Data Mining and Optimization

Lecture 9: Transformer

Liu Yang

Nanjing University

Spring, 2025

Table of Contents

注意力Seq2Seq模型的问题

- 在Transformer产生之前，基于RNN，尤其是基于LSTM和GRU等带有门控机制的循环神经网络架构的编码器-解码器结构，一直被认为是序列转换的最佳模型；
- 基于RNN的架构存在两个问题：
  - 难以开展并行处理，RNN属于序列模型，需要以一个接一个的序列化方式处理信息，注意力权重需要等待序列全部输入模型之后才能确定，即“从头看到尾”；
  - 容易造成历史信息的稀释，当两个相关词语相距较远时，RNN中存储的信息会不断受到稀释，翻译效果不佳。

Transformer的诞生

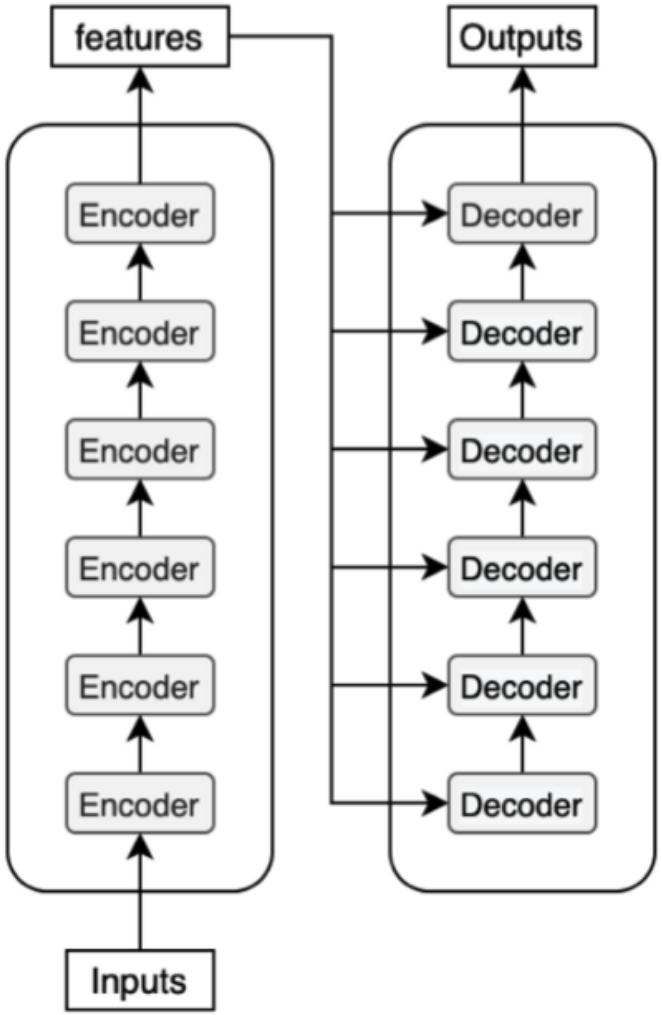
- 2017年，Ashish Vaswani等8名来自谷歌大脑和谷歌研究院的学者联合发表文章"Attention Is All You Need"，提出RNN结构不是必须的，标志着Transformer模型的诞生；
- Transformer的诞生可谓是NLP领域的“一声炸雷”，宣告了RNN时代的终结，在NLP领域引发了彻底“Transformer化"的风潮，诸多有着“逆天”表现的大语言模型都是基于Transformer模型构造的。

Table of Contents

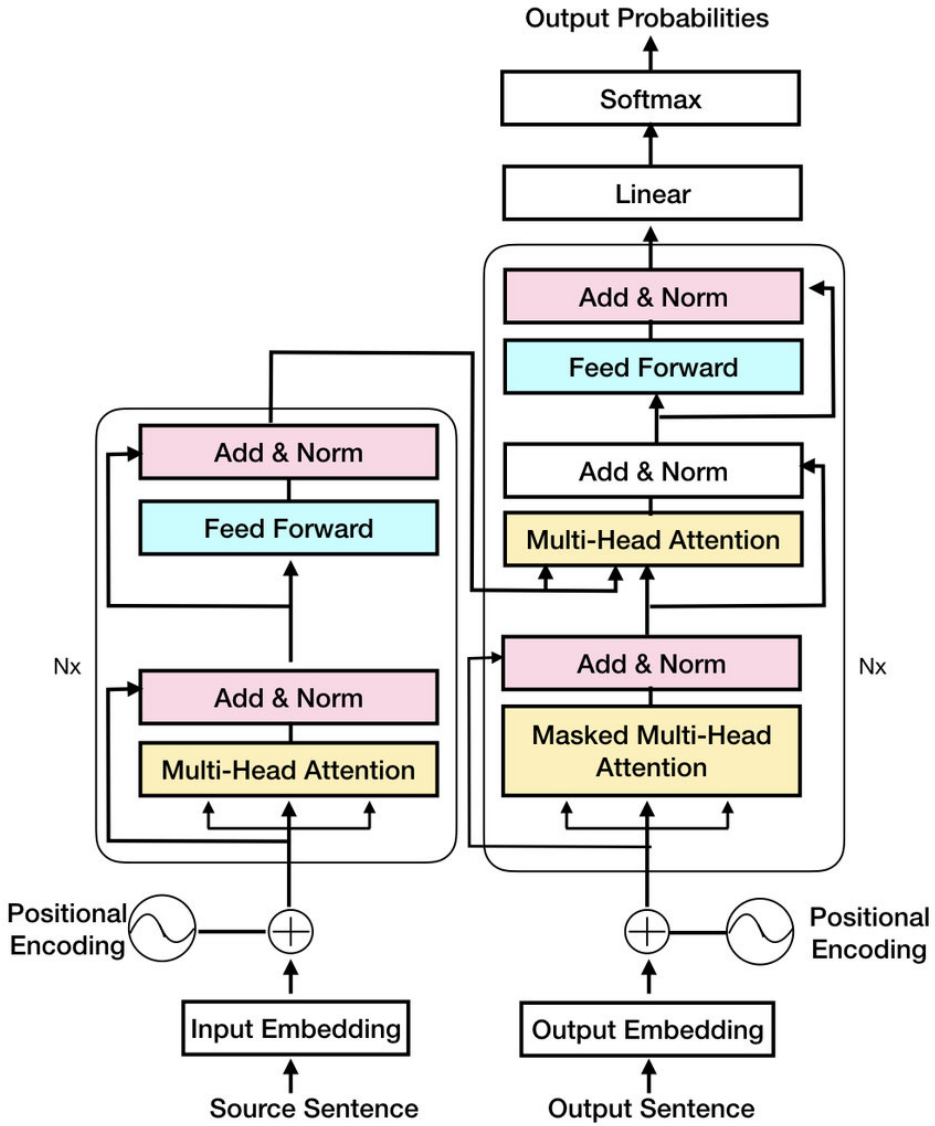
Transformer的编码器-解码器架构

- Transformer也采用编码器-解码器架构，但编码器和解码器不再是RNN，取而代之的编码器栈(encoder stack)和解码器栈(decoder stack)。所谓“栈”就是将同一结构重复堆叠多次，具体在Transformer中，编码器栈和解码器栈中分别包含了6个具有相同结构的编码器和解码器；
- 输入序列整体送入编码器栈，多个编码器对其进行逐级加工，每个编码器只以上一级编码器的输出作为输入（第一个编码器以原始序列作为输入）。编码器栈采用真正的并行方式处理数据，在每级加工时，看到的都是完整序列，输出的也是完整序列对应的特征表示；
- 解码器栈中每一个解码器（不包括第一个解码器）除了以上一级解码器的输出作为输入，还同时以整个编码器栈的输出（即最后一个编码器的输出）作为输入。解码器栈采用自回归(auto-regressive)方式逐个输出序列，即在生成当前的目标词汇时，会用到已经生成的全部词汇。

Transformer的编码器-解码器架构



Transformer的编码器-解码器架构



编码器结构

- Transformer编码器栈中每一个编码器包含两个以串联方式组织的子模块：
  - 多头注意力模块(Multi-Head Attention)：采用自注意力机制；
  - 全连接前馈网络；

  两个子模块外还添加了两个残差连接，并且在其后还分别设置有两个归一化操作；

- 对于某一编码器，假设输入和输出序列分别为$x = \{x_1, \cdots, x_T\}$和$y = \{y_1, \cdots, y_T\}$，其中T为序列长度，则上述子模块对数据的加工过程可表示为

$$\begin{aligned} x' &= LayerNorm(x + AttLayer(x)) \\ y &= LayerNorm(x' + FwdLayer(x')), \end{aligned} \tag{1}$$

  其中，$AttLayer(x)$表示作用在输入序列x上的多头注意力，其输出也是一个具有T个元素的向量序列，$LayerNorm(\cdot)$表示对序列中的向量进行“减均值，除标准差”的归一化操作。

编码器结构

- 在Transformer中，从词嵌入开始到每层的输入与输出，所有向量都具有相同的维度$d_{model} = 512$。在(1)中，$x, x', y \in R^{T \times d_{model}}$，即$T$个$d_{model}$的向量；
- $FwdLayer(x')$表示作用在注意力子模块输出序列x'之上的全连接前馈网络处理，即针对向量序列中的每一个512维向量逐个进行相同的操作，所有向量共享参数，而中间层的维度为$d_{ff} = 2048$。其中第一个全连接层操作后设置有ReLU激活函数。针对注意力输出序列中的第t个$(t = 1, \cdots, T)$向量x'_t，上述前馈神经网络的计算方式可表示为

$$FwdLayer(x'_i) = ReLU(x'_i W_1 + b_1) W_2 + b_2, \tag{2}$$

  - $W_1 \in R^{d_{model} \times d_{ff}}$和$W_2 \in R^{d_{ff} \times d_{model}}$均为线性变换的参数矩阵；
  - $b_1 \in R^{d_{ff}}$和$b_2 \in R^{d_{model}}$均为线性变换的偏置向量；
  - $ReLU(x) = \max\{0, x\}$为ReLU激活函数。

解码器结构

- Transformer解码器栈中每一个解码器包含三个以串联方式组织的子模块：
  - 掩码多头注意力(Masked Multi-Head Attention)；
  - 多头注意力；
  - 全连接前馈网络；

  三个子模块外还添加了三个残差连接，并且在其后还分别设置有三个归一化操作；

- 第一个掩码多头注意力模块，简单理解为多头注意力模块的“掩码版本”，其输入来自上一个解码器的输出；
- 第二个多头注意力模块的完整名称是“编码器-解码器注意力”(encoder-decoder attention)模块，该模块一方面以掩码多头注意力模块的输出作为输入，另一方面还有编码器栈的输出作为输入；
- 最后一个全连接前馈网络模块，其结构与工作方式与编码器中的全连接前馈网络相同。

Table of Contents

词嵌入

- Transformer中采取线性词嵌入方法，即$x_t W_E$；
- 首先将输入序列中每个词用独热编码(one-hot encoding)表示为一个$|V|$维的向量，记为$x_t \in R^{|V|}$，其中$|V|$为词典中不同词的数量，也叫词典长度；
- $W_E \in R^{|V| \times d_{model}}$为可学习的嵌入矩阵，上述变换将得到一个新的$d_{model}$维的嵌入特征表示，特别地，当$d_{model} << |V|$时，线性嵌入同时起到了降维作用；
- 例子：假设我们有一个小词典，长度$|V| = 4$，我们期望得到的词嵌入维度$d_{model} = 3$，再假设某词对应的独热编码为$x_t = (0, 1, 0, 0)$，那么基于线性变换的词嵌入可表示为

$$x_t W_E = (0, 1, 0, 0) \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} = (w_{21}, w_{22}, w_{23}), \tag{3}$$

不难看出，独热编码作用在嵌入矩阵，等价于行向量抽取。

位置编码

- 无论什么语言，词都是有序的，同一个词出现在句子中的不同位置，对其编码应当有所不同。比如“我爱我的女儿”，第一个“我”和第二个“我”，尽管都是“我”，但其语义是不同的，仅靠词嵌入向量表达不同位置的词是不够的，需要在模型中引入某种表达位置的机制；
- 对位置进行表示最常用的手段是在词嵌入向量的基础上再添加一个和其位置有关的矫正向量，包括位置嵌入(positional embedding)和位置编码(positional encoding)两种具体实现方式；
- 位置嵌入也是通过线性变换方式实现的，只不过这里独热编码的长度为最大句子所包含的词数，其中的独热元素位置为当前词在句子中的位置，位置嵌入矩阵也随着整个模型的训练进行一体化训练，GPT和BERT等大语言模型都是采用这种方式；
- 位置编码则是通过预先构造好的矫正向量实现的，这些矫正向量具有不同位置、不同取值的固定向量，Transformer模型即采用这种方式。

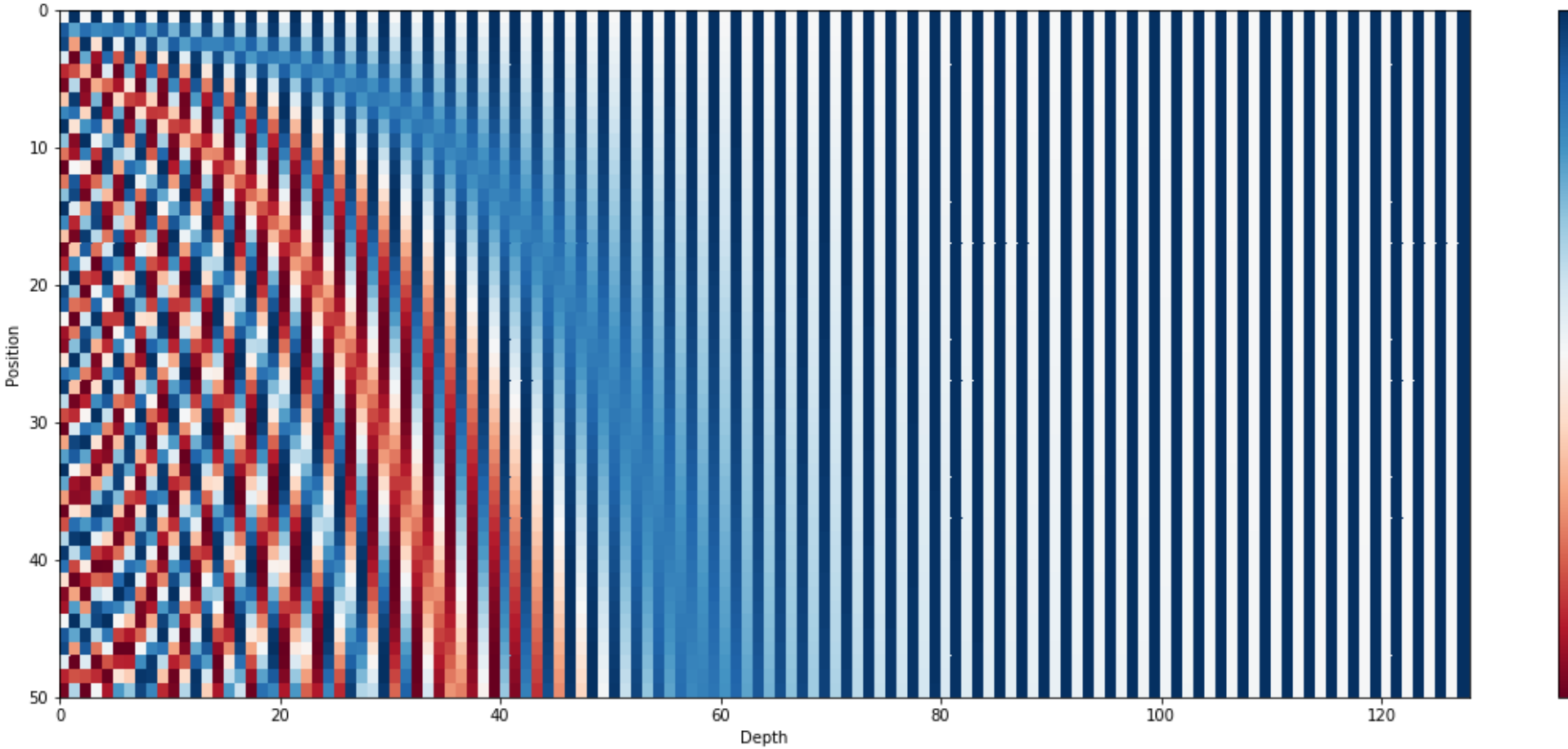
位置编码

- Transformer使用正弦和余弦函数来构造位置编码

$$
\begin{aligned}
PE(pos, 2i) &= sin\left(\frac{1}{10000^{2i/d_{model}}} pos\right) \\
PE(pos, 2i+1) &= cos\left(\frac{1}{10000^{2i/d_{model}}} pos\right),
\end{aligned} \tag{4}
$$

其中，pos表示位置，i表示特征维度，$PE(p, \cdot) \in R^{d_{model}}$表示给第$p$个词嵌入向量添加的位置矫正向量。

位置编码



Transformer的输入

面对一个输入序列，Transformer为其构造模型输入的过程为：

- 首先，针对序列中的每一个词，将其独热编码与此嵌入矩阵相乘，得到对应的词嵌入向量序列；
- 然后，按照词所在序列的位置，利用(4)，为每个词构造位置编码向量，词嵌入向量和位置编码向量的维度均为d_{model}；
- 最后，将上述两组向量求和，即得到带有位置信息的词向量序列，该向量序列即作为Transformer的输入。

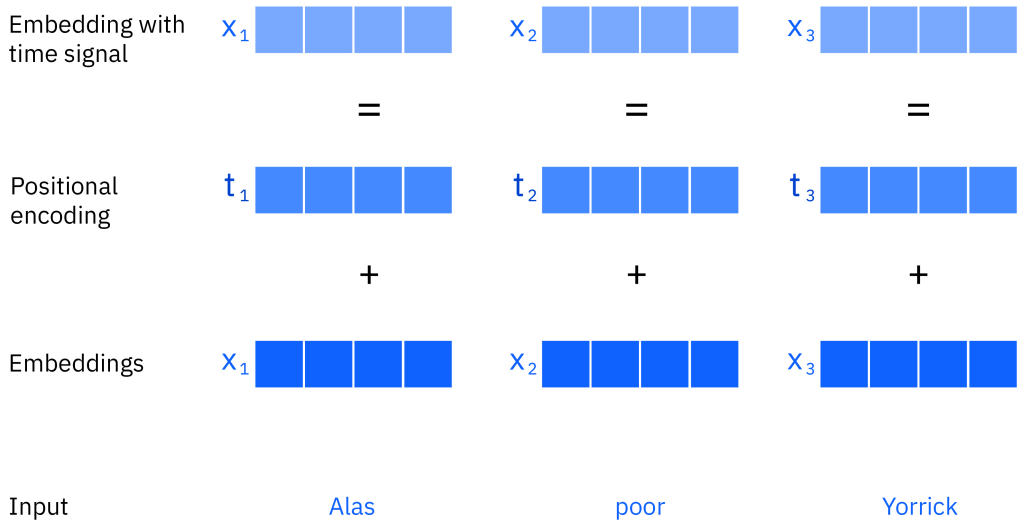


Table of Contents

Transformer的输出

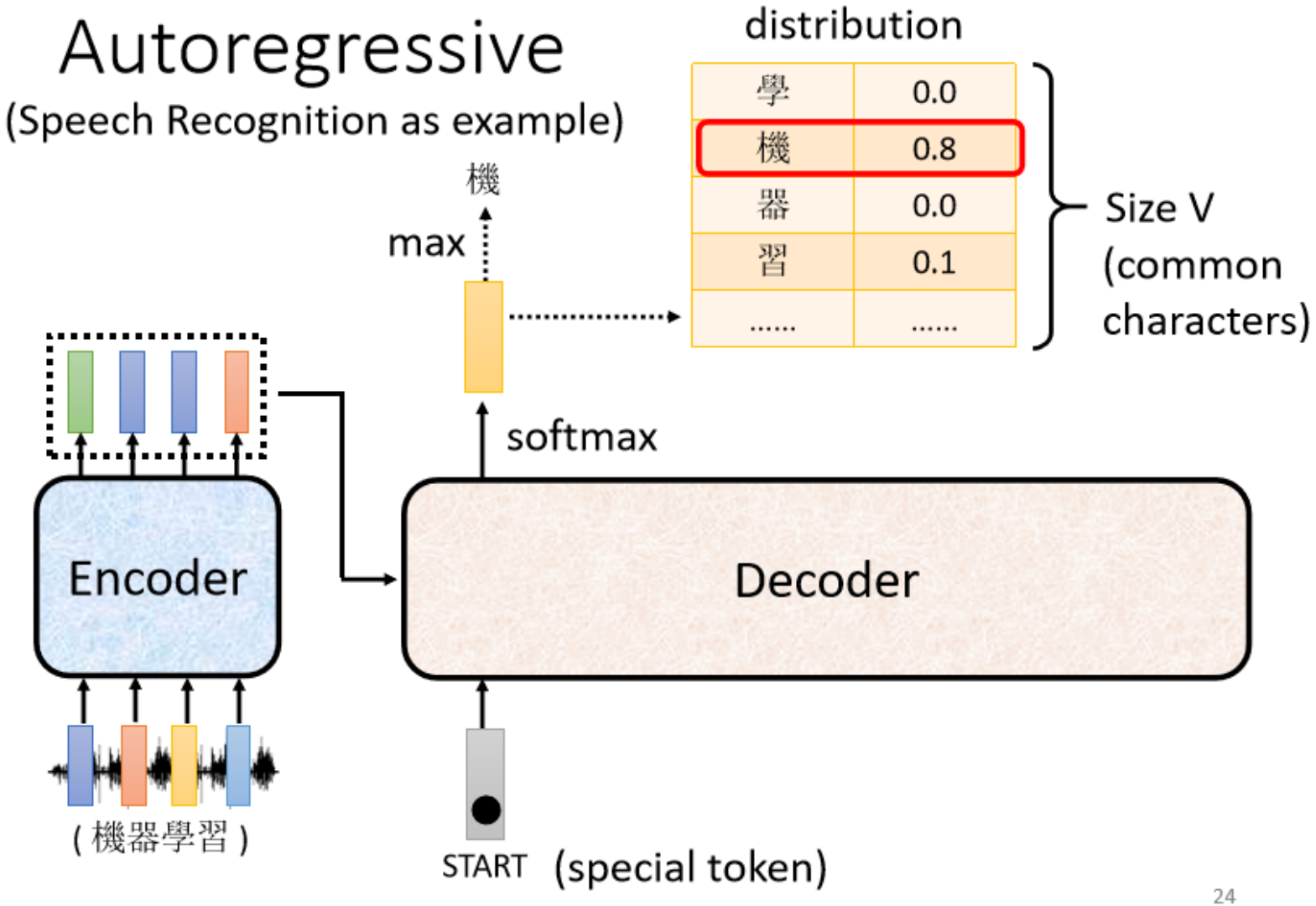
- Transformer的输出由解码器栈完成，一个接一个地输出，且在构造当前步骤输出时，除了依赖编码器栈得到的输入序列的内部编码外，还依赖当前步骤以前的所有已经产生的输出，即自回归模式；
- Transformer编码与自回归生成目标序列过程可简单表示为

$$\begin{aligned} h_1, \cdots, h_T &= encoder(x_1, \cdots, x_T) \\ y_t &= decoder(h_1, \cdots, h_T, y_1, \cdots, y_{t-1}), \end{aligned} \tag{5}$$

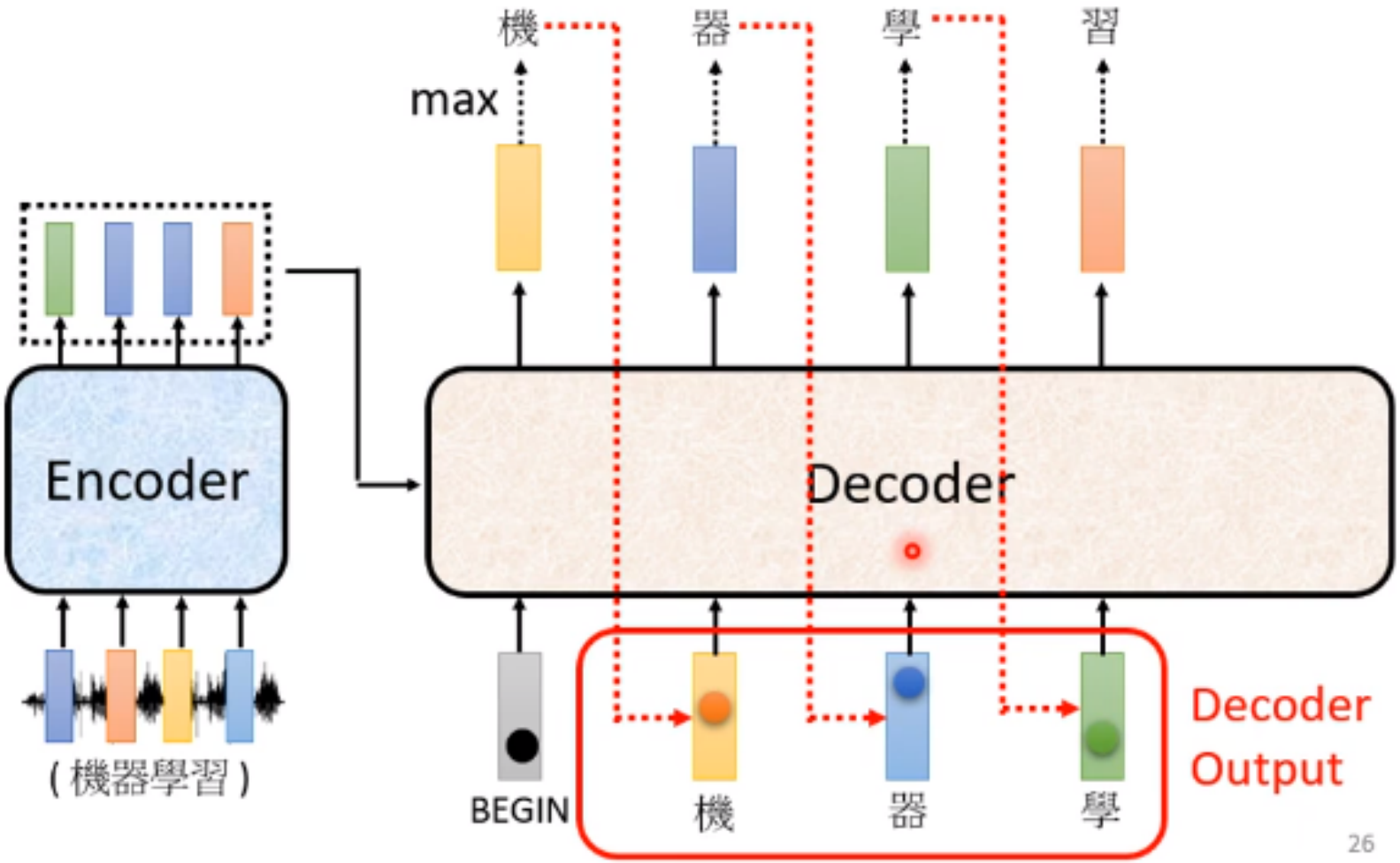
其中，$x_1, \cdots, x_T$表示长度为T的输入序列，$h_1, \cdots, h_T$为编码器栈对输入序列处理后得到的输出（维度为d_{model}），y_t（维度为d_{model}）为解码器栈产生的第t个输出，$y_1, \cdots, y_{t-1}$表示第t步之前的输出序列；

- 在机器翻译任务中，为了直接预测目标词汇，Transformer在上述向量后再进行一次线性变换，将d_{model}维的向量变换为一个$|V'|$维向量，这里$|V'|$为目标语言的词典长度，然后利用softmax函数对该向量归一化。从概率视角看，词汇生成过程就是对条件概率$p(y_t|y_1, \cdots, y_{t-1}, x_1, \cdots, x_T)$的建模过程。

Transformer的输出



Transformer的输出



Transformer的输出

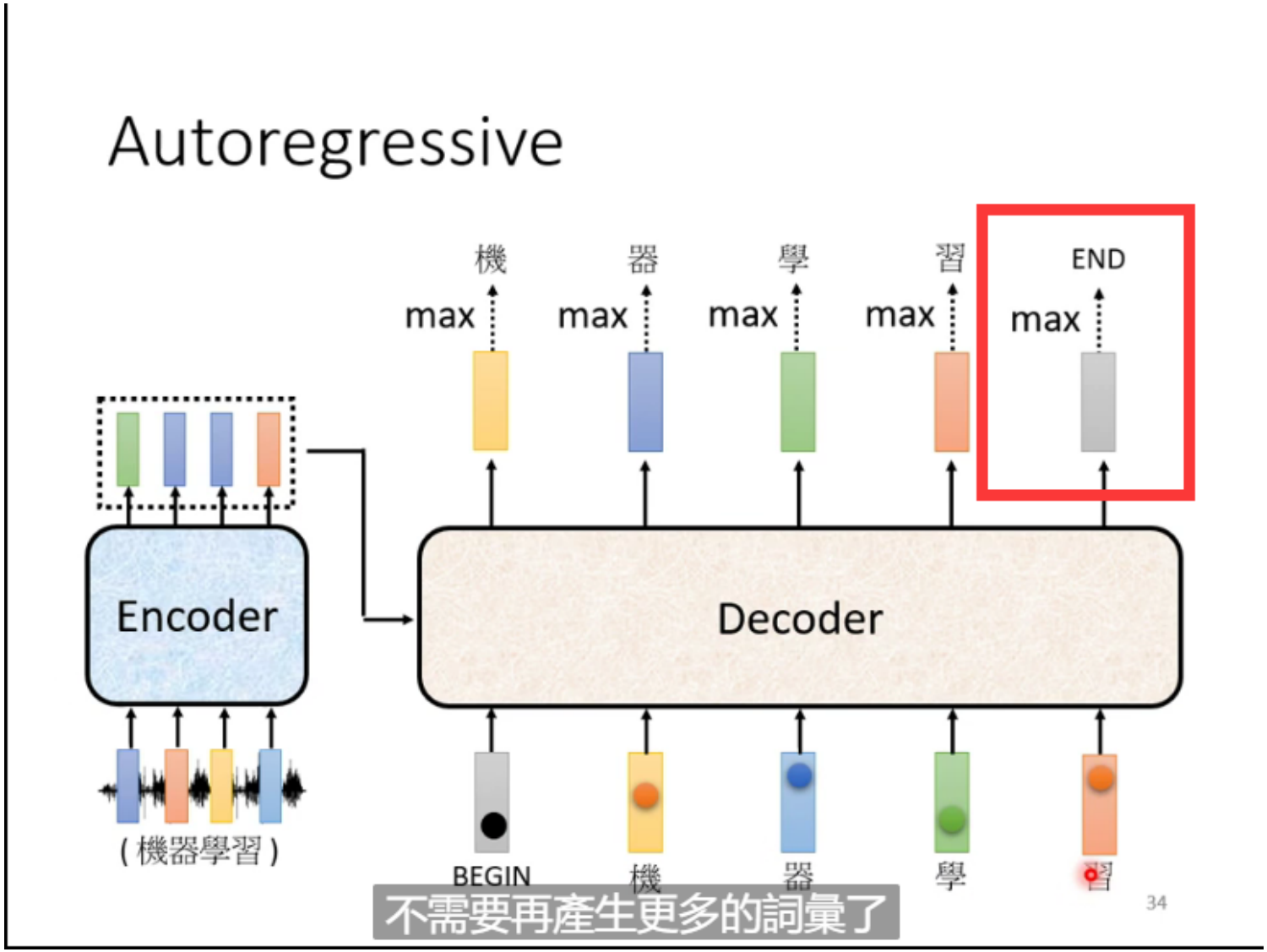


Table of Contents

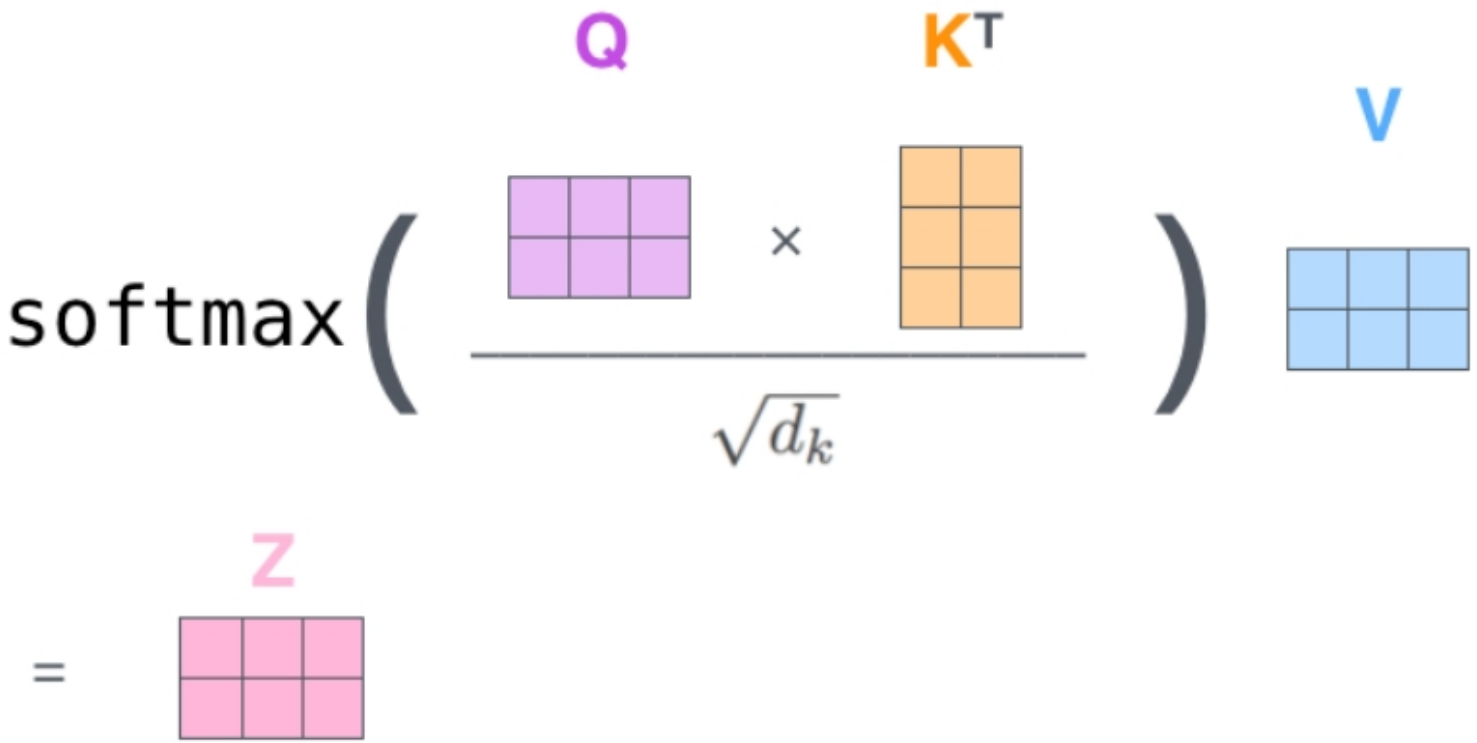
缩放点积注意力

- 设查询向量的维度为d_k，Q为将M_q个查询向量作为行向量拼成的矩阵形式，即$Q \in R^{M_q \times d_k}$；
- K为矩阵表示的键集合，设键向量的个数为M_k，且键向量的维度与查询向量的维度相同，因此有$K \in R^{M_k \times d_k}$；
- V为矩阵表示的值集合设值向量的维度为d_v，且值向量的个数与键向量个数相同，因此有$V \in R^{M_k \times d_v}$；
- 缩放点积注意力定义为

$$attention(Q, K, V) = softmax\left(\frac{QK'}{\sqrt{d_k}}\right)V, \tag{6}$$

计算结果为一个$M_q \times d_v$维矩阵，其中每一行都表示对应查询向量与所有键向量作用形成注意力权重后，对所有值向量进行加权求和的结果。

缩放点积注意力



多头注意力

- 在Transformer中，编码器和解码器实际使用的都是多头注意力机制，即多路执行并融合的缩放点积注意力，也可理解为多个视角下的注意力机制。该注意力基本操作流程包括四个步骤；
- 多路线性变换：针对Q, K, V，分别进行h个线性变换，h就是分出的“头数”，其中的第i个头的查询、键和值集合可表示为

$$Q_i = QW_i^Q, K_i = KW_i^K, V_i = QW_i^V, \tag{7}$$

  其中，$W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}$均为可学习的线性变换参数矩阵；
- 为了控制计算量，一般在分头后，都会将查询向量和值向量的维度适当降低，在Transformer基础版本中，头数$h = 8$，而$d_k = d_v = d_{model}/h = 64$，最后再通过线性变换将维度恢复为$d_{model}$。

多头注意力

- 多路缩放点积注意力计算：基于每个头的查询、键和值集合，执行缩放点积注意力的计算，即

$$H_i = attention(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i'}{\sqrt{d_k}}\right) V_i; \tag{8}$$

- 多路融合：通过拼接操作，将h路缩放点积注意力结果进行整合，有

$$H_f = concat(H_1, \cdots, H_h), \tag{9}$$

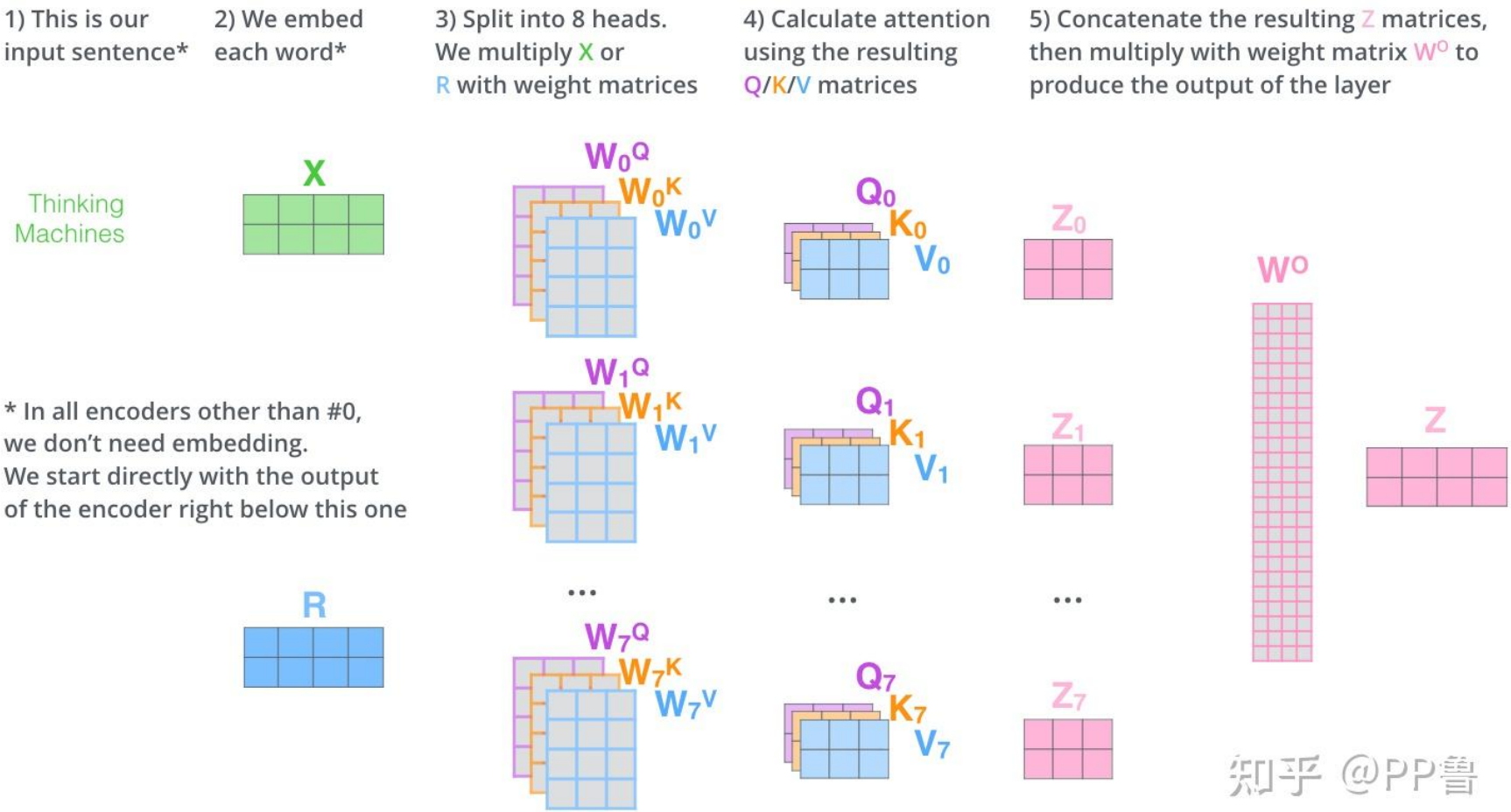
其中，拼接方式以“横铺”方式进行，因此有$H_f \in R^{M_q \times hd_v}$；

- 再次线性变换对拼接结果进一步融合，同时恢复到我们需要的向量维度，有

$$attention_m(Q, K, V) = H_f W^O, \tag{10}$$

其中，$W^O \in R^{hd_v \times d_{model}}$为可学习的线性变换参数矩阵。

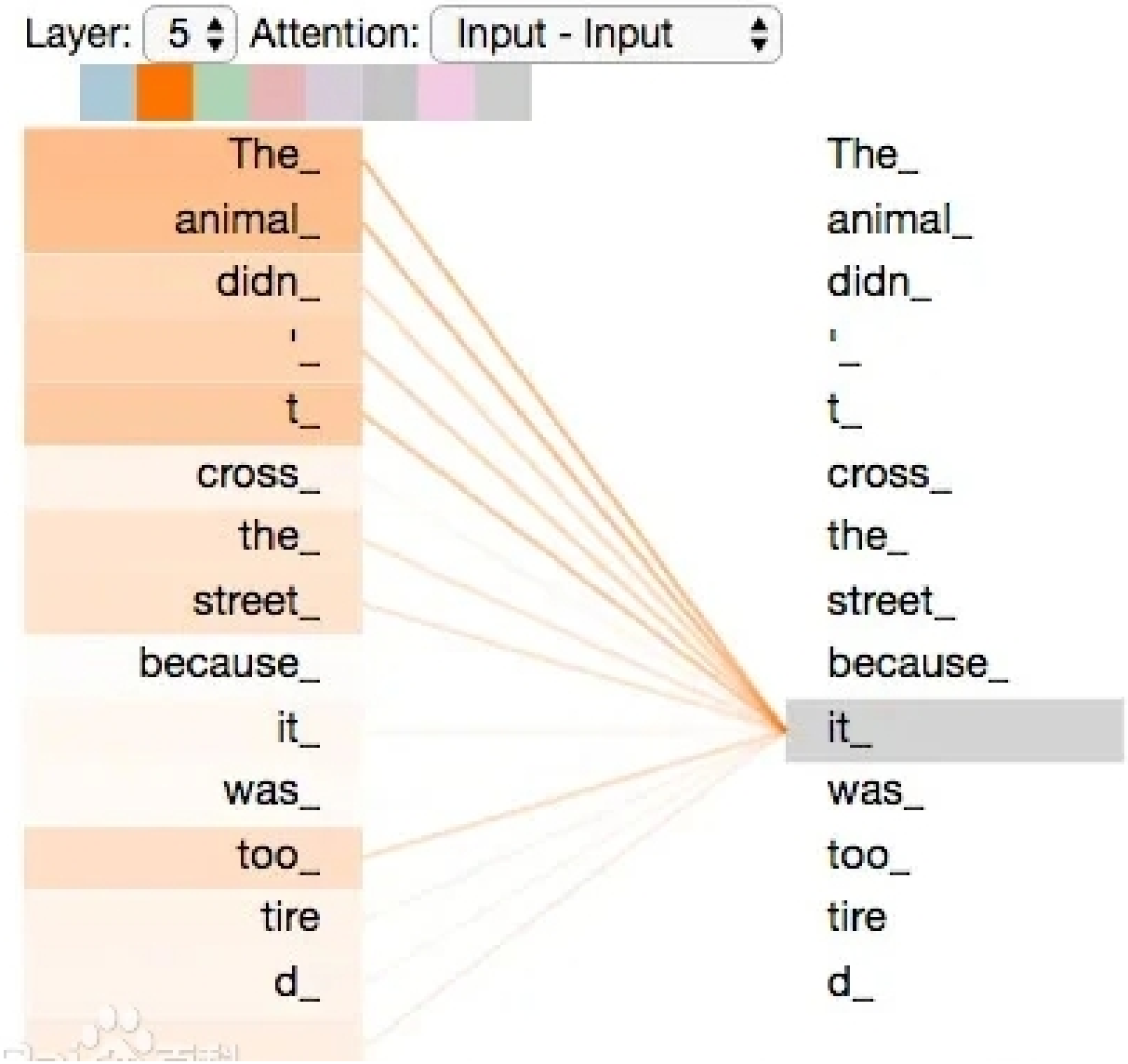
多头注意力


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking
Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z
知乎 @PP喵

编码器中的标准注意力模块

- Transformer每个编码器中具有唯一的注意力模块，该模块即为标准多头注意力模块，属于VVV模式，其中，查询集合、键集合以及值集合均为上一个编码器对输入序列作用得到的编码向量序列（如果是第一个编码器，则为带有位置信息的词嵌入向量序列）；
- 编码的核心工作就是捕捉输入序列中元素与元素之间的关系，使每个元素对应的编码向量充分蕴含上下文语义信息；
- 在实现方式上，Transformer编码器一次性看到一个完整的序列，以并行的方式对每个元素进行编码。

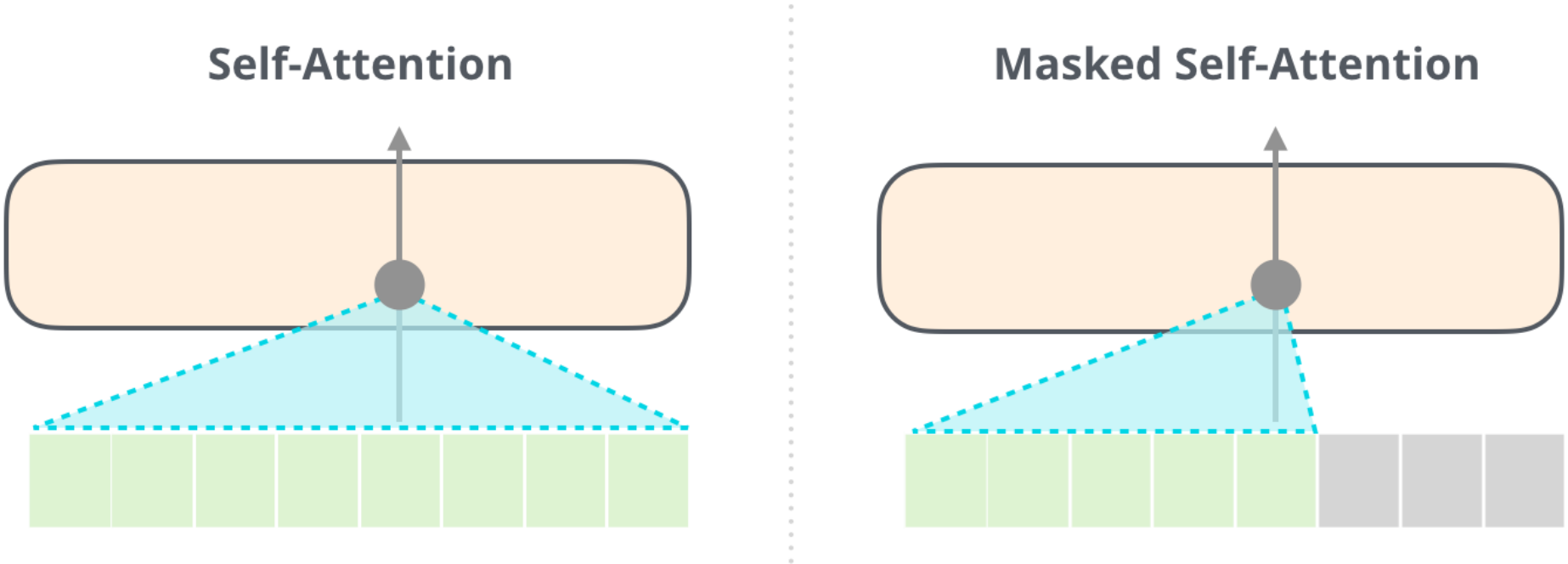
编码器中的标准注意力模块



解码器中的掩码自注意力模块

- Transformer每个解码器包括两个注意力模块，其中第一个也具有多头注意力结构，也也属于VVV模式；
- 相较于编码器中的自注意力模块，该模块添加了掩码(mask)机制，这是因为Transformer的解码过程采用自回归方式进行，在产生目标序列时，词是一个一个“蹦”出来的，这就意味着在产生当前目标词汇时，我们根本不知道下面会输出什么；
- 所以在不知后续“剧情”的情况下，只能用到当前生成步骤及其之前的输入序列进行注意力计算。在训练环节，一个输入的源序列及其对应的目标序列构成一组训练样本，Transformer以并行的方式一次性将目标序列送入解码器栈，并且一次性获得所有对应的输出，因此我们需要有一种机制能够挡住当前预测步骤之后的输入序列，使其不得参与当前输出的生成；
- 我们明明知道完整的目标序列，故意对后续输入“装作不知道”，只在前序输入上计算并应用注意力。

普通自注意力v.s.带掩码自注意力



解码器中的掩码自注意力模块

- 在缩放点积注意力中，可以很容易通过掩码矩阵的方式实现掩码自注意力

$$attention_{ms}(Q, K, V) = softmax\left(\frac{QK' \circ M}{\sqrt{d_k}}\right)V, \tag{11}$$

其中，M为一个$M_q \times M_k$的二值掩码矩阵，其上三角元素位置表示遮掩位置，符号"$\circ$"表示矩阵的逐元素相乘；

- (11)中，$QK' \circ M$即表示将QK'的上三角部分“抹去”，随后的$softmax$函数即按行在未抹去的元素上进行归一化。

例子：掩码自注意力

- 设查询向量有4个，分别记作$q_1, \cdots, q_4$，键值集合的元素个数也为4个，分别记为$k_1, \cdots, k_4$和$v_1, \cdots, v_4$，值向量的维度$k_v = 6$。将上述向量分别作行向量排布，即得到矩阵Q, K, V

$$Q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix}, K = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{pmatrix}, V = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}, \tag{12}$$

- $QK' \circ M$可表示为

$$QK' \circ M = \begin{pmatrix} q_1 k_1' & q_1 k_2' & q_1 k_3' & q_1 k_4' \\ q_2 k_1' & q_2 k_2' & q_2 k_3' & q_2 k_4' \\ q_3 k_1' & q_3 k_2' & q_3 k_3' & q_3 k_4' \\ q_4 k_1' & q_4 k_2' & q_4 k_3' & q_4 k_4' \end{pmatrix} \circ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \tag{13}$$

可以看出，经过掩码矩阵M的按位置遮掩，QK'的上三角区域被无效化，其余部分呈阶梯状。

例子：掩码自注意力

- 按照(11)计算注意力机制的输出，有

$$
\begin{aligned}
attention_{ms}(Q,K,V) &= softmax\left(\frac{QK'\circ M}{\sqrt{d_k}}\right)V \\
&= \begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix},
\end{aligned} \tag{14}
$$

其中，$\sum_{j=1}^{i}\alpha_{ij}=1\ (i=1,\cdots,4)$;

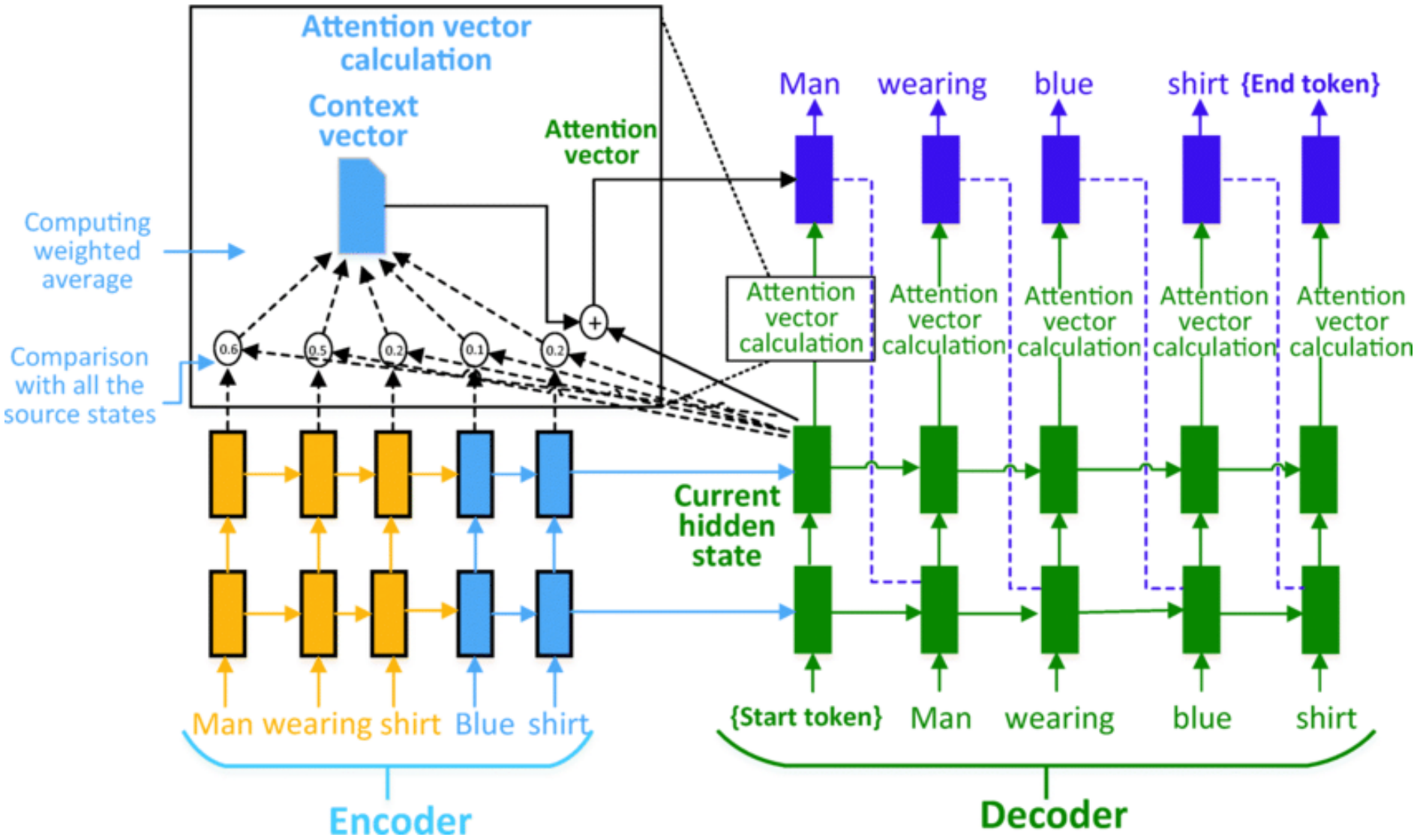
- 上述注意力机制的输出为一个4×6矩阵，每个行向量分别表示以四个查询向量对应的掩码注意力输出，表示为

$$
\begin{aligned}
attention_{ms}(q_1,K,V) &= \alpha_{11}v_1 \\
attention_{ms}(q_2,K,V) &= \alpha_{21}v_1+\alpha_{22}v_2 \\
attention_{ms}(q_3,K,V) &= \alpha_{31}v_1+\alpha_{32}v_2+\alpha_{33}v_3 \\
attention_{ms}(q_4,K,V) &= \alpha_{41}v_1+\alpha_{42}v_2+\alpha_{43}v_3+\alpha_{44}v_4.
\end{aligned} \tag{15}
$$

解码器中的互注意力模块

- Transformer解码器中的第二个注意力模块称为“编码器-解码器注意力”模块，是一种构造于编码器和解码器之间的互注意力模块，属于典型的QVV模式；
- 在编码器-解码器注意力模块中，查询集合来自当前解码器上一个模块的输出，而键值集合相同，均为编码器栈的输出序列，即为输入序列构建的内部编码序列；
- 编码器-解码器注意力模块的输入是上一级掩码注意力模块对目标序列的加工输出，针对其中的某一个向量，该模块以其作为查询向量，构建其与编码器栈输出序列中所有编码向量间的相似关系，然后通过归一化操作将这些相似关系转化为注意力权重，最后再利用这些权重对所有编码向量进行加权求和，得到对应的输出。

编码器-解码器注意力模块



编码器-解码器注意力模块的功能

- 构建编码器到解码器的信息桥梁：解码器产生输入序列的内部编码，再产生目标输出，而该过程必须依赖编码器的信息输入，Transformer中只有编码器-解码器注意力模块扮演了从输入到输出信息桥梁的角色；
- 在产生输出时有侧重地参考全部输入信息：输出什么得看输入，但是输入不见得对输出都有帮助，这就需要输出时有侧重地参考输入信息；
- 与掩码自注意力模块形成“主内”与“主外”的搭配：序列内部元素之间的关系很重要，掩码子注意力模块探索的正是这一序列内部关系，然后，在序列转换任务中，还需要在理清序列内部关系基础上，充分参考并理解输入序列的语义，才能产生与输入语义相符合的目标序列。

Table of Contents

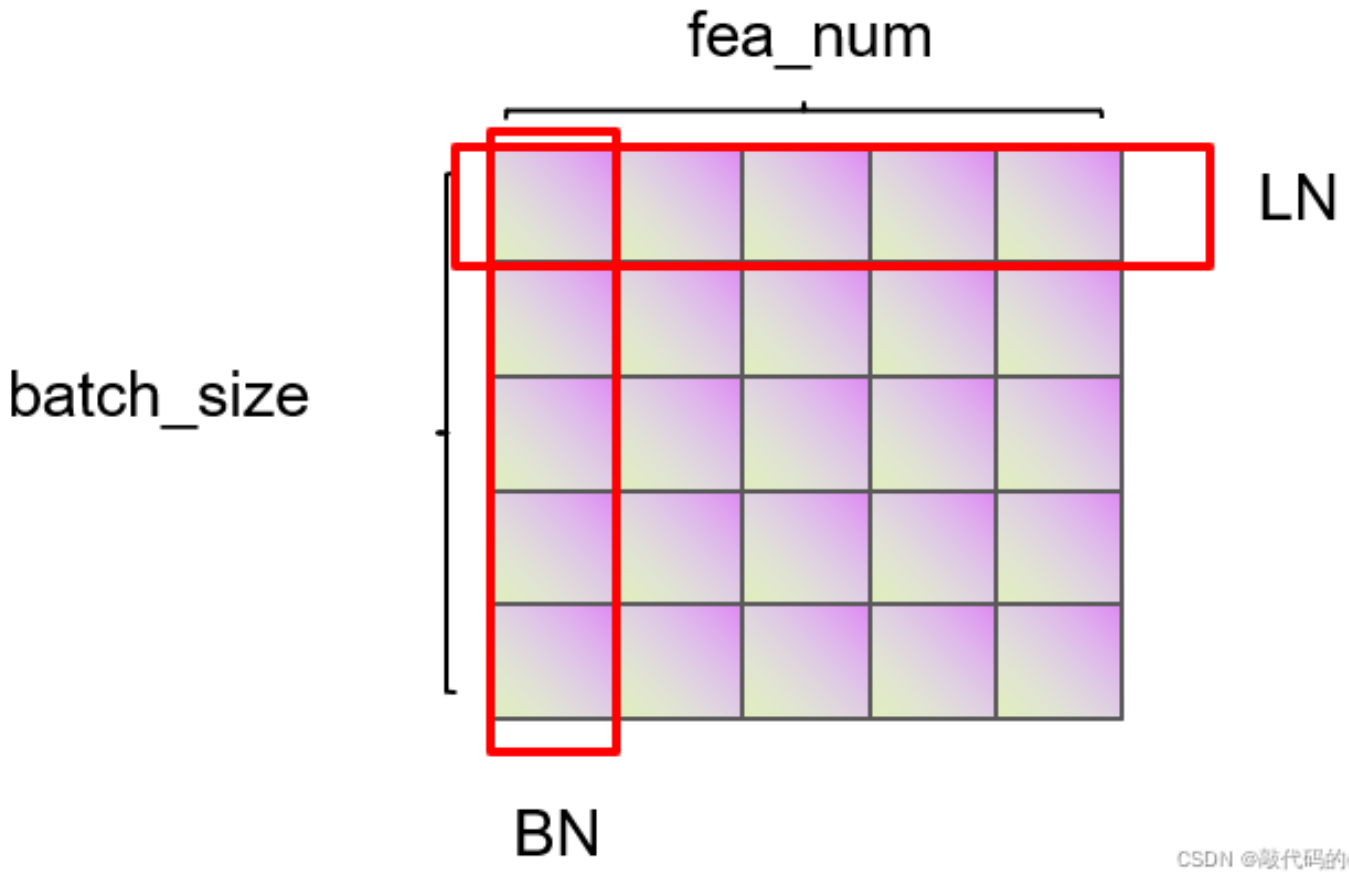
BatchNorm

- BatchNorm：将不同样本相同维度的特征处理为相同的分布，即减样本均值除样本标准差，使不同维度的特征近似服从标准正态分布；
- BatchNorm主要适用于计算机视觉领域，以跨样本的方式开展归一化，因此不会破坏不同样本同一特征之间的关系，这一性质决定了经过归一化操作后，样本之间仍然具有可比性，但是，特征与特征之间不再具有可比较性。

LayerNorm

- 对于一个NLP模型，一个批次的输入包括若干个独立的句子，每个句子又由不定长度的词构成，句子中的每个词又被表达为一个定长向量；
- 按照BatchNorm的思路，将批次中不同样本同一维度特征视为相同分布，即对所有句子处于相同位置的词向量执行减均值除标准差的操作，是极不合理的，这是由于
  - 由于句子长短不一，某些位置有值有些位置为空，无法执行归一化运算；
  - 归一化的目的是将具有相同性质的数据转化为标准正态分布，其结果并不会破坏数据之间的可比较性，但是不同句子相同位置的词汇不具有相同性质，也不需要进行比较；
- 由于Transformer在构建句内关系时使用的自注意力机制，正是将某个词与同句中其他词“比”出的相似度，我们要保留句子中词和词之间的可比较性，就需要对整个句子进行归一化操作，这就是LayerNorm——减一个句子的均值除一个句子的标准差；
- LayerNorm类似于在一个句子中找到一个“语义中心”，然后将句子中的所有词都聚集在这个中心的周围，而句中词与词之间的关系不会遭到破坏。

BatchNorm与LayerNorm



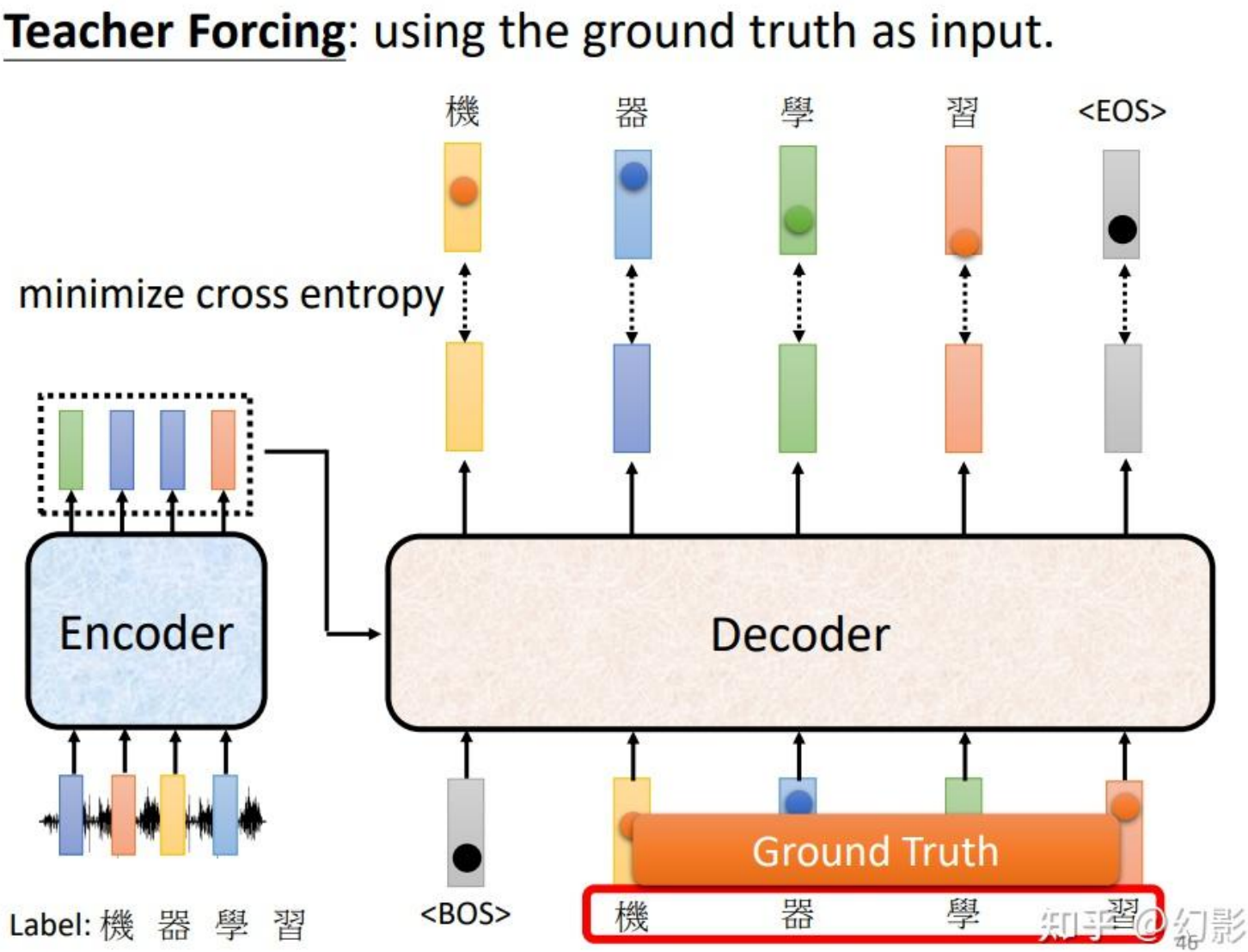
模型训练的自回归模式

- 所谓自回归预测，就是基于已经产生的预测结果预测下一步的输出，Transformer在预测环节应用了该模式；
- 自回归模式存在一个巨大弊端，那就是容易产生错误的累积，可谓“一步错步步错，且越错越离谱”，这在模型训练的初始阶段体现得更加明显，因为此时解码器还是一个“小白”，很难得到靠谱的预测结果，错误累积问题就变得极其严重，模型训练也变得极不稳定，很难收敛，除此以外，自回归模式只能以串行方式进行，很难以并行化的方式开展训练以提升效率。

模型训练的Teacher Forcing模式

- 所谓Teacher Forcing模式，就是在训练时不在以模型预测作为输入，取而代之的是以目标真值(ground truth)作为输入，"Teacher"自然指的就是真值；
- 在训练阶段，假设一组训练样本为$\{(x_1, \cdots, x_n), (\tilde{y}_1, \cdots, \tilde{y}_m)\}$，其中，$x_1, \cdots, x_n$为输入序列，$\tilde{y}_1, \cdots, \tilde{y}_m$为目标真值序列，我们以$y_1, y_2, \cdots$表示模型的预测结果；
- 在第t个生成步骤，自回归模式和Teacher Forcing模式下的目标生产可以分别表示为$y_t = generate(y_1, \cdots, y_{t-1})$和$y_t = generate(\tilde{y}_1, \cdots, \tilde{y}_{t-1})$；
- 不难看出，Teacher Forcing模式下的模型训练以“看着标准答案做题”的方式进行——看到的都是正确答案，只要做好当前一步的预测就好，从而能够在训练的时候不断矫正模型的预测结果，消除错误的累积效应，增强模型训练的稳定性，收敛速度也得以大幅提升，同时，我们可以一次性输入全部目标序列，以并行的方式一次性输出完整的目标序列，训练效率大幅提升。

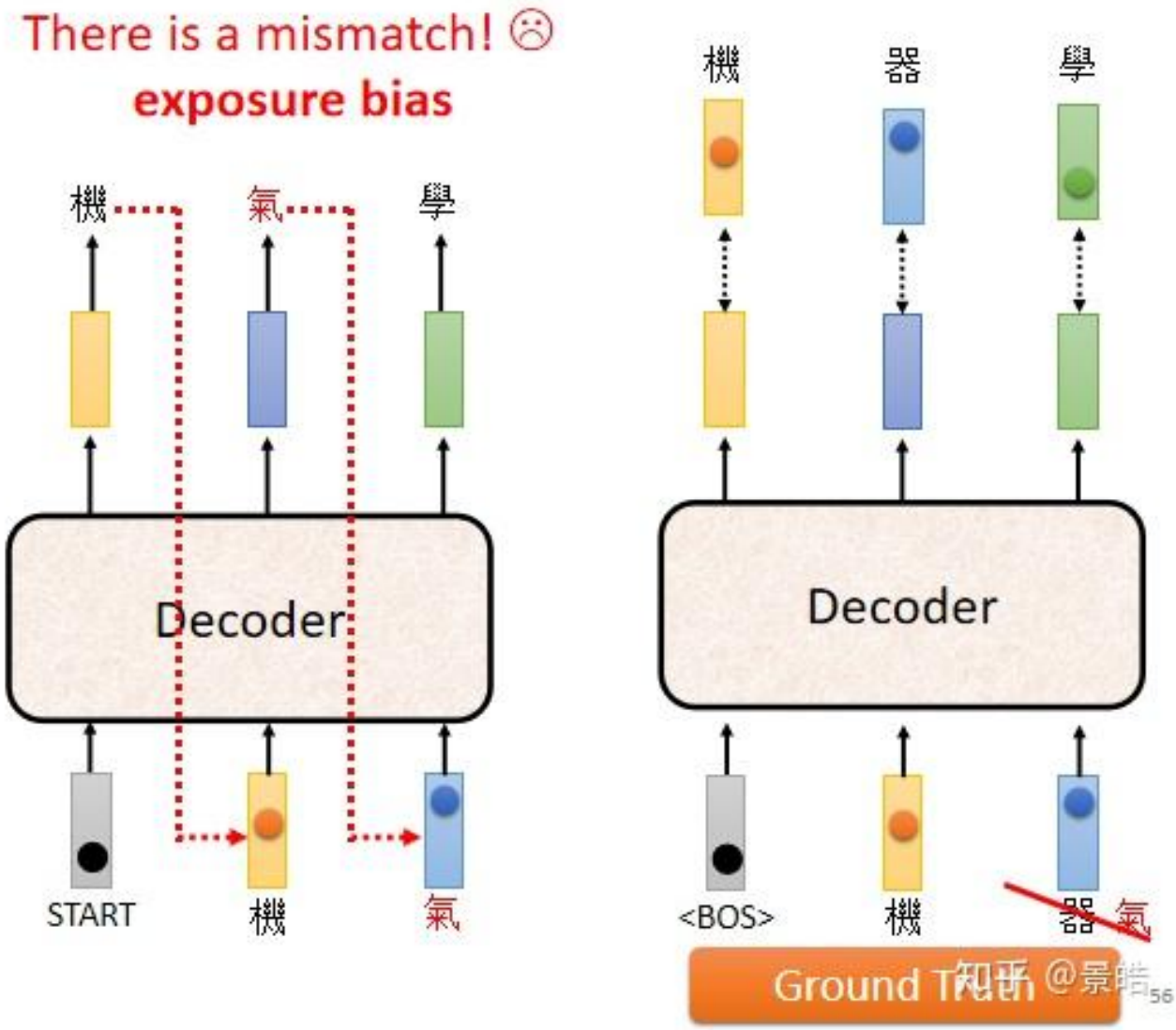
模型训练的Teacher Forcing模式



模型训练的Teacher Forcing模式

- 自回归模式“纯粹靠自己”，Teacher Forcing模式走到了“纯粹靠标准答案”的另一个极端。显然这是有问题的，毕竟到了真正的序列转换任务中，“老师”没了，还是得靠自己。这种因Teacher Forcing模式导致的模型在训练环节和预测环节存在行为差异的现象被称为“曝光误差”(exposure bias)；
- 2015年谷歌提出“计划采样”(scheduled sampling)方法，其核心思想非常简单：既然自回归模式的全靠自身预测结果和Teacher Forcing模式的全靠真值均不可取，那就取折中方案，即在每一步的预测时，以一定的概率随机选择是用模型输出还是用真值，而选择概率时随着训练的推进不断调整的：模型在训练初期尚属“小白”，应该更多依赖真值，需将使用真值的概率调整得高一些。随着模型的不断训练，“小白”逐渐变成“大佬”，故对模型自身得输出变得越来越信任，使用模型输出的概率也逐渐增大。

计划采样


There is a mismatch! ☹
exposure bias
機
氣
學
Decoder
START
機
氣
機
器
學
Decoder
<BOS>
機
器
氣
Ground Truth
知乎 @景皓

全局最优输出

- 在Transformer的预测环节，如何决定最优的输出序列？全局最优输出序列是指在整个词空间中，使得输出序列联合概率取得最大词汇的组合；
- 将输出序列记作$y = y_1, \cdots, y_m$，将其中的最优序列记作y^*，即满足$y^* = \arg\max p(y_1, \cdots, y_m)$。例如，有一个小词典，其中只有3个词，分别记作A，B和C，假设输出序列长度为2（即$m = 2$），则在词空间中，输出序列可能的情况包含9种，即：AA、AB、AC、BA、BB、BC、CA、CB、CC。倘若其中某个组合的概率在9个概率中取得最大值，那么该组合就是全局最优输出序列；
- 按照联合分布的链式法则，可以将联合分布展开为条件概率的连乘形式，即有

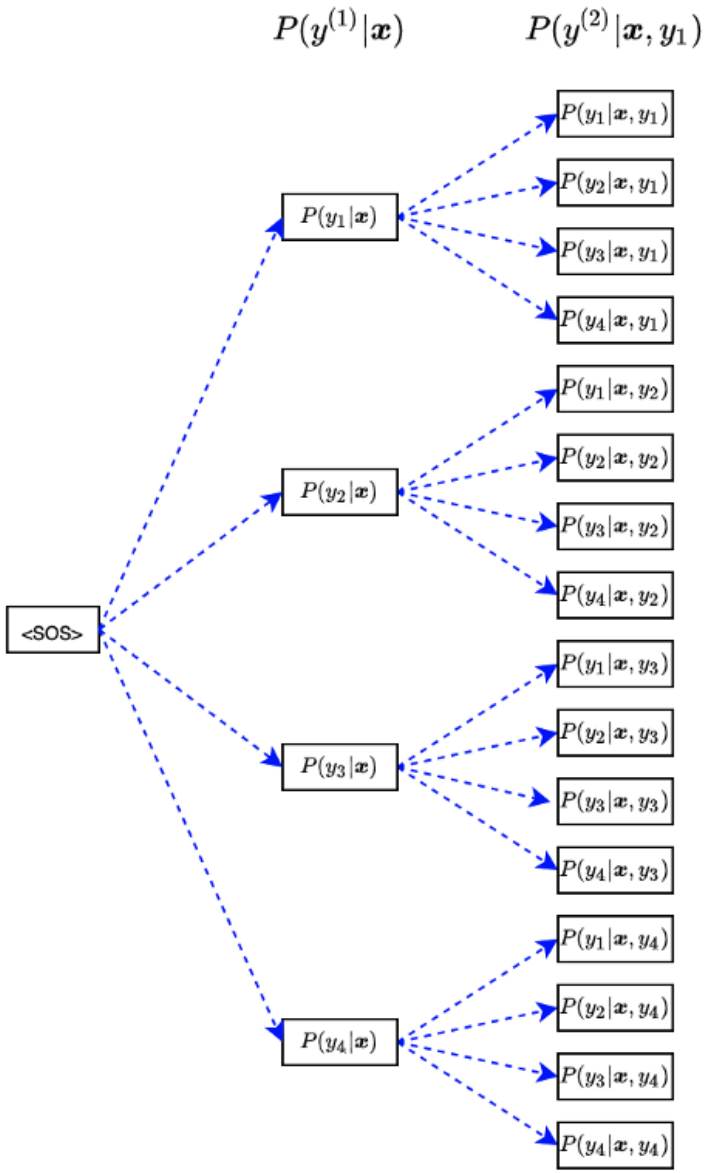
$$p(y_1, \cdots, y_m) = p(y_1)p(y_2|y_1)\cdots p(y_m|y_1, \cdots, y_{m-1}), \tag{16}$$

不难看出，要获得全局最优输出，需要遍历词序列的所有可能组合并在其中挑选概率最大者，这种构造输出序列的方式被称为穷举搜索(exhaustive search)法。

穷举搜索

- 对于自回归模式的序列模型，穷举搜索意味着在进行第t步预测时，无论在$t-1$步输出了什么，在当前步骤要让输入序列$y_1, \cdots, y_{t-1}$取遍所有可能的词汇组合作为模型输入；
- 穷举搜索看的是联合分布表示的全局最优，“不计一城一地之得失”；
- 当我们使用一个大词典时，穷举搜索的计算量将暴增，直接导致该方法无法实际使用。

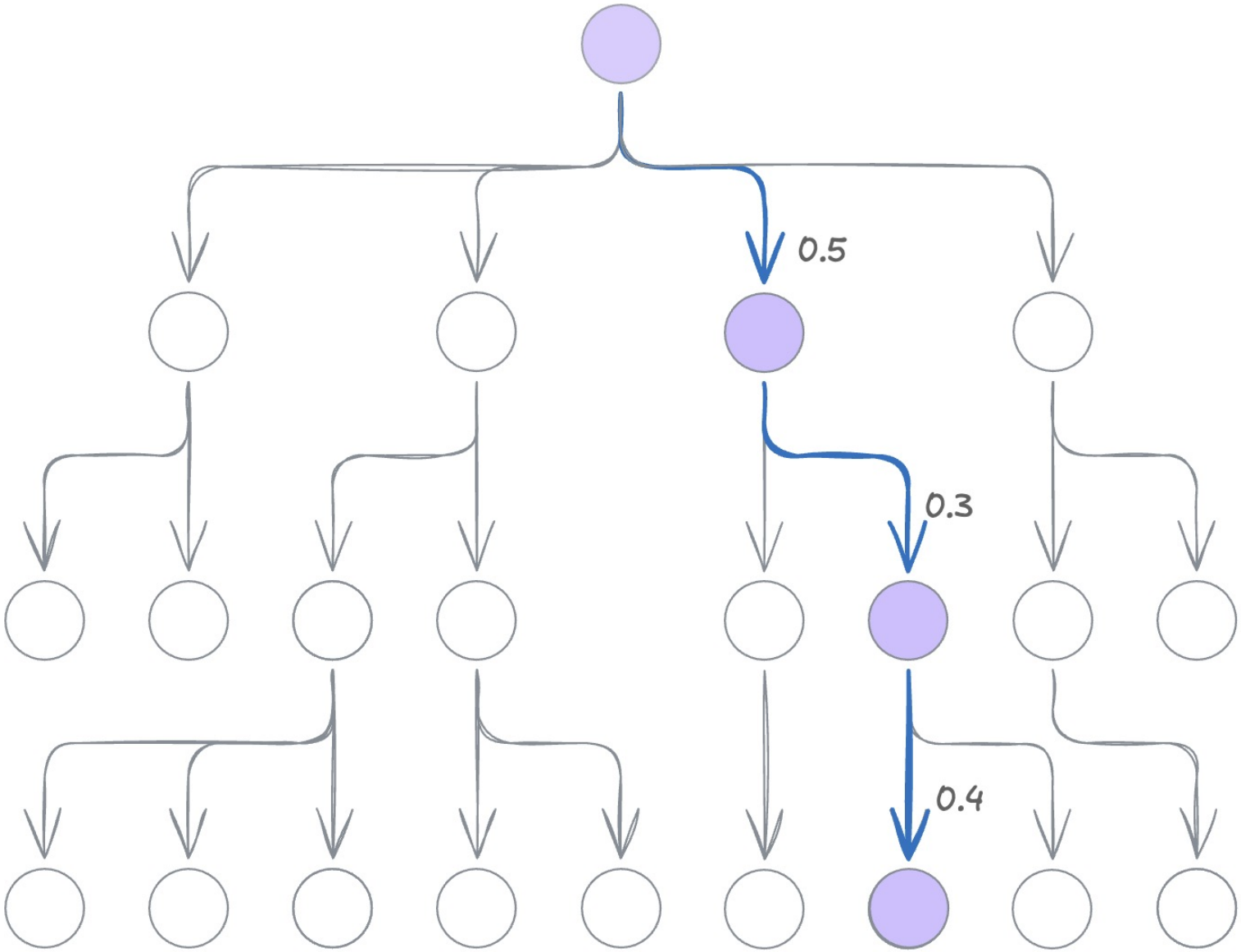
穷举搜索


$P(y^{(1)}|\boldsymbol{x})$
$P(y^{(2)}|\boldsymbol{x}, y_1)$
<SOS>
$P(y_1|\boldsymbol{x})$
$P(y_2|\boldsymbol{x})$
$P(y_3|\boldsymbol{x})$
$P(y_4|\boldsymbol{x})$
$P(y_1|\boldsymbol{x}, y_1)$
$P(y_2|\boldsymbol{x}, y_1)$
$P(y_3|\boldsymbol{x}, y_1)$
$P(y_4|\boldsymbol{x}, y_1)$
$P(y_1|\boldsymbol{x}, y_2)$
$P(y_2|\boldsymbol{x}, y_2)$
$P(y_3|\boldsymbol{x}, y_2)$
$P(y_4|\boldsymbol{x}, y_2)$
$P(y_1|\boldsymbol{x}, y_3)$
$P(y_2|\boldsymbol{x}, y_3)$
$P(y_3|\boldsymbol{x}, y_3)$
$P(y_4|\boldsymbol{x}, y_3)$
$P(y_1|\boldsymbol{x}, y_4)$
$P(y_2|\boldsymbol{x}, y_4)$
$P(y_3|\boldsymbol{x}, y_4)$
$P(y_4|\boldsymbol{x}, y_4)$

贪心搜索

- 所谓贪心搜索(greedy search)法，就是在进行第t步预测时，以$t-1$步得到具有最大概率的预测词汇作为输入，以该词汇作为条件计算条件概率，然后将使得该条件概率达到最大值的词汇作为第t步输出；
- 贪心搜索不会关心输出序列的联合概率是否达到最大值，因此输出序列不见得是全局最优的；
- 穷举搜索是“放眼全局”，计算量太大，无法实用；贪心搜索则“只看眼前”，目光短浅，容易产生误差累积。

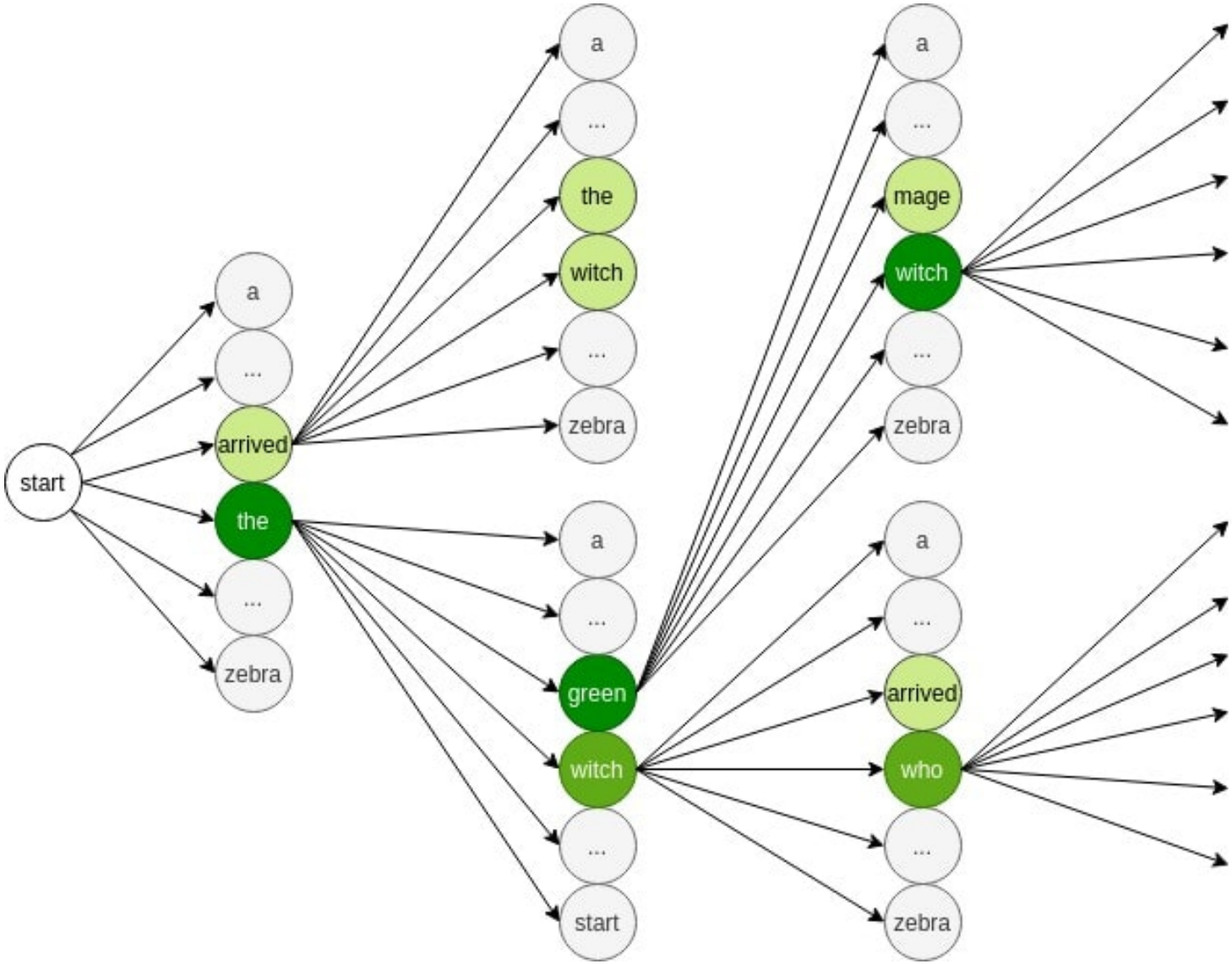
贪心搜索



集束搜索

- 集束搜索(beam search)介于穷举搜索与贪心搜索之间，Transformer在进行序列生成时使用的就是该方法；
- 在集束搜索中，在第t步的词预测中，既不像穷举搜索那样用到全部词的组合，也不像贪心搜索那样只用到前面最大的那个预测词汇，而是用到上一步概率值排在前k个的词预测作为当前步骤的输入，其中，k被称为“集束宽度”(beam width)，在Transformer中，集束宽度设置为4；
- 相较于穷举搜索和贪心搜索，集束搜索在计算量和准确性方面进行了平衡，当$k = |V|$时，集束搜索就变成了穷举搜索，其中$|V|$为词典大小，而当$k = 1$时，集束搜索就退化为贪心搜索。

集束搜索


start
a
...
arrived
the
...
zebra
a
...
the
witch
...
zebra
a
...
green
witch
...
start
a
...
mage
witch
...
zebra
a
...
arrived
who
...
zebra