

软性注意力机制

所谓注意力机制,就是在输入的信息中选择有用的信息,交由神经网络处理.

如 $X = [x_1, x_2, \dots, x_N]$ 中只选择一些进行计算

(一) 普通模式

软性注意力不是随便挑几个信息,而是将所有信息进行加权平均后输入神经网络

硬性注意力就是选择一些信息(如随机选择,或选概率最高的信息)

注意力值的计算分为两步: (1) 在所有输入信息上计算注意力分布

(2) 根据注意力分布来计算输入信息的加权平均

1. 注意力分布

把输入向量 X 看作一个信息存储器,现在给定一个查询向量 q , 用来查询并选择 X 中的某些信息。软性注意力就是从所有信息中都抽一些,但相关信息抽取得多一些。

定义一个注意力变量 $z \in [1, N]$ 表示被选择的信息的索引位置, $z=i$ 表示选择第 i 个位置的信息, 计算在给定 q 和 X 的情况下, 选择第 i 个输入信息的概率 α_i

$$\begin{aligned}\alpha_i &= p(z=i | X, q) = \text{softmax}(s(x_i, q)) \\ &= \frac{\exp(s(x_i, q))}{\sum_{j=1}^N \exp(s(x_j, q))}\end{aligned}$$

其中 $s(x_i, q)$ 是注意力打分函数, 有以下几种形式:

加性模型 $s(x_i, q) = V^T \tanh(Wx_i + Uq)$

点积模型 $s(x_i, q) = x_i^T q$

缩放点积模型 $s(x_i, q) = \frac{x_i^T q}{\sqrt{d}}$

双线性模型 $s(x_i, q) = x_i^T W q$

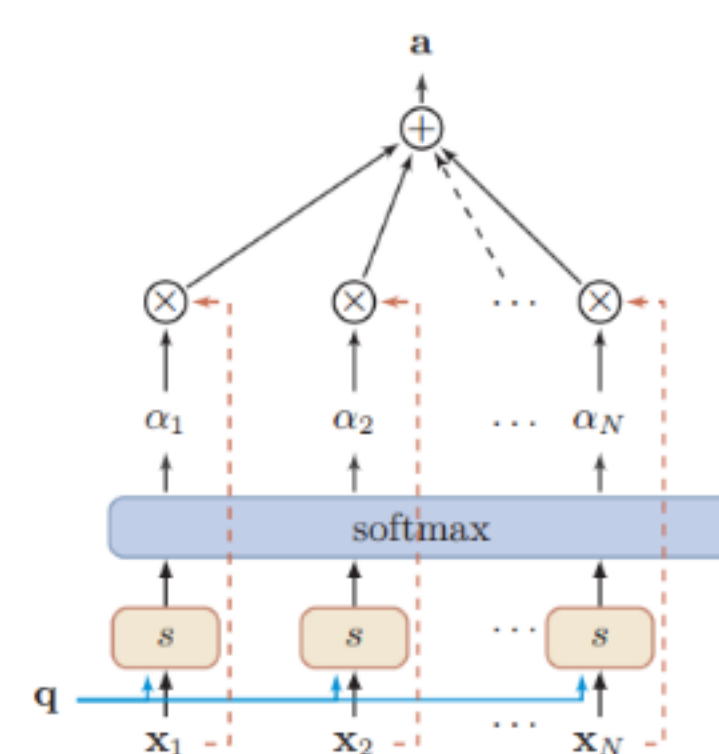
W, U, V 都是可学习的网络参数, d 是输入向量的维度。

于是我们得到了 α_i 构成的概率分布向量 $[\alpha_1, \alpha_2, \dots, \alpha_N]$, 称为注意力分布

2. 加权平均

使用注意力分布对 X 进行加权平均, 得 Attention 值。

$$\text{att}(X, q) = \sum_{i=1}^N \alpha_i x_i$$



(二) 键值对注意力模式

用键值对来表示输入信息 $X_i = (K, V) = [(K_1, V_1), (K_2, V_2), \dots, (K_n, V_n)]$

$K(\text{key})$: K 代表 X_i 的索引信息, 物理层面它是一个高维数字向量, 语义层面它是 X_i 的“特征、属性”, 如同性、概念、实体等

$V(\text{Value})$: V 表示 X_i 的实际内容, V 是 X_i 被提炼、压缩、聚合后的结果
不直接用 X_i 是因为 X 一般维度不均, 要将其降/升维与网络空间对齐; 另一方面 X 要被解构成 K 和 V , 让各自的信息更干净

这时查询向量 Q 可以看作一个寻址器, 当 $K=Q$ 时, 便提出 V 来

软性注意力机制则是计算 K 与 Q 的相似度来决定要提取 V 中多少内容

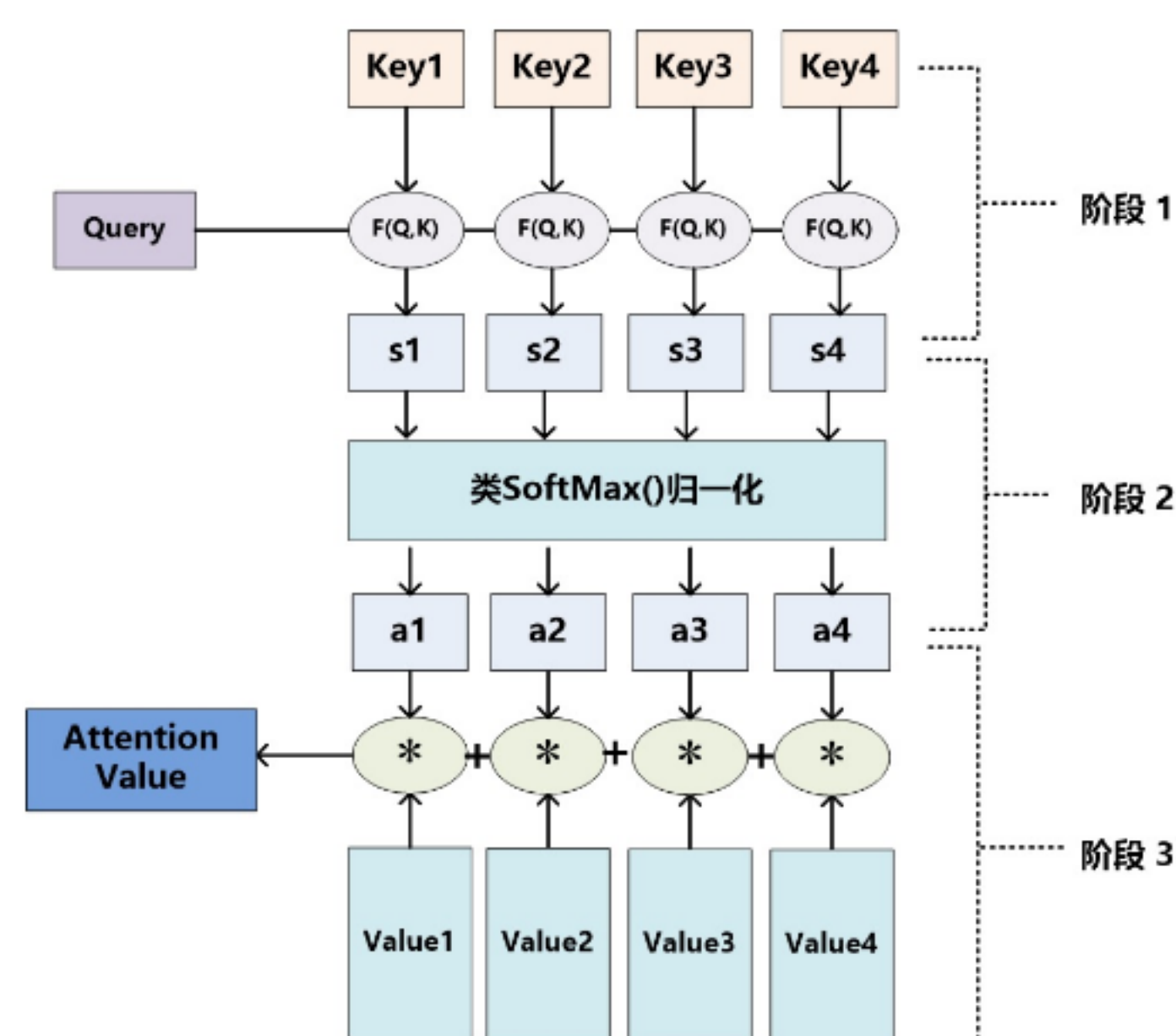
即计算每个 V 的权重, 最后加权求和得到 Attention 值

步骤: 1. 计算 Q 和 K 的相似度(注意力得分) $S_i = F(Q, K_i)$

相似度可以用上面提到的加性/点积模型或余弦相似度来计算

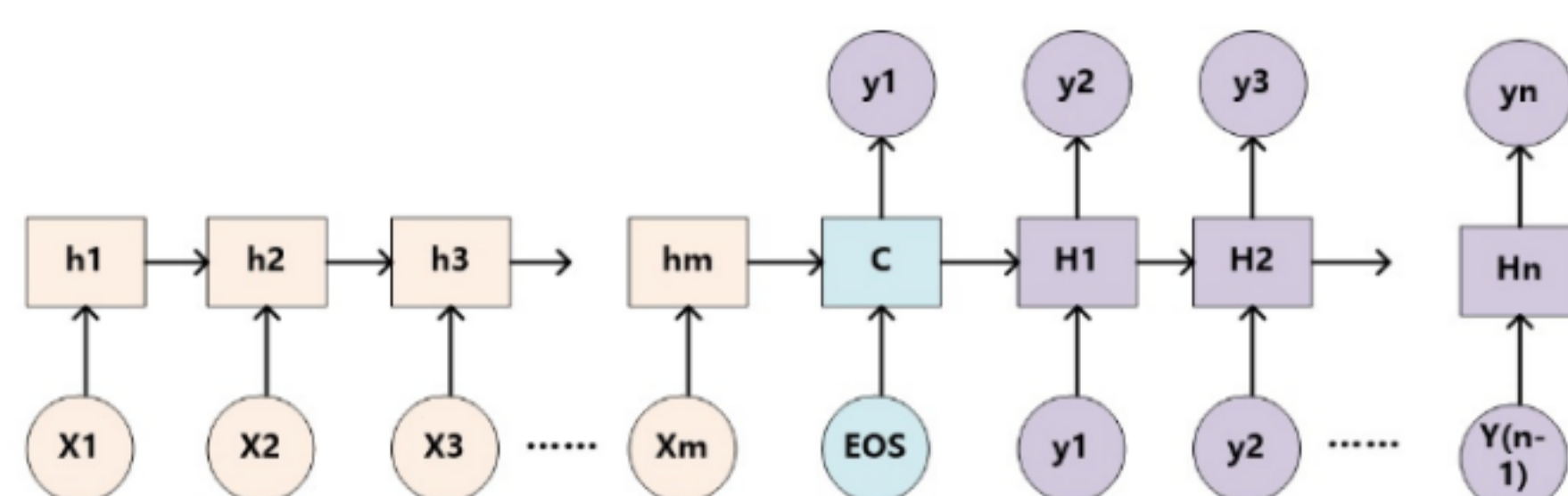
2. 计算权重 $\alpha_i = \text{softmax}(S_i) = \frac{\exp(S_i)}{\sum_{j=1}^N \exp(S_j)}$

3. 加权求和 $\text{Attention}(K, V, Q) = \sum_{i=1}^N \alpha_i V_i$



Encoder - Decoder 框架

(一) 未引入注意力机制的 RNN E-D 框架



长度不固定的序列 $X \xrightarrow{\text{Encoder}}$ 长度固定的向量 $C \xrightarrow{\text{Decoder}}$ 长度不固定的序列 Y

给定输入序列 $X = [x_1, x_2, \dots, x_n]$, 这一解码-编码过程实际上是在求 $y =$

$[y_1, y_2, \dots, y_T]$ 的条件概率分布, 即

$$P(y) = P(y_1, y_2, \dots, y_T | x_1, x_2, \dots, x_n)$$

可以转化为: $P(y) = \prod_{t=1}^T P(y_t | \{y_1, \dots, y_{t-1}\}, c)$, 即在得到向量 c 和之前所有词 $\{y_1, y_2, \dots, y_{t-1}\}$ 后可以预测第 t 个词 y_t

可以用三个步骤来算这个概率 $P(y_t | \{y_1, \dots, y_{t-1}\}, c)$

1、将 x 中的元素输入 Encoder 的 RNN 网络中, 计算隐状态 h_t , 然后再把所有的隐状态 $[h_1, h_2, \dots, h_T]$ 整合为一个语义表示向量 c

$h_t = f_1(x_t, h_{t-1})$ 通过前一刻的隐状态和现在的输入更新隐状态

$c = g(\{h_1, h_2, \dots, h_T\})$ 通过 g (一般是直接取 h_T) 生成 c

2、Decoder 根据 c 、上一刻预测向量 y_{t-1} 和隐状态 s_{t-1} 更新隐状态

$$s_t = f_2(y_{t-1}, s_{t-1}, c)$$

3、根据 c 、 y_{t-1} 和 s_t 预测 y_t

$$P(y_t | \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

问题: 1、信息瓶颈: 所有信息都要挤进一个定长向量 c , 导致信息越多, c 的质量越低

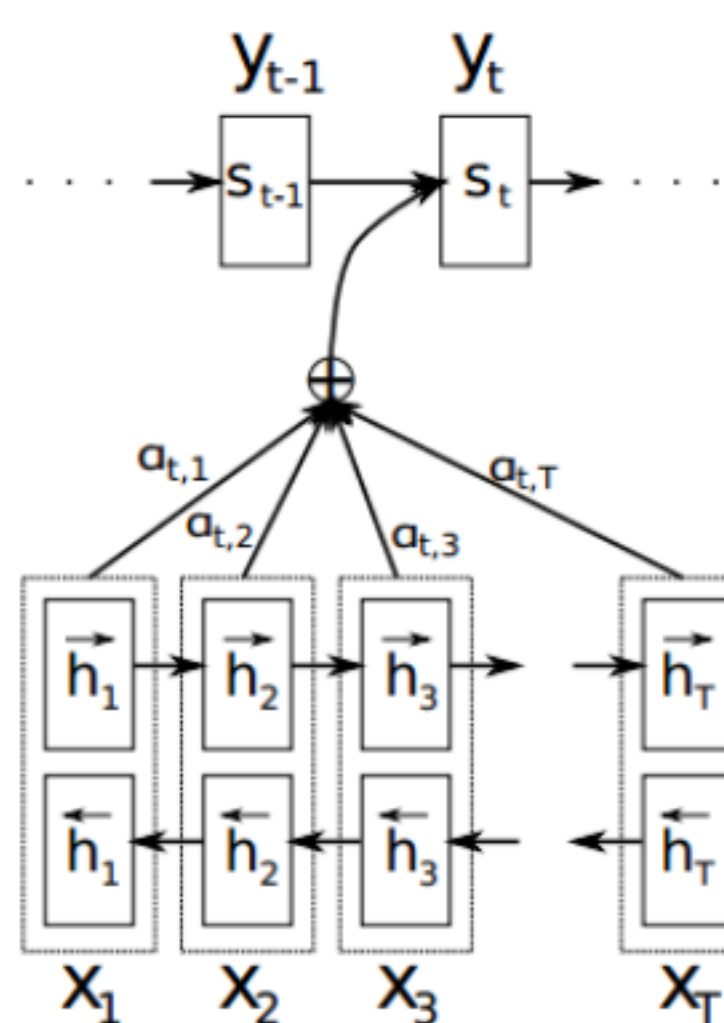
2、长距离依赖: 一个个输入逐个编码, 在输入序列很长时, 前面的输入信息会被大幅度稀释

3、注意力涣散: c 一成不变, 对于每个 y_t , c 中所有信息同等重要

(二) 引入注意力机制的 RNN E-D 框架

在预测 y_t 时, 不使用 c , 而是动态地更新 c_t , 即

$$P(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$



具体步骤如下:

1. 输入 X , 计算 Encoder 隐状态 h_t

$$\begin{aligned} h_t^{(1)} &= f_1(U^{(1)}h_{t-1}^{(1)} + W^{(1)}x_t + b^{(1)}) && \text{正向计算} \\ h_t^{(2)} &= f_2(U^{(2)}h_{t-1}^{(2)} + W^{(2)}x_t + b^{(2)}) && \text{反向计算} \\ h_t &= h_t^{(1)} \oplus h_t^{(2)} && \text{向量拼接} \end{aligned}$$

2. 计算 Decoder 隐状态 s_t

$$s_t = f_3(y_{t-1}, s_{t-1}, C_t)$$

3. 更新 C_t

$$e_{ij} = a(s_{t-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^K \exp(e_{ik})}$$

$$C_i = \sum_{j=1}^T \alpha_{ij} h_j$$

e_{ij} 表示用上一刻的状态 s_{t-1} 与输入序列的状态去匹配, 计算与每个词输入时的状态的匹配得分

α_{ij} 是对 e_{ij} 的归一化结果, 也即注意力分布

C_i 是根据 α_{ij} 对输入句子里所有状态进行加权求和, 得出当前时刻的左隔上下文向量 C_i

该过程便可以得出预测词更应该关注输入序列的哪个词

4. 预测

$$P(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, C_i)$$

注: 以上流程为 $s_{t-1} \rightarrow C_t \rightarrow s_t$ 这样一个小循环, 在初始时:

$$s_0 = h_T \quad y_0 = \langle \text{SOS} \rangle \text{ 向量} \quad h_j = \text{已知}$$

自注意力机制

前面我们提到过键值对注意力模式, 在传统的交叉注意力机制中, Q 通常来源于 Decoder, 而 K, V 则来源于 Encoder。在自注意力机制中, K, Q, V 全都来自 Encoder。

自注意力全局依赖建模能力更强 (无论隔多远, 模型都可捕捉到语义关系)、训练并约化 (矩阵运算, 无需按 t 分步计算)、可以根据不同上下文赋予同一词不同语义向量

复习一下Attention值的计算:

1. 计算 Q 和 K 的相似度 $S_i = F(Q, K)$

2. 计算权重

$$\alpha_i = \text{softmax}(S_i) = \frac{\exp(S_i)}{\sum_{j=1}^N \exp(S_j)}$$

4. 加权求和

$$\text{att}(K, V), (Q) = \sum_{i=1}^N \alpha_i V_i$$

$$\text{即 att} = \sum_{i=1}^N \alpha_i V_i = \sum_{i=1}^N \frac{\exp(S(K_i, Q))}{\sum_j \exp(S(K_j, Q))} V_i$$

一般来说 S 用的是点积模式, 即 $S(K_i, Q) = \frac{Q \cdot K_i}{\sqrt{d_k}}$

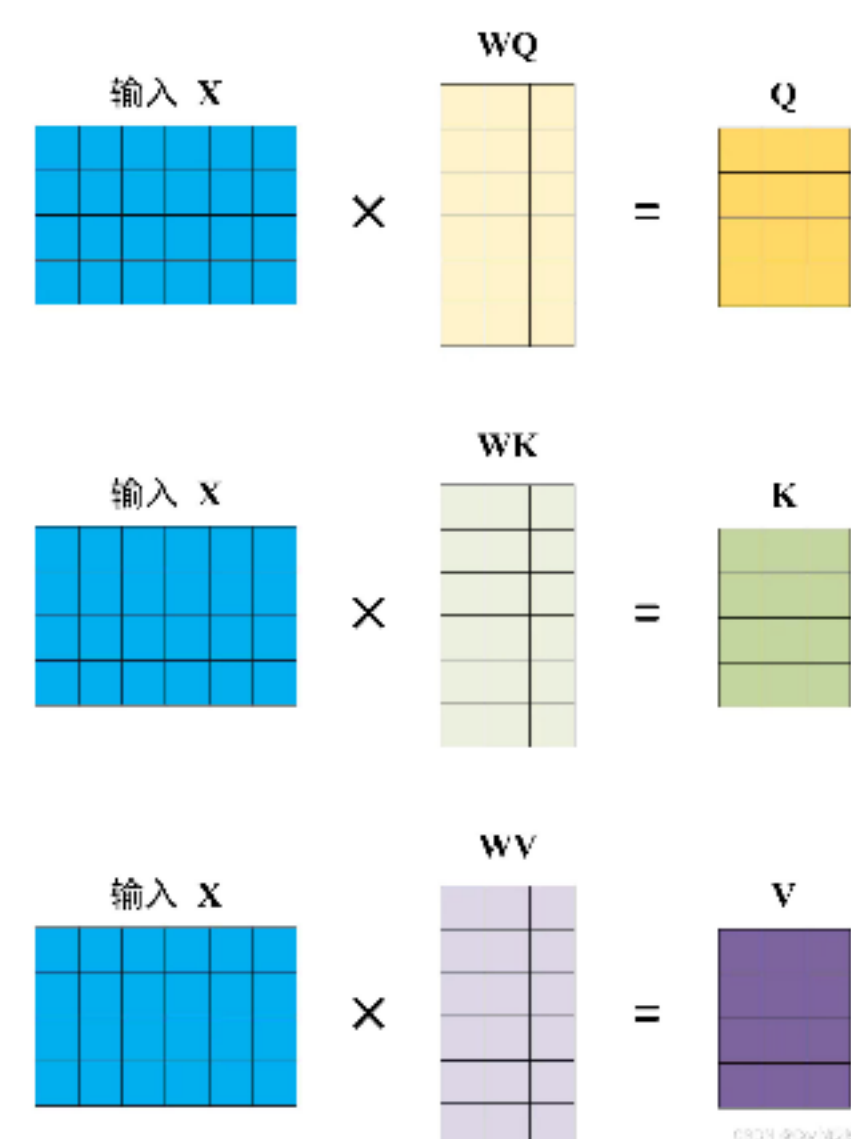
最终可以写成:

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

K、Q、V 矩阵运算

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

W^Q, W^K, W^V 是模型在训练中学习到的参数矩阵, 在推理中不会变化



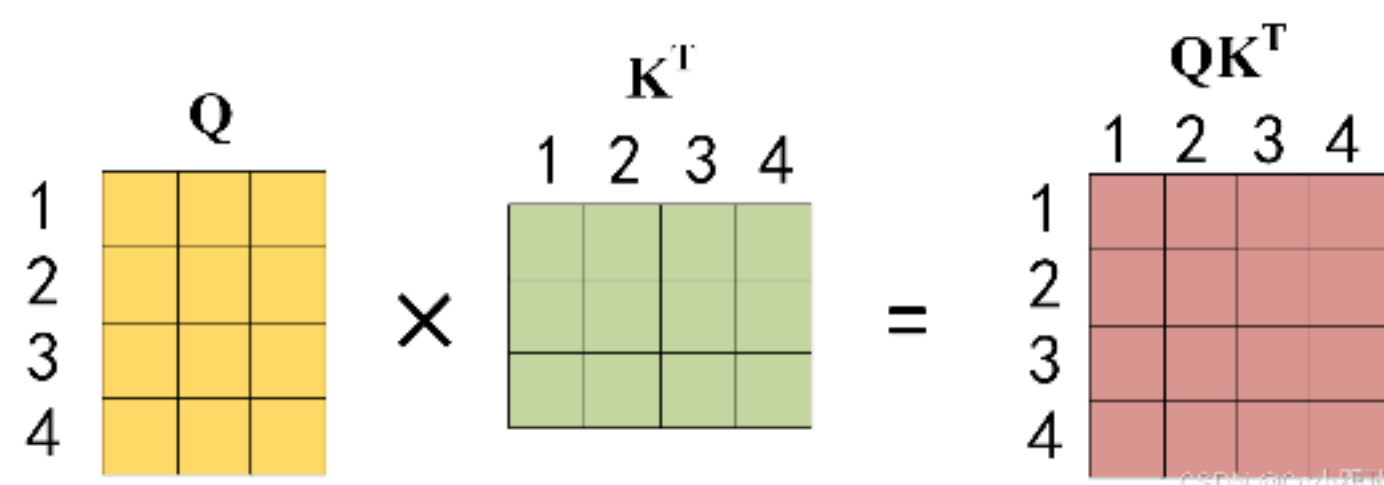
self-Attention 架构

我们回到该架构来拆解流程

0. 输入 X, 算出 K、Q、V

1. MatMul

Q 与 K^T 矩阵相乘, 得到相似度矩阵



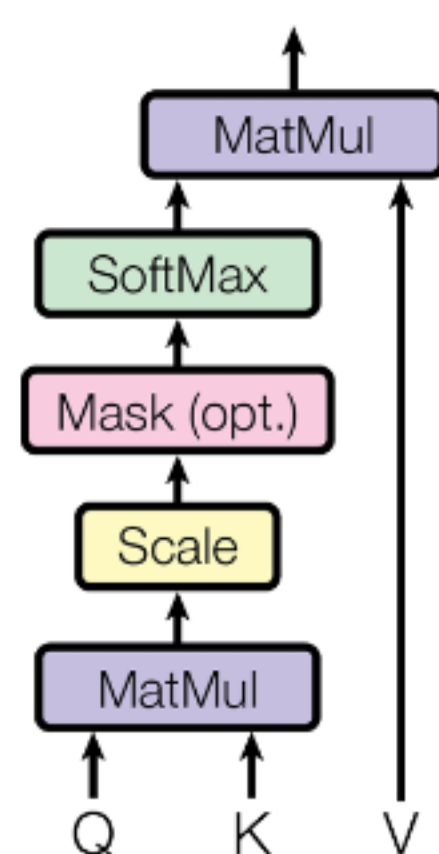
2. Scale

将 QK^T 除以 $\sqrt{d_k}$

3. Mask

对得分矩阵进行掩码处理, 遮蔽一些无意义的标签和未来信息 (自回归生成不能提前看未来信息, 具体来说, 是用一个上三角矩阵形式)

Scaled Dot-Product Attention



的Mask遮住未来位置)

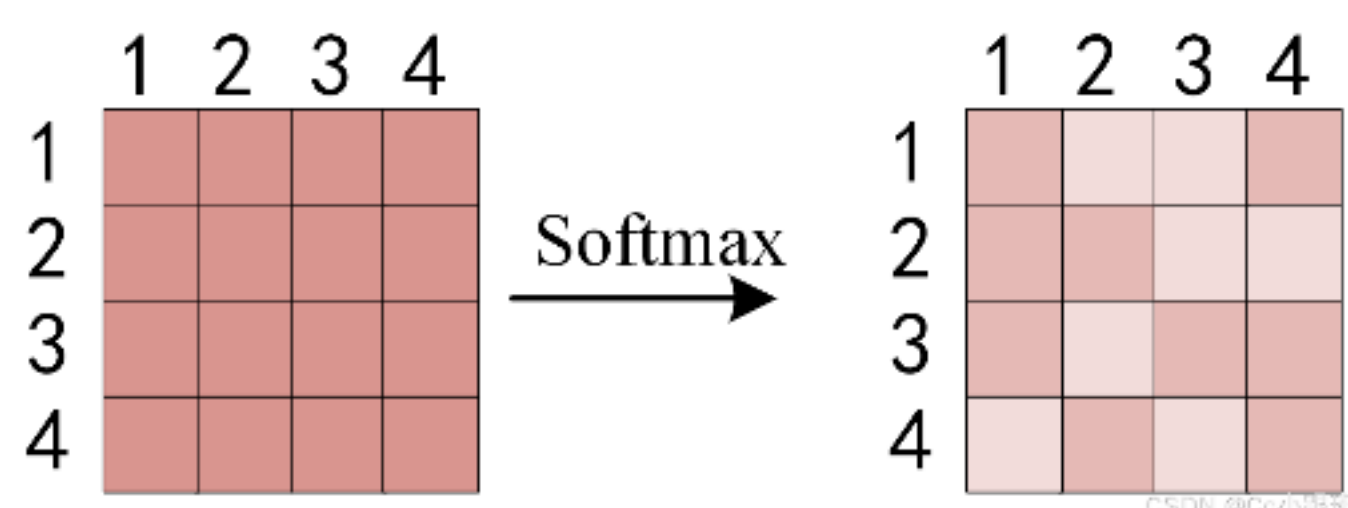
训练阶段是一定要Mask的,模型正在生成 $\hat{y}_t$,若不把答案 y_t 遮住,那模型会直接抄答案

推理阶段从原理上来说不用,因为未来信息本就没有生成,但是为了配合一些并行计算、批量处理的工具依然要Mask.而且Mask还可以处理一些无信息的标签(如占位符<PAD>)

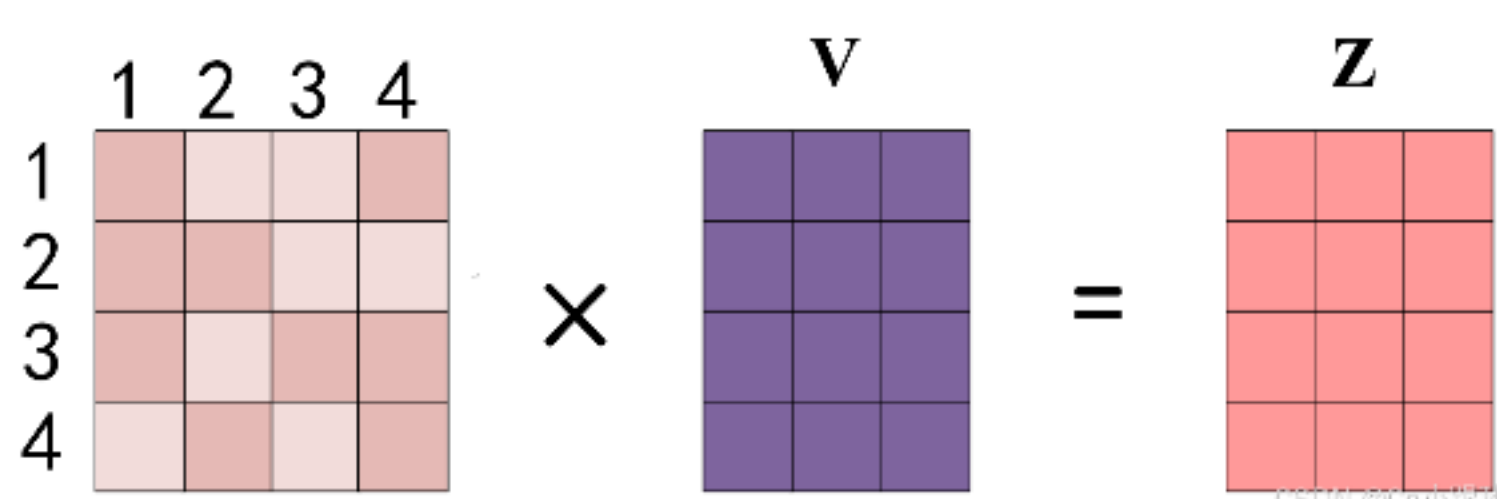
数学角度上,Mask就是一个矩阵,在无需遮蔽的位置上为0,否则为 $-\infty$.在Mask阶段,将该矩阵与得分矩阵相加.

4. $\text{Softmax}(\frac{QK^T}{\sqrt{d_k}})$ 得到权重矩阵

第 i 行表示第 i 个词与其它所有词的权重系数



5. 将权重矩阵与 V 相乘



$$\text{输出 } Z = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

多头注意力

'头'即是一组 W_Q, W_K, W_V 权重,如果我们只使用一个头,那它很可能将大部分注意力放在某一特性上,如词性,而忽略掉句子的其它信息。所以我们提出'多头'注意力,让每个头关注一个特性(如语法结构,语义关联,指代关系等),并最终整合所有信息

计算机制

设序列长度为 L 模型隐藏层维度是 $d_{\text{model}} = 512$,设头数为8

1. 降维拆分

不再使用512维的 K, Q, V ,而是给每个头分配独立的权重矩阵 W_i^Q, W_i^K, W_i^V
得 $XW_i^Q = Q_i, XW_i^K = K_i, XW_i^V = V_i$, $Q_i/K_i/V_i$ 都是64维($64 \times 8 = 512$)
(W_i^Q, W_i^K, W_i^V 的大小为 512×64)

2. 并行计算注意力

对于每个头,都计算 $\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$

每个 $head_i$ 的大小为 $L \times 64$

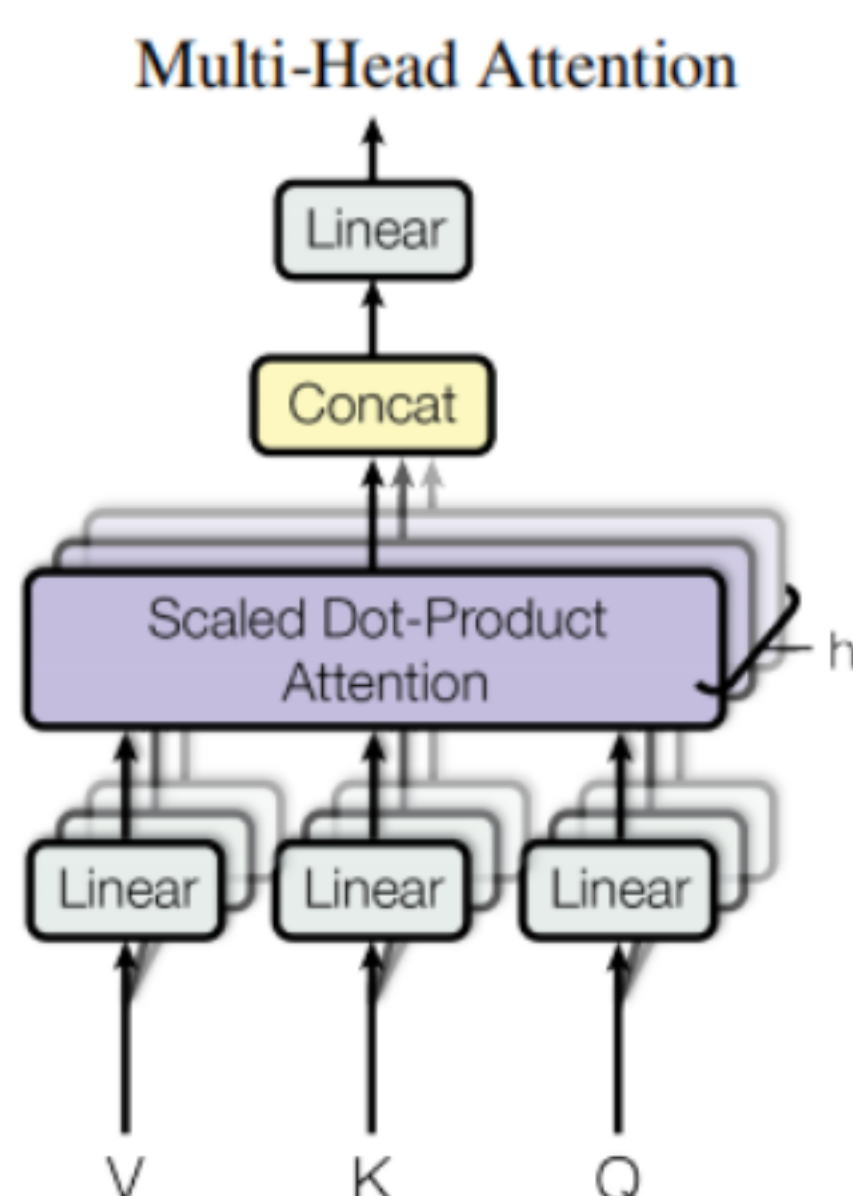
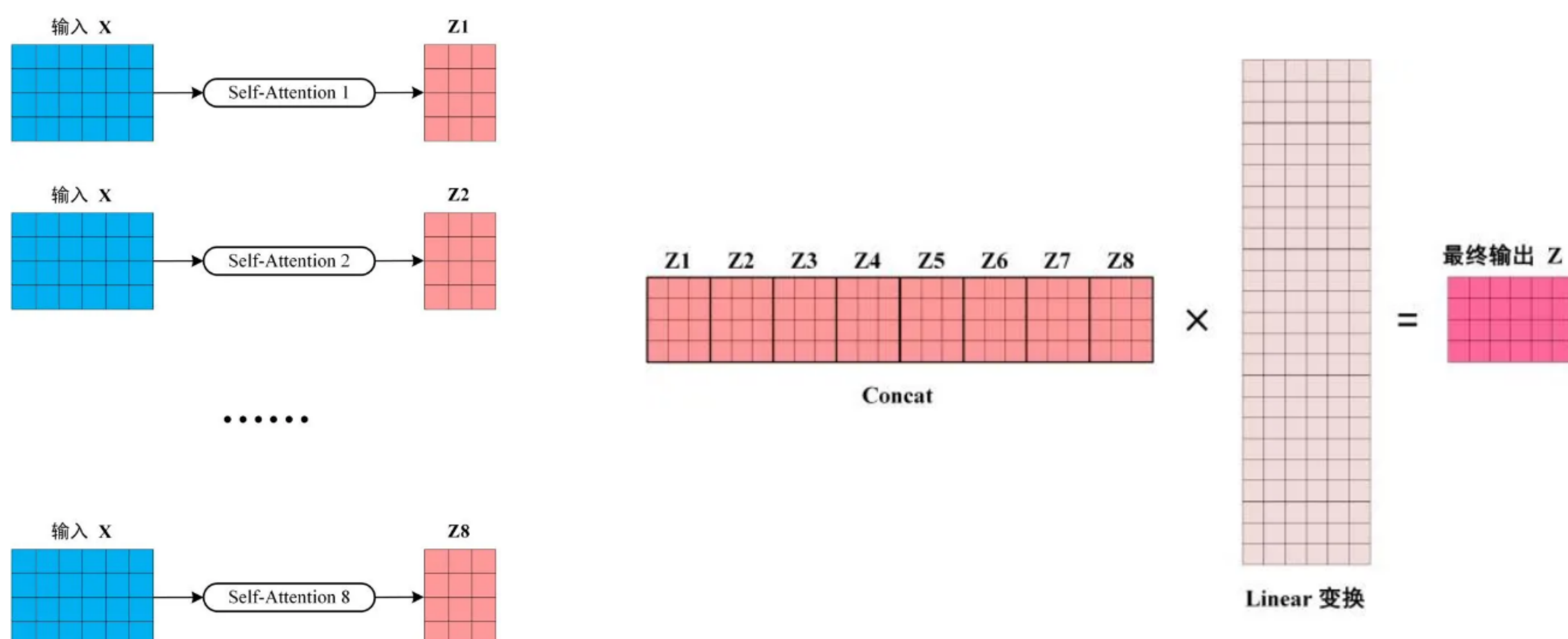
3. 拼接 Contact

将 8 个 $head_i$ 首尾相接, 重新变为 512 维

4. 输出

将这个 $L \times 512$ 的矩阵乘一个输出权重矩阵 $W^O (512 \times 512)$ 进行最终输出

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



位置编码

Self-Attention 会丧失语序信息, 原因如下:

我们设 $X = [x_1, x_2, x_3]$, 在 $S = QK^T$ 中, 任意两个词的注意力得分

$S_{ij} = (x_i W_Q)(x_j W_K)^T = x_i W_Q W_K^T x_j^T$, 我们发现, S_{ij} 只与 x_i, x_j 的值有关, 而跟 i, j 无关, 即我们交换 X 中的顺序, S 中的数值集合是不变的, 只是在矩阵

中的位置变了。

到] $Output = \sum_j \text{Softmax}(S_{ij}) V_j$ 中, 因为求和满足交换律, 所以结果不变。这意味着模型只关注谁与谁相关, 而不关心语序。于是我们必须引入位置编码。

实现思路: 将词的语义向量和位置向量相加并作为输入

$$Input = \text{WordEmbedding} + \text{PositionalEncoding}$$

位置向量: 1. 正弦余弦编码 (原始Transformer方案)

对于位置 pos (第几个词) 和维度索引 i (512维中的第几维)

$$\begin{cases} PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right) \\ PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right) \end{cases}$$

特点: • 唯一性, 每个 pos 的编码独一无二

• 有界性, PE 不可能数值过大而淹没语义信息

• 相对位置关系, 根据三角函数关系, $PE_{pos+k} = PE \times \text{旋转矩阵}$, 模型便可知道词的相对位置关系

• 波长渐变, 低维频率高, 变化快, 捕捉精细位置, 高维频率低, 变化慢, 捕捉全局位置 (想象时针和秒针)

• 无需学习参数, 可适应任意序列长度

2. 学习式绝对位置编码 (BERT, GPT-2)

每个位置对应一个可训练向量, 再和词嵌入相加

$$\text{PositionEmbedding} = \text{nn.Embedding}(\text{maxlen}, \text{dmodel})$$

优点: 简单灵活

缺点: 受最大长度限制

不能自然外推到比训练时更长的序列

每个位置独立学习, 相对位置关系只靠 Self-Attention

3. 相对位置编码 (TS)

在注意力计算中引入 token 之间的相对距离

$$\text{score}_{ij} = q_i k_j + b_{i-j}$$

b_{i-j} 是位置 i 和 j 之间的相对位置偏置

4. RoPE 旋转位置编码 (现代主流)

不把 PE 加到 WordEmbedding 上, 而是对 Query 和 Key 向量做旋转
把向量按维度两两分组, 对每一组应用一个与位置相关的旋转角

$$q'_{2i} = q_{2i} \cos(m\theta_i) - q_{2i+1} \sin(m\theta_i)$$

$$q'_{2i+1} = q_{2i} \sin(m\theta_i) + q_{2i+1} \cos(m\theta_i)$$

其中 m 是位置, $\theta_i = 10000^{-2i/d}$ 。Key 也做同样旋转

关键性质: $(R_m q)^T (R_n k) = q^T R_{n-m} k$

即 score 依赖相对位置 $n-m$, 使得 RoPE 既有绝对位置编码的实现又有相对位置编码的效果 (该效果优于正余弦编码)

5. ALiBi

类似相对位置编码, $\text{score}_{ij} = q_i k_j - m \cdot |i-j|$

Transformer

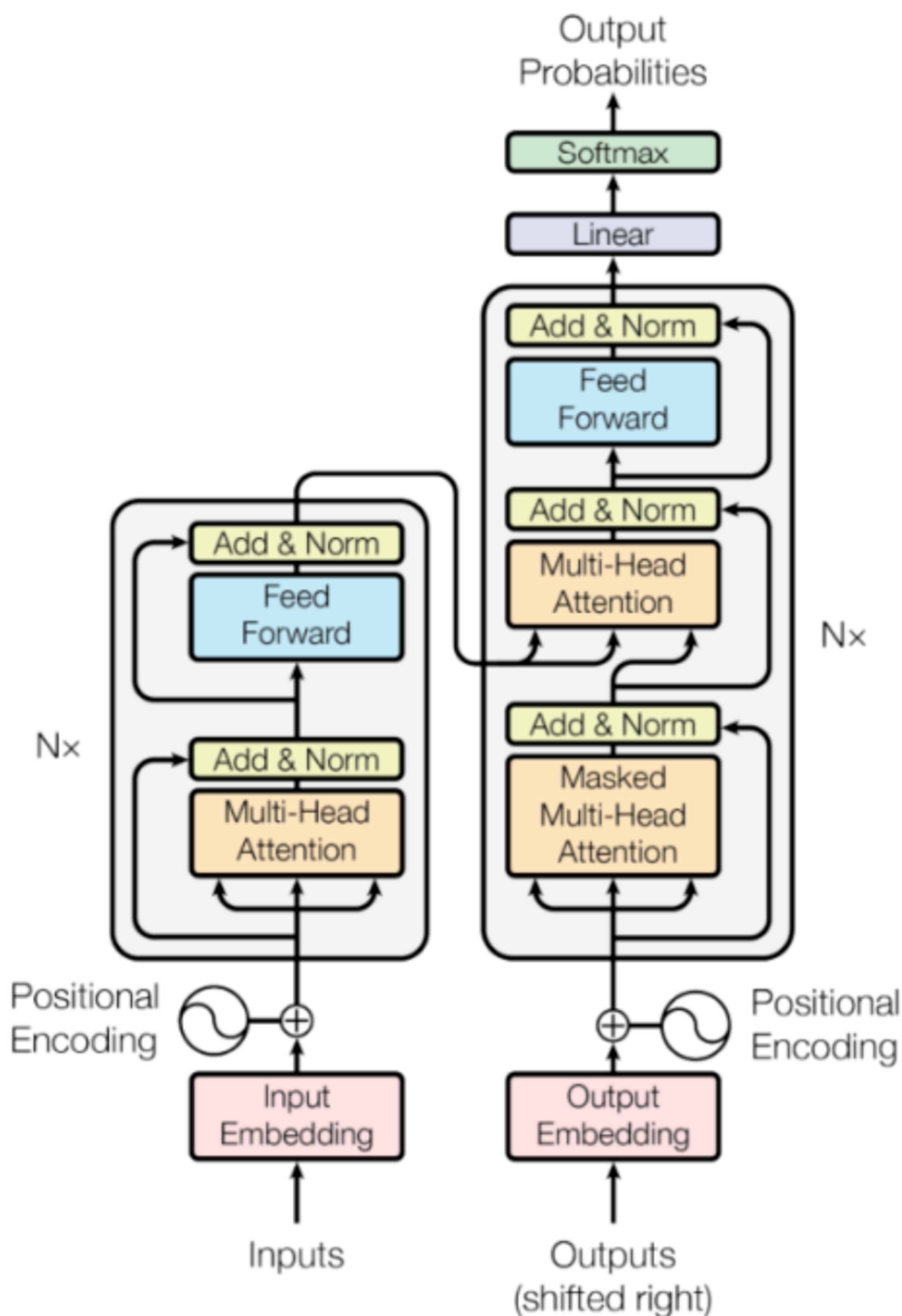
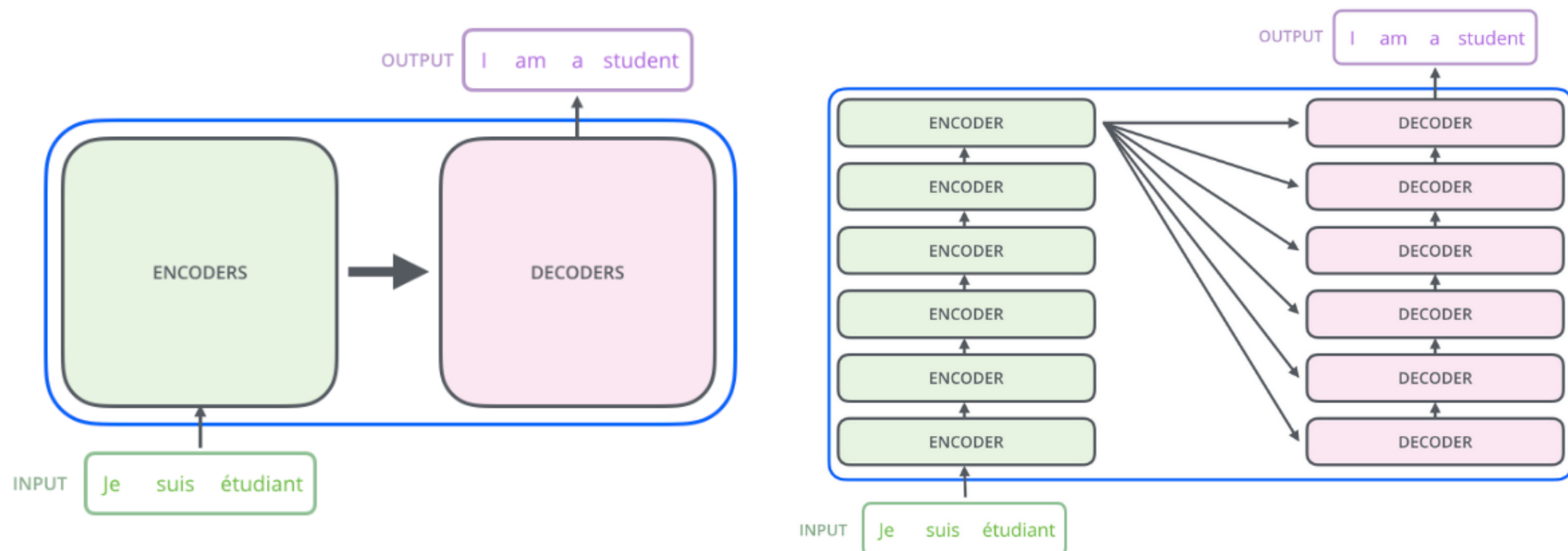
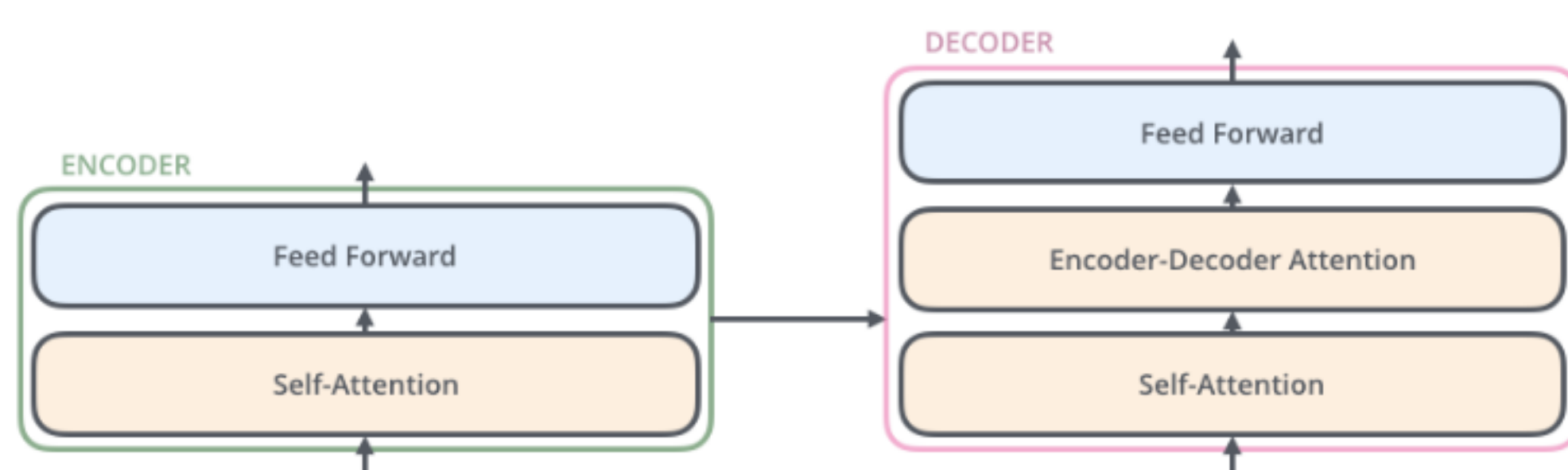


Figure 1: The Transformer - model architecture.

本质上是一个 E-D 架构, 但 E、D 都是由多层 Encoder/Decoder 组成的

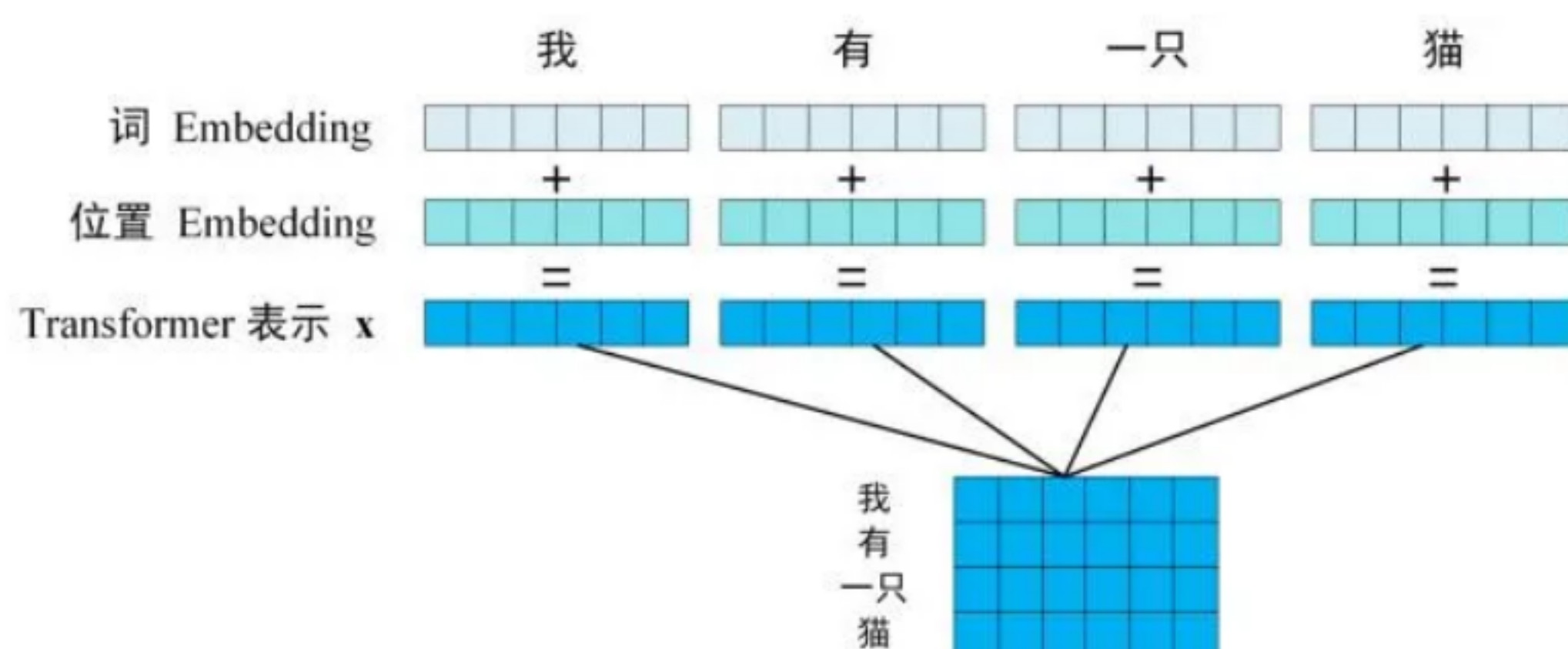


每个 Encoder 由 Self-Attention 层和 FFN 层两层组成, Decoder 则多一个 Attention 层
每一层后还接了一个残差连接层 ADD & Norm

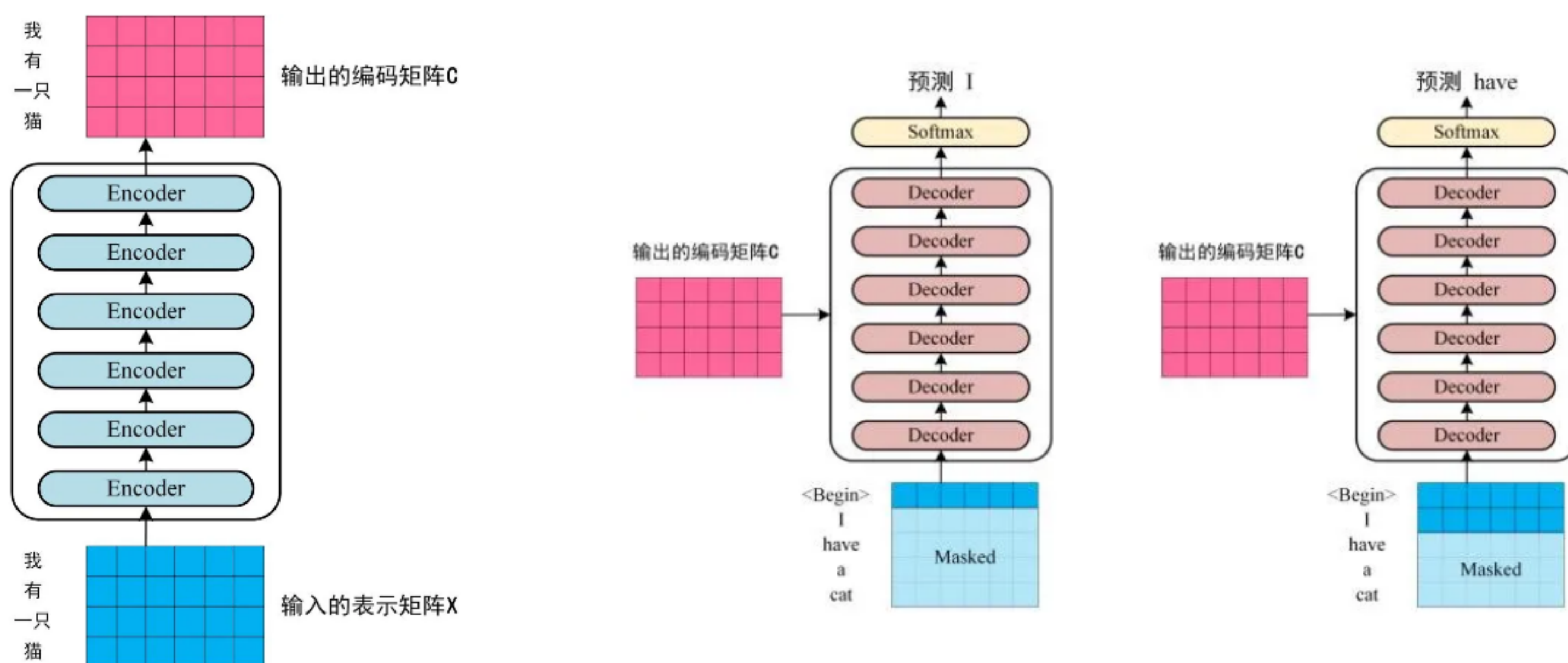


大致工作流程:

1. 计算每个单词的词向量和位置编码, 相加后得到输入



2. 输入 Encoder, 经过 6 个 Encoder block 后可以得到句子中所有单词的编码信息 C, C 的维度与输入完全一致



3. 将 C 输入 Decoder, Decoder 会根据当前翻译过的单词 1~i 翻译下一个单词 i+1, 并遮住 (Mask) i+1 之后的单词

Encoder 结构

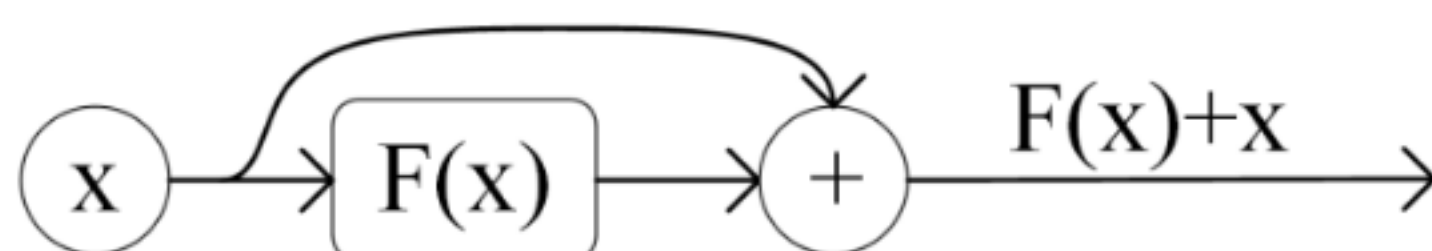
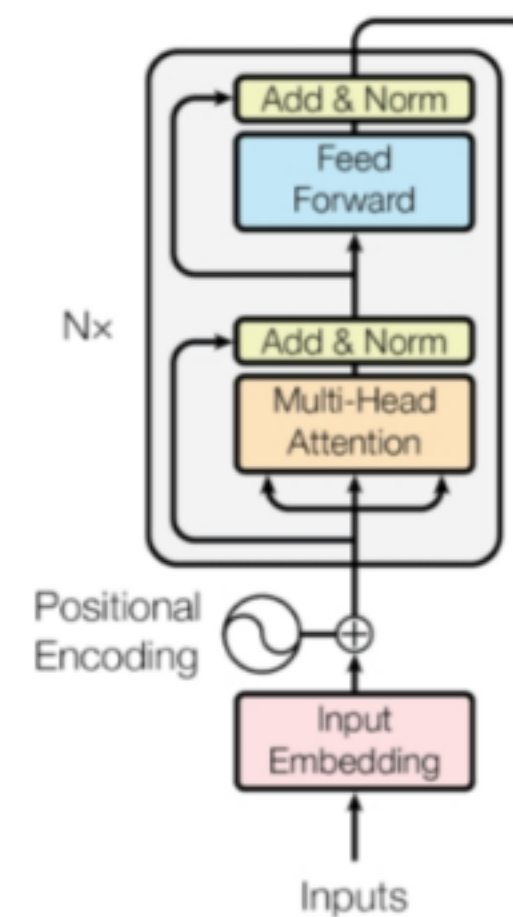
Add & Norm 层

由 Add 和 Norm 两部分组成, 其计算公式如下

$$\text{LayerNorm}(X + \text{MultiHeadAttention}(X))$$

$$\text{LayerNorm}(X + \text{FFN}(X))$$

Add 指 $X + \text{MultiHeadAttention}(X) / X + \text{FFN}(X)$, 是一种残差连接, 用于解决多层网络训练问题



Norm 指 Layer Normalization, 将每层神经元输入转化为均值和方差都一样的, 用于加快收敛, 公式如下 (具体展开有点多, 此处省略)

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i$$

$$\sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma_i \hat{x}_i + \beta_i$$

FFN 层

一个两层的全连接层, 第一层激活函数为 Relu, 第二层不使用激活函数
整层公式如下:

$$\max(0, xW_1 + b_1)W_2 + b_2$$

Decoder 结构

一个 Decoder block 由两个 Multi-Head Attention 和一个 FFN 构成

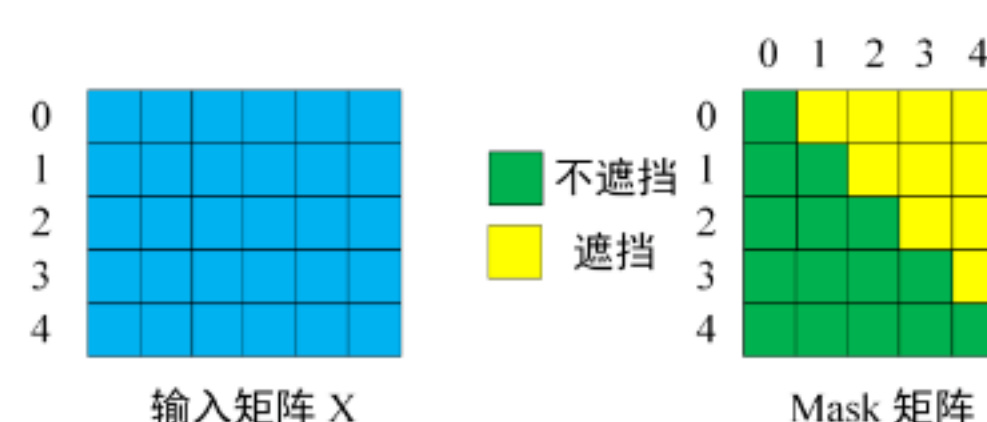
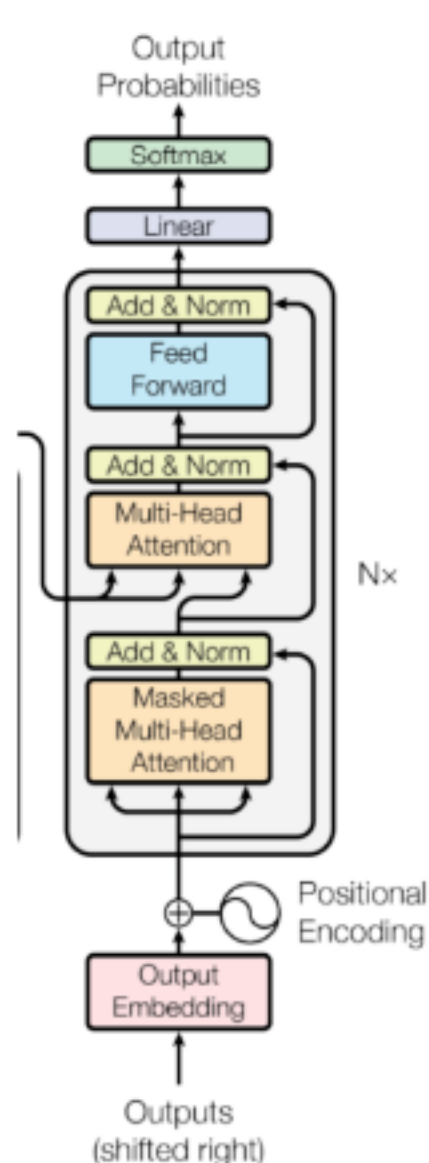
输入 = 非训练阶段仅是一个起始符 (< sos > / < bos >)

训练阶段为目标序列向右偏移一位的结果 (I love you → < sos > I love you)

第一个 Multi-Head Attention 层

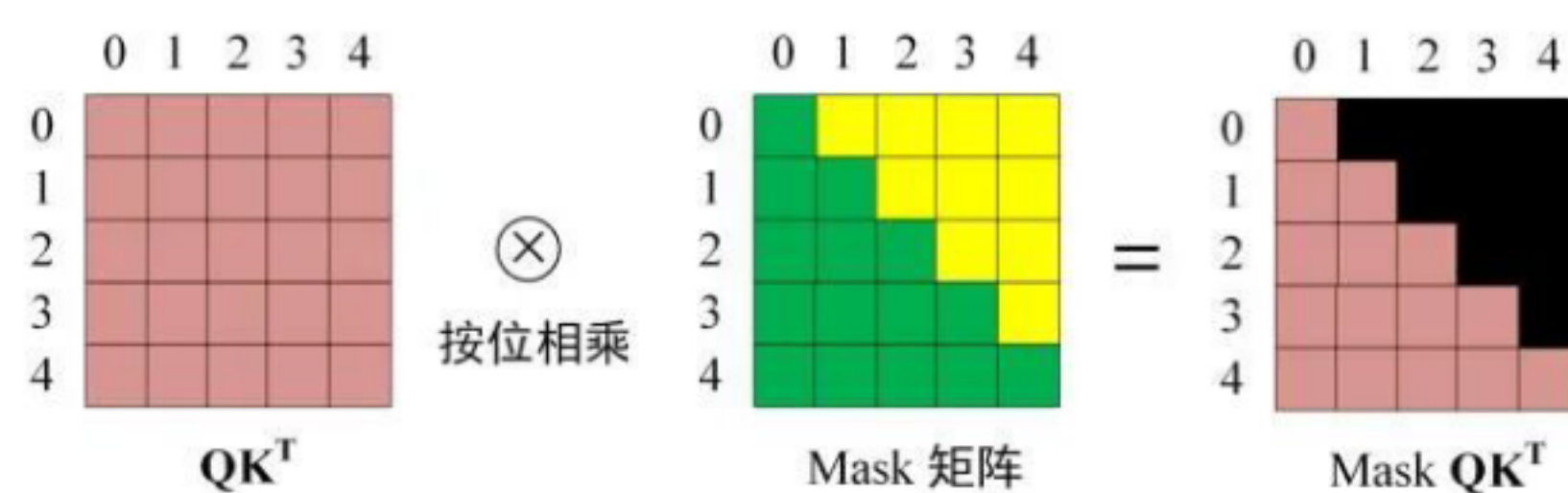
采用 Mask 操作, 具体步骤如下

第一步: 得到输入 X 和 Mask 矩阵



第二步：用 X 得到 Q, K, V ，计算 QK^T

第三步：将 QK^T 与 Mask 矩阵按位相乘，让第 i 个词不能得知 $i+1$ 及之后的信息，再 $\text{softmax}(\text{Mask } QK^T)$



第四步： $Z = \text{softmax}(\text{Mask } QK^T) V$

第五步：拼接各头的 Z_i ， $Z = \text{concat}(Z_1, \dots, Z_h) W^O$

第二个 Multi-Head Attention

用 Encoder 的输出 C 计算 K, V ，用第一个 Multi-Head Attention 的输出 Z 计算 Q ，后续计算完全一致

Linear 线性层

普通的全连接层，做一次仿射变换 $y = xW + b$ ，将 Decoder 的 d_{model} 维结果映射为词表大小 vocab-size

Softmax 预测输出

由于 Mask 的存在，第 i 个单词只知道它之前的信息。Softmax 层根据矩阵的每一行预测下一个单词

