碎语

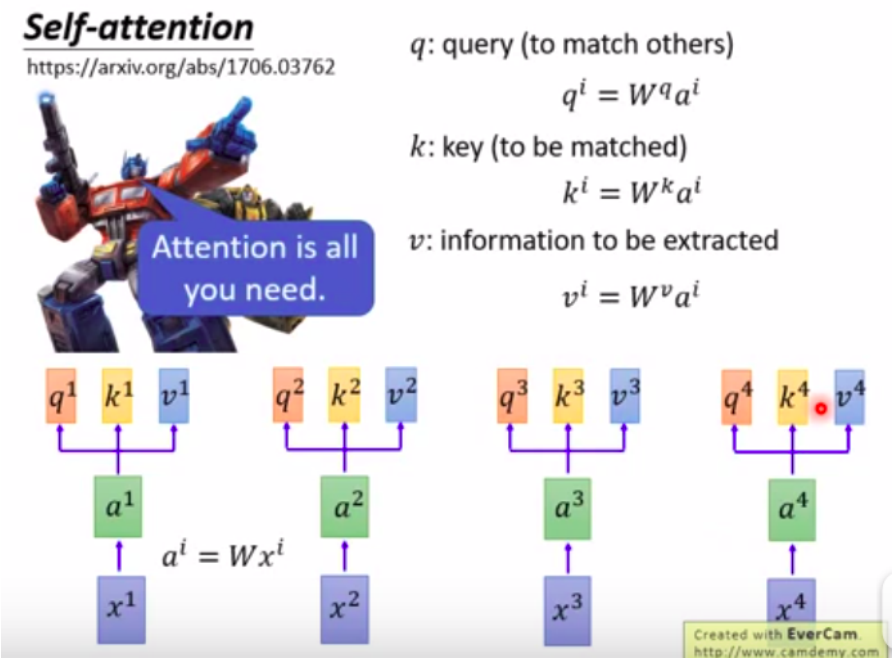
> 1. *本人先看了下面的视频*
>
> 2. *阅读了《Attention is all you need》发现文章内容和视频讲的基本一样，细节论文不清楚，。*
> 3. *所以，又看了一遍视频。*

视频笔记

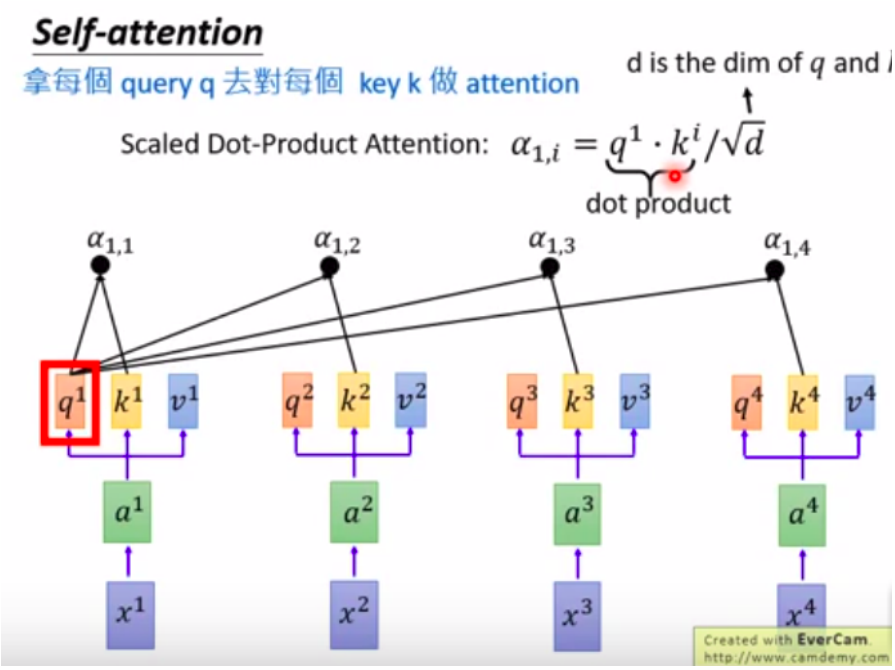
> *https://www.bilibili.com/video/av56239558?from=search&seid=4166606460205828626*

self-attention

> *得到：q,k,v*

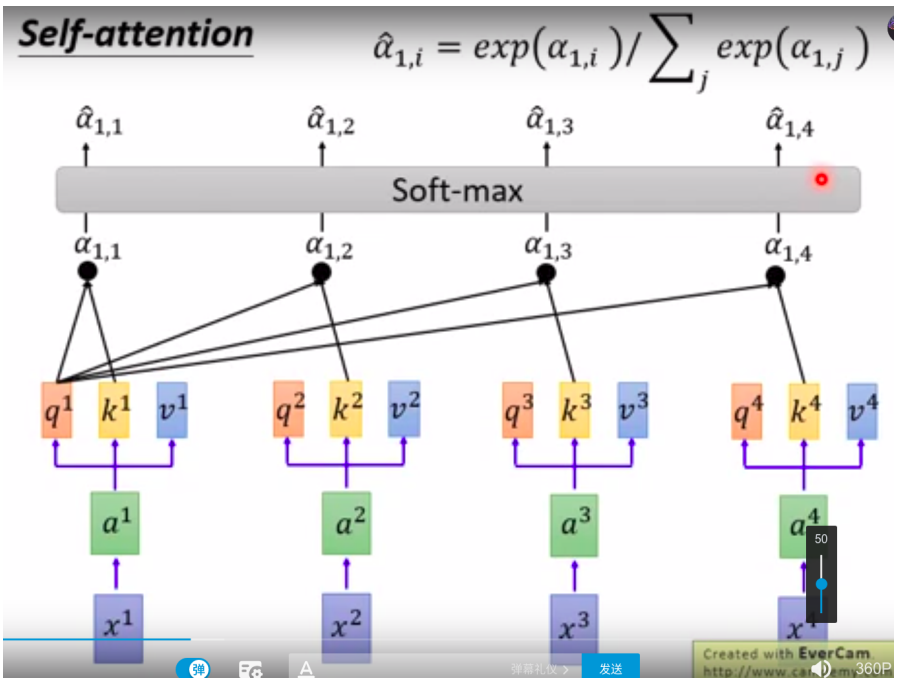


> *接下来，计算attention*

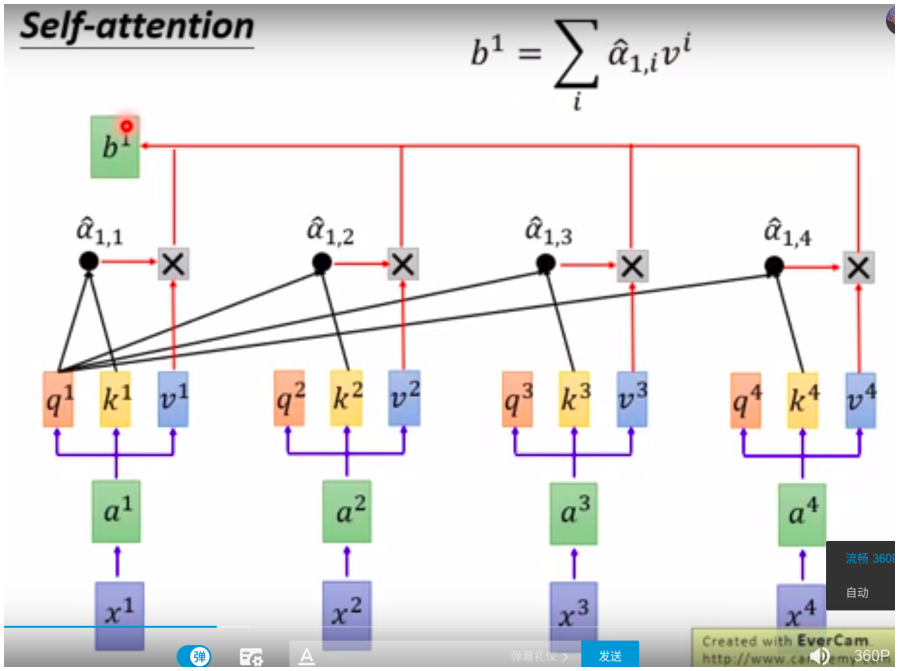


> *doct product: 点积 （attention is all you need文章中，用此方法计算相似性）。*
>
> *为什么除以根号d:因为点积是相乘求和，维度越大，相应数值越大。*

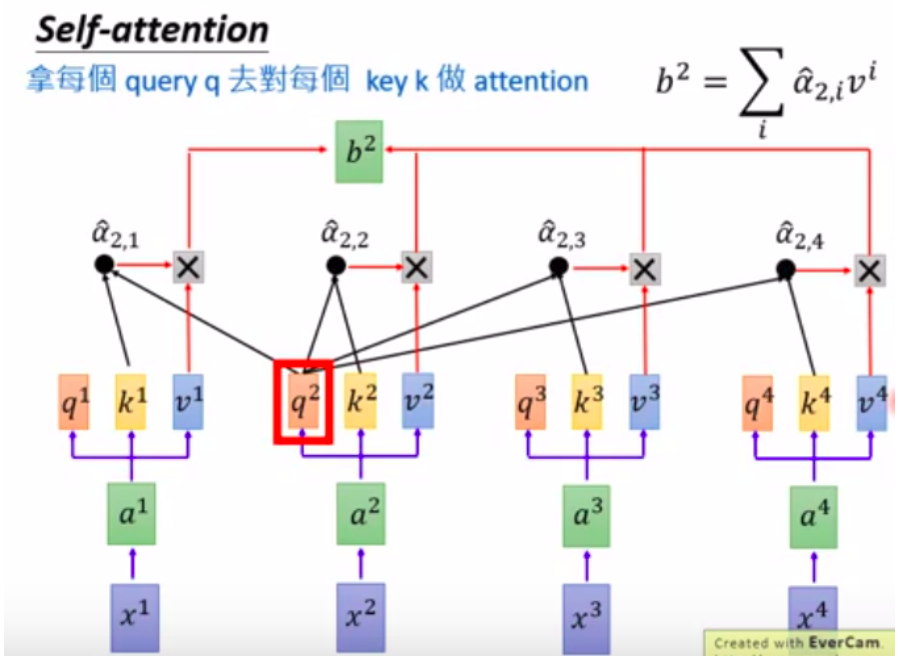
> *接下来，数值变概率（softmax）*



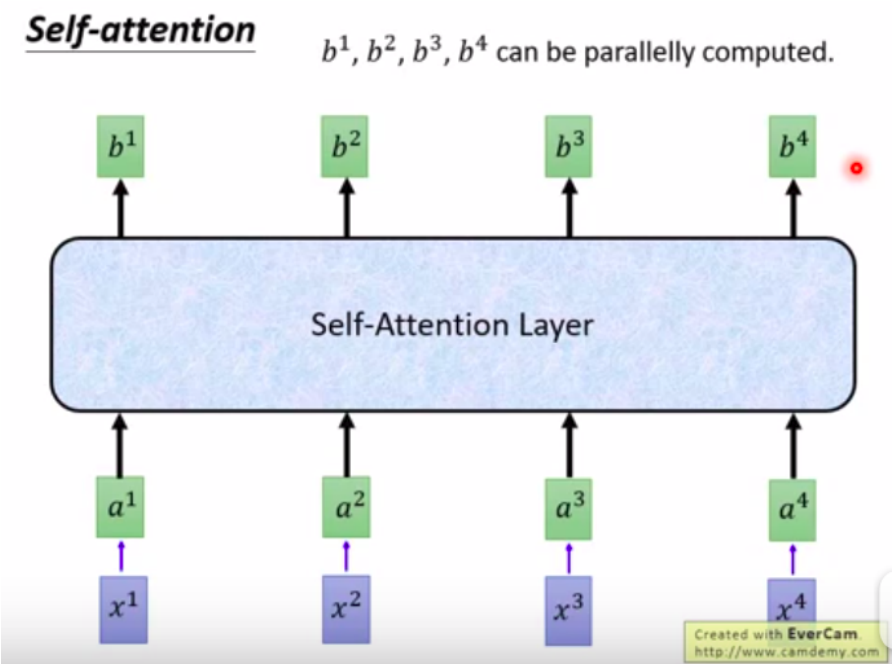
> *第一个输出：（这个输出，用了全部输出的信息）*


Self-attention
$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$
b^1
$\hat{\alpha}_{1,1}$ $\hat{\alpha}_{1,2}$ $\hat{\alpha}_{1,3}$ $\hat{\alpha}_{1,4}$
q^1 k^1 v^1 q^2 k^2 v^2 q^3 k^3 v^3 q^4 k^4 v^4
a^1 a^2 a^3 a^4
x^1 x^2 x^3 x^4
Created with EverCam
360P

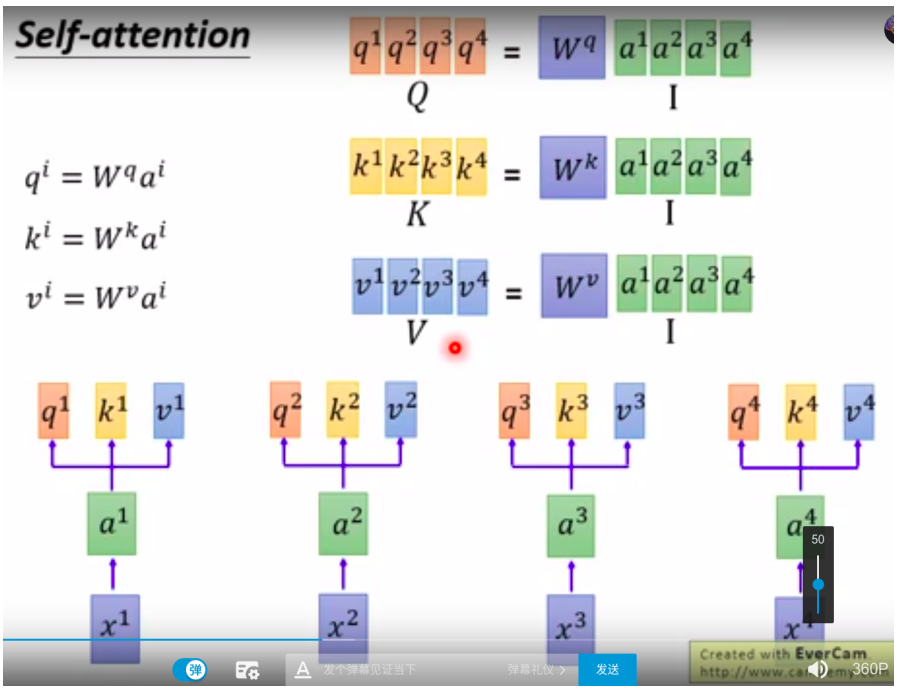
> *第二个输出：*


Self-attention
拿每個 query q 去對每個 key k 做 attention
$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$
b^2
$\hat{\alpha}_{2,1}$ $\hat{\alpha}_{2,2}$ $\hat{\alpha}_{2,3}$ $\hat{\alpha}_{2,4}$
q^1 k^1 v^1 q^2 k^2 v^2 q^3 k^3 v^3 q^4 k^4 v^4
a^1 a^2 a^3 a^4
x^1 x^2 x^3 x^4
Created with EverCam

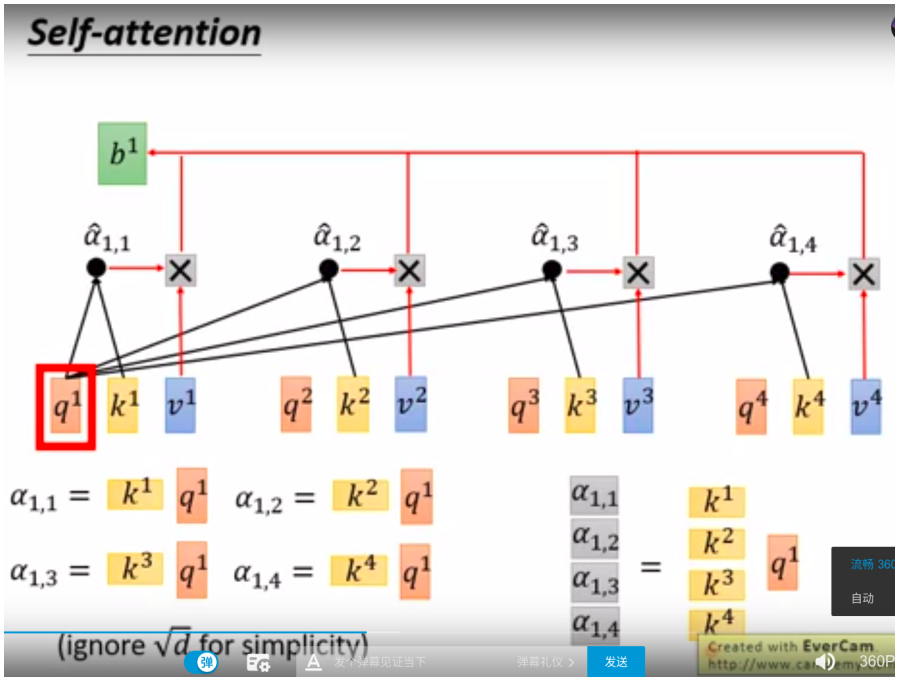
> *整体展示：*


Self-attention
b^1, b^2, b^3, b^4 can be parallelly computed.
b^1 b^2 b^3 b^4
Self-Attention Layer
a^1 a^2 a^3 a^4
x^1 x^2 x^3 x^4
Created with EverCam.
http://www.camdemy.com

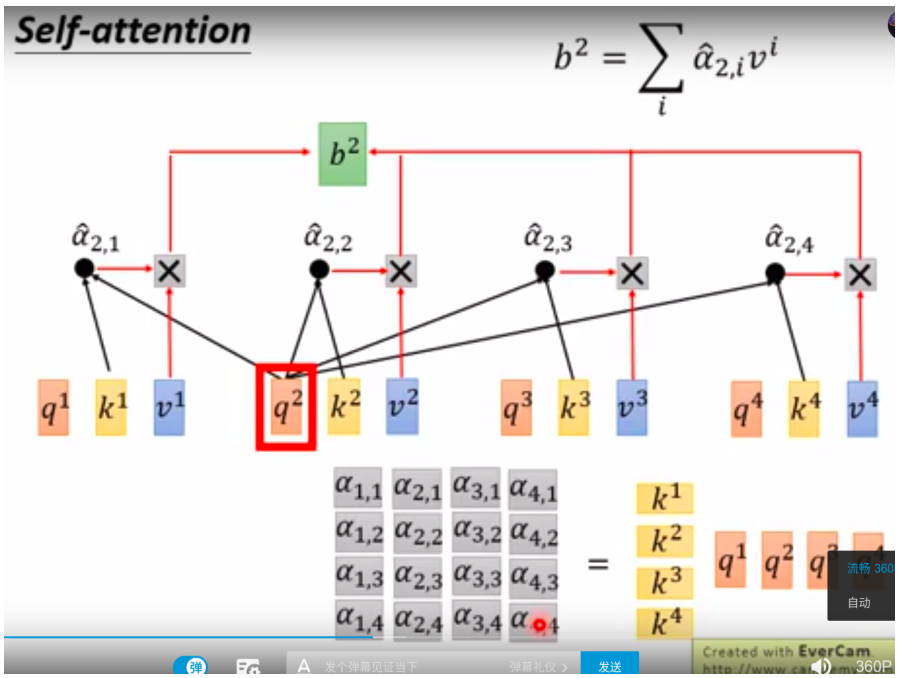
矩阵运算平行化


Self-attention
$q^i = W^q a^i$
$k^i = W^k a^i$
$v^i = W^v a^i$
$q^1 q^2 q^3 q^4 = W^q a^1 a^2 a^3 a^4$
Q I
$k^1 k^2 k^3 k^4 = W^k a^1 a^2 a^3 a^4$
K I
$v^1 v^2 v^3 v^4 = W^v a^1 a^2 a^3 a^4$
V I
q^1 k^1 v^1 q^2 k^2 v^2 q^3 k^3 v^3 q^4 k^4 v^4
a^1 a^2 a^3 a^4
x^1 x^2 x^3 x^4
50
发个弹幕见证当下
弹幕礼仪 >
发送
Created with EverCam.
http://www.camdemy.com
360P

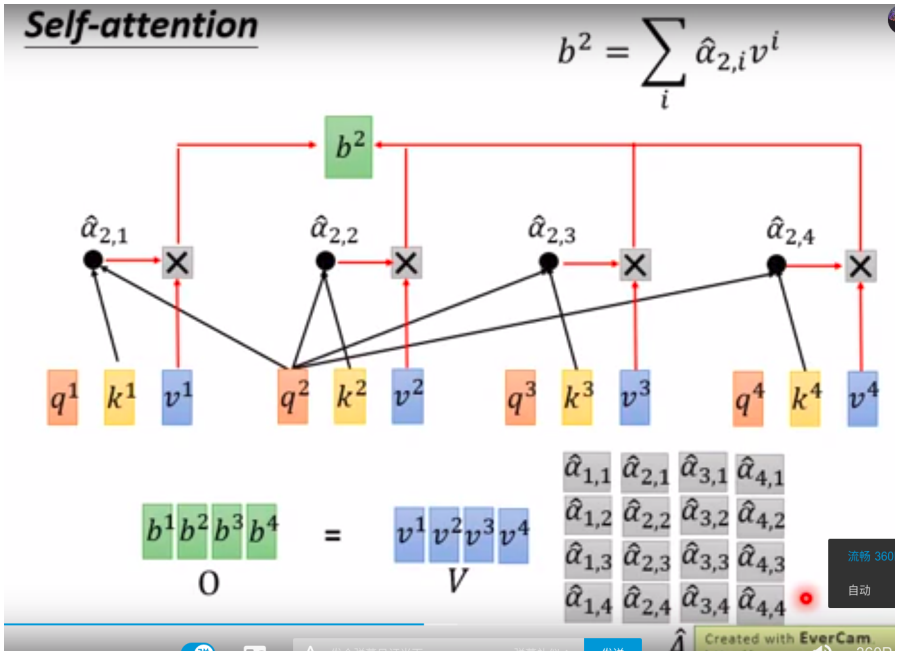
> 一个



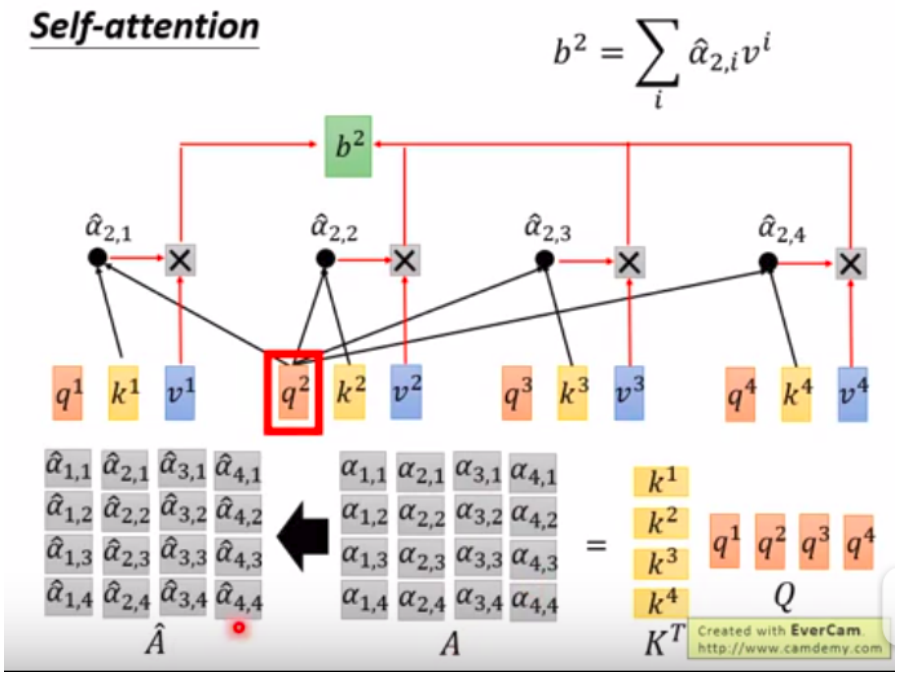
> 第二个



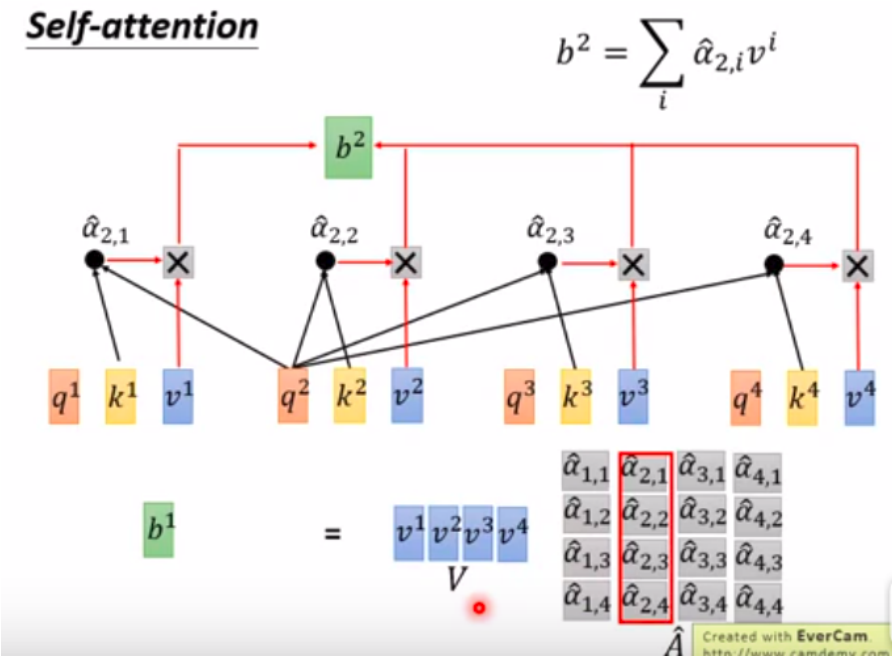
> 整个过程，可以转化为上面



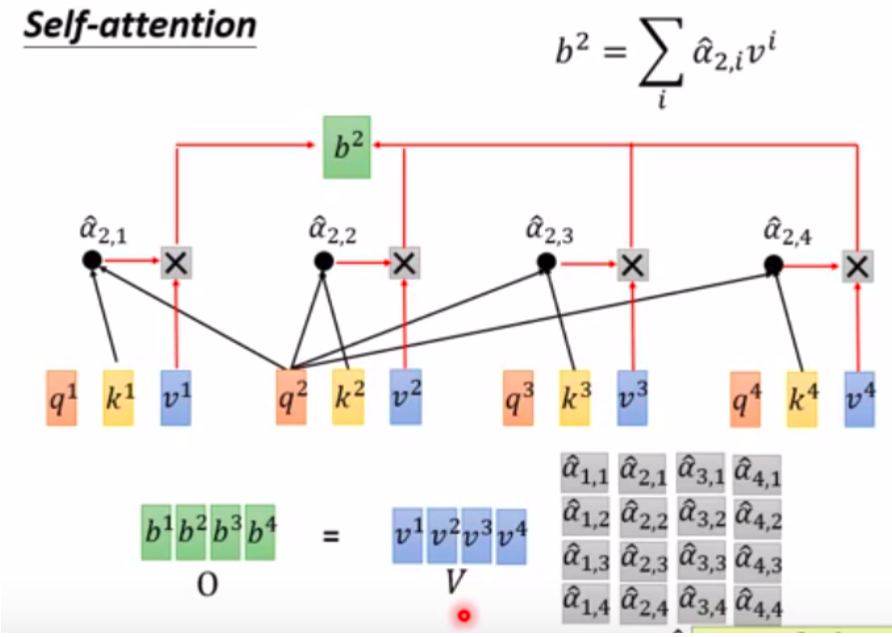
> *softmax*



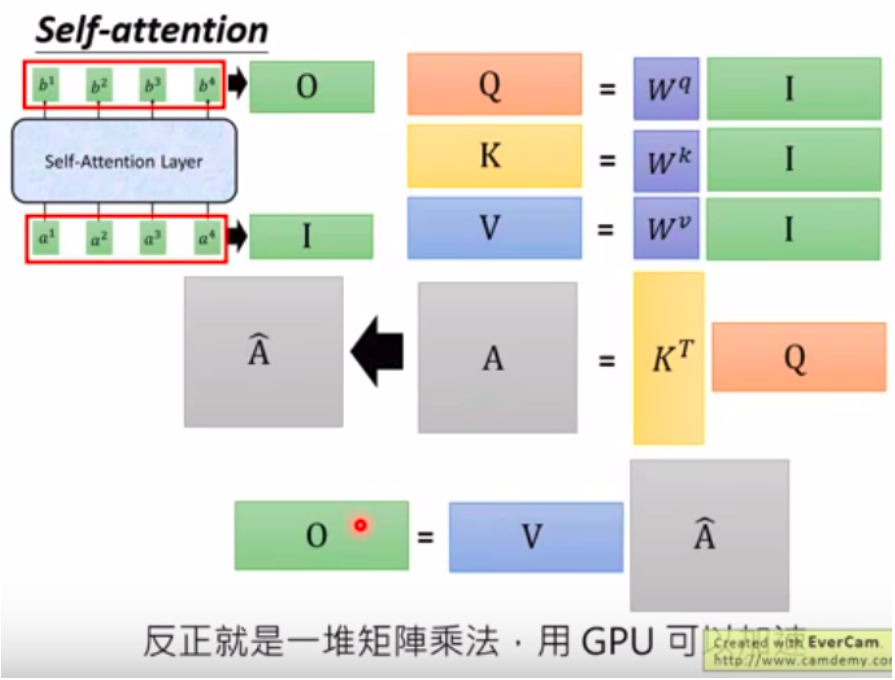
> *输出求解*



> *所有输出*

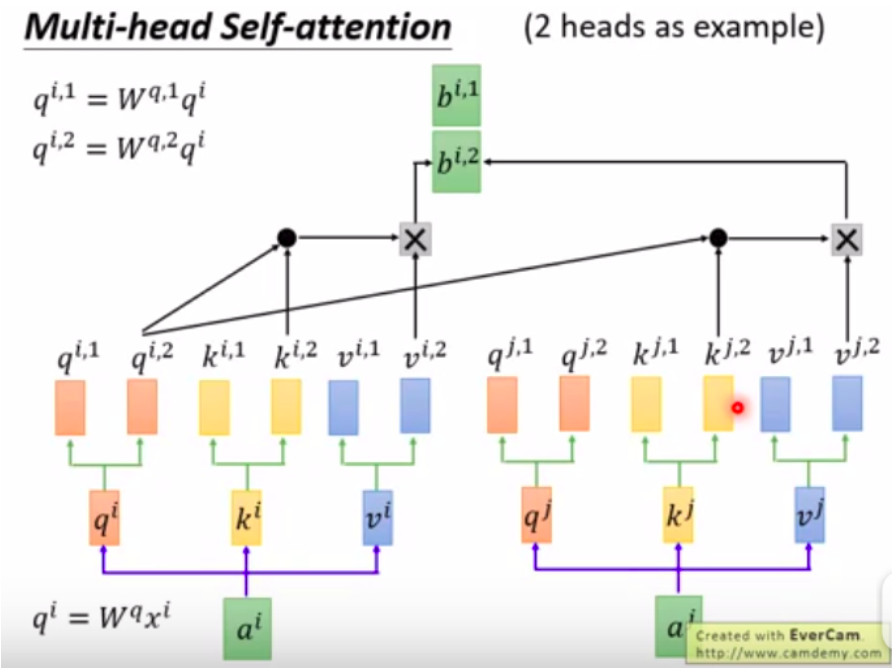


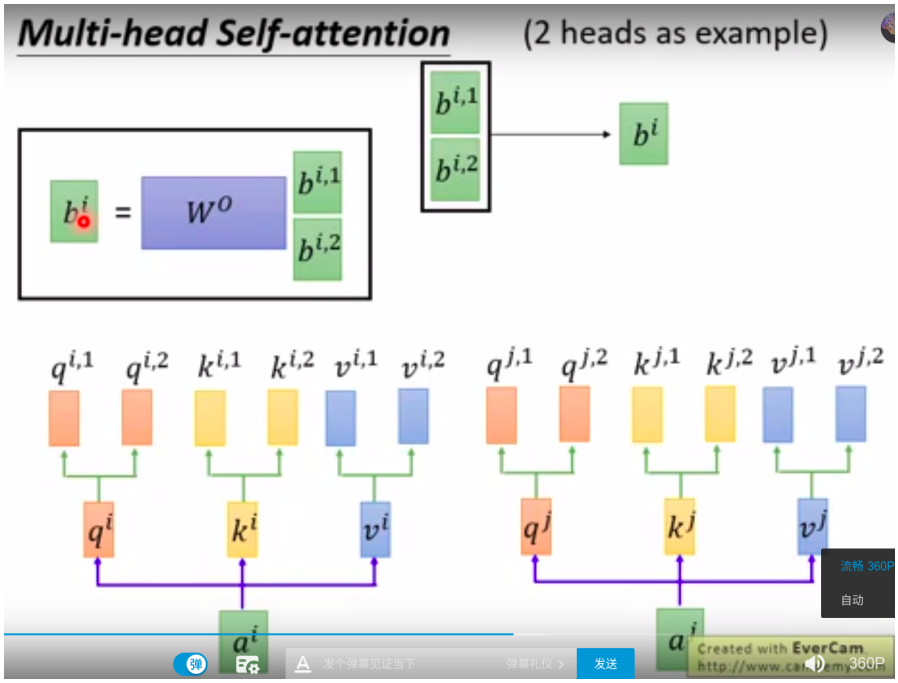
> *换个形式再看一遍*



> *重点：GPU加速矩阵乘法*

Multi-head self-attention

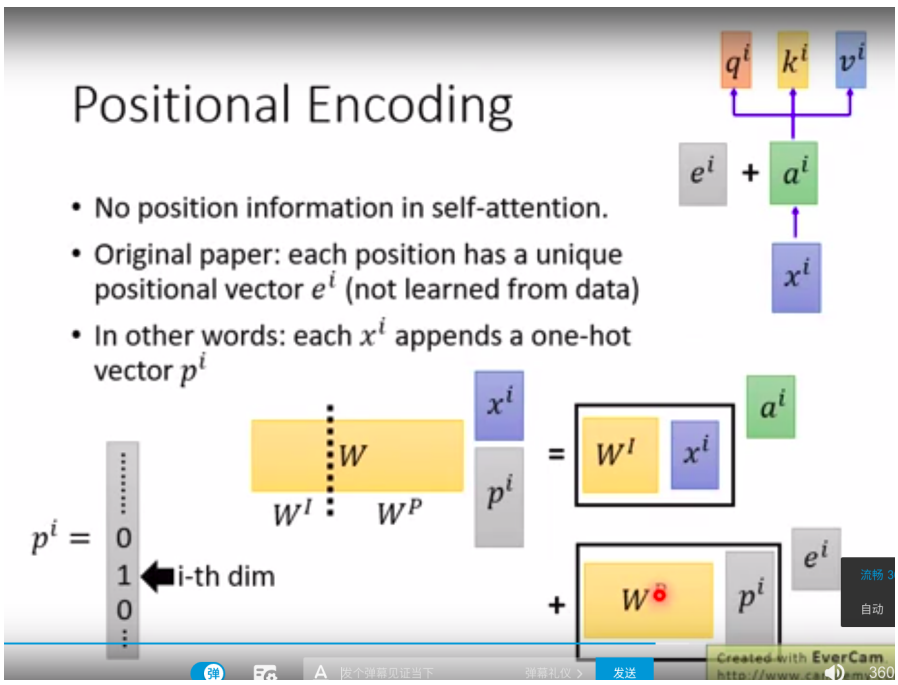




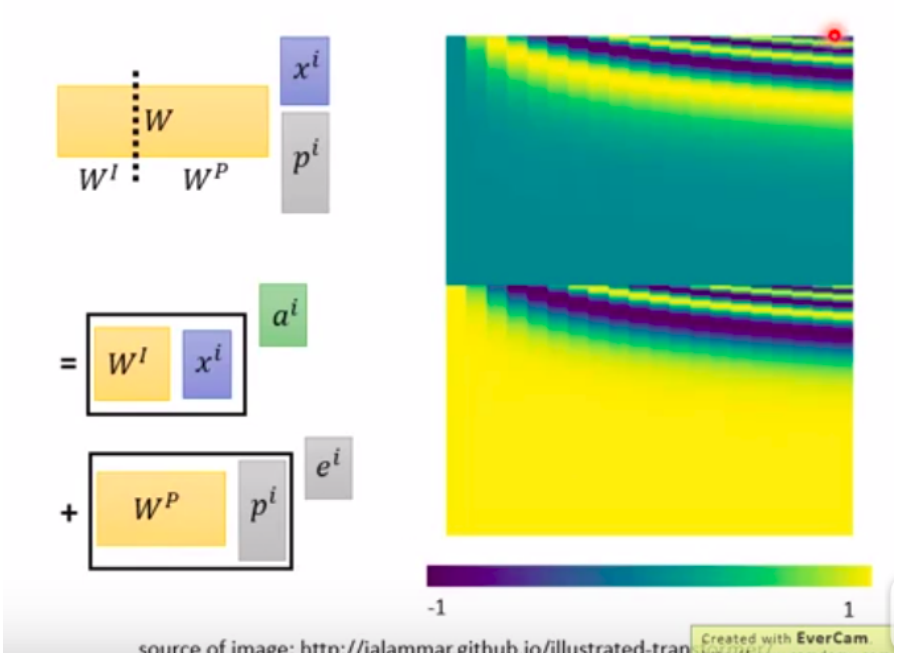
> *多头有什么好处?*
>
> *不同的头，关注可以不同，多个关注点。论文里说，避免了原来attention的平均化。*

Position Encoding

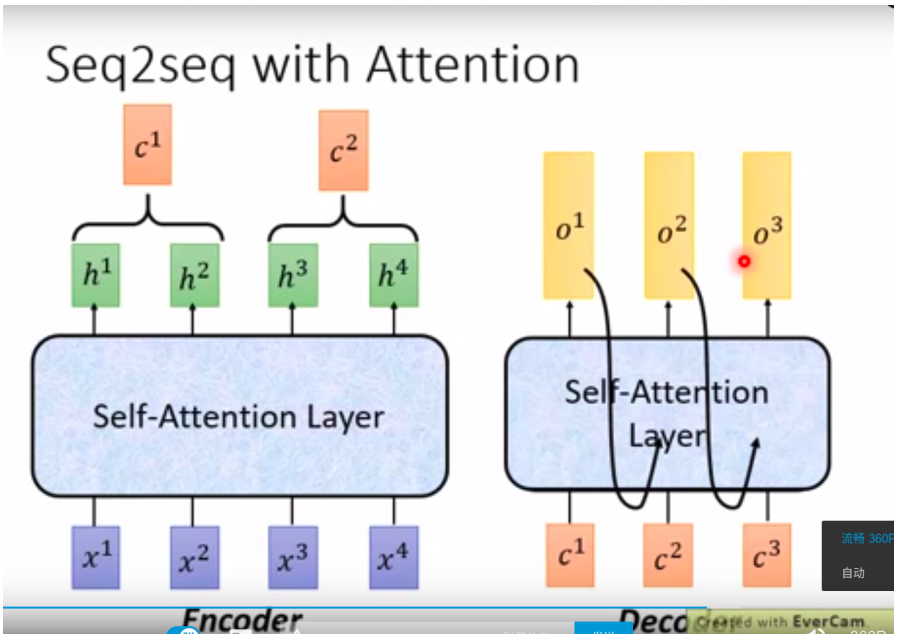
注意，原有self-attention对位置不敏感，没有位置信息



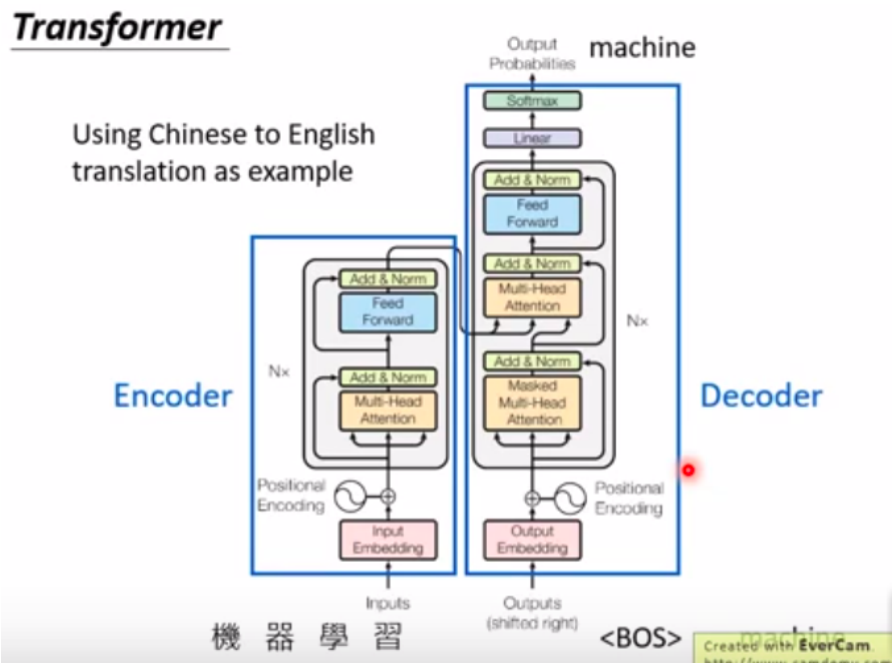
> 1. ei是人根据式子人工设的。
> 2. 为什么ei和ai是相加，不是concat进x里?看上面公式计算，计算后他们等价。
> 3. 有人试过，learn Wp,但是并不好，所以有人，通过式子人工手设wp。如下图:

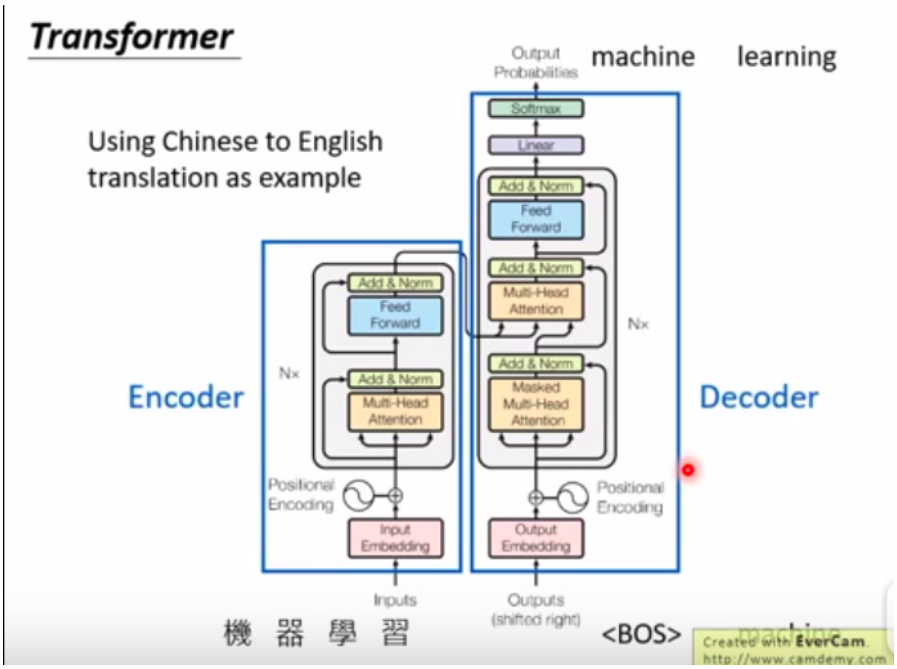


self-attention 在seq2seq

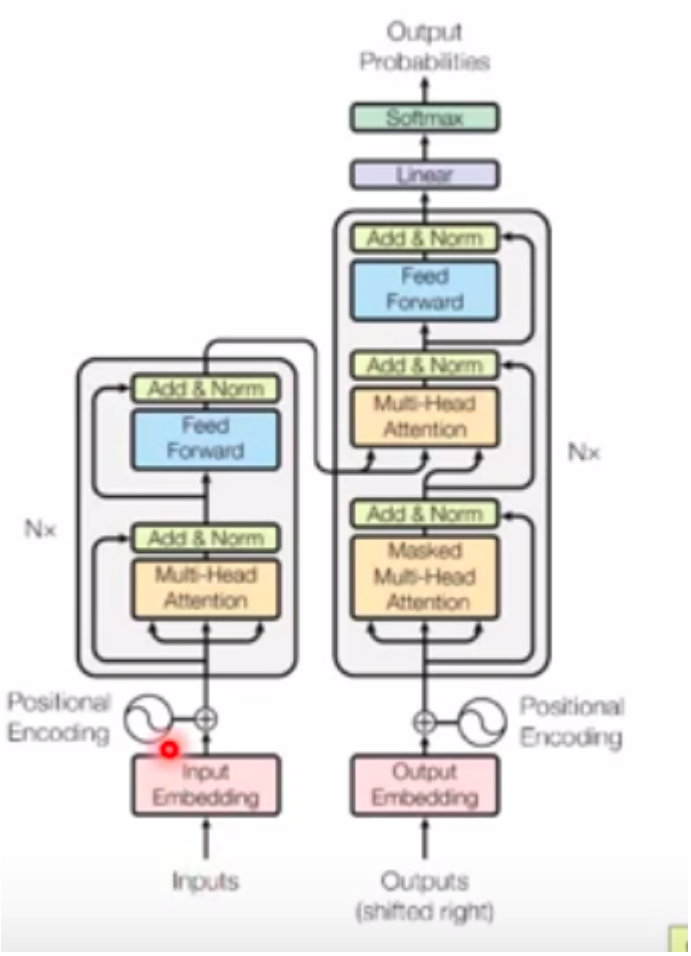


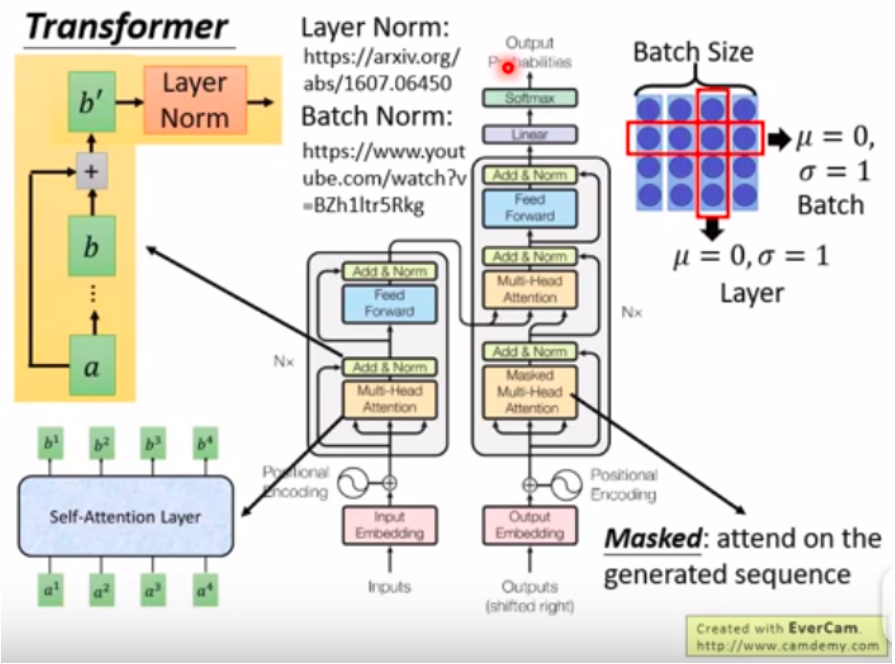
Transformer





分析layer





encoder

> 1. *input -> embedding = vctor | + position -> 最后输出*
> 2. *进入灰色块（循环N次）*

3. *灰色第一层（multi-head）:输入一个sequnce得到另一个等长的sequnce。*
4. *下一层：（如图左上）ADD: multi_input + mulit_output ——> layernorm.*
5. *batch_norm和layer_norm区别：（右上角）：batch横向（注意，横向是batch_size），整个batch里同一维度做norm，layer相反，不考虑batch，一个样本内做norm，一般layer_norm搭配RNN。*
6. *下一层：feed Forward.*
7. *最后 ADD & Norm*

decoder

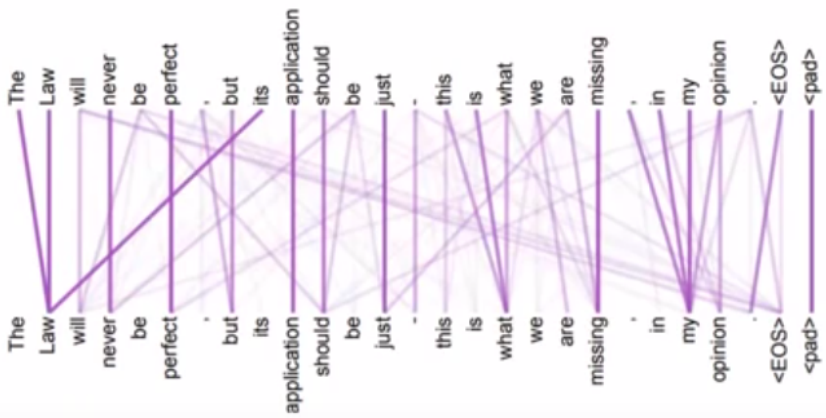
1. *input-> 前一个timestep产生的output。*
2. *embeddding+postion进入灰色block。*

1. *灰色第一层： Mask_multi,mask表示只attention产生出来的句子。*
2. *Add & Norm*
3. *下一层multi_attention对的是encoder的输出。*
4. *Add & Norm*
5. *feed*
6. *Add & Norm*

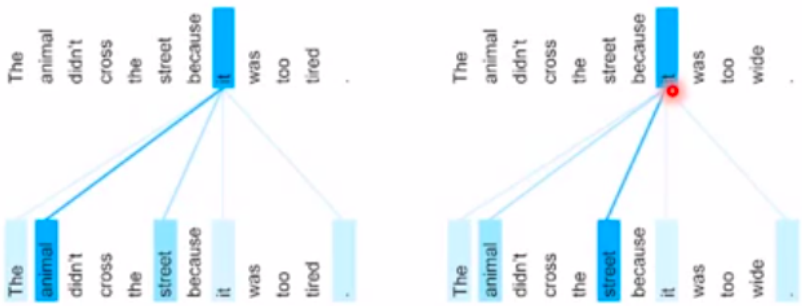
1. *linner*
2. *softmax*

Attetion visualization

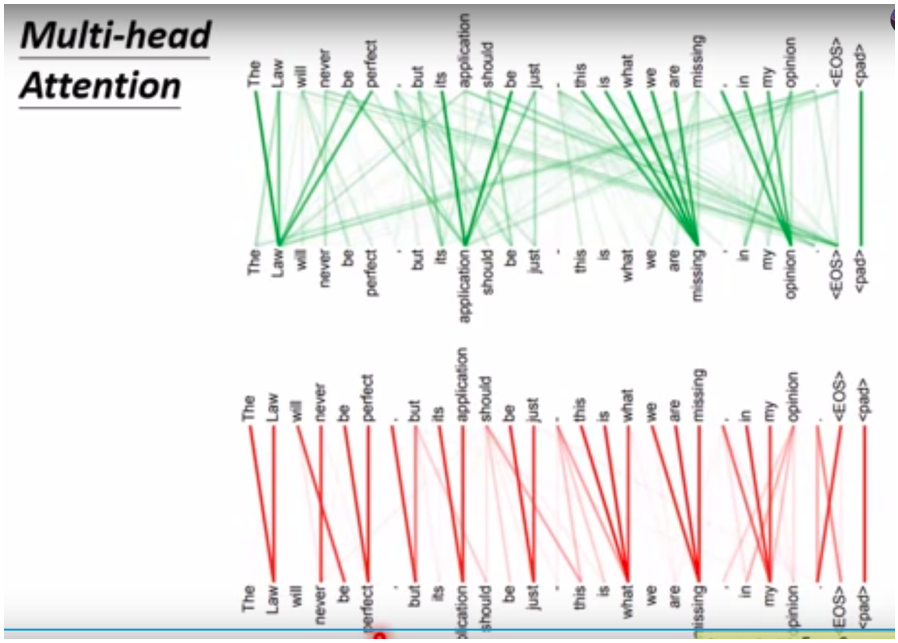
Attention Visualization



Attention Visualization



The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).



> *上面那个是：全局attention*
>
> *下面那个是：局部attention*

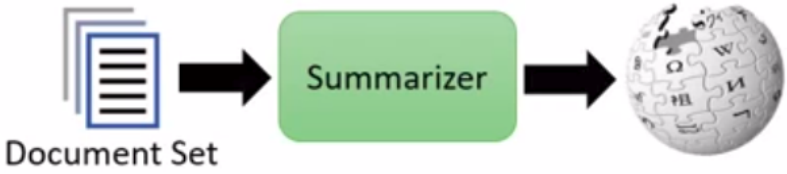
Transformer 例子

第一个(wiki)

> *摘要:*
>
> *自动写一个wiki*

Example Application

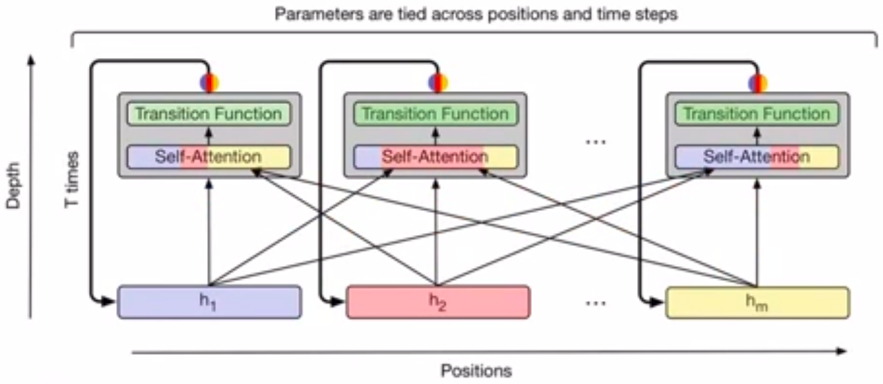
- If you can use seq2seq, you can use transformer.



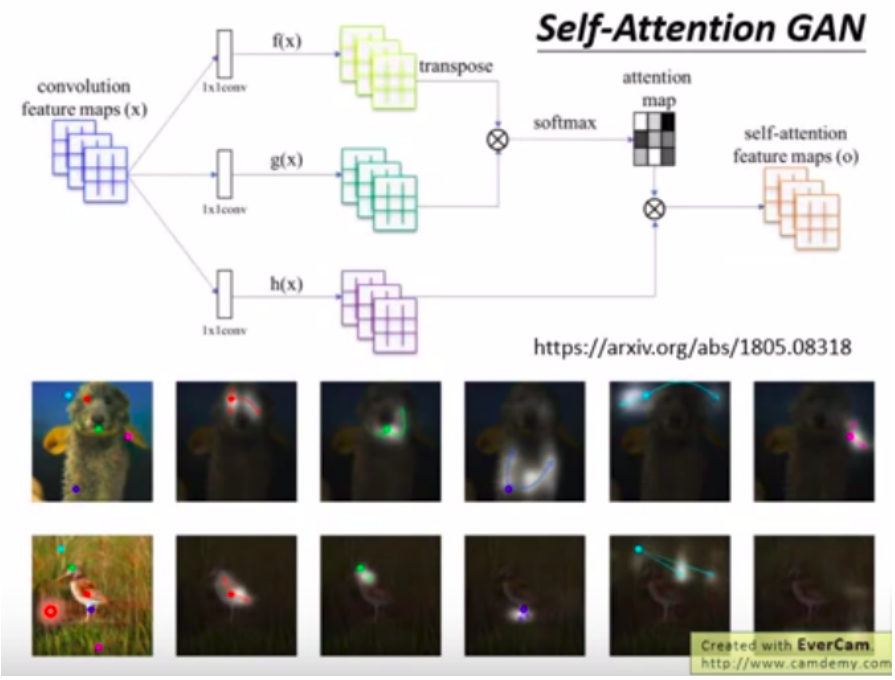
Dataset	Input	Output	# examples
Gigaword (Graff & Cieri, 2003)	10^1	10^1	10^6
CNN/DailyMail (Nallapati et al., 2016)	10^2–10^3	10^1	10^5
WikiSum (ours)	10^2–10^6	10^1–10^3	10^6

第二个（universal Transformer）

Universal Transformer



第三个（影像）


Self-Attention GAN
convolution feature maps (x)
f(x)
g(x)
h(x)
1x1conv
1x1conv
1x1conv
transpose
softmax
attention map
self-attention feature maps (o)
https://arxiv.org/abs/1805.08318
Created with EverCam.
http://www.camdemy.com