Transformer简介

2025.09.05

目录

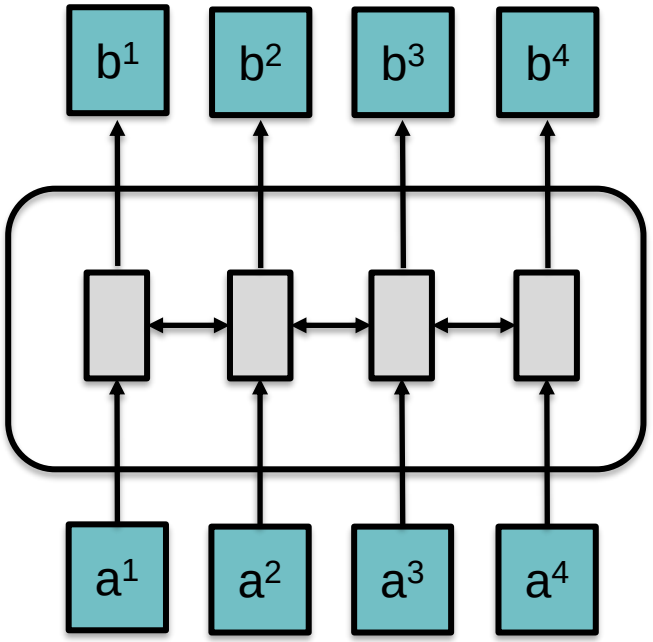
1、Transformer背景

问题引出：

- **为什么使用Transformer?**

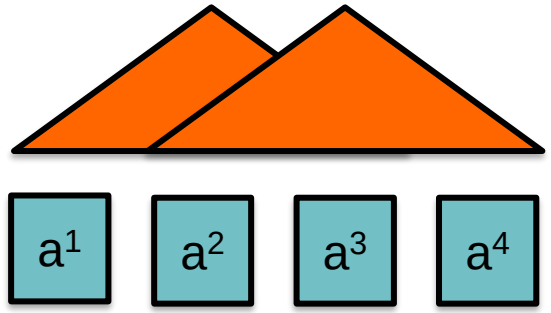
原因：

- **RNN并行困难**
- **CNN关注局部**



Next Layer

- **第n个单元的生成需要前n个单元的信息**
- **RNN: Hard to parallel!**



Using CNN to replace RNN

- **CNN也可以考虑更长的信息，但需叠加多层**

Previous layer

1、Transformer背景

问题引出：

➢ Transformer是什么？ (What)

最早的应用场景：**Seq2Seq**（机器翻译）

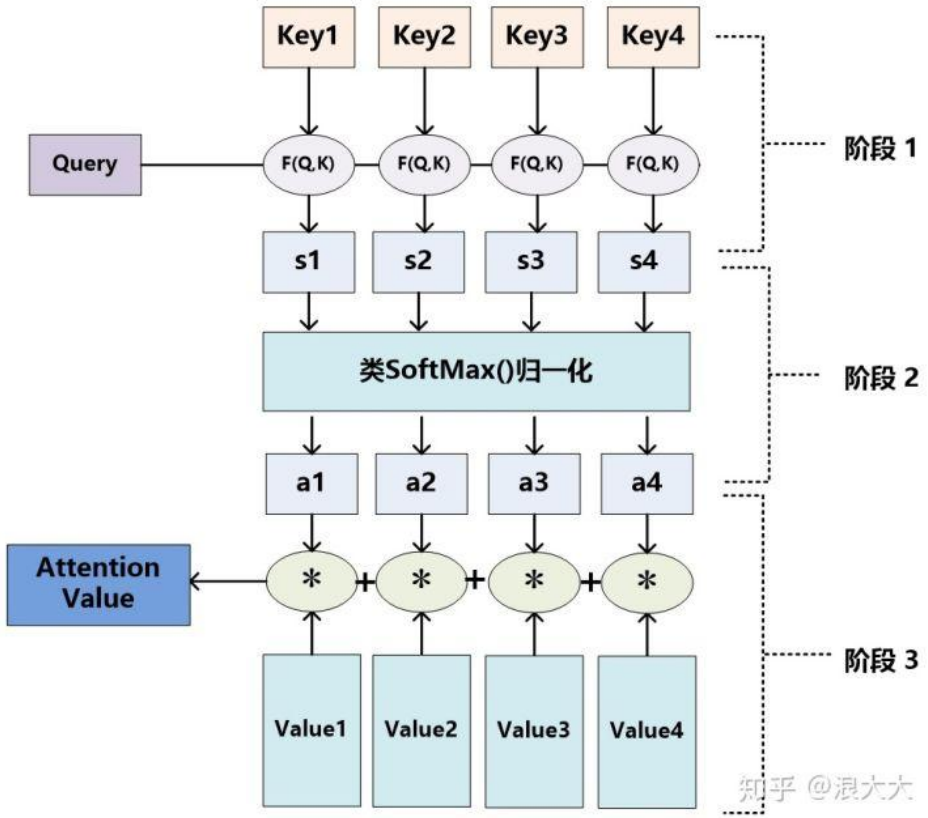
Transformer的组成：

- **Self-attention是 Attention变体，擅长捕获数据/特征的内部相关性**
- **Self-attention 组成 Multi-head Self-attention**
- **Multi-head Self-attention 反复利用组成Transformer**

1、Transformer背景

问题引出：

- **Attention是什么？ (What)**



Query、Key、Value：

query 、 key-value 的概念其实来源于推荐系统。基本原理是：给定一个 **query**，计算**query** 与 **key** 的相关性，然后根据**query** 与 **key** 的相关性去找到最合适的 **value**。

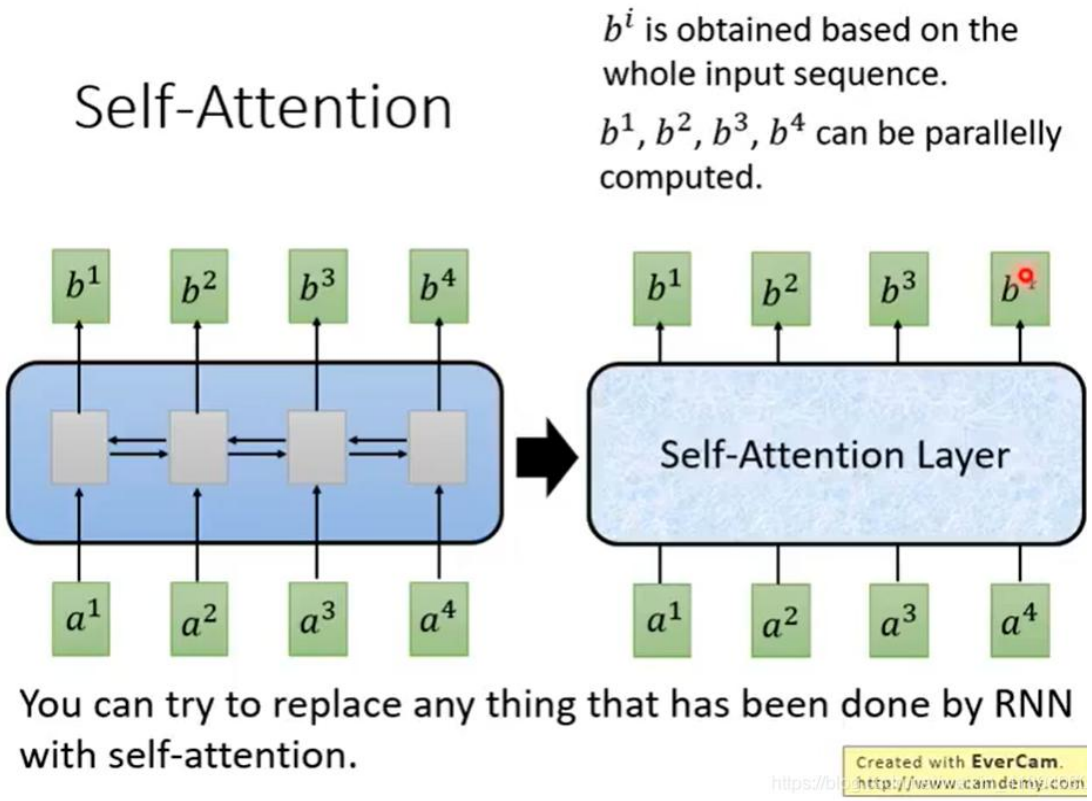
Attention机制的具体计算过程：

- 第一个过程是根据Query和Key计算权重系数（阶段1，阶段2）
- 第二个过程根据权重系数对Value进行加权求和（阶段3）
- 详细地，第一个过程分为两个阶段，阶段1根据query和key计算相似度/相关性，阶段2作归一化处理，生成权重。

1、Transformer背景

问题引出：

- **Self-attention是什么?**



Self-attention：

- 定义：Self-attention，有时称为 intra-attention，是一种将单个序列的不同位置关联起来计算序列表示的注意机制。
- 理解：自注意力机制是注意力机制的变体，其减少了对外部信息的依赖，更擅长捕捉数据或特征的内部相关性。
- 例如：**NLP**中，通过计算单词间的互相影响，来解决长距离依赖问题。（The animal didn't cross the street because it was too tired.）

Attention与Self-Attention的区别：

- 对于输入的一句话，**Attention**关注其中哪个单词更重要，寻找感兴趣单词。而**Self-Attention**通过建立不同单词的关系，来寻找哪个单词与哪个单词的关系更加紧密，例如，代词消解（**it**表示什么？）

目录

2、主要算法分析

Self-attention

https://arxiv.org/abs/1706.03762

q: query (to match others)

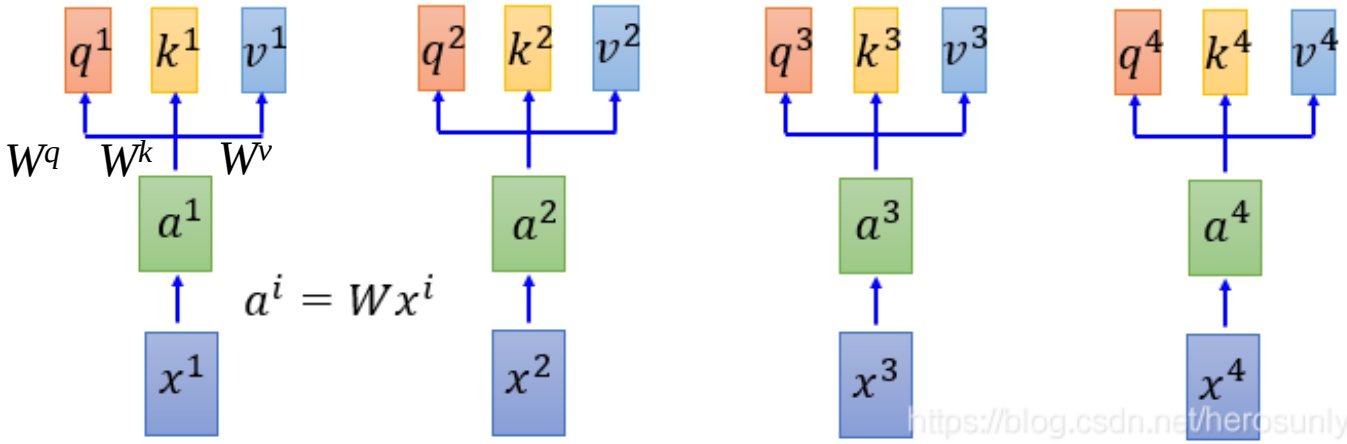
$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

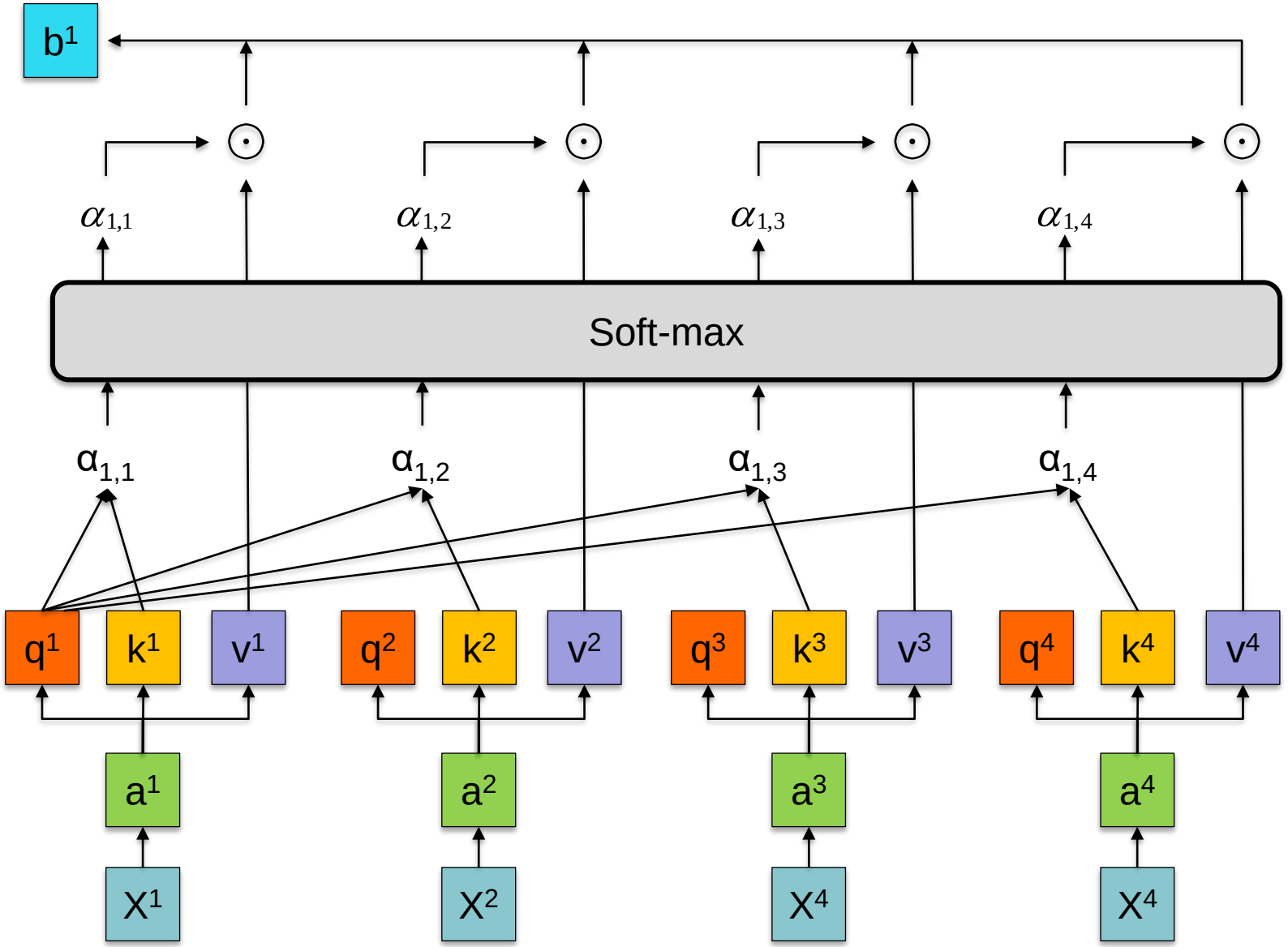
v: information to be extracted

$$v^i = W^v a^i$$



- 输入Sequence x1~x4，
- 乘上W matrix 得到embedding a1~a4，
- 丢入Self-attention：分别乘上三个不同的transformation matrix，产生三个不同的vector q，k，v。

2、主要算法分析



$$\alpha_{1,i} = \overbrace{q^1 \cdot k^i}^{\text{dot product}} / \sqrt{d}$$

- 拿每个 query 对每个 key 作点乘
- d是q和k的维度，除d的原因：q和k做点积的数值会随着维度增大

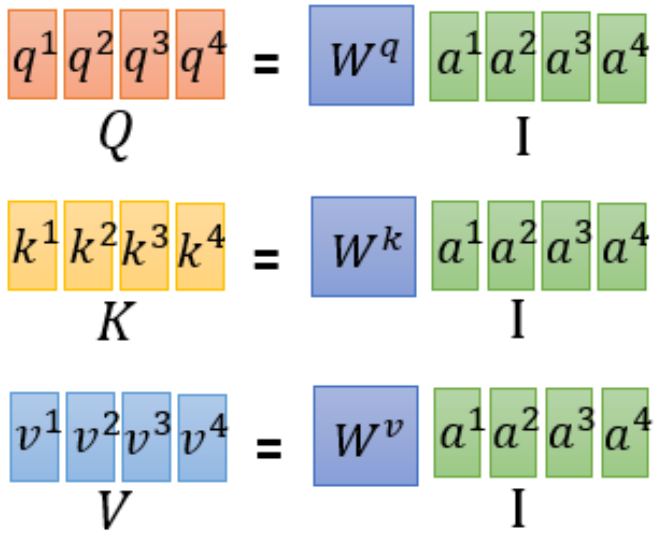
$$\hat{\alpha}_{1,i} = \exp\left(\alpha_{1,i}\right) / \sum_j \exp\left(\alpha_{1,j}\right)$$

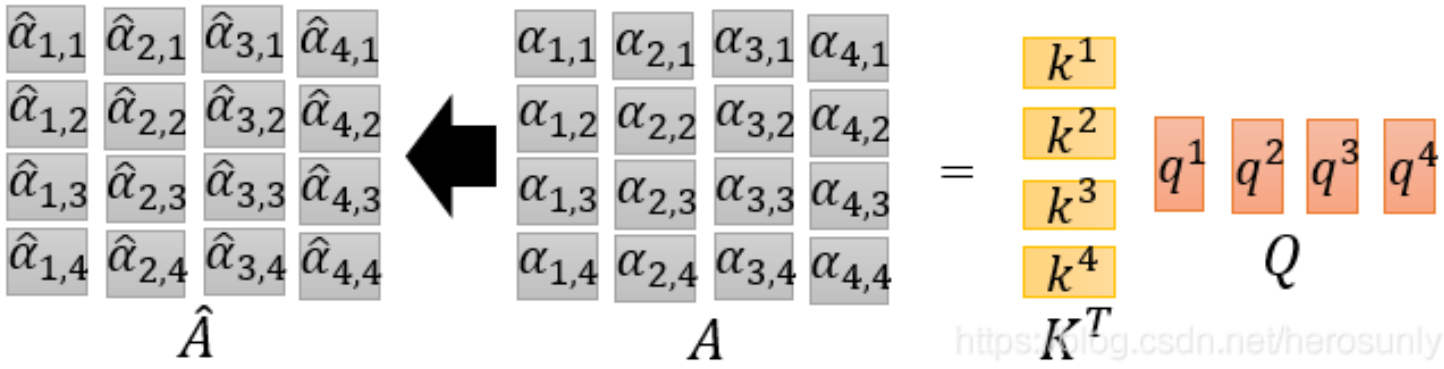
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

- b^1, b^2, b^3, b^4 可以并行计算

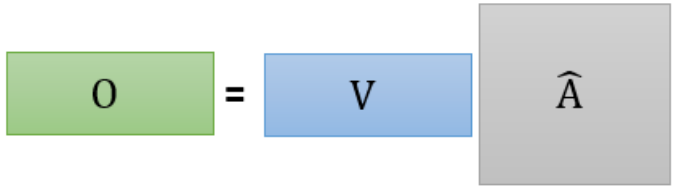
2、主要算法分析

整合后的矩阵表示

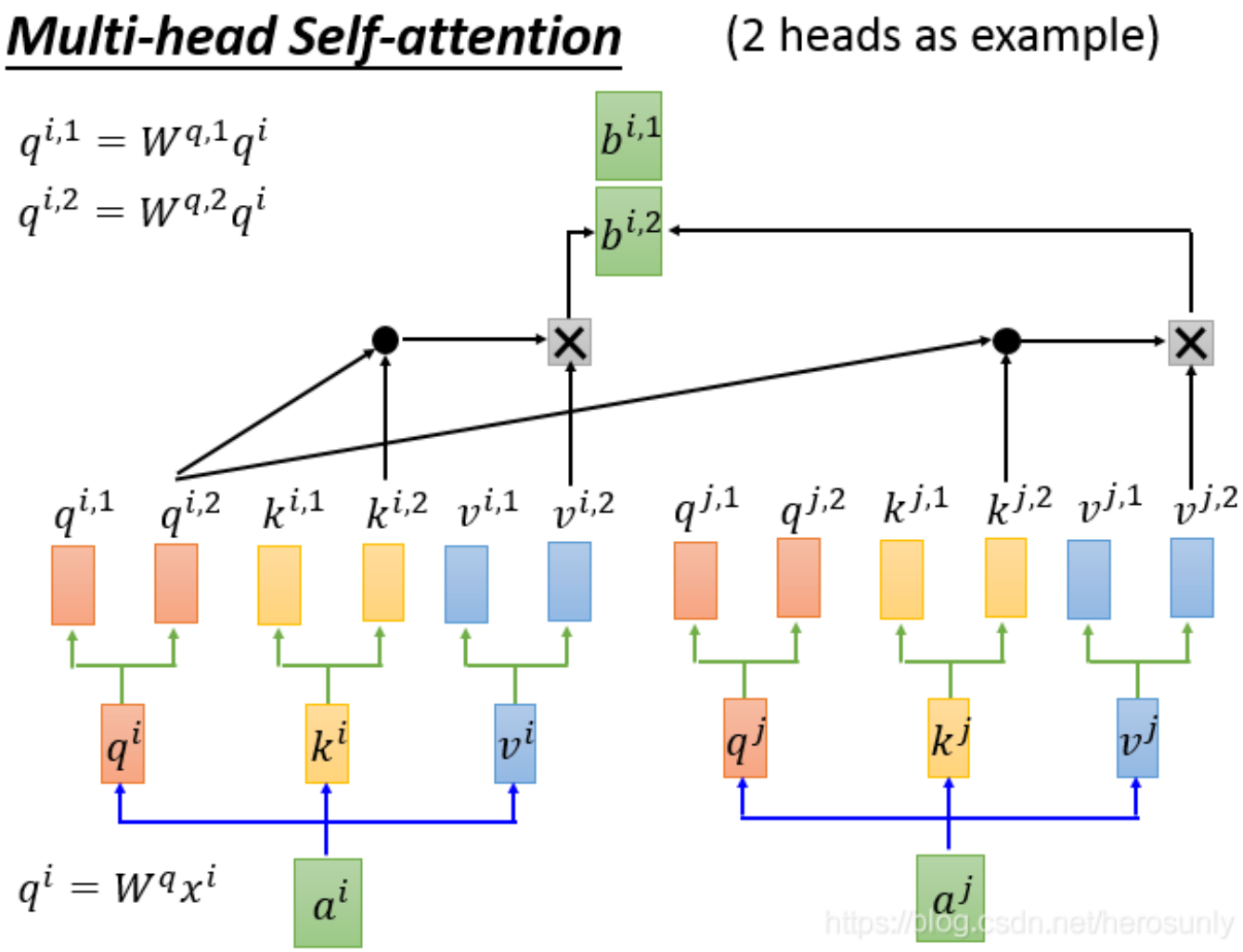




一系列矩阵乘法，可用**GPU**加速。

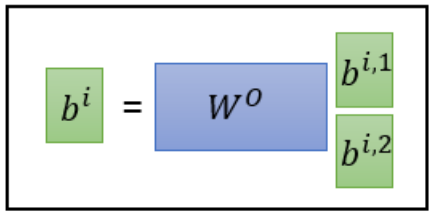


2、主要算法分析



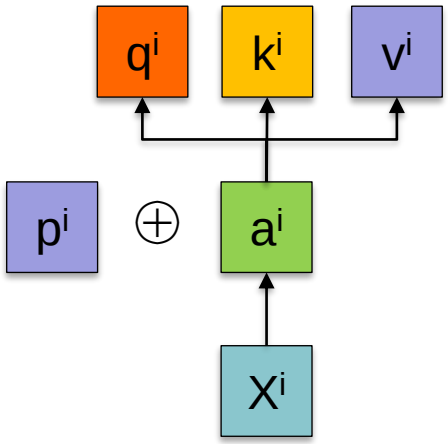
为什么要设置不同的head？

- 不同的head关注的信息可能是不同的，有的head关注的是局部信息，有的关注的是较长距离的信息(类似分组卷积，同样使用多个卷积核)。



将$b^{i,1}$，$b^{i,2}$作concatenate操作，并乘权重生成b^i

2、主要算法分析



Positional Encoding

- Self-attention中不体现位置信息

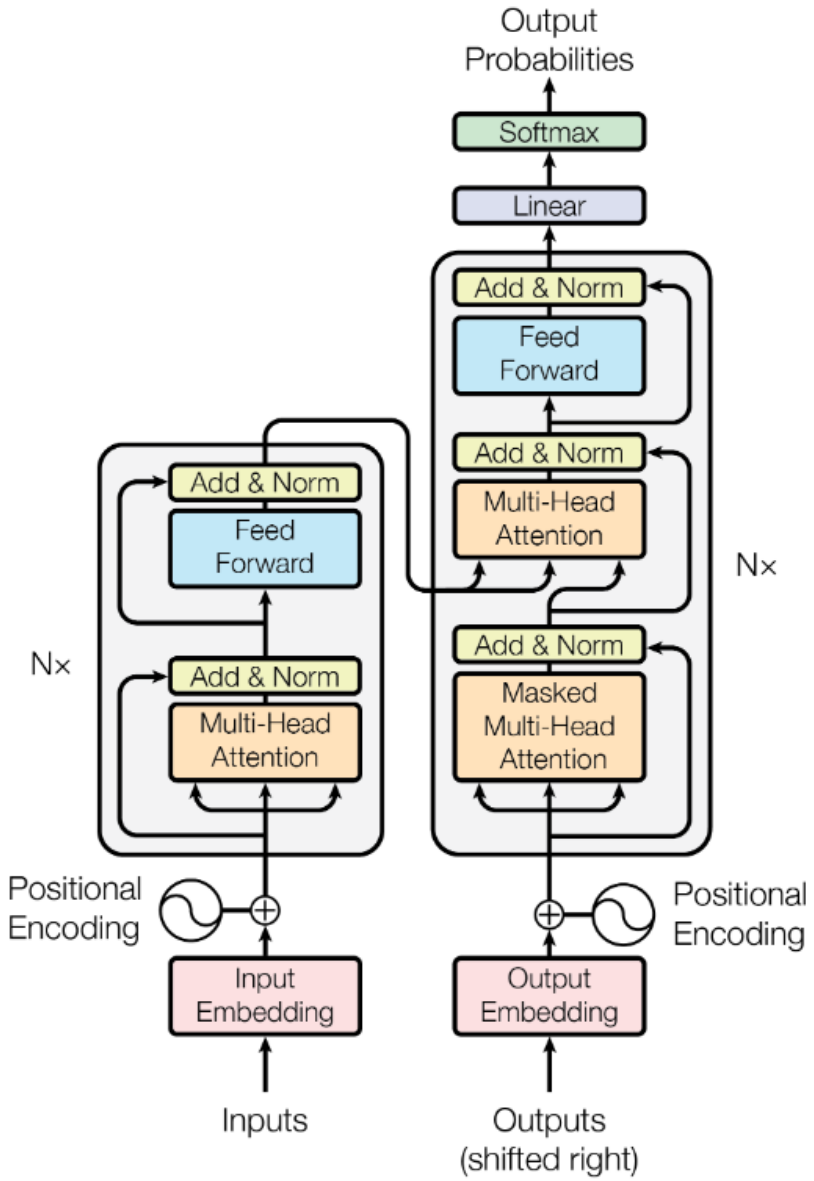
$$\left[W^I, W^p\right]\begin{bmatrix} x^i \\ p^i \end{bmatrix} = W^I * x^i + W^p * p^i = a^i + e^i$$

$p^i =$ 0 1 0 ⋮ ← i-th dim

为啥用add而不用concatenate？

- 通过矩阵变化，意义一样
- W^p是人工设置。

2、主要算法分析



Transformer 是第一个完全依赖自注意力来计算其输入和输出表示而不是使用序列对齐的**RNN**和**CNN**。

编码器：

- **N=6**个相同**layer**，每个**layer**包括两个子层，每个子层采用残差连接，并添加层归一化。
- 每个子层的输出是**LayerNorm(x+Sublayer(x))**，其中 **Sublayer** 是子层自己的实现函数。
- 为促进残差连接，每个子层和嵌入层产出d_{model}**=512**

解码器：

- **N=6**个相同**layer** ，每个**layer**包括三个层，添加对编码器堆栈输出执行**multi-head attention**。
- 修改了解码器堆栈中的 **self-attention** 子层，以防止位置关注后续位置。

2、主要算法分析

动图示例

注意力在此模型中的应用在三个方面：

- **Encoder给Deocder：query来自以前的decoder layer，而key和value来自encoder的输出。这使得decoder中的每个位置都能关注到输入序列中的所有位置。 这是模仿序列到序列模型中典型的Encoder-Decoder的attention机制**

- **Encoder之间：在self-attention层中，所有的key 、 value和query都来自同一个地方，在这里是encoder中前一层的输出，encoder中的每个位置都可以关注到encoder上一层的所有位置**

- **Decoder之间： decoder中的self-attention层允许decoder中的每个位置关注decoder中直到并包括该位置的所有位置。防止decoder中的左向信息流以保留自回归特性。**

2、主要算法分析

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- 对于序列处理，**Self-Attention**和**CNN**可并行计算，而**RNN**依赖于前面的输出。
- 对于**Complexity per layer**，**Self-Attention**建立了**n**个单词之间的联系，彼此相连，故为n^2，现在每个单词映射到**d**维度，所有是n^2*d。**Recurrent**的输入和中间语义的维度都是**d**，一共**n**个单词，则为d^2*n。**CNN**的卷积核大小为**k**，个数为**n**（**n**个输入，步长为**1**平移），**d**同时是卷积输入和输入的维度，所以$k*n*d^2$。对于**Self-Attention(restricted)**，只在局部捕获自注意力机制，**r**代替其中一个**n**。
- **Maximun Path Length**表示网络中远程依赖之间的路径长度， 学习长距离依赖是许多序列转化模型的关键挑战。输入和输出的任意组成路径越短，学习远距离依赖更容易。**Self-Attention**彼此相连，故而最短路径为**1**。

2、主要算法分析

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

2、主要算法分析

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

表分析：

- 在表3（A），改变了注意头数和key和value的维度，计算量不变。头的数量有影响（不一定越大越好）
- 表3（B），注意到降低key尺寸会损害模型的性能
- 从表3的（c）和（D）可以看到，更大的模型性能更好，dropout对模型过拟合很有帮助。
- (E) 是用学习的位置嵌入代替我们的正弦位置编码，并观察到与基本模型几乎相同的结果

参数解释：

- N表示Encoder or Decoder的块数
- dmodel：为促进残差连接，模型子层和嵌入层产生维度为512的输出
- dff：两个FC的中间映射
- h：head的数量
- $value_{\varepsilon_{ls}} = 0.1$：标签平滑

Q&A