

预训练语言模型（上）

Transformer

王嘉宁

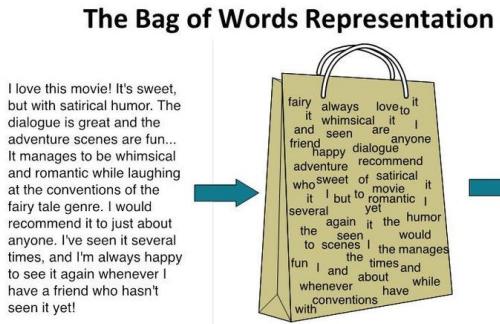
2023-05-30

深度学习课程

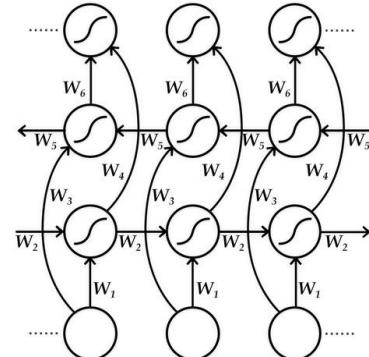


DaSE
Data Science
& Engineering

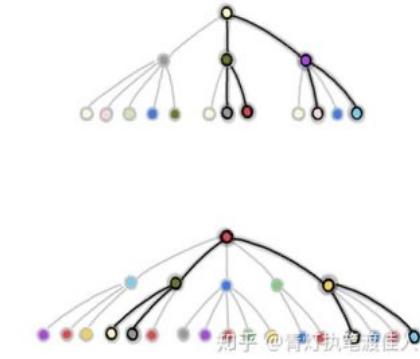
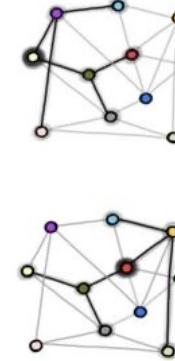
你所知道的表征提取模型有哪些？



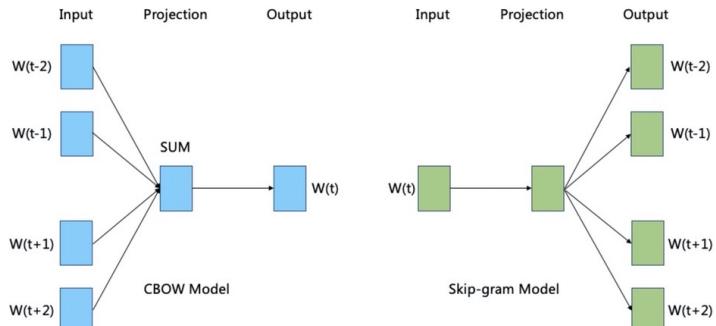
词袋模型



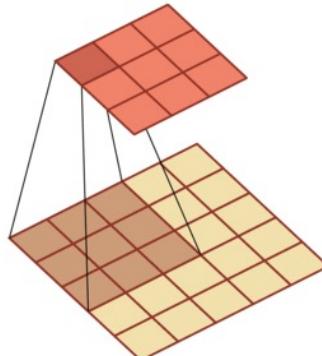
循环神经网络



图神经网络

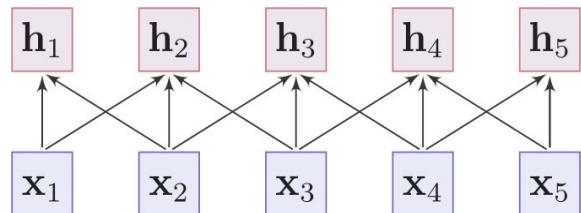


词向量模型

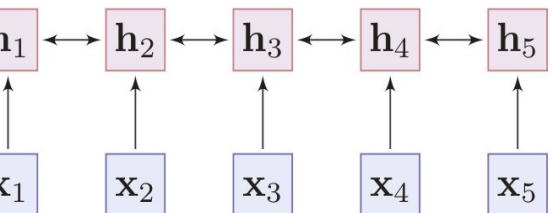


卷积神经网络

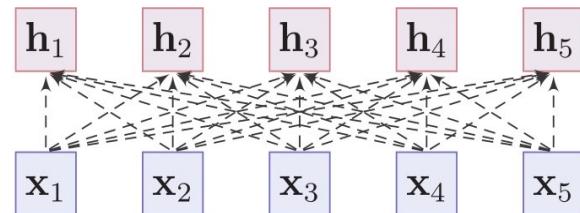
更多。。。



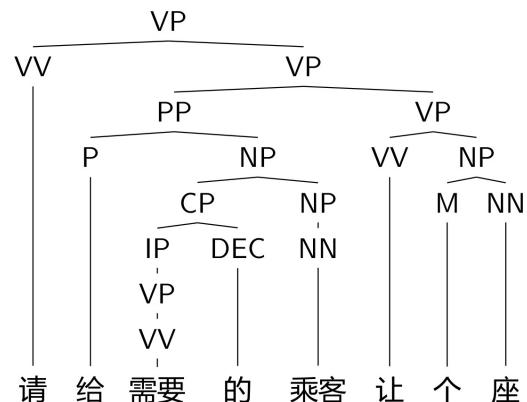
(a) Convolutional model



(b) Sequential model



(c) Fully-connected graph-based model



自然语言：序列数据到图数据的转换

循环神经网络：

- 捕捉时序信息
- 通过记忆单元获得全局信息
- 线性马尔可夫性；

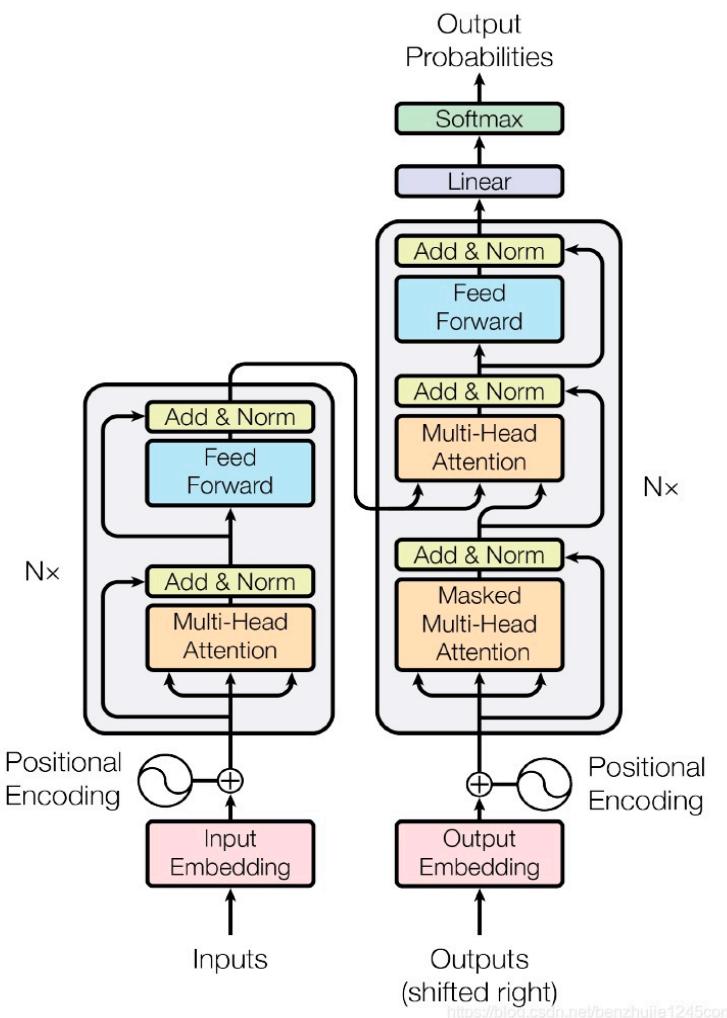
循环神经网络：

- 捕捉细粒度局部信息；
- 通过多层卷积获得全局信息；

图神经网络：

- 全连接图/连通图；
- 不受到位置远近的约束；
- 可获得局部和全局信息；

Transformer模型



什么是Transformer? (变压器?)

Transformer由谷歌公司于2017年发表在NIPS的论文《Attention Is All You Need》中提出的完全使用Attention机制实现的端到端机器翻译模型。

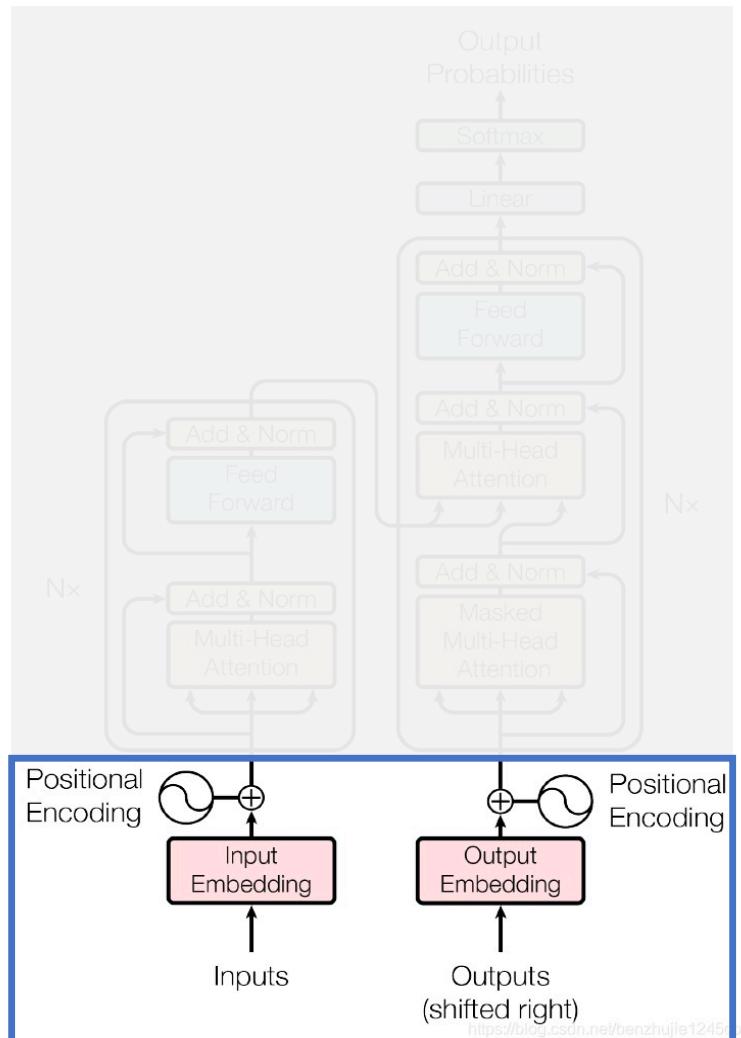
其主要贡献：

- 完全使用Attention机制；
- 可并行训练；
- 缓解了RNN、CNN的局部最优问题；
- 增加深度，进一步提高泛化能力；

解决问题：

- Transformer如何解决没有位置信息问题？
- 只有Attention，如何提高泛化能力？
- 如何进行文本生成？

Transformer模型——表征



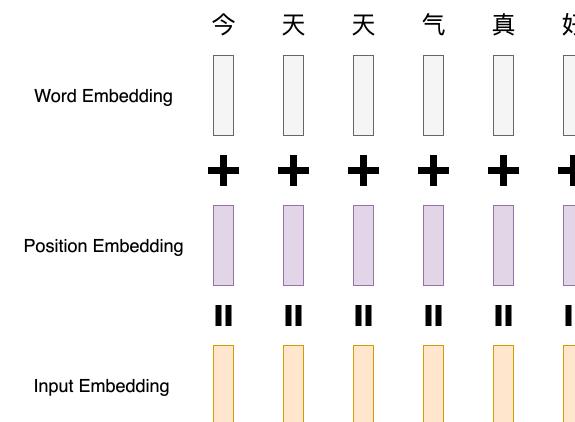
Embedding层：旨在对输入的文本Token进行表征

Token是什么？

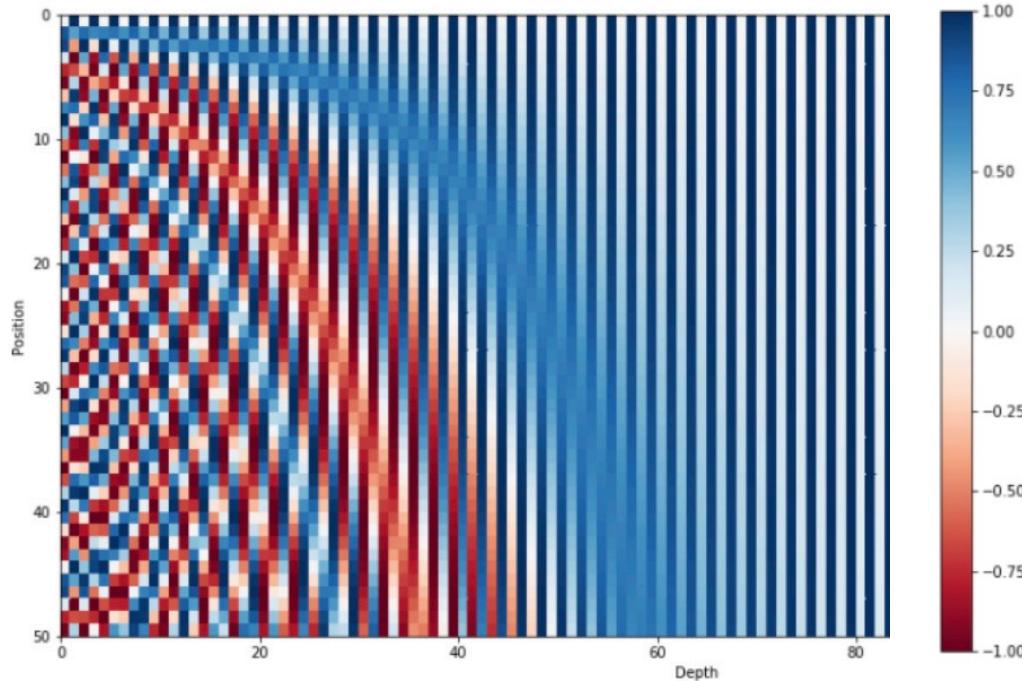
Token是NLP分词工具将文本分词后的最小单位，常用的分词方法由Word Piece、BPE等。

Input Embedding包含两部分内容：

- **词向量（Word Embedding）**：可以是随机初始化（正态分布初始化），可以使用预训练的Word2Vec等；
- **位置表征（Position Embedding）**：用于保存不同位置的信息，使得Transformer模型能够学习到位置信息



Transformer模型——表征



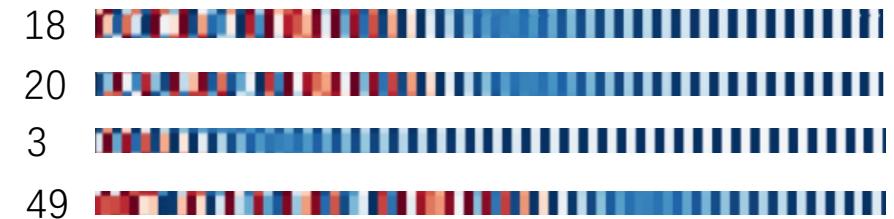
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

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

Position Embedding

采用正余弦函数来定义不同位置的表征向量。

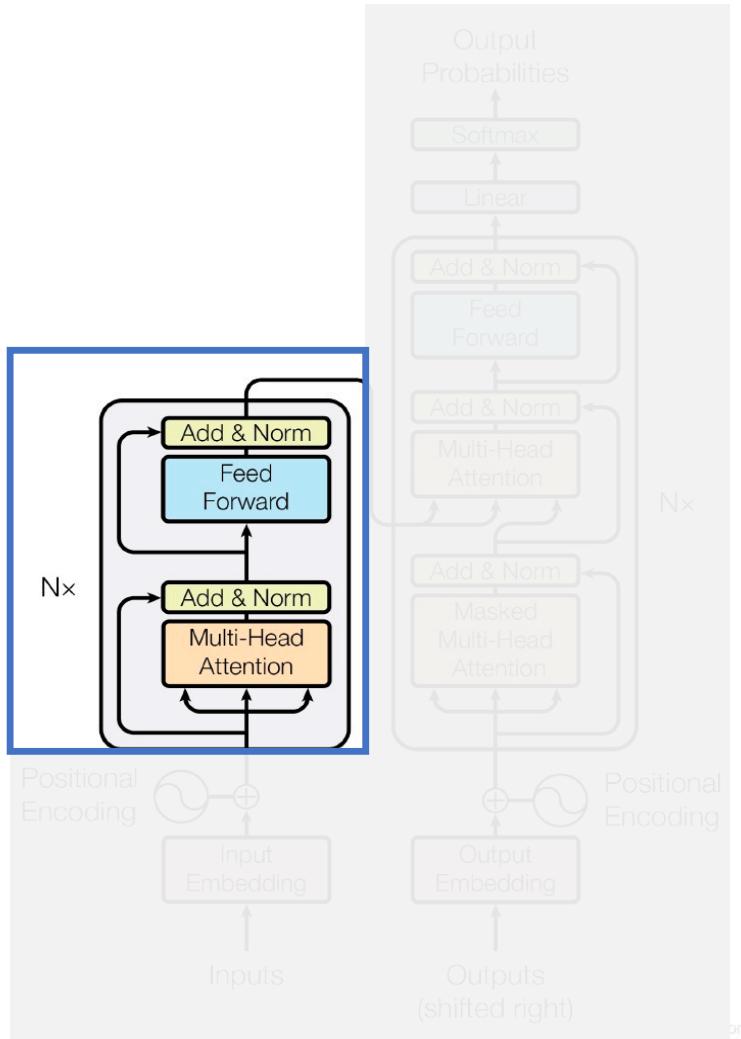
- 对于某个Position，其向量的奇数位元素为余弦值，偶数位元素位正弦值；
- 绝对位置 \times ，相对位置 \checkmark ；
- 相邻的位置的位置表征向量也很相似；
- 可以表征任意长度的位置，避免位置OOV问题；



(1) 性质一：两个位置编码的点积(dot product)仅取决于偏移量 $\triangle t$ ，也即两个位置编码的点积可以反应出两个位置编码间的距离。

(2) 性质二：位置编码的点积是无向的，即 $PE_t^T * PE_{t+\triangle t} = PE_t^T * PE_{t-\triangle t}$

Transformer模型——Encoder



Transformer Encoder

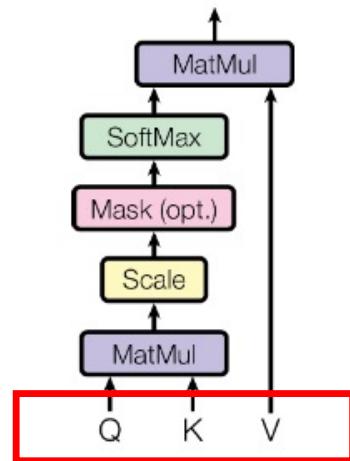
Encoder由若干Transformer Block组成，其主要目的是对输入的文本进行双向编码。

Transformer Block包含如下几个模块：

- Multi-head Attention：由多个Self-Attention组成
- 残差链接&Layer Normalization
- Feed Forward

Transformer模型——Encoder

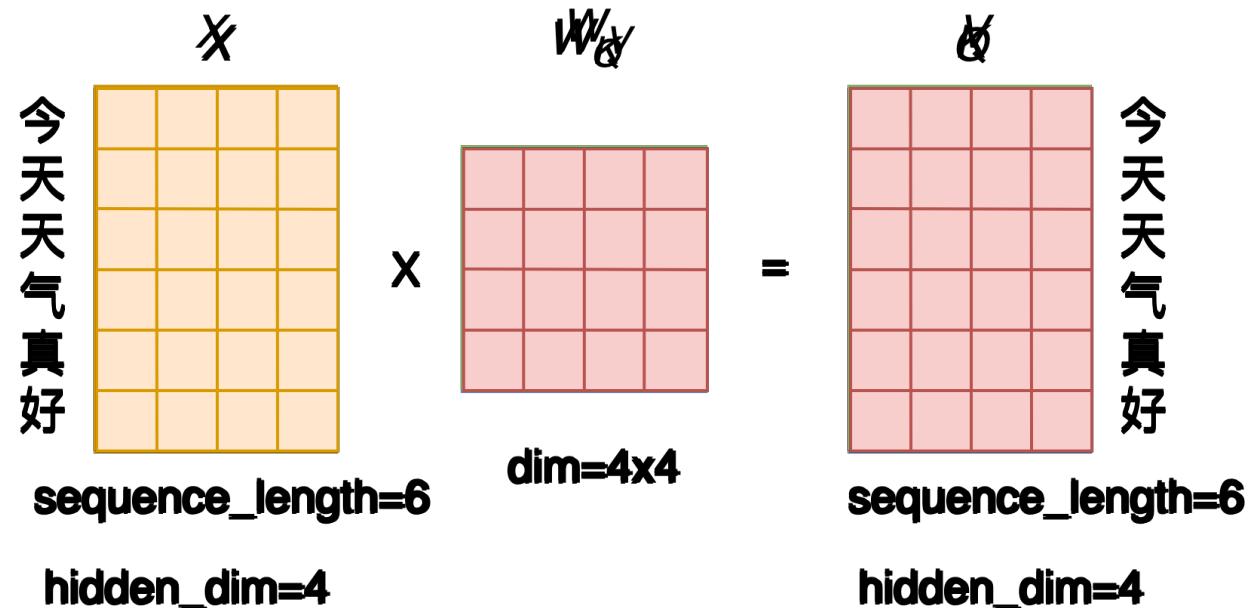
Scaled Dot-Product Attention



备注：Encoder中是双向编码，所以无需进行Masking。

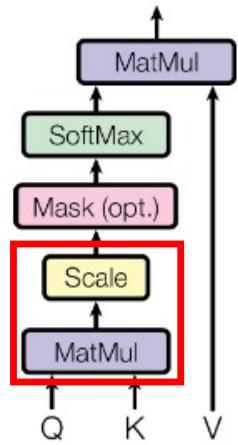
Transformer Encoder——Scaled Dot-Product Attention

$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$



Transformer模型——Encoder

Scaled Dot-Product Attention

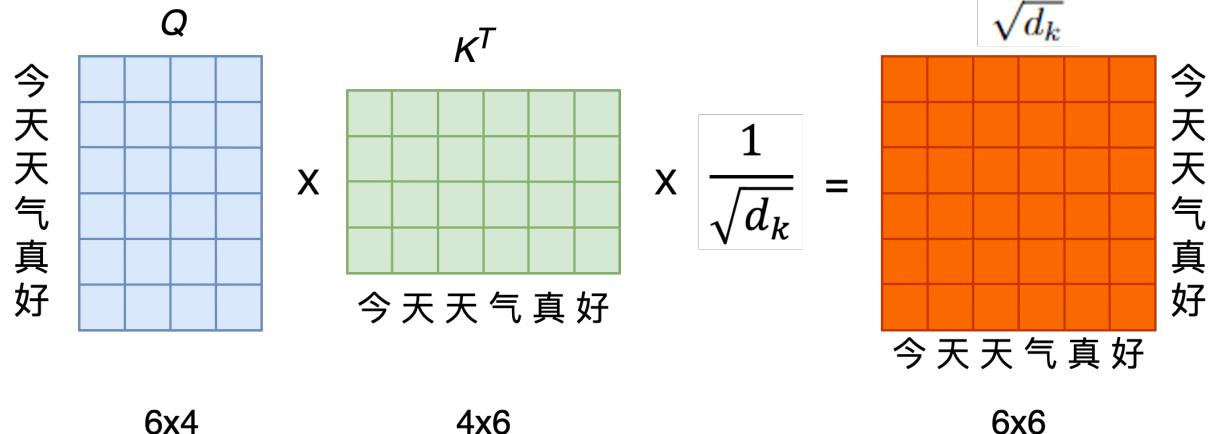


备注：Encoder中是双向编码，所以无需进行Masking。

Transformer Encoder——Scaled Dot-Product Attention

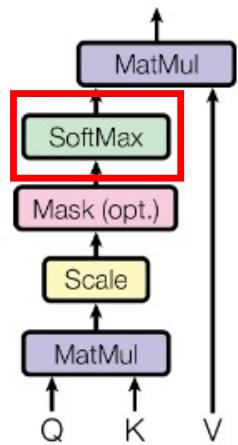
$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

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



Transformer模型——Encoder

Scaled Dot-Product Attention

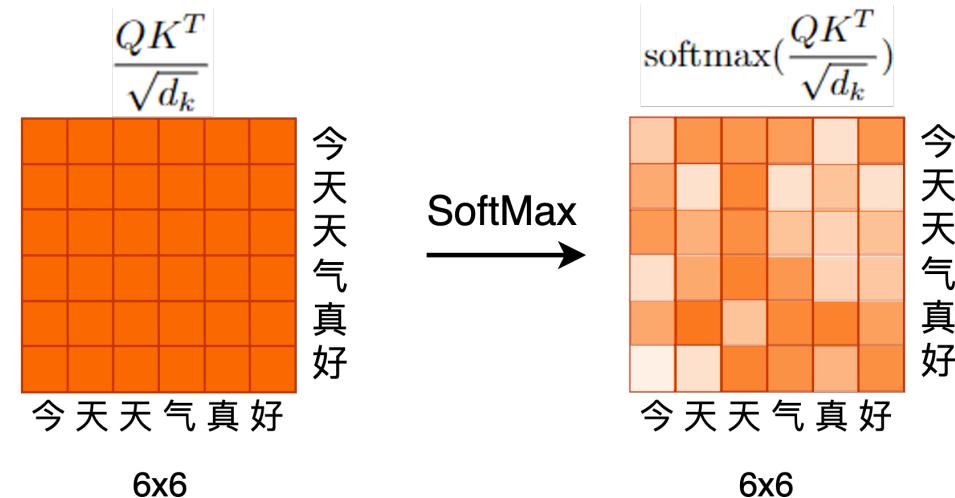


备注：Encoder中是双向编码，所以无需进行Masking。

Transformer Encoder——Scaled Dot-Product Attention

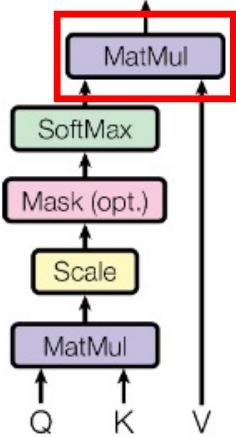
$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

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



Transformer模型——Encoder

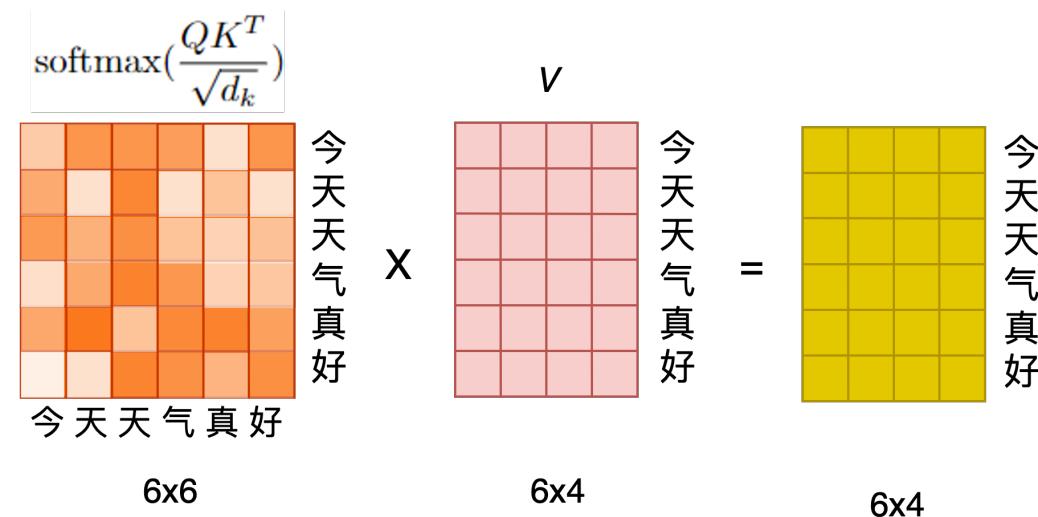
Scaled Dot-Product Attention



Transformer Encoder——Scaled Dot-Product Attention

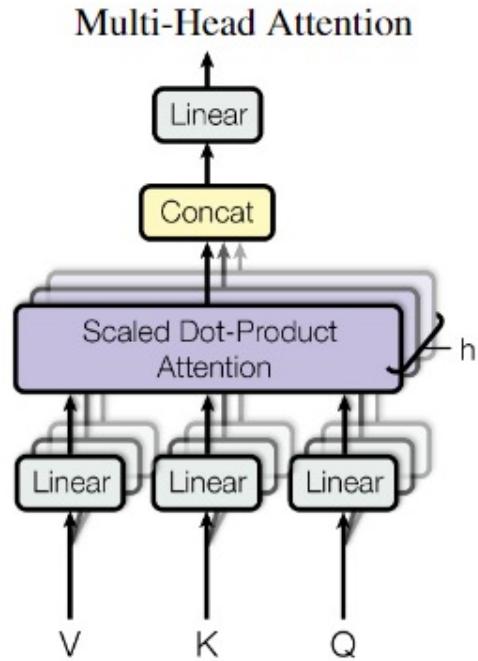
$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

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



备注：Encoder中是双向编码，所以无需进行Masking。

Transformer模型——Encoder



多头注意力 (Multi-head Attention)

Transformer Encoder——Scaled Dot-Product Attention

$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

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

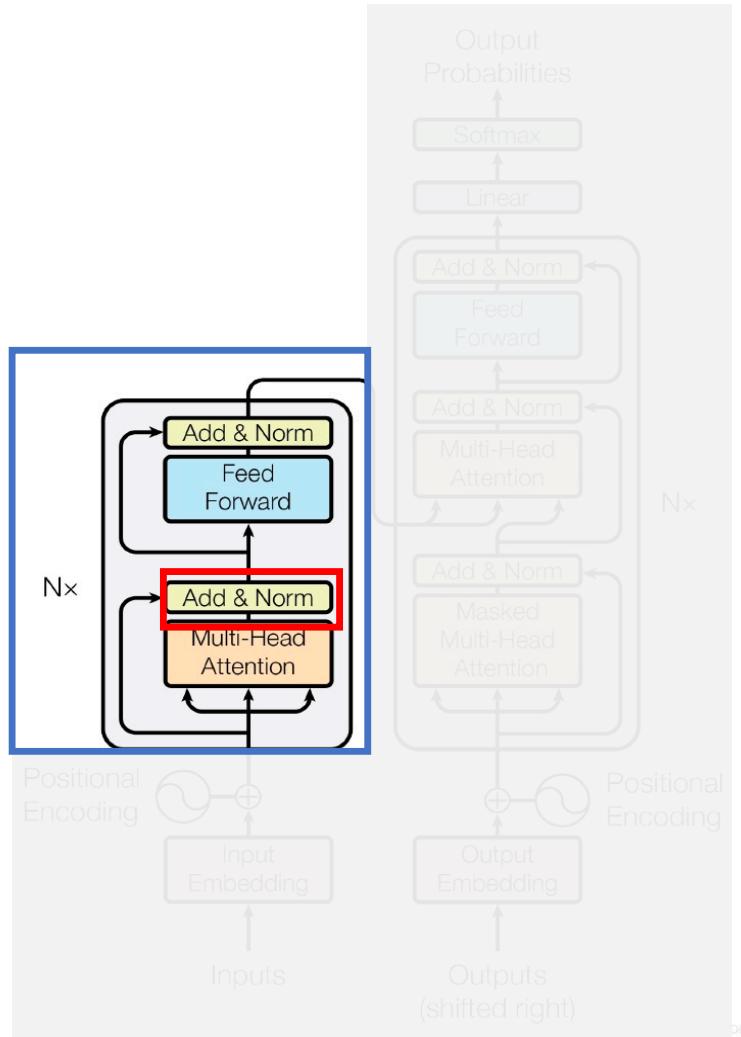
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

思考：

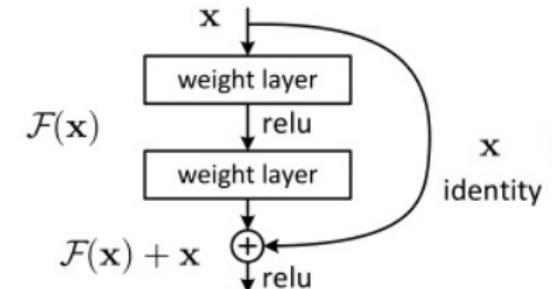
- 多头注意力的作用是什么？和CNN有什么区别？
- 假设Embedding维度为768，一共有12个Head，每个Head对应的Attention的维度是多少？

Transformer模型——Encoder



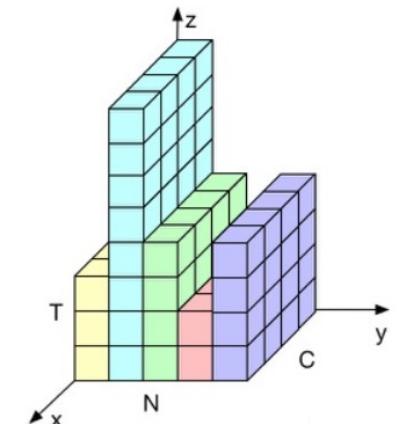
残差链接

Multi-head Attention的输入与输出进行相加，避免模型过深导致梯度消失或梯度爆炸问题。



Layer Normalization

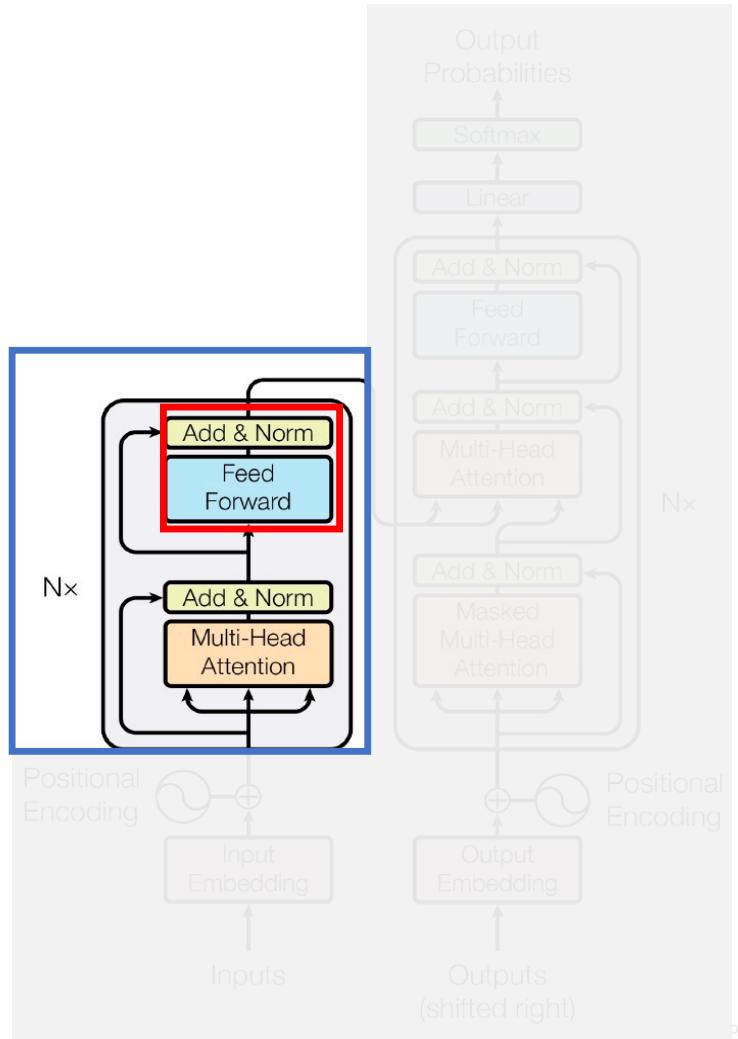
- Token之间存在相关性；
- 对于一批数据，样本之间是独立的；
- 每个Token对应的768个特征进行归一化；



思考：

- 为什么要使用归一化？不使用会怎么样？
- 为什么Transformer不能使用Batch Normalization？

Transformer模型——Encoder



Feed Forward Network

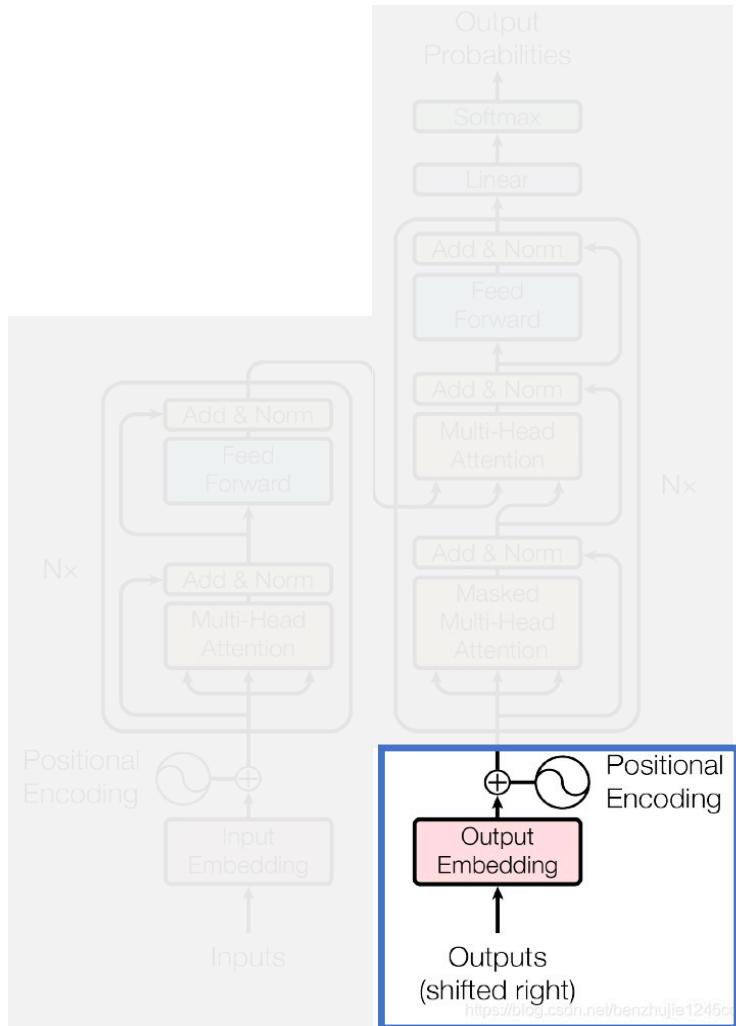
本质上是一个使用ReLU的MLP

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

思考：

- 为什么要再接一层MLP？

Transformer模型——Decoder



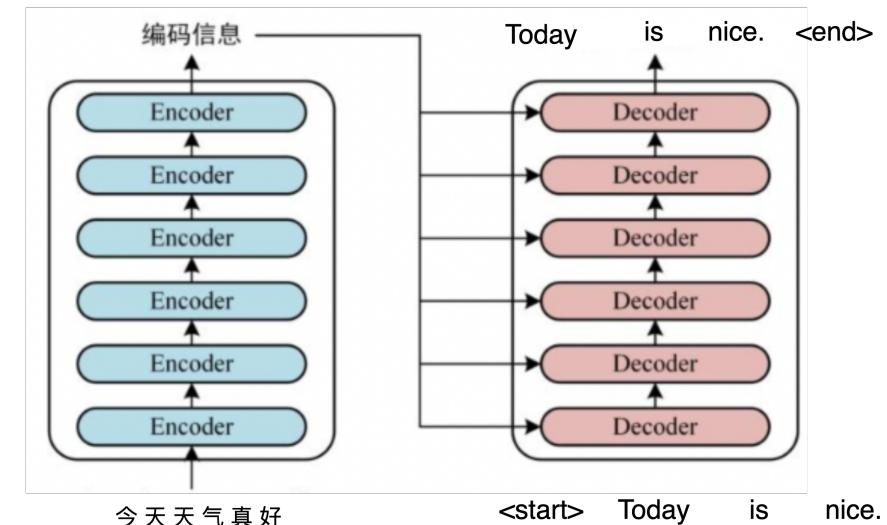
Auto-regressive Generation

Transformer采用自回归式生成模式，因此在每个时刻t，只能生成一个token，其可以定义为一个条件概率：

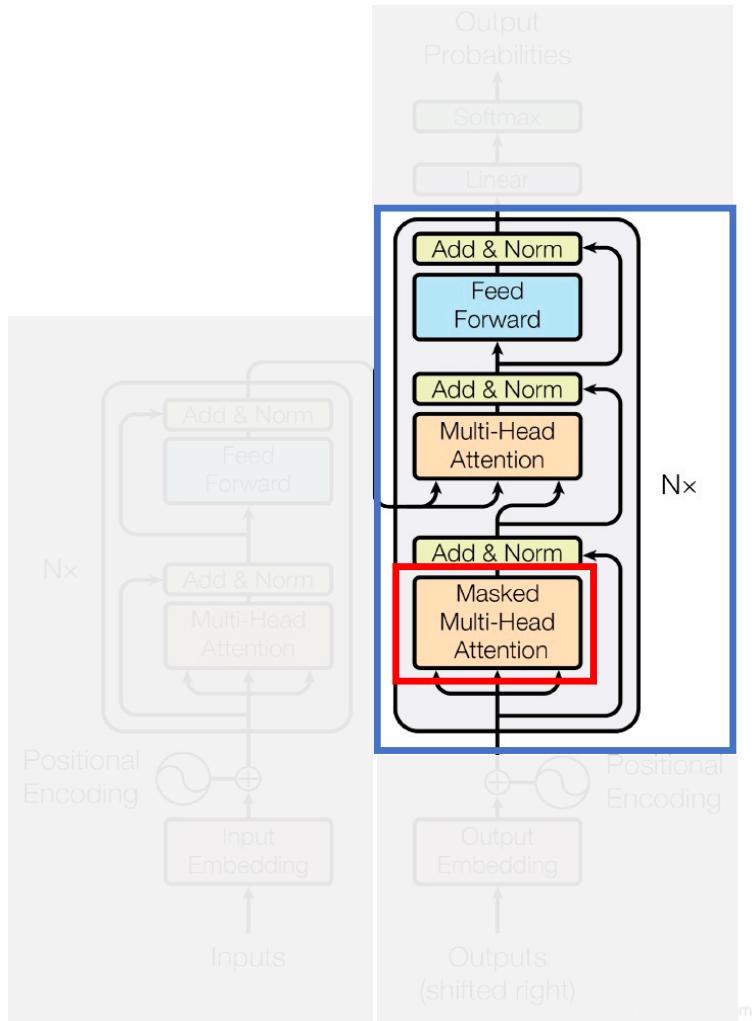
$$y_t = \arg \max_y p_\theta(y|y_{<t}, h, x_t) = \arg \max_y p_\theta(y|\mathbf{y}_{t-1}, \mathbf{h}, \mathbf{x}_t)$$

因此在t时刻，需要模型接收到如下信息：

- Encoder阶段对所有输入h的表征信息；
- 上一个生成的token y_t ；
- 当前时刻的输入token x_t 。



Transformer模型——Decoder



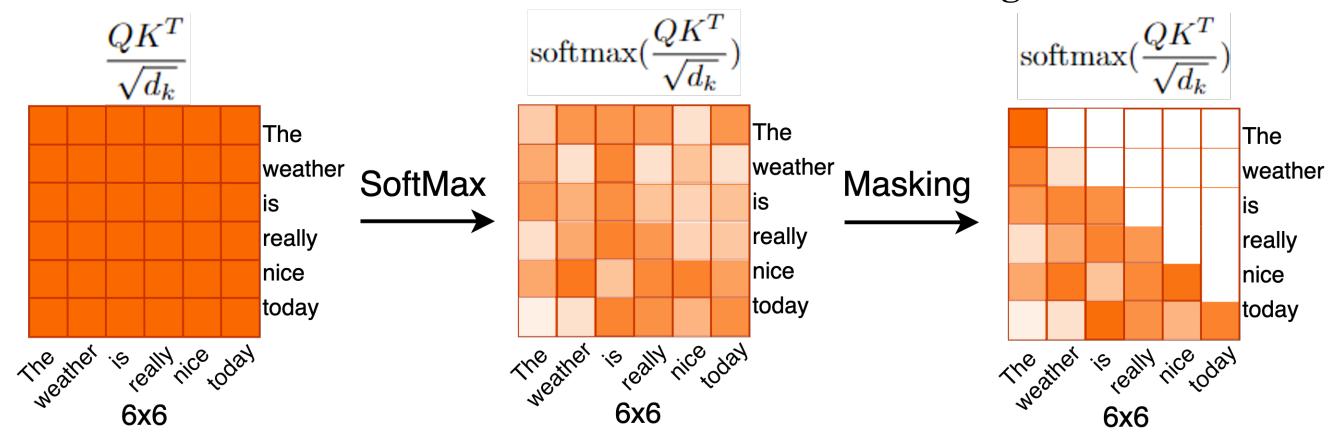
Transformer Block

在Decoder中，每个Transformer Block与Encoder阶段类似，区别在于：

- 有两层Multi-head Attention；
- 新增了一个Masked Self-Attention；
- 在第二层Multi-head Attention中，将Encoder和Decoder的信息同时融合。

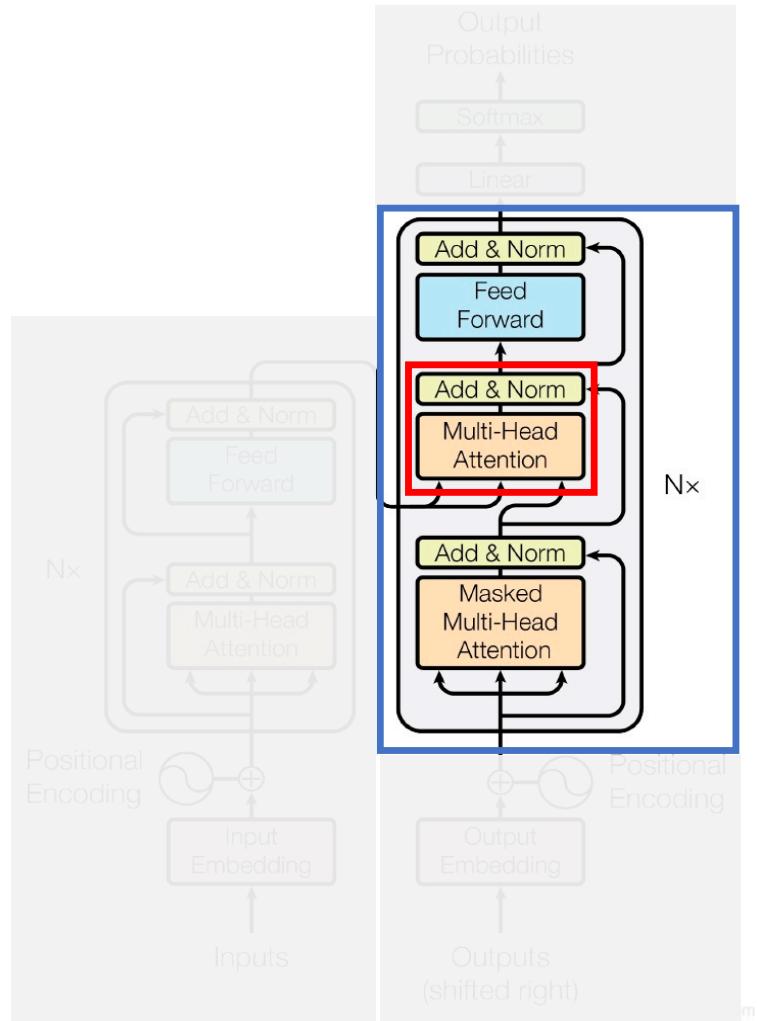
第一层Attention——Masked Multi-head Attention

- 输入：训练阶段，QKV全部来自于Decoder的输入；
- Attention矩阵，对“未来”的token进行Masking。



思考：为什么要对Attention矩阵进行Masking？

Transformer模型——Decoder



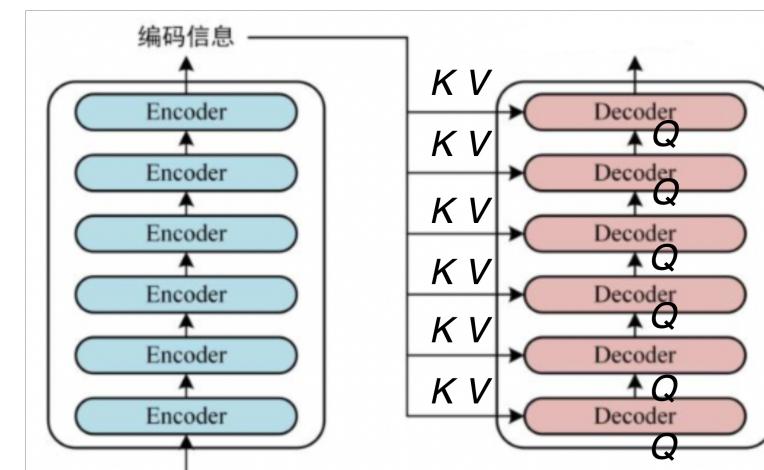
Transformer Block

在Decoder中，每个Transformer Block与Encoder阶段类似，区别在于：

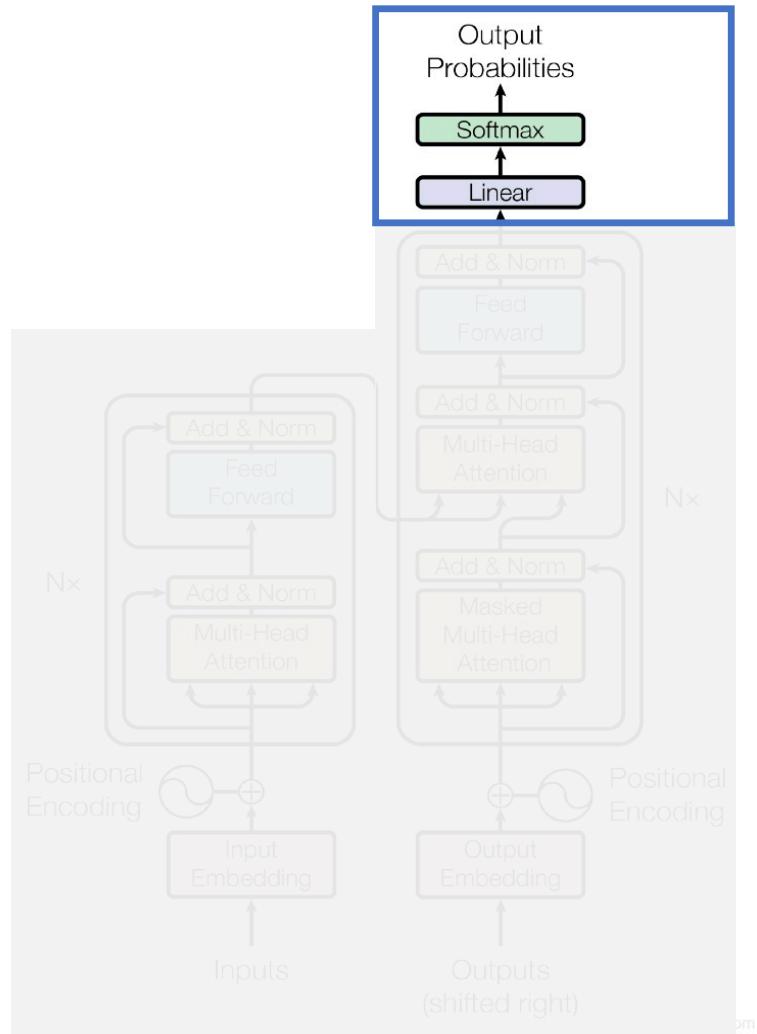
- 有两层Multi-head Attention；
- 新增了一个Masked Self-Attention；
- 在第二层Multi-head Attention中，将Encoder和Decoder的信息同时融合。

第二层Attention——Multi-head Attention

- 输入：K和V来自于Encoder的输出，Q来自于第一层Attention的输出。

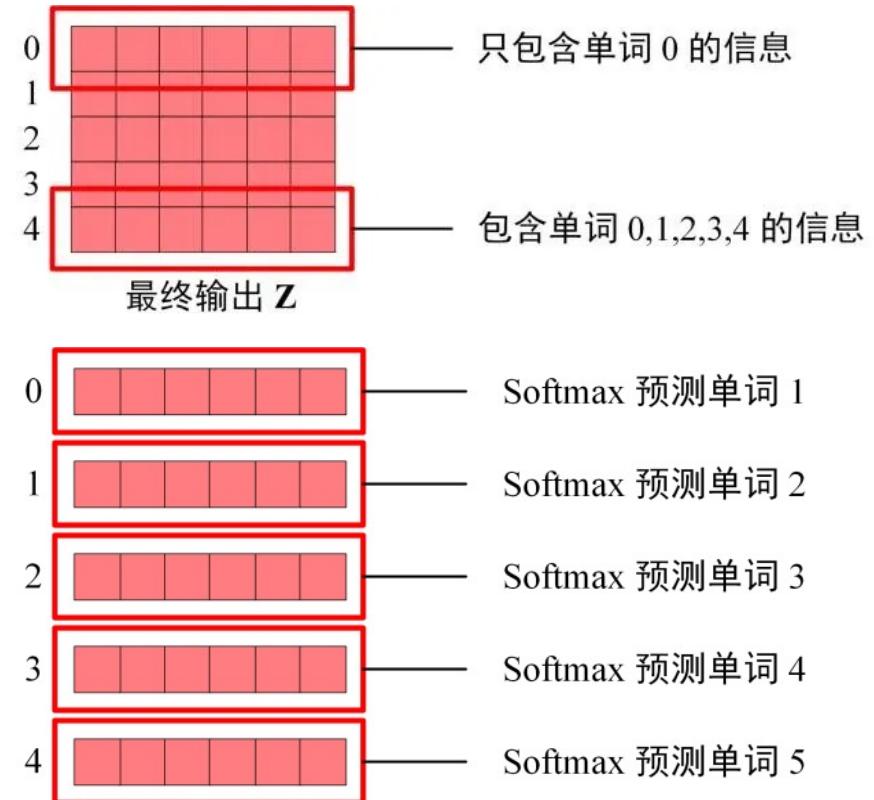


Transformer模型——Decoder

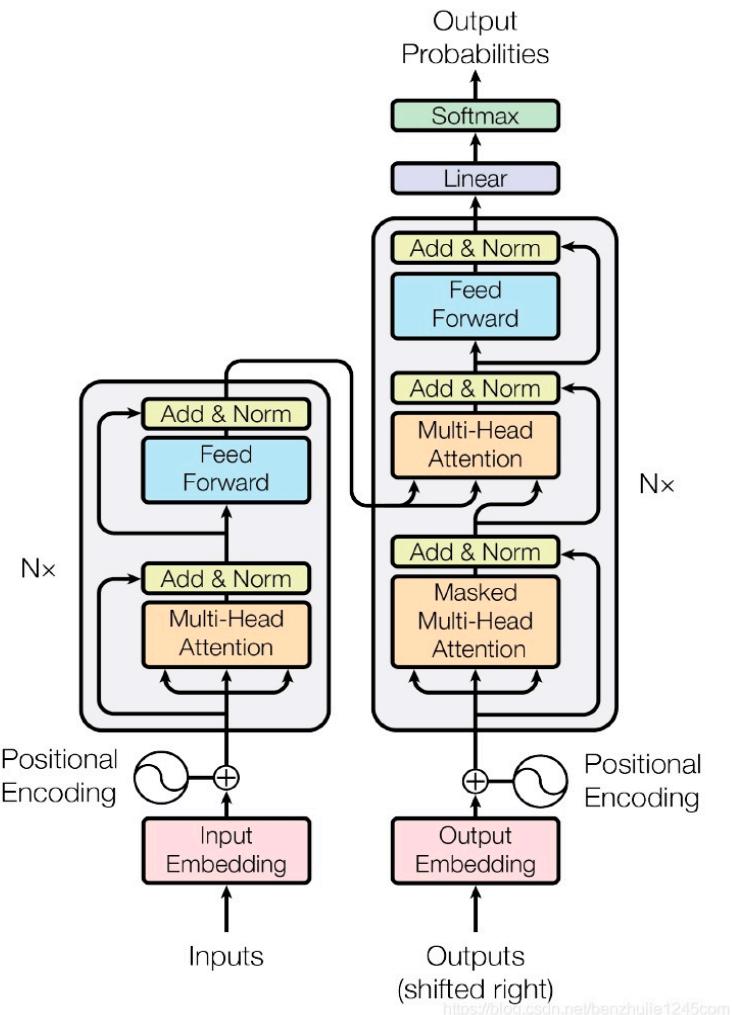


Output

在当前位置输出一个词表上的概率分布，代表预测每个词的概率。



Transformer模型——Decoder



Model	BLEU		Training Cost (FLOPs)									
	EN-DE	EN-FR	EN-DE	EN-FR								
ByteNet [15]	23.75											
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$								
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$								
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$								
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$								
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$								
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$								
ConvS2S Ensemble [8]	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	41.0	$2.3 \cdot 10^{19}$									
	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)									300K	4.92	25.7	
big	6	1024	4096	16						4.33	26.4	213

思考：Transformer里为什么用Layer Normalization？

Layer Normalization蕴含着独立同分布的假设，即输入的不同特征在分布上是相似的。

Transformer使用Layer Normalization的原因：

- 保证训练和推理阶段的一致性：Batch Normalization在训练和推理过程中对每个小批次的统计特性进行归一化，而在推理时很难应用到单个样本上。相比之下，Layer Normalization对单个样本进行归一化，使得在训练和推理时具有一致的表现，更易于应用于在线推理或单个样本的情况；
- 序列数据的适应性：Transformer中的自注意力机制对序列数据进行操作，不同位置的特征具有不同的重要性。Layer Normalization逐位置对特征进行归一化，更适应序列中不同位置的变化和特征重要性的差异；
- BN更加注重样本之间的相关性，但是NLP领域内输入的batch样本之间对应特征通常没有什么相关性，然而样本内不同的token之间却又相关性，因此采用LN更合适。

思考：Transformer时间复杂度是多少？如何进行优化？

要将自注意力（self-attention）的复杂度从 $O(n^2)$ 降低到 $O(n)$ ，可以采用以下有效方案：

- **自注意力矩阵的稀疏化：**在计算自注意力时，可以通过设置适当的阈值，将注意力权重矩阵中接近于零的元素设为零，从而将自注意力矩阵稀疏化。这样做可以减少计算量，并将复杂度从 $O(n^2)$ 降低到 $O(n)$ 。注意力矩阵的稀疏化方法可以通过使用稀疏矩阵存储格式（如COO、CSR等）来实现。

CNN	$O(kLd^2)$	$O(1)$	$O(\log_k(L))$
模型	每层复杂度	序列操作数	最大路径长度

- **自注意力的分块计算：**将输入序列划分为较小的块，在每个块内部计算自注意力，而不是对整个序列进行计算。这种分块计算可以减少计算复杂度。在计算自注意力时，每个块只需要关注其自身和相邻块的元素，而不必考虑整个序列的所有元素。

- **近似自注意力：**使用一些近似方法来计算自注意力，以减少计算复杂度。例如，可以使用低秩近似方法（如SVD、LU分解等）对注意力权重矩阵进行近似，从而将计算复杂度从 $O(n^2)$ 降低到 $O(n)$ 。

- **基于局部性的自注意力：**在计算自注意力时，可以引入局部性假设，即假设每个位置的注意力只与其附近的一小部分位置有关。通过限制注意力的作用范围，可以将复杂度降低到 $O(n)$ 。这些方法可以结合使用，以进一步降低自注意力的复杂度。但需要注意的是，降低复杂度的同时可能会引入一定的信息损失或近似误差，因此需要在具体应用中进行权衡和调整。

思考：Multi-head Attention的代码实现？

本节课你掌握了多少？

- 了解什么是Transformer？
- Transformer与CNN & RNN的区别？
- Transformer的输入表征有哪些，分别有什么作用？
- Transformer如何解决位置表征问题？
- 什么是Self-Attention？
- Self-Attention中的QKV分别是什么？
- Transformer的时间复杂度如何降低？
- Multi-head的作用是什么？
- Transformer中的Layer Normalization的作用？
- Masked Attention的作用是什么？
- Feed Forward Layer与残差链接是什么？
- Encoder与Decoder之间如何进行信息传递？
- Transformer如何进行文本生成的？
- 为什么Transformer可以并行训练？
- Self-Attention的代码实现？