} - 1$$

512 维 两两一组

每一组 \sin & \cos

共享 1 个频率 ω_i

共 256 组

从 0 开始 编号

∴ 最后一组 编号 255

$$255 \times 2 = 510$$

\sin / \cos 函数波长由 W_i 决定

则从 2π 增长到 $2\pi * 10000$

$$W_i = \frac{1}{10000^{2\pi/d_{model}}}$$

$$T=0 \quad W_i = 1 \quad 2\pi$$

$$T=255 \quad 1/W_i = 10000^{2\pi/d_{model}} = 10000$$

$$\text{波长 } t = \frac{2\pi}{W_i} \quad (\text{差不多是 } 1)$$

3.2 位置化

$$T=50 \quad d_{model}=128$$

Positional Embedding
| Encode

$$pos=0$$

$$pos=60$$

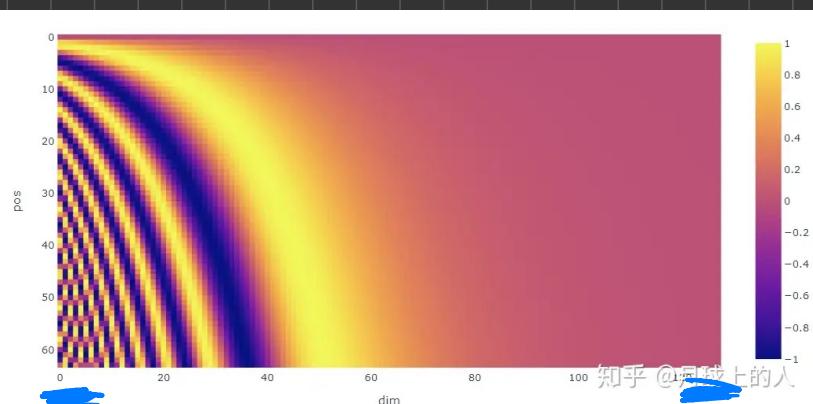
越低位变化越快

2 进制 编码

0 :	0 0 0 0	8 :	1 0 0 0
1 :	0 0 0 1	9 :	1 0 0 1
2 :	0 0 1 0	10 :	1 0 1 0
3 :	0 0 1 1	11 :	1 0 1 1
4 :	0 1 0 0	12 :	1 1 0 0
5 :	0 1 0 1	13 :	1 1 0 1
6 :	0 1 1 0	14 :	1 1 1 0
7 :	0 1 1 1	15 :	1 1 1 1

低位

为什么这样简单的sines和cosines的组合可以表达位置信息呢？一开始的确有点难理解。别着急，这边举个二进制的例子就明白了。可以观察一下下面这个表，我们将数字用二进制表示出来。可以发现，每个比特位的变化率是不一样的，越低位的变化越快，红色位置0和1每个数字会变化一次，而黄色位，每8个数字才会变化一次。



越低位变化越快

高位变化慢

case1: $\dim = 0 \rightarrow \dim = 128$

低位 \rightarrow 高位

固定 \dim 看变化 低位振荡快

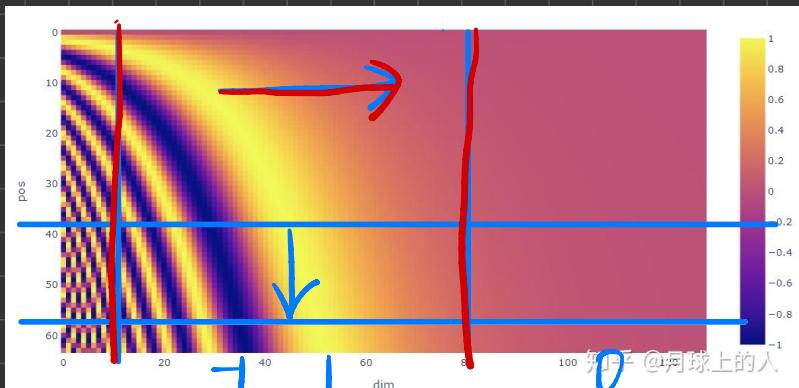
变化大

(1) 看颜色 -1, 1 (类比2进制0, 1)

(2) 看变化频率

case2: 固定 pos

低位变化快 高位几乎不变



但为什么是0呢？

pos 变大 | 波的波段越长
| 0越少

观察 pos

低位 & 高位

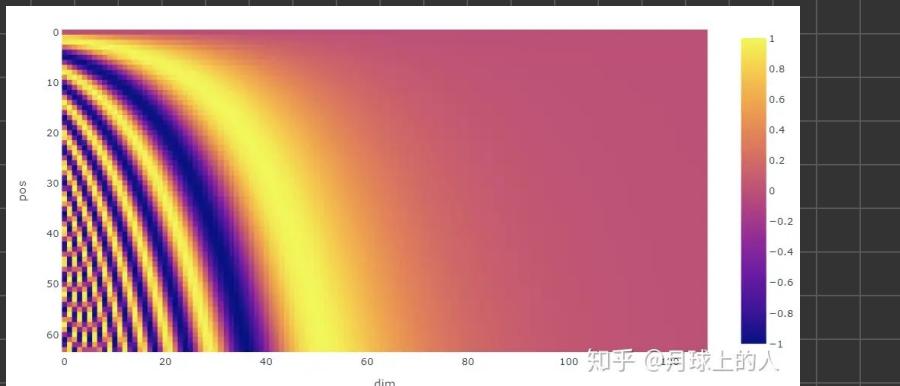
Angel1 dim

{ 固定 dim | (类比2进制) 变化很快
dim 变大 | (趋于0) why (H. I. X.)

Angel2 pos

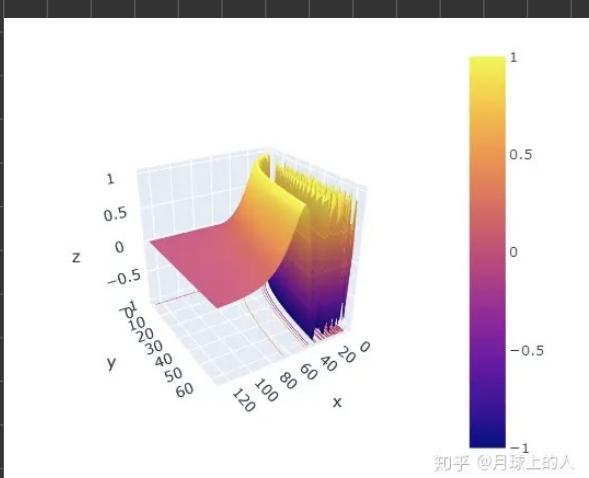
{ 固定 pos
pos 变大

观察 dim



128维 2D 示意图

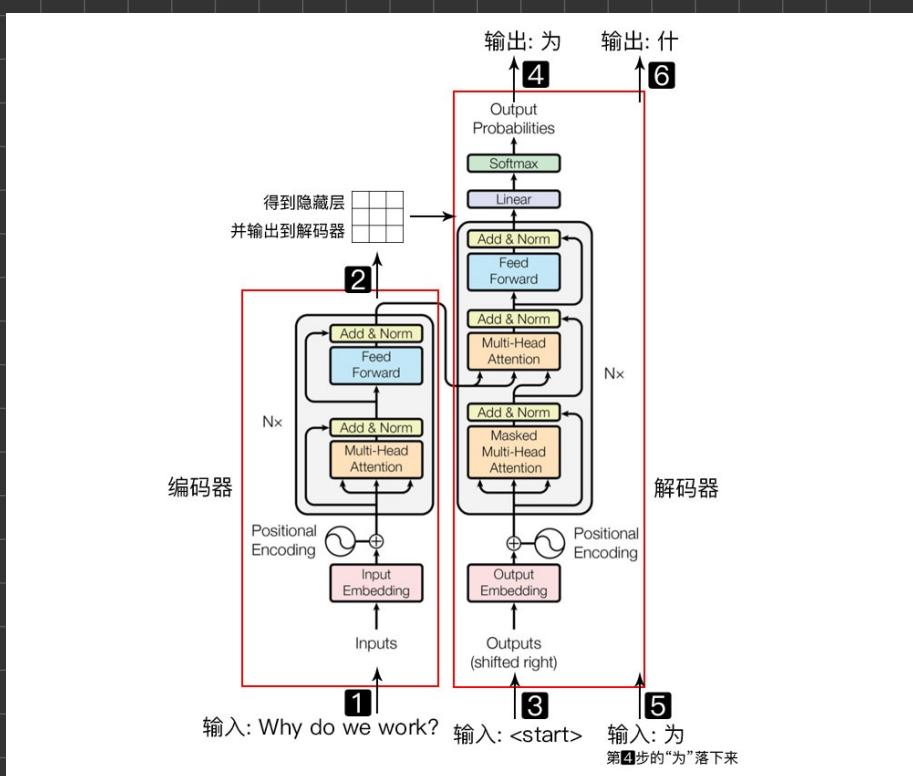
一文教你彻底理解Transformer中Positional Encoding
- 月球上的人的文章 - 知乎
<https://zhuanlan.zhihu.com/p/338592312>



128维 3D 示意图

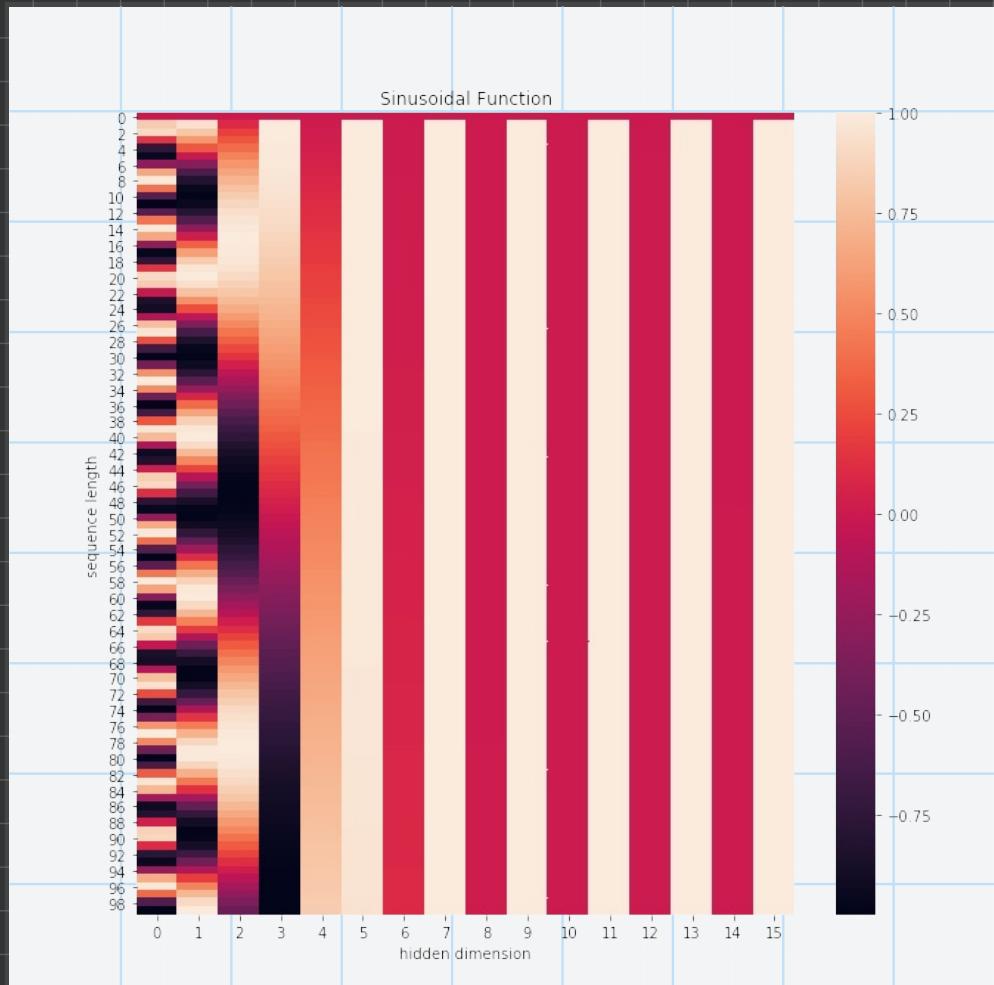
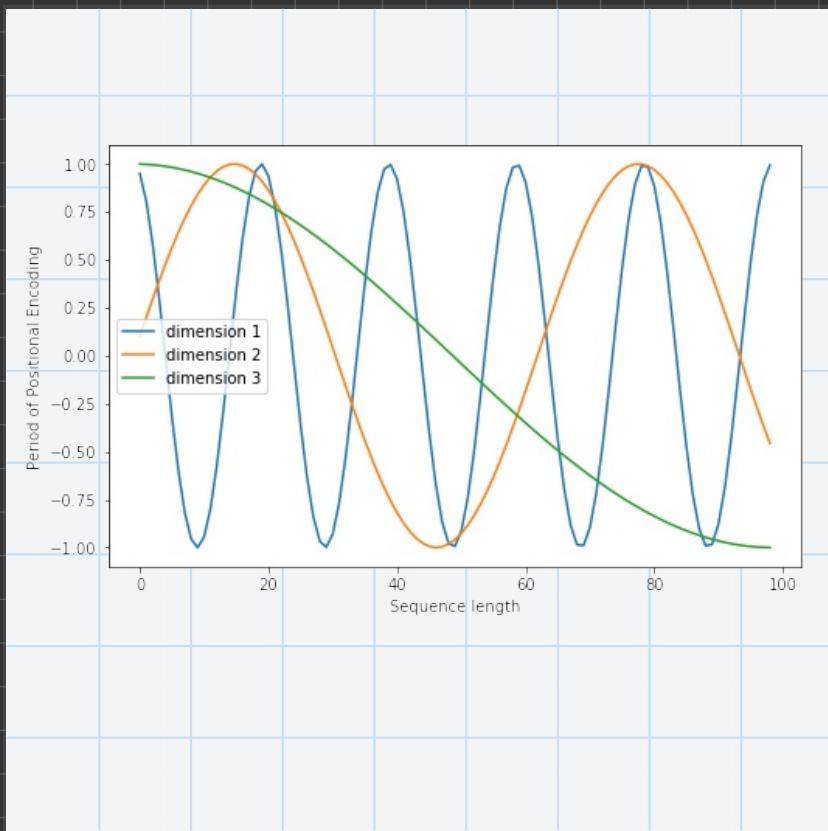
$\left\{ \begin{array}{l} pos=20-\text{条曲线} \\ 0\text{维}-\text{点} \end{array} \right.$

学习框架



- ✓ positional encoding
- ☐ self Attention mechanism
- ☐ 残差连接 & LayerNorm
- ☐ Encoder ☐ decoder

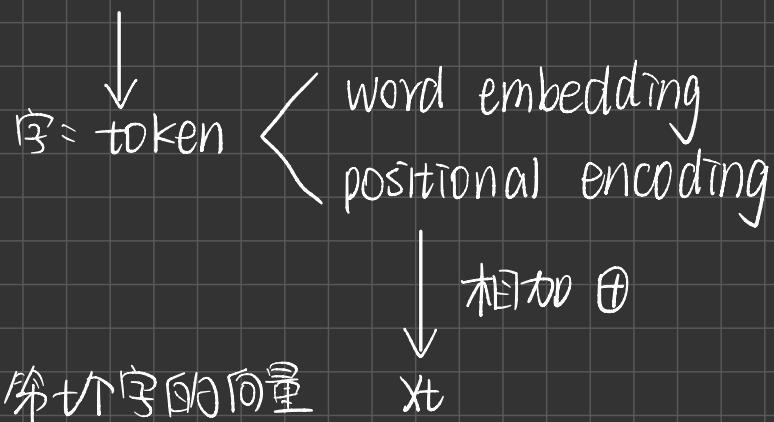
位置编码 纵向观察 Embedding-dimension ↑
周期变化平缓



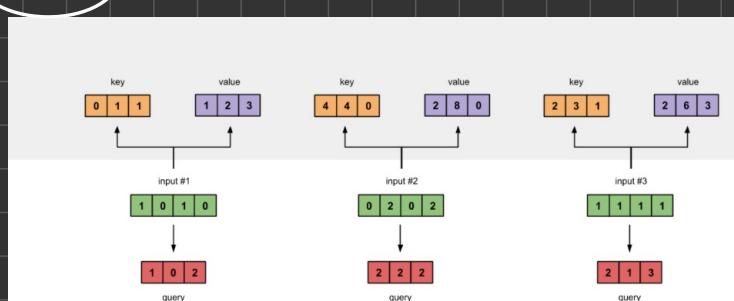
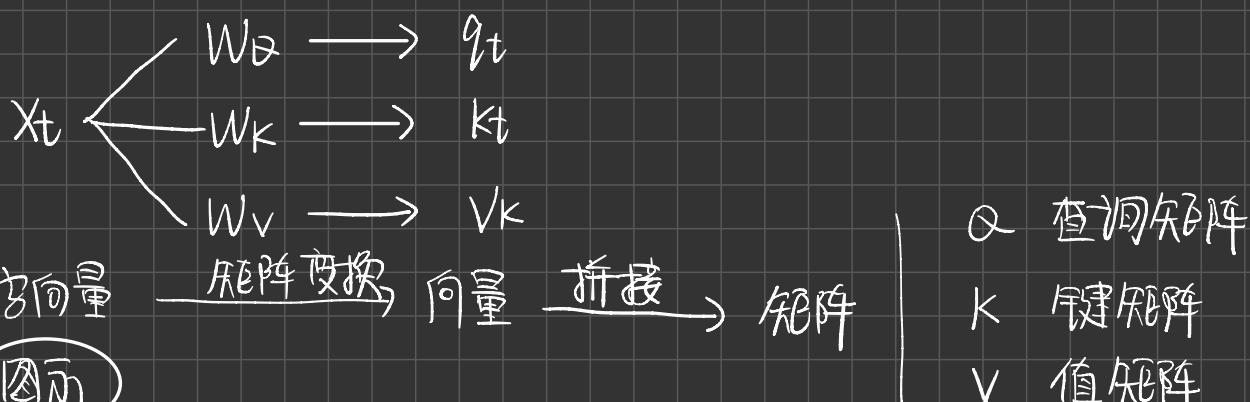
Part 2: self - attention 回答什么是self - attention (eg)

(1) 输入 token x_t

<https://wmathor.com/index.php/archives/1438/>



(2) x_t 由 3 个矩阵作用 得到 3 个向量

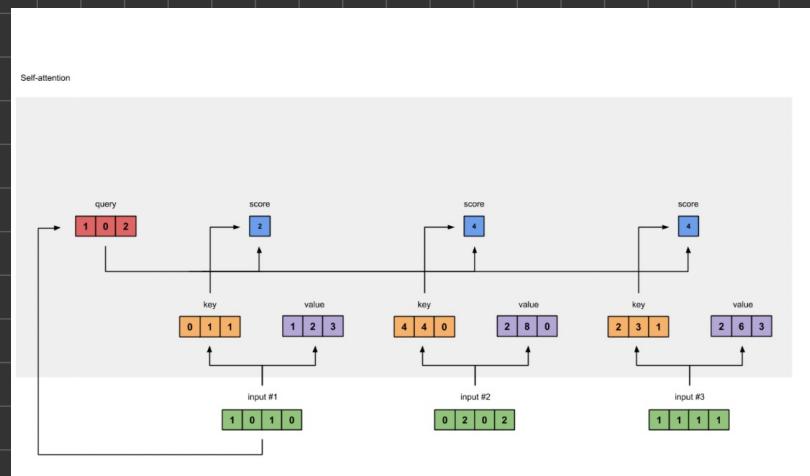


计算注意力权重

向量 $q_1 \times$ 键矩阵 K

$$K = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

转置



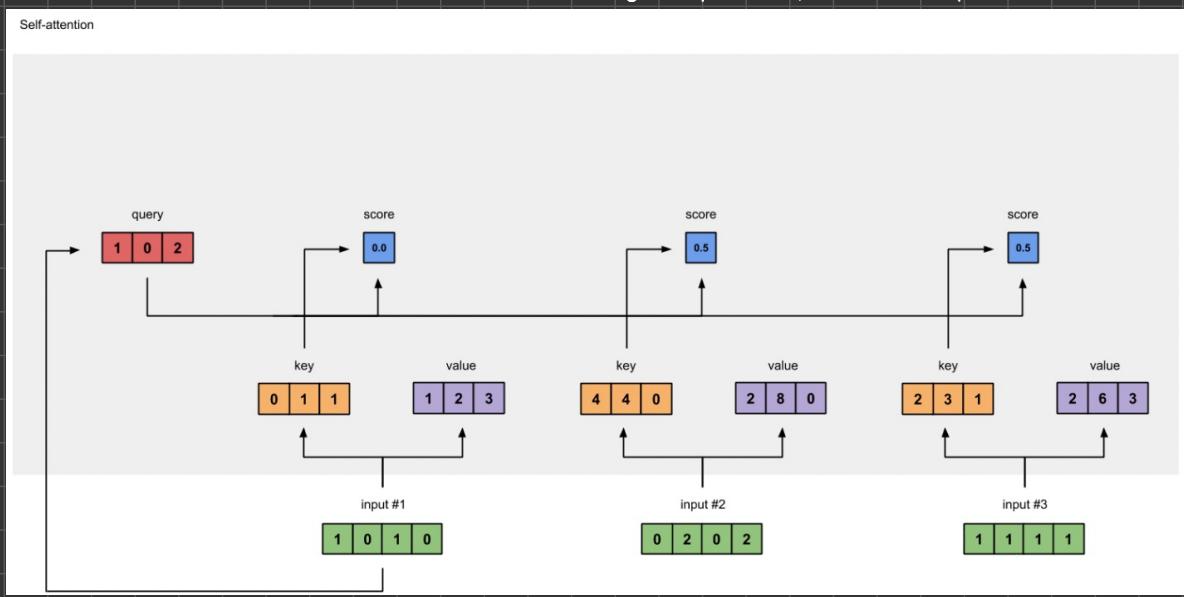
$$[1, 0, 2] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [2, 4, 4]$$

$$q_1 \times K^T$$

softmax 和为 1

$$\text{softmax}(2, 4, 4) = \left[\frac{e^2}{e^2 + e^4 + e^4}, \frac{e^4}{e^2 + e^4 + e^4}, \frac{e^4}{e^2 + e^4 + e^4} \right]$$

$$= [0.0634, 0.4683, 0.4683]$$

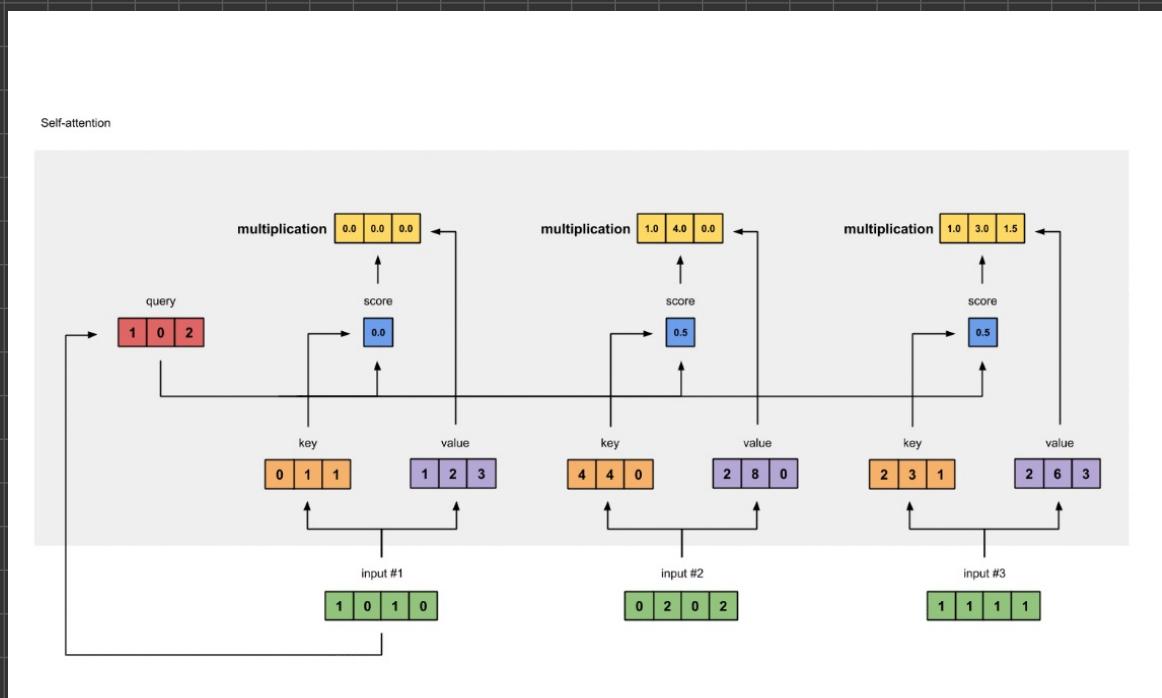


权重 \times 对应字的值向量 v_t

$$0.0 \times [1, 2, 3] = [0, 0, 0]$$

$$0.5 \times [2, 8, 0] = [1.4, 0, 0]$$

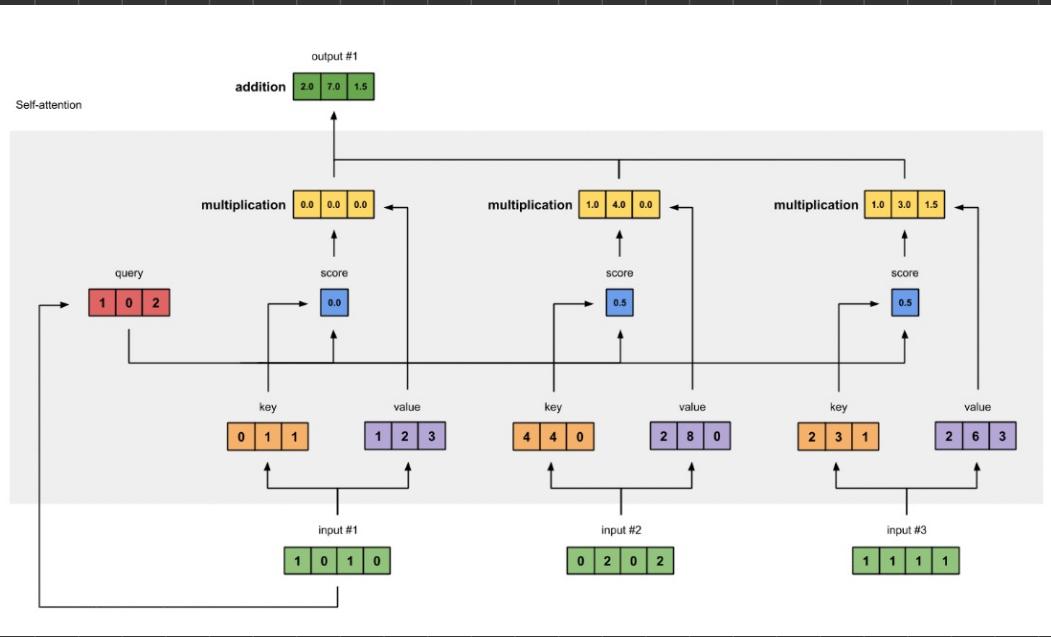
$$0.5 \times [2, 6, 3] = [1.3, 1.5, 0]$$



OUTPUT #1

权重大化后的值向量求和 得到第 1 个字的输出

$$\begin{aligned} & [0, 0, 0] + [1, 4, 0] + [1, 3, 1.5] \\ & = [2, 7, 1.5] \end{aligned}$$



OUTPUT #2, OUTPUT #3

对所有输入向量执行相同的操作，得到 self-attention
所有输入

step1：第1个字的查询向量 $q_1 \times$ 键矩阵 K

$$q_1 K^T$$

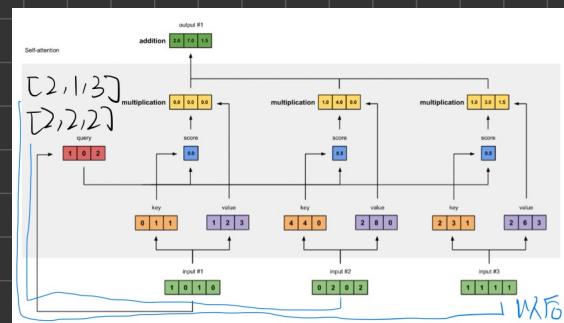
权重 \downarrow $\text{softmax}(q_1, K^T)$

step2: 权重 $\times V_t$

\downarrow
step3: $\text{Output\#1} = \sum (\text{权重} \times V_t)$

$$\sum (\text{softmax}(q_1, K^T) \times V_t)$$

$$t=1, 2, 3$$



具体来说

- ① Input: $[1, 0, 1, 0]$ Input #1
 $[0, 2, 0, 2]$ Input #2
 $[1, 1, 1, 1]$ Input #3

② deal1 = train 有

- $[0, 1, 1]$ key#1 $[1, 2, 3]$ value#1
 $[4, 4, 0]$ key#2 $[2, 8, 0]$ value#2
 $[2, 3, 1]$ key#3 $[2, 6, 3]$ value#3

- $[1, 0, 2]$ query#1
 $[2, 2, 2]$ query#2
 $[2, 1, 3]$ query#3

self attention

input = # key = # query = # value

那？普通的呢？

deal2:

Output_i = sum (softmax($q_i K^T$) $\times V_i$)

$$q_2 \quad [2, 2, 2] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [, ,]$$

$$\text{softmax} (, ,) = \frac{a_1 \ a_2 \ a_3}{\text{矩阵表示}(不写)}$$

$$a_1 * V_1 = a_1 * [1, 2, 3]$$
$$a_2 * V_2 = a_2 * [2, 8, 0]$$
$$a_3 * V_3 = a_3 * [2, 6, 3]$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} V^T$$
$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 2 & 8 & 0 \\ 3 & 0 & 3 \end{bmatrix}$$
$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix}$$

必须但没
必要

相加即得输出 done

③ Output

$$\text{Output} \#1 = [2, 7, 1.5]$$

$$\text{Output} \#2 = [2, 8, 0]$$

$$\uparrow \quad \text{Output} \#3 = [2, 7.8, 0.3]$$

self-attention 是什么? def
eg ✓

?

矩阵计算

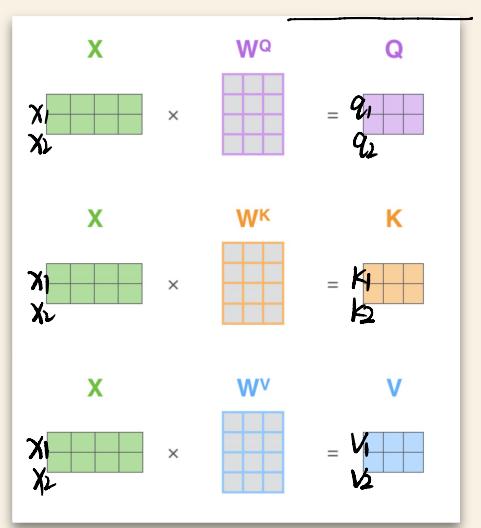
向量计算变成矩阵的形式

- VR 计算出所有时刻的输出

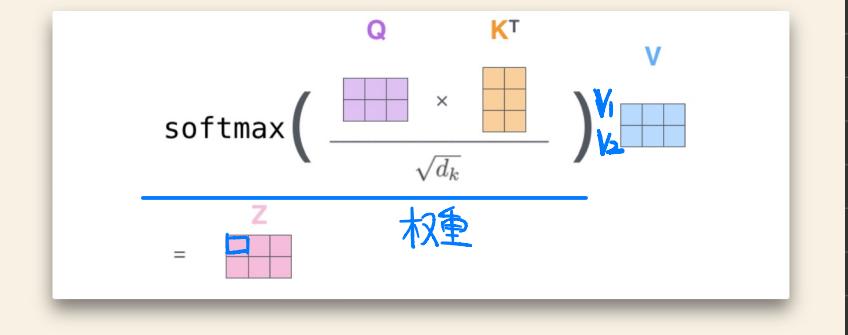
图示: ✓ Step1

Step2

第一步就不是计算某个时刻的 q_t, k_t, v_t 了，而是一次计算所有时刻的 Q, K 和 V 。计算过程如下图所示，这里的输入是一个矩阵 X ，矩阵第 t 行为第 t 个词的向量表示 x_t



接下来将 Q 和 K^T 相乘，然后除以 $\sqrt{d_k}$ (这是论文中提到的一个 trick)，经过 softmax 以后再乘以 V 得到输出



数 * 向量 = 向量

$$a_1 * v_1 = \underline{\hspace{2cm}}$$

$$a_2 * v_2 = \underline{\hspace{2cm}}$$

$$a_3 * v_3 = \underline{\hspace{2cm}}$$

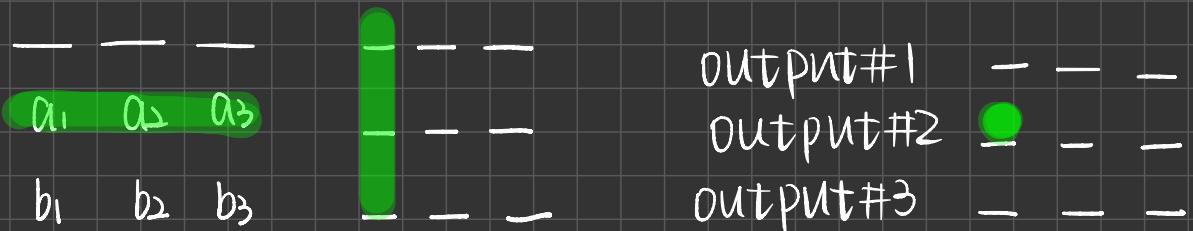
$$= \text{output} \#2$$

$$b_1 * v_1 = \underline{\hspace{2cm}}$$

$$b_2 * v_2 = \underline{\hspace{2cm}}$$

$$b_3 * v_3 = \underline{\hspace{2cm}}$$

$$= \text{output} \#3$$



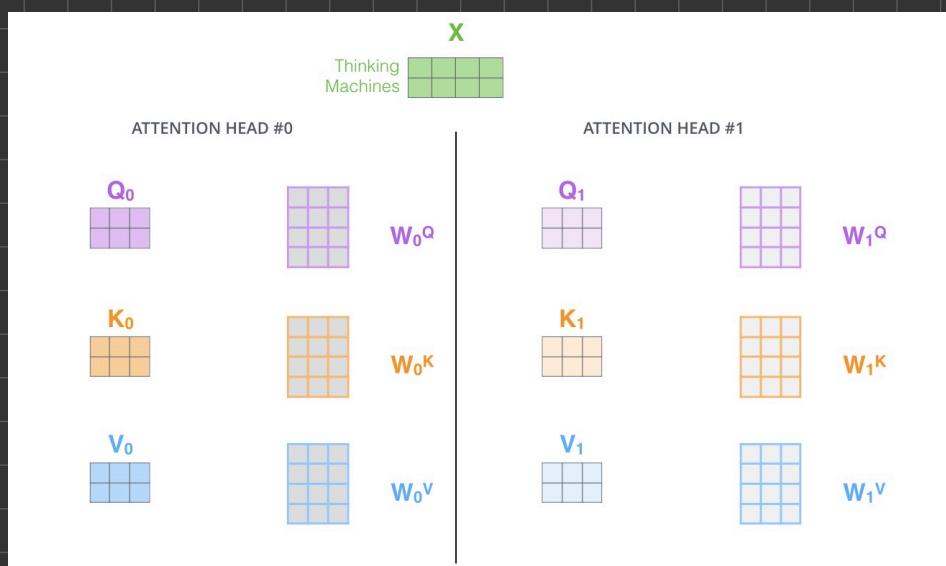
Output #2 的第1个元素怎么来的？

$\Leftrightarrow a_1, a_2, a_3$ 分别乘以 v_1, v_2, v_3 的第1个元素再相加

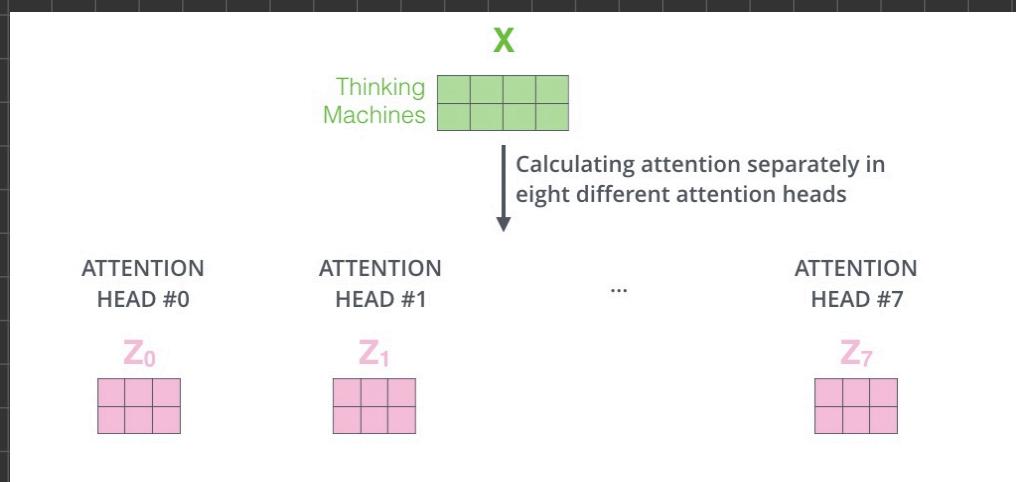
part3 Multi-Head Attention

Idea: 定义多组 Q, K, V 分别关注不同的上下文

线性变换从一组 (W^Q, W^K, W^V) 多组 W_0^Q, W_0^K, W_0^V , W_1^Q, W_1^K, W_1^V

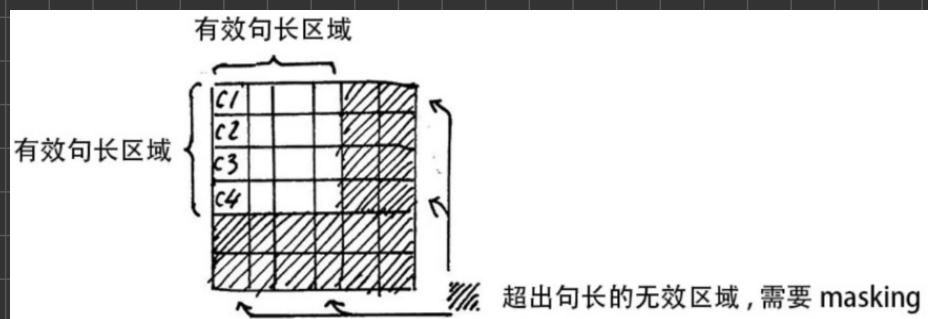


对于输入矩阵 X 每一组 Q, K, V 得到一个输出矩阵 Z



padding mask

(问：为什么介绍？是什么？有什么用？)



问题在于 batch

实际应用中 Input $X.shape = [batch_size, sequence_length]$

batch_size：一个 batch 由多个不等长的句子组成

sequence_length：句长

⇒ 目的：1. VR 计算多句话

(1) padding mask

(2) Why? Jdk

X :

batch size	-----

sequence_length

按最长句子对余下句子补齐 (补0)

★ 又有问题 softmax 取指数 0 有值的

∴ 做 1 个 mask (self-attention)

让无效区域不参与运算

★ mask ?

为了让无效区域不参与运算

⇒ 无效区域 + 负偏置，即

$$z_{\text{illegal}} = z_{\text{illegal}} + \text{bias}_{\text{illegal}}$$

$$\text{bias}_{\text{illegal}} \rightarrow -\infty$$

$$e^{-\infty} = 0$$

Add softmax 取极

$$D(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Part 4. 残差连接 & Layer Normalization

✓ 残差连接是什么? Xembedding + Self-Attention (Q, K, V)

□ Layer Normalization

def (什么是 Layer Normalization)

作用: 网络中的隐藏层归一化为标准正态分布

for: 加快训练, 加速收敛

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_{ij}$$

(矩阵列求和求均值)

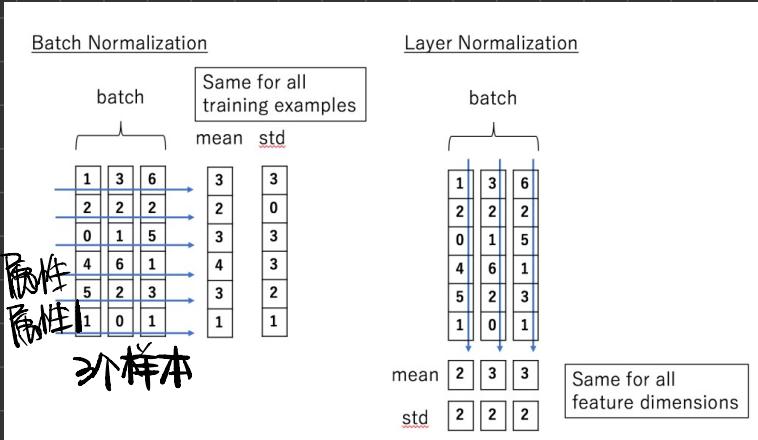
$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_{ij} - \mu_j)^2$$

(矩阵列求和求方差)

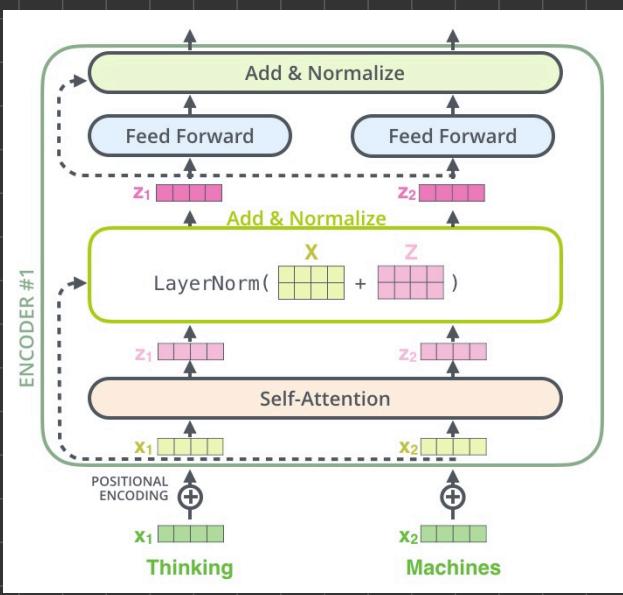
$$\text{则 } \text{Layer Norm}(x) = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

(每一列的每一个元素 减去列均值 / 列标准差)

(ϵ : 防止分母为0)



□ Encoder #1



$x_1 \ x_2 \xrightarrow{\text{self-attention}} z_1, z_2$
和输入 x_1, x_2 残差连接
↓
经过 LayerNorm 输出经全连接层 } LayerNorm
残差连接
↓
输出给下一个 Encoder

Q: [什么叫 Encoder Block 中的 Feed Forward 权权重共享?]
(Why?)

(为什么 LayerNorm 正规化加速了训练)

□ Transformer Encoder 整体结构

初始化过程

(1) 向量 & 位置编码

$$X = \text{Embedding-Lookup}(x) + \text{Positional-Encoding}$$

(2) 自注意力机制

$$Q = \text{Linear}_q(X) = X \cdot W_Q$$

$$K = \text{Linear}_k(X) = X \cdot W_K$$

$$V = \text{Linear}_v(X) = X \cdot W_V$$

$$X_{\text{Attention}} = \text{self-Attention}(Q, K, V)$$

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

(3)

self-attention 残差连接 & LayerNormalization

$$X_{\text{attention}} = X + X_{\text{attention}}$$

$$X_{\text{attention}} = \text{LayerNorm}(X_{\text{attention}})$$

⑭) FeedForward

两层线性映射 & 激活函数

$$X_{\text{hidden}} = \text{Linear}(\text{ReLU}(\text{Linear}(X_{\text{attention}})))$$

⑮) FeedForward 残差连接 & LayerNormalization

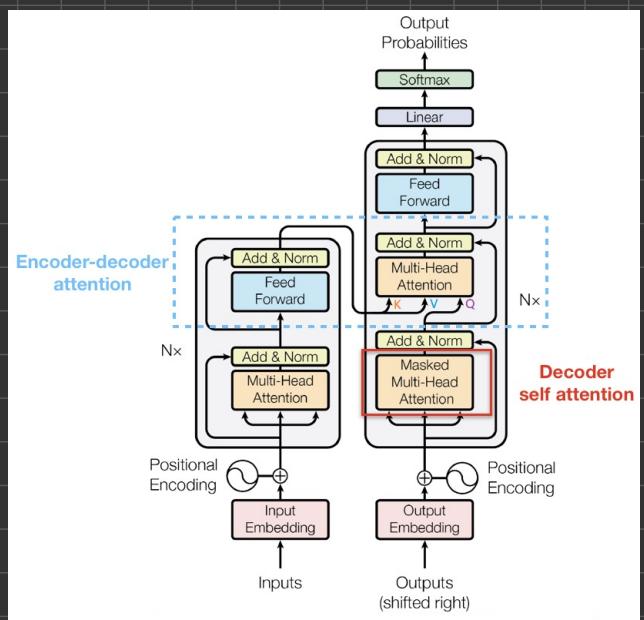
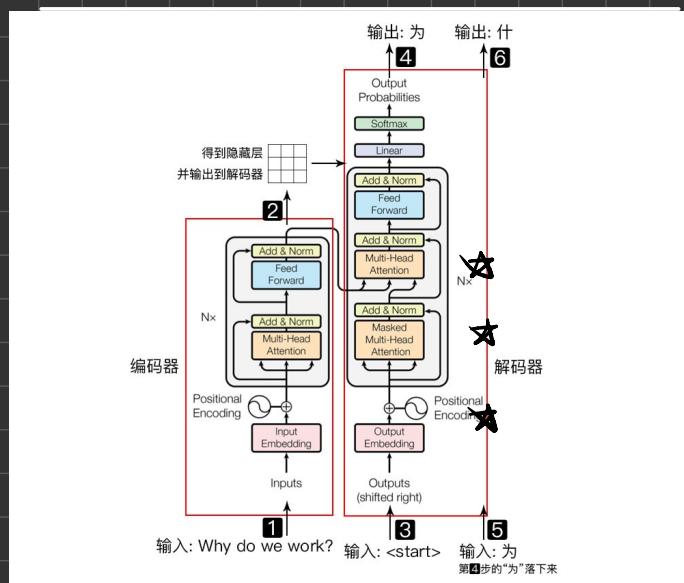
$$X_{\text{hidden}} = X_{\text{attention}} + X_{\text{hidden}}$$

$$X_{\text{hidden}} = \text{LayerNorm}(X_{\text{hidden}})$$

其中 $X_{\text{hidden}} \in \mathbb{R}^{\text{batch_size} * \text{seq_len} * \text{embed_dim}}$

□ Transformer Decoder 整体结构

只说了 \uparrow encoder, 就没了 \downarrow



decoder 结构

- masked multi-head self-Attention
- Multi-Head Encoder-Decoder Attention
- FeedForward Network

每一个部分都有残差连接 & Layer Normalization

Masked Self-Attention

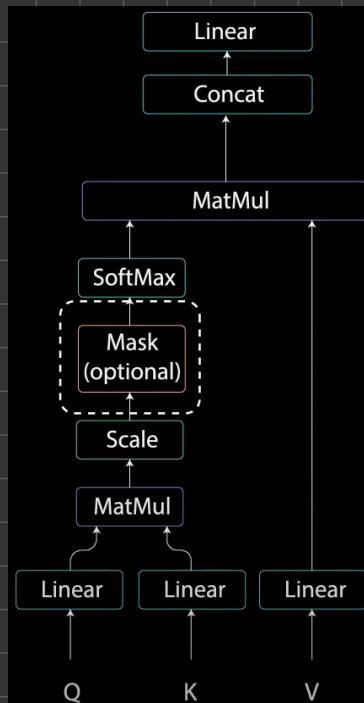
原因：训练过程是并行的

ground truth 全暴露在 Decoder 中

i) MASK

具体什么意思？ eg.

[所以 Transformer 没有监督训练]



举个例子，Decoder 的 ground truth 为 "<start> I am fine"，我们将这个句子输入到 Decoder 中，经过 WordEmbedding 和 Positional Encoding 之后，将得到的矩阵做三次线性变换 (W_Q, W_K, W_V)。然后进行 self-attention 操作，首先通过 $\frac{Q \times K^T}{\sqrt{d_k}}$ 得到 Scaled Scores，接下来非常关键，我们要对 Scaled Scores 进行 Mask，举个例子，当我们输入 "I" 时，模型目前仅知道包括 "I" 在内之前所有字的信息，即 "<start>" 和 "I" 的信息，不应该让其知道 "I" 之后词的信息。道理很简单，我们做预测的时候是按照顺序一个字一个字的预测，怎么能这个字都没预测完，就已经知道后面字的信息了呢？Mask 非常简单，首先生成一个下三角全 0，上三角全为负无穷的矩阵，然后将其与 Scaled Scores 相加即可

□ scale w/o mask softmax 之后 = 0 再与 V 相乘 不会有影响

Scaled Scores				
<start>		I	am	fine
<start>	0.7	0.1	0.1	0.1
I	0.1	0.6	0.2	0.1
am	0.1	0.3	0.6	0.1
fine	0.1	0.3	0.3	0.3

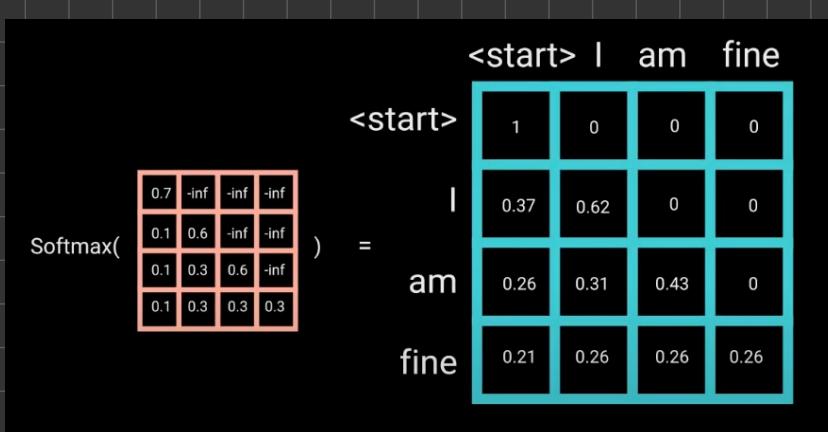
+

Look-Ahead Mask				
<start>		I	am	fine
<start>	0	-inf	-inf	-inf
I	0	0	-inf	-inf
am	0	0	0	-inf
fine	0	0	0	0

=

Masked Scores				
<start>		I	am	fine
<start>	0.7	-inf	-inf	-inf
I	0.1	0.6	-inf	-inf
am	0.1	0.3	0.6	-inf
fine	0.1	0.3	0.3	0.3

□ mask 都是 -inf → 0 矩阵即为每个字之间权重

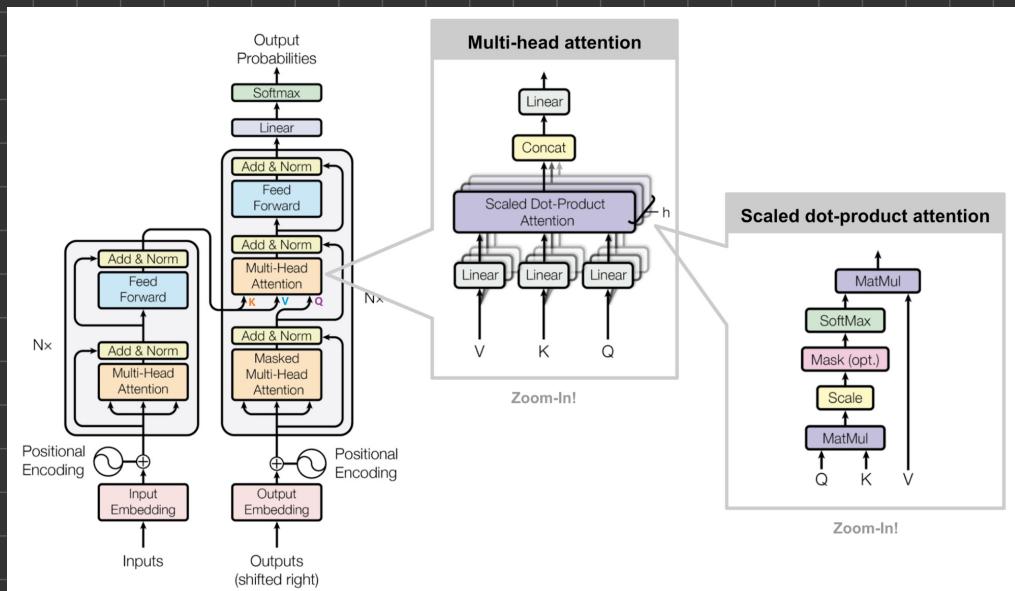


□ Masked Encoder-Decoder Attention

(1) why name Encoder-Decoder Attention

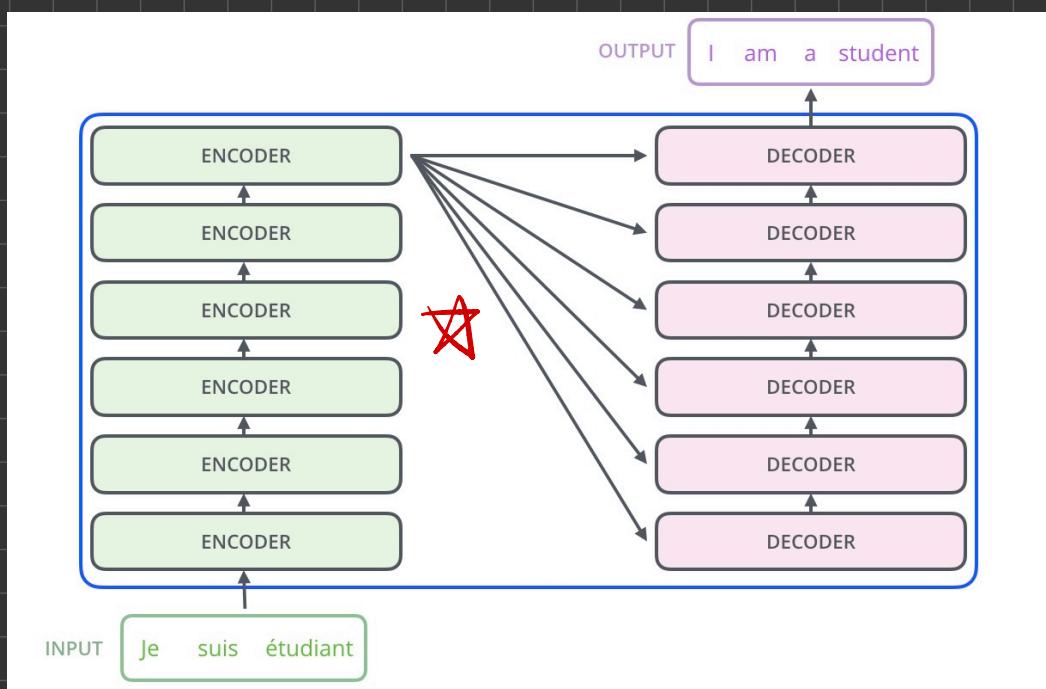
Ans: K, V 为 Encoder 编码出

Q 为 Decoder Masked Self-Attention 编码出



[Summary]

并行结构 LX 完整的



24.08.29