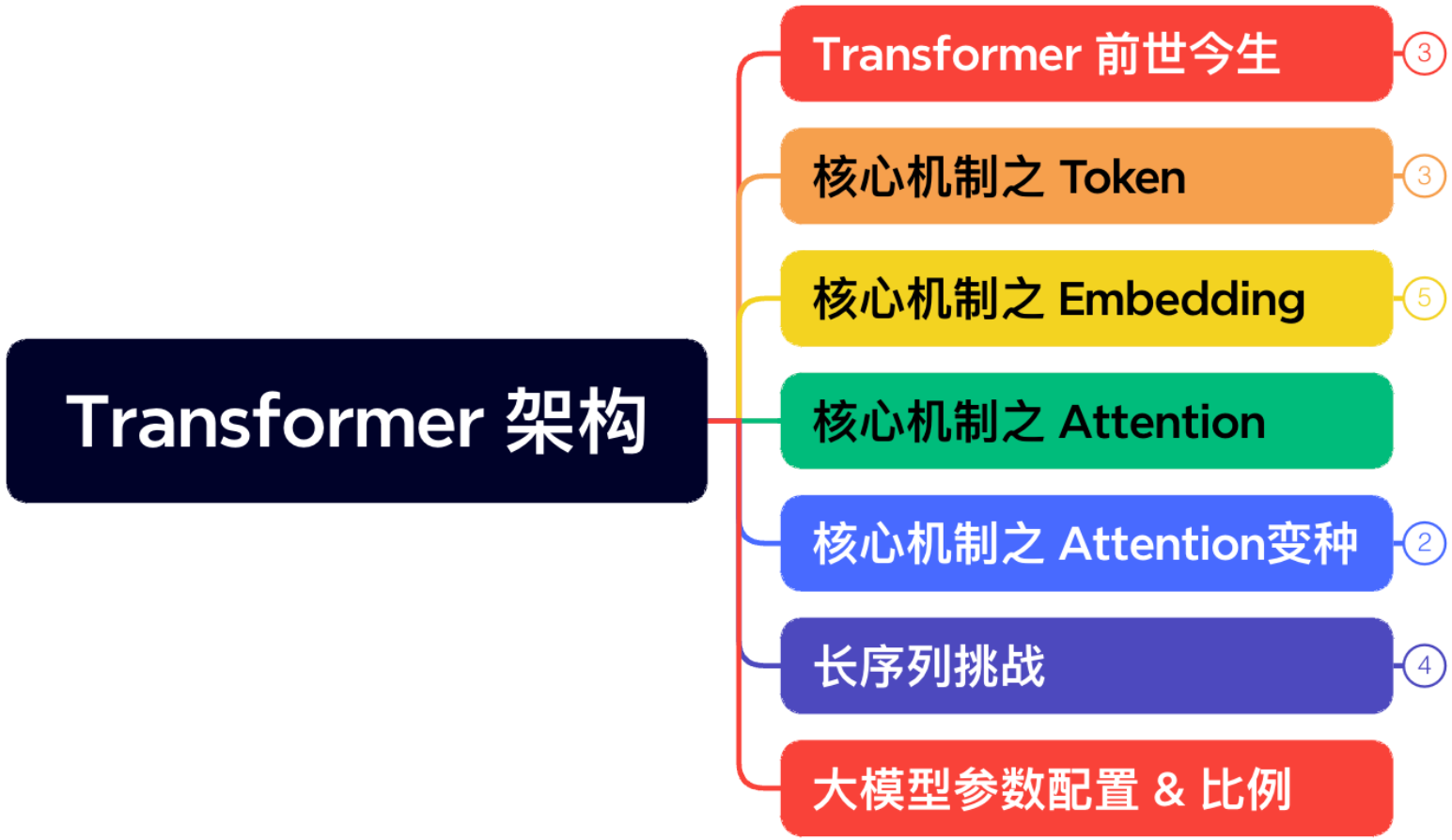



自注意力Attention全解析


ZOMI

Contenet


Transformer 架构
Transformer 前世今生 3
核心机制之 Token 3
核心机制之 Embedding 5
核心机制之 Attention
核心机制之 Attention变种 2
长序列挑战 4
大模型参数配置 & 比例


好好想想

核心问题：

- 传统序列模型局限性——长程依赖丢失、并行计算困难。

解决方案：

- Attention核心思想——通过权重分配聚焦关键信息。

视频目录大纲

- Attention 的由来
- Attention 技术原理
- Self-Attention 核心原理
- Multi-Head Attention 延伸
- Masked-self Attention 原理

Attention == Self-Attention?

- 注意力机制！ = 自注意机制！ =多头自注意力机制
- Attention != Self-Attention != Mulit-Head Self-Attention

Attention == Self-Attention?

- **Attention：**
  - 一种通用的动态信息筛选机制，通过为输入的不同部分分配权重，使模型关注与当前任务最相关的信息。
  - 通过计算Q与K的相似度得到注意力权重，再对V进行加权求和，形成新的语义表征。

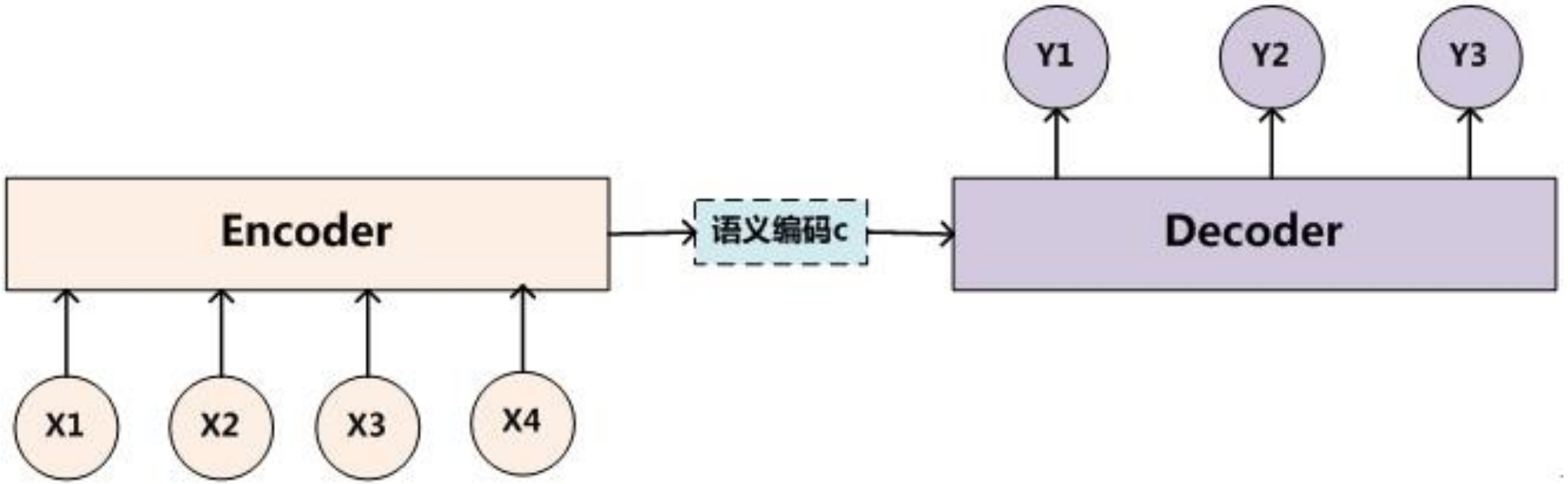
- **Self-Attention：**
  - Attention的特殊形式，要求Q、K、V同源（均来自同一输入序列）。它通过让序列中的每个元素与其他所有元素交互，捕捉内部依赖关系，

01

Attention 由来

encoder-decoder 框架

- 文本生成任务，encoder表示一个对文章的阅读和理解，decoder就是生成文本摘要的过程；
- 对于问答系统，encoder就是一个问句，decoder就对应一个回答过程。

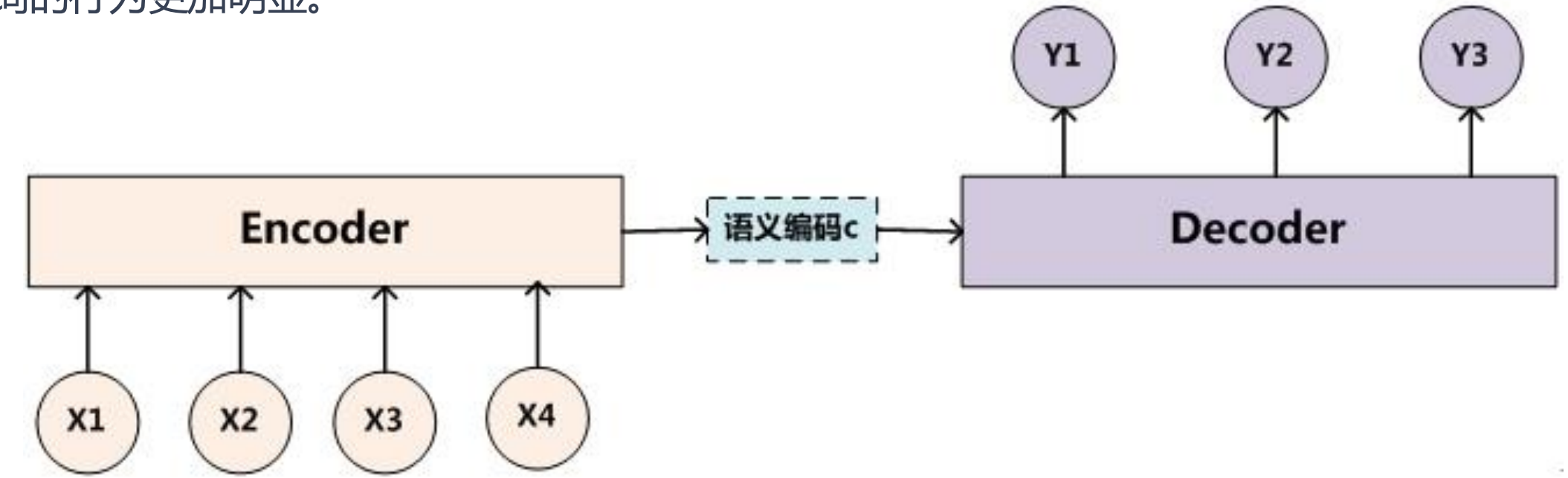


Attention 由来

- 提出Attention技术希望可以让模型关注输入的相关部分。e.g. “我是一个学生”，希望模型可以在翻译 student时，更关注‘学生’这个词，而不是其他位子的词。

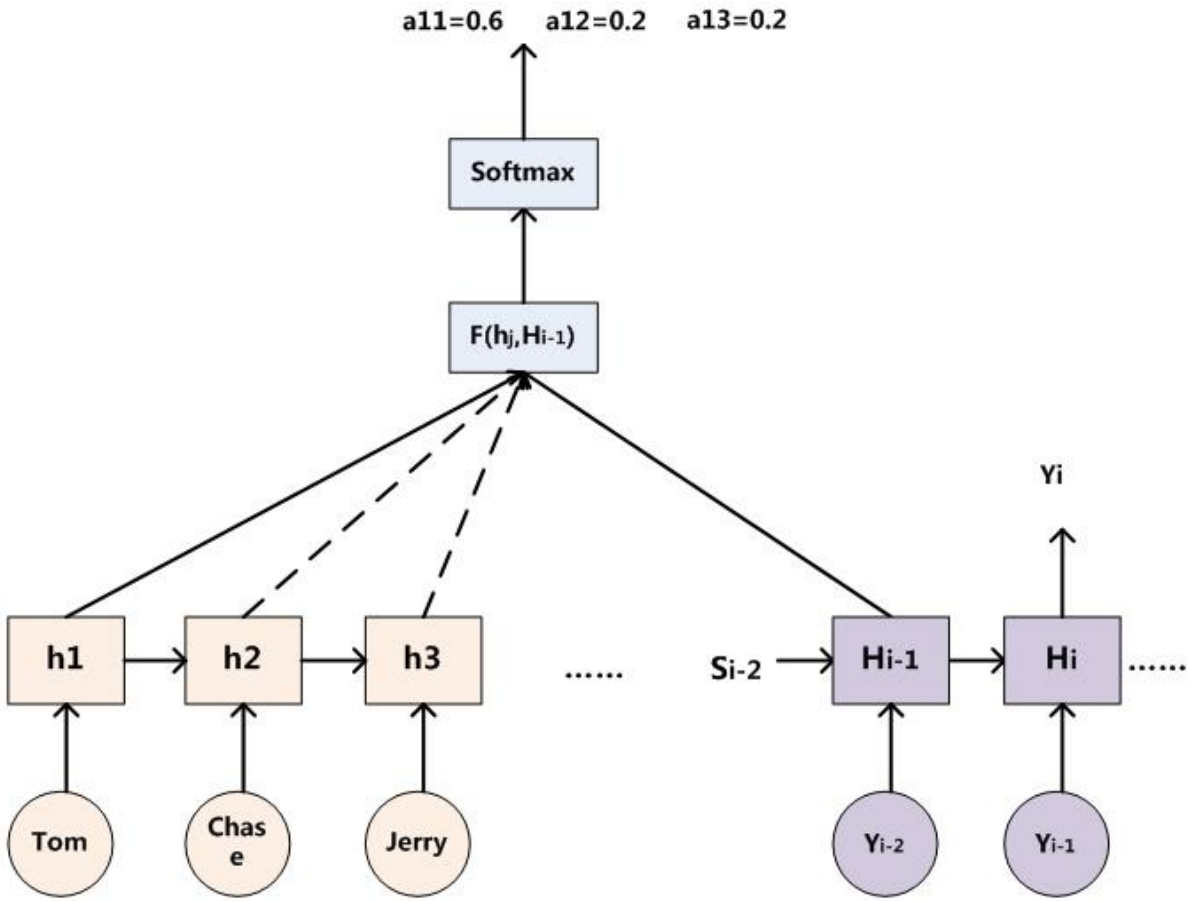
Attention 提出

- encoder-decoder 框架中对于每个输出 yi 的预测，它们使用的是同一个语义编码 C，这其实是很反直觉的。
- 翻译每个词时，注意力在原句子中是有所侧重，而不是漫无目的搜索，处理长句时，聚焦特定词的行为更加明显。



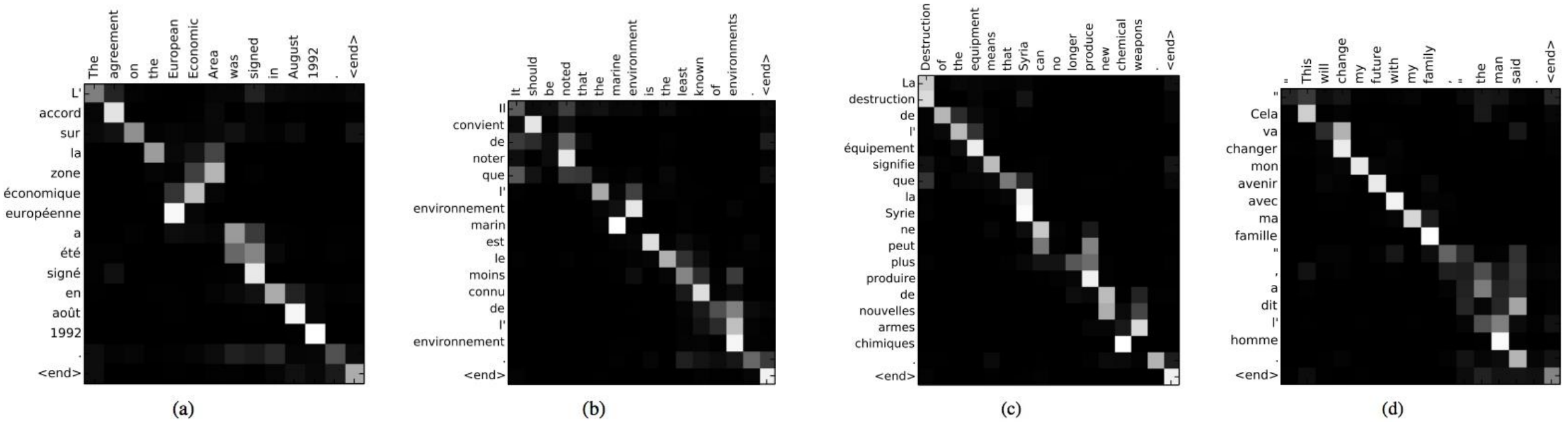
Attention 提出

- 当前所需预测 yi，利用前一时间隐藏层输出 $Hi-1$ 和 encoder 过程中每个时间隐藏层输出 hj 进行 source 计算 $F(hj, Hi-1)$，再经过 Softmax 得到对应 source 每个位置概率。



Attention 在 encoder-decoder 下效果

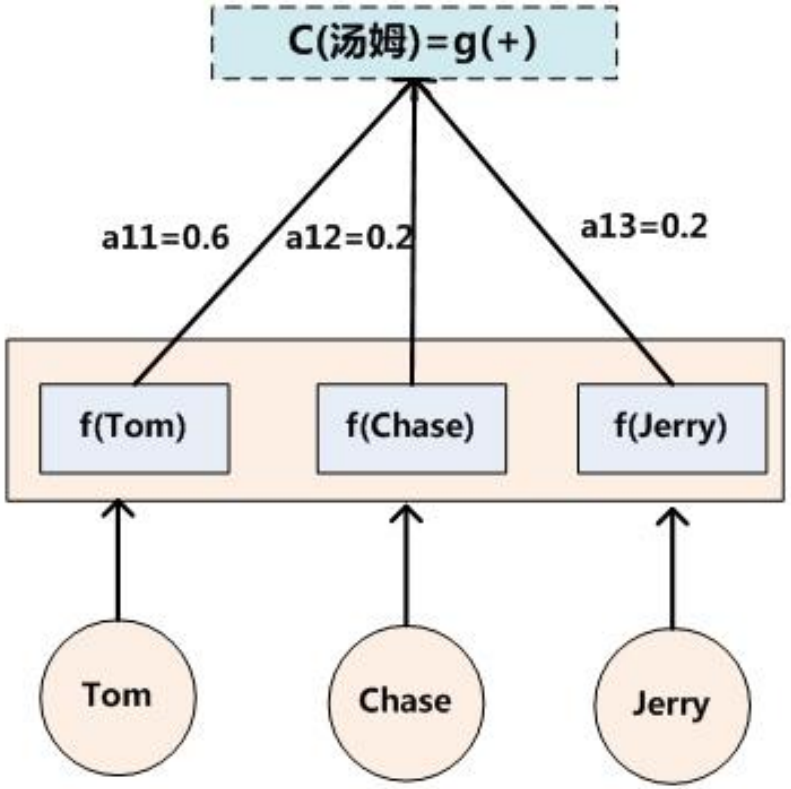
- 英法翻译时的 attention 计算的示意图，可以发现亮的地方所对应的英语和法语是相当关联。



https://arxiv.org/pdf/1409.0473.pdf

Attention 概率分布

- 得到了每个 yi 的 attention 概率分布后，通过加权求和的方式，可以得到预测 yi 时的语义编码 C，其中 $a11, a12, a13$ 分别表示预测第一个词时使用的 attention 概率分布。



Attention 概率分布

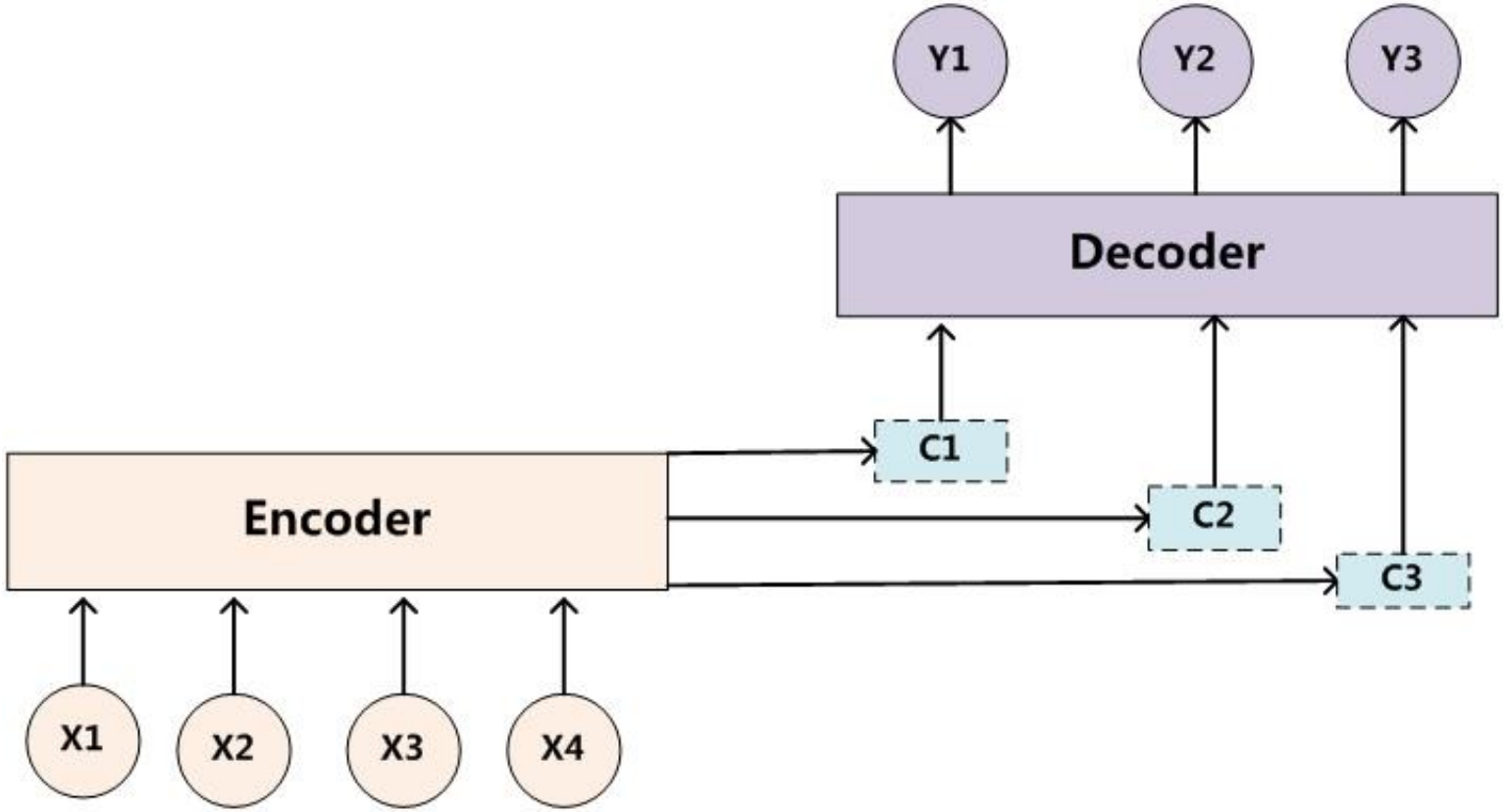
- 利用 attention 分布，对 source 中每个时间的隐藏层状态 $h1, h2, h3$ 进行加权求和，就得到预测第一个词时，应该使用的语义编码 $C1$，即：

$$C_i = \sum_{j=1}^{L} a_{ij} h_j$$

- 其中 L 表示 source 长度，hj 表示 source 每个时刻的隐藏层输出。

Attention 概率分布

- 类似可以求出预测“追逐”和“杰瑞”时的语义编码 $C2$ 和 $C3$。对比之前 $yi = g(C, y1, y2, \ldots, yi-1)$ 使用同一个语义编码 C，这里每一次预测都会先计算与之匹配的语义编码 Ci。



02

Attention 核心原理

Attention 机制本质

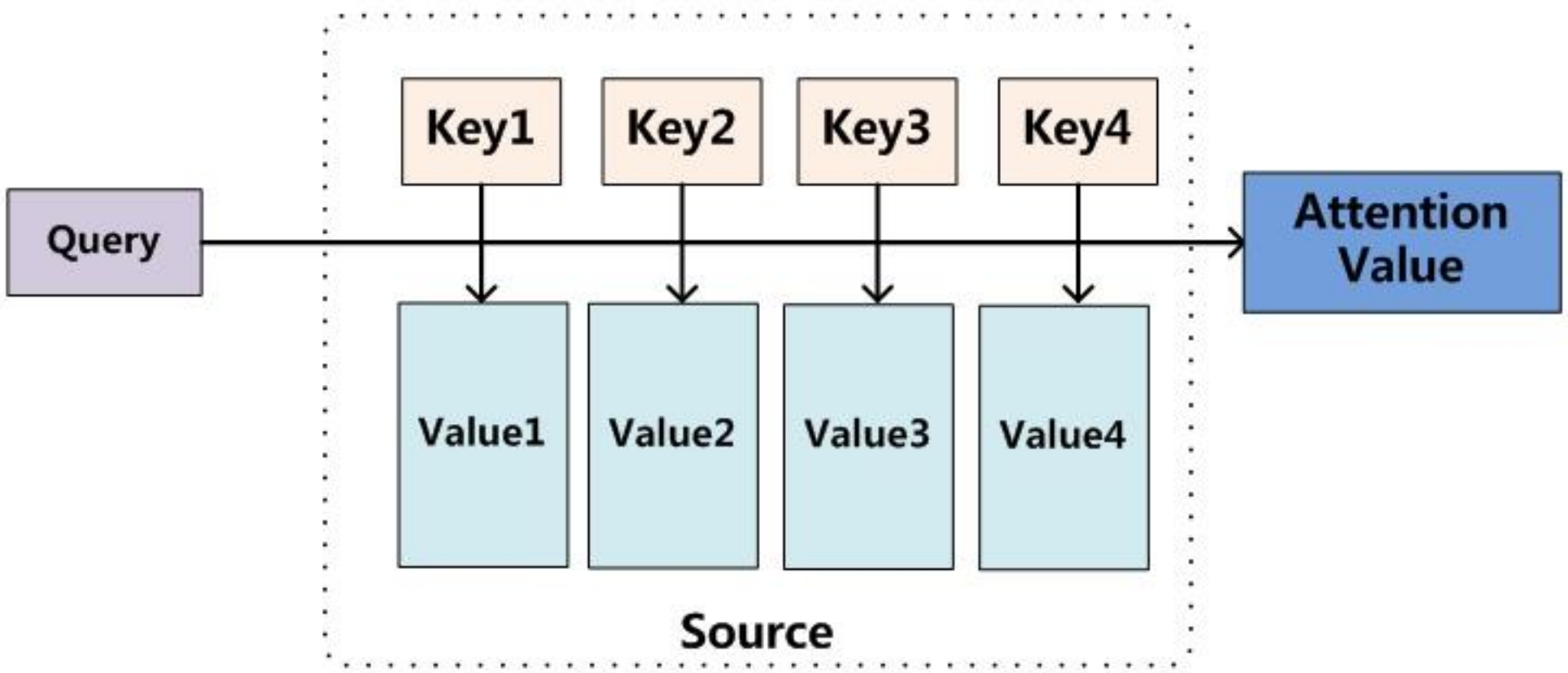
- 将source看作是一系列的对，对于每个输入query，attention机制会将其和每个key进行比对，得到对应value的权重系数，然后根据计算的每个权重，对所有value进行加权求和。

$$Attention(Query, Score) = \sum_{i=1}^{L} f(Query, Key_i) * Value_i$$

- 其中 L 表示 Source 的长度。
- 机器翻译时，Key 和 Value 相同，都是 Source 中每个时刻隐藏层状态。

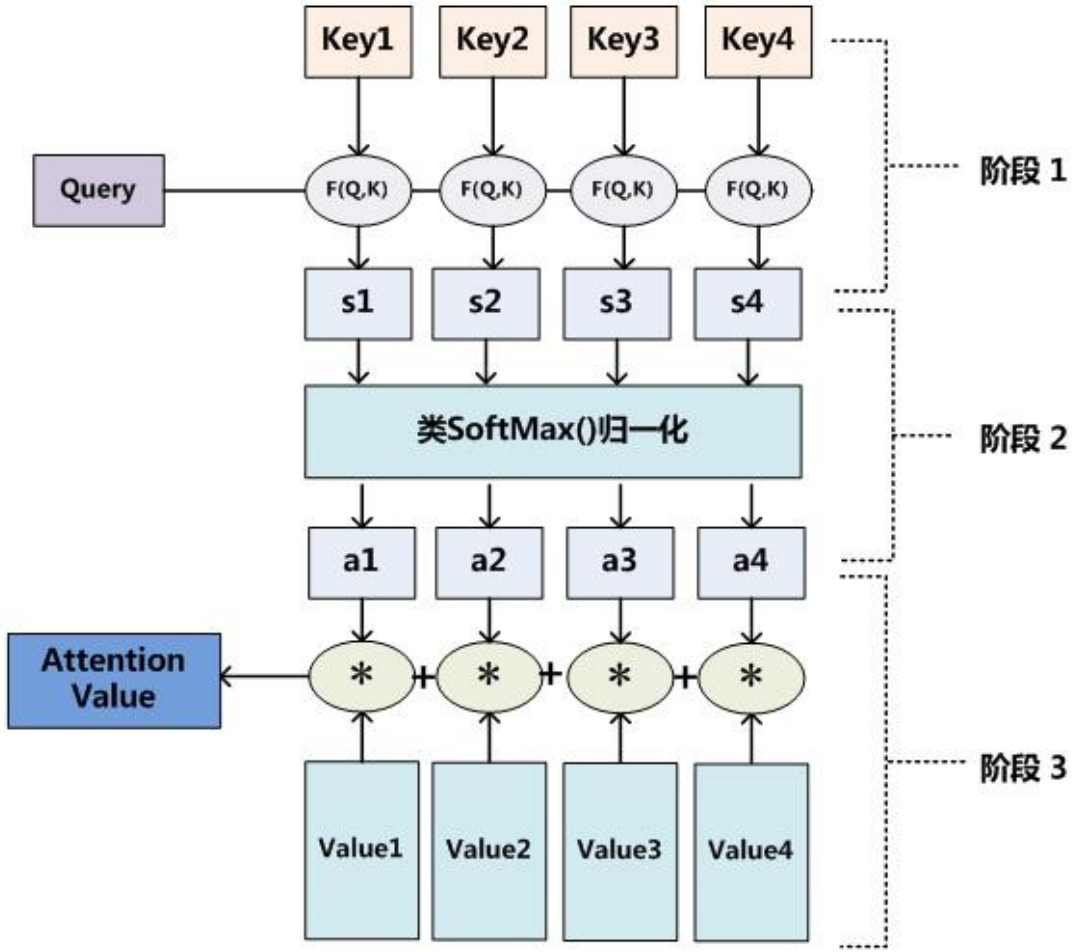
Attention 机制本质

- 将 source 看作是一系列的对，对于每个输入 query，attention 机制会将其和每个 key 进行比对，得到对应 value 权重系数，然后根据计算的每个权重，对所有 value 进行加权求和。



Attention 机制本质

- 这里的 attention 仍然是学习如何从大量信息中筛选出和当前 query 最相关的信息，权重越大的 value 说明越重要。



Attention Step1

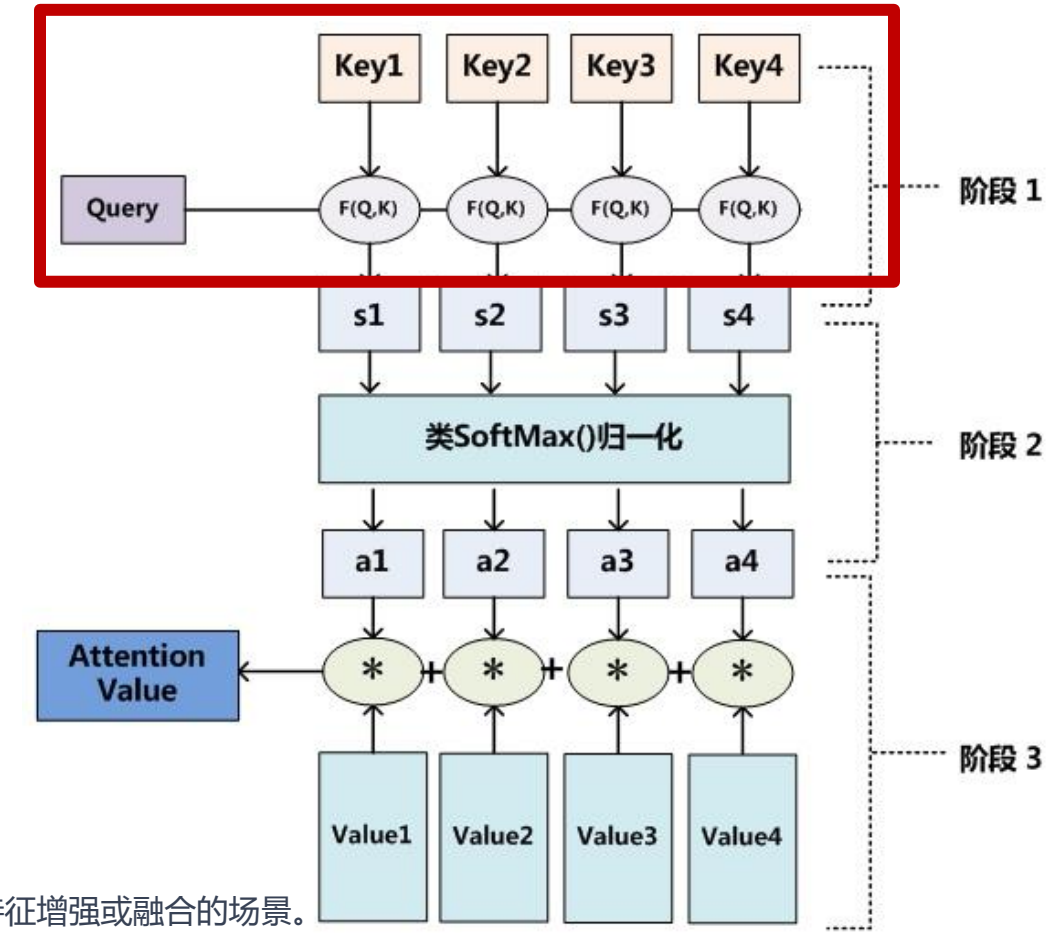
- 将 Query 和每一个 key 进行计算 $F(Query, Keyi)$，得到该 query 和每个 value 的相关性 si：

$$F(Q, K_i) = Q^T K_i$$

$$F(Q, K_i) = Q^T W_a K_i$$

$$F(Q, K_i) = W_a[Q; K_i]$$

$$F(Q, K_i) = v_a^T tanh[W_a Q + U_a K_i]$$

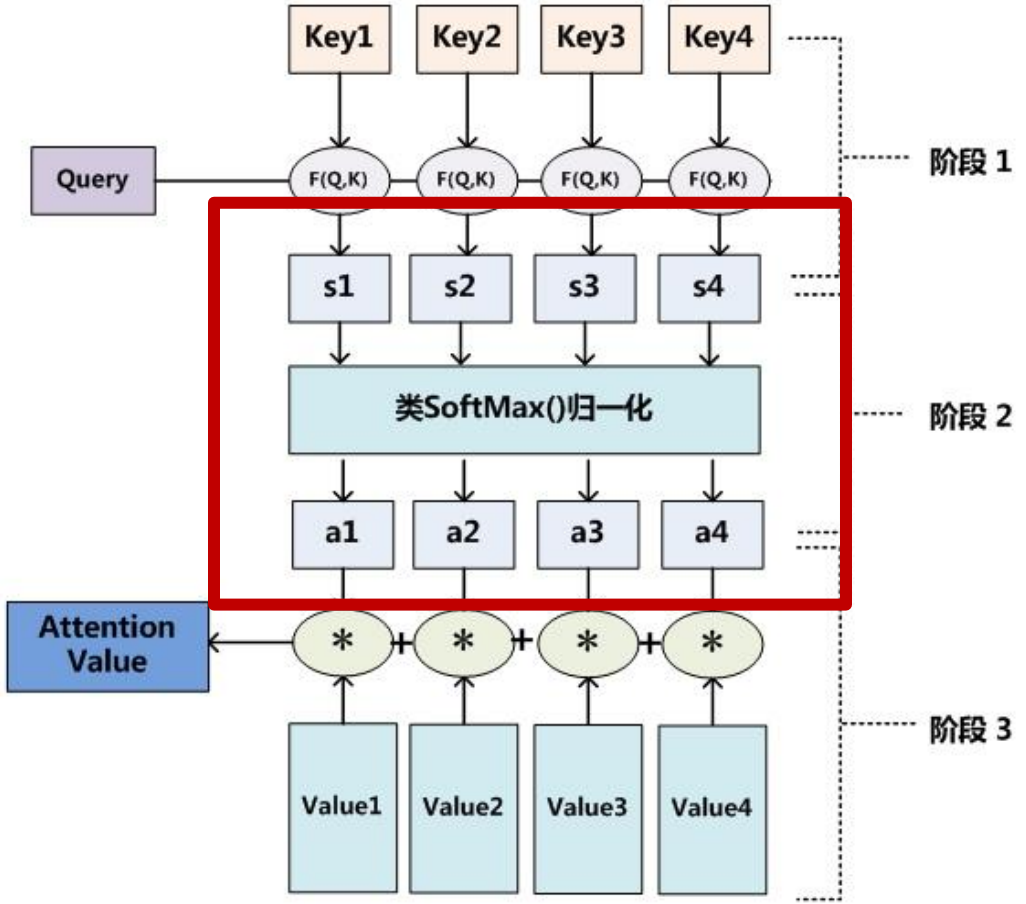


- 矩阵点乘的核心意义在于对齐位置的元素级交互，适用于需要保留空间结构、进行局部特征增强或融合的场景。

Attention Step2

- 将 Step1 得到的相关性，进行 Softmax 归一化，使其符合概率分布的形式 ai。

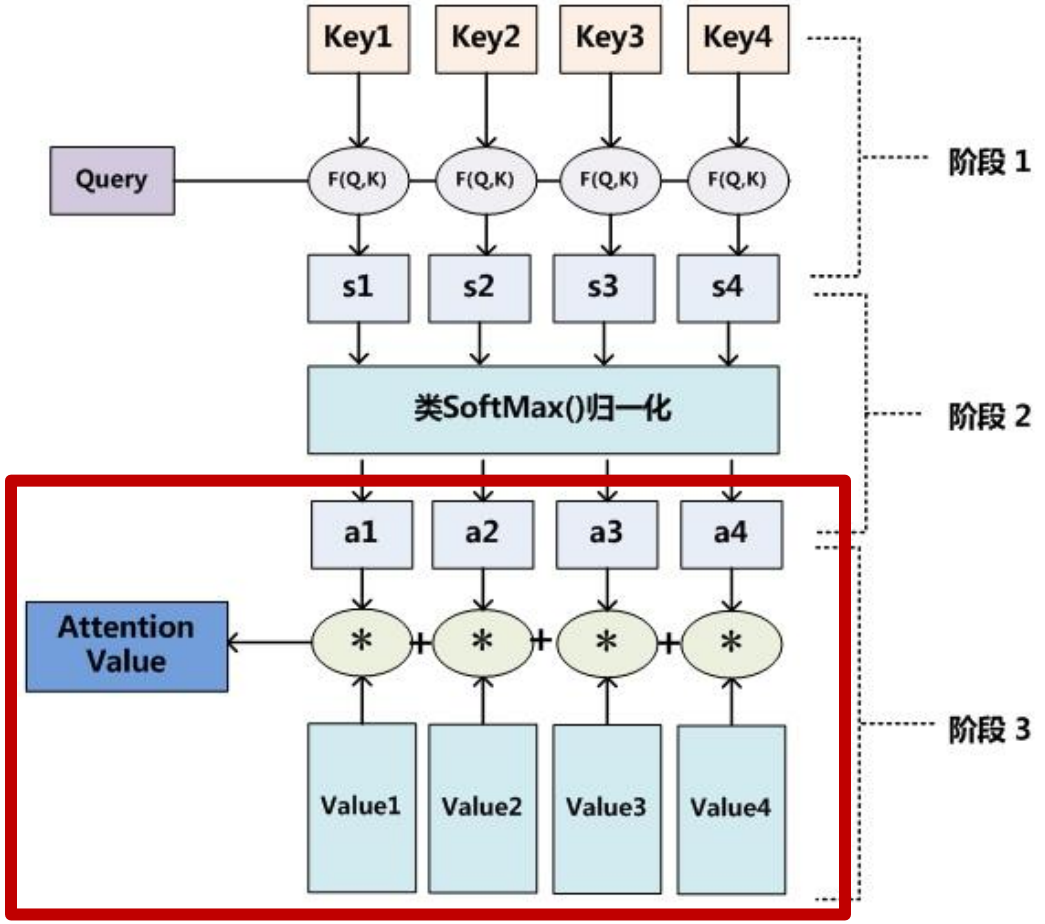
$$a_i = softmax(f(Q, K_i)) = \frac{\exp(f(Q,K_i))}{\sum_j \exp(f(Q,K_j))}$$



Attention Step2

- 根据 Step2 权重系数 a_i，对所有 $value_i$ 进行加权求和：

$$Attention(Query, Score) = \sum_{i=1}^{L} a_i \cdot Value_i$$



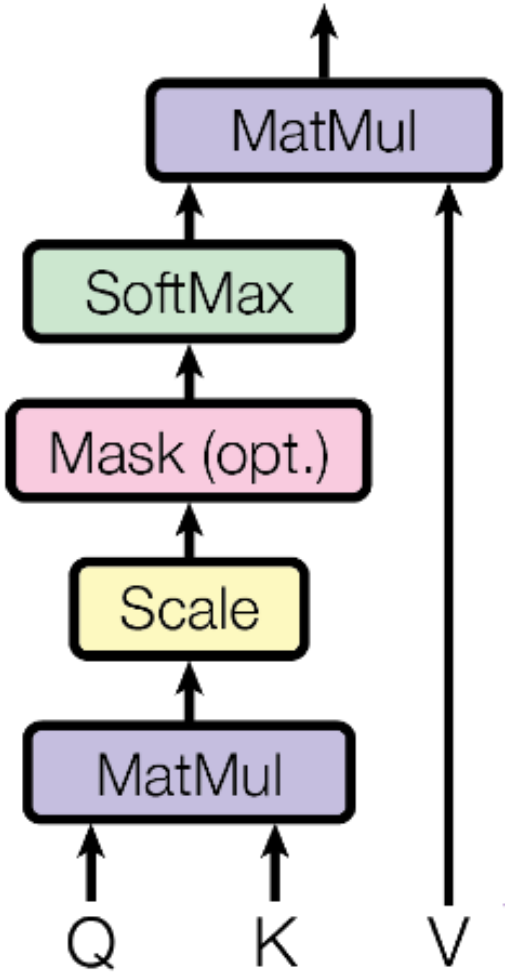
03

Self Attention

核心原理

Self-Attention（自注意力机制）

- 使输入序列中的每个元素能够关注并加权整个序列中的其他元素，生成新的输出表示，不依赖外部信息或历史状态。
  1. Self-Attention允许输入序列中的每个元素都与序列中的其他所有元素进行交互。
  2. 通过计算每个元素对其他所有元素的注意力权重，然后将权重应用于对应元素的表示。
  3. Self-Attention不依赖于外部信息或先前的隐藏状态，完全基于输入序列本身。



Attention == Self-Attention?

- **Attention 计算公式：**

$$\text{Attention}(Q_d, K_e, V_e) = \text{softmax}\left(\frac{Q_d K_e^T}{\sqrt{d_k}}\right) V_e$$

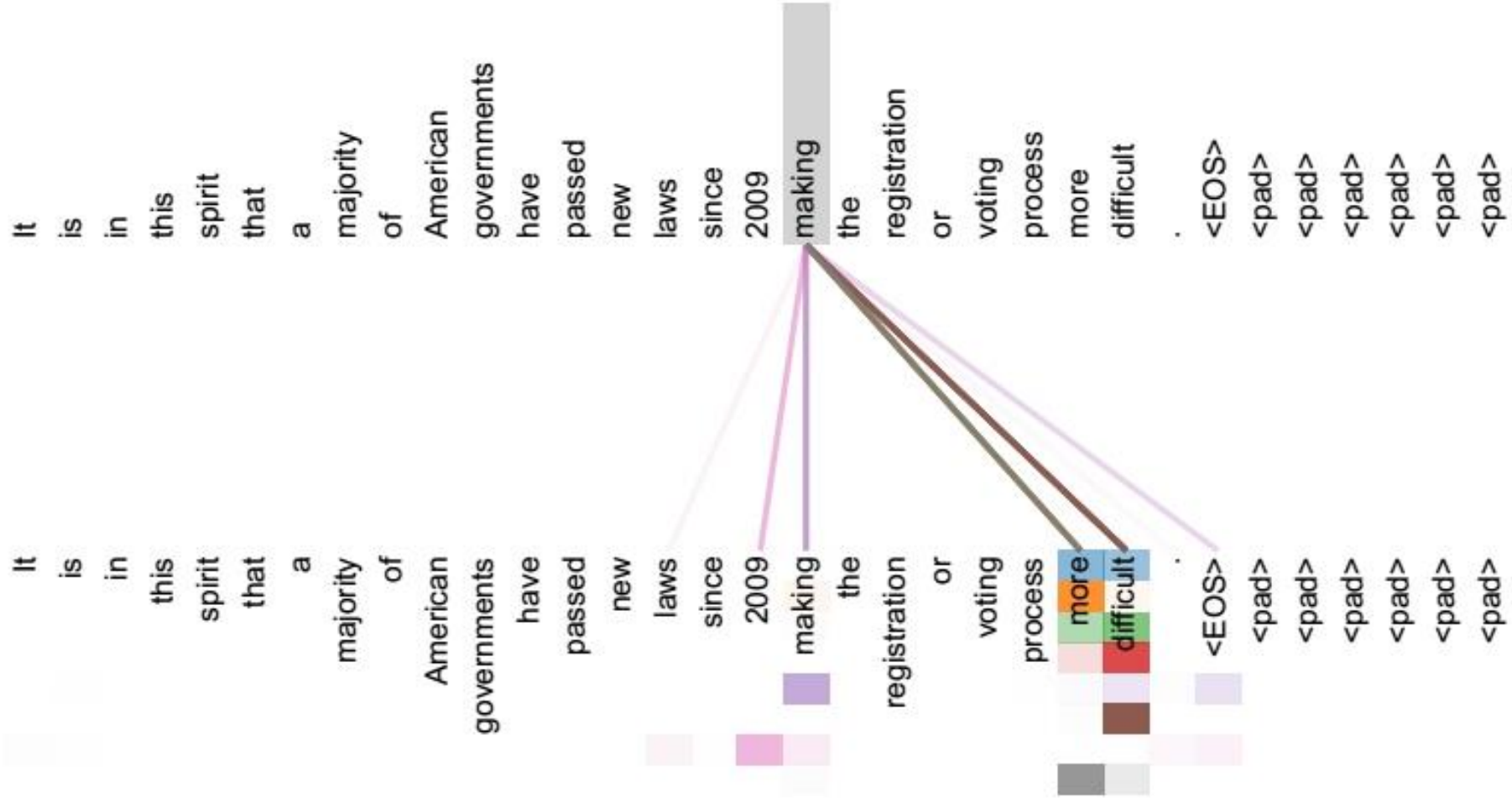
- **Self-Attention 计算公式：**

$$\text{SelfAttention}(X) = \text{softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_k}}\right) (XW_V)$$

- target = source 下 attention 机制，被看作source自身或者target自身发生的attention机制。

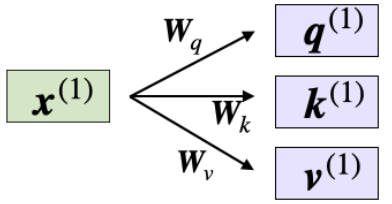
self attention学到了哪些特征呢?

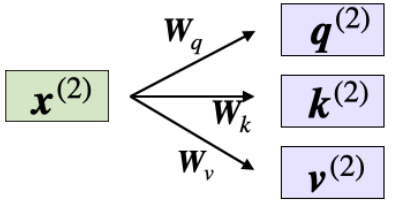
- self attention 一定程度上可以捕获一个句子中单词之间的一些句法特征或者语义特征。

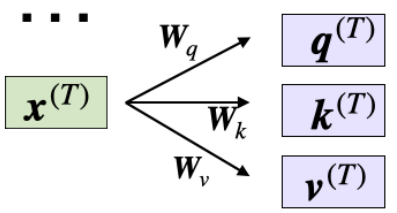


Step1： Defining the Weight Matrices

- The respective query, key and value sequences are obtained via matrix multiplication between the we ight matrices W and the embedded inputs x:
- Query sequence:
  - $\boldsymbol{q}(i) = \boldsymbol{W}_q \boldsymbol{x}(i)\ for\ i \in [1, T]$
- Key sequence:
  - $\boldsymbol{k}(i) = \boldsymbol{W}_k \boldsymbol{x}(i)\ for\ i \in [1, T]$
- Value sequence:
  - $\boldsymbol{v}(i) = \boldsymbol{W}_v\, \boldsymbol{x}(i)\ for\ i \in [1, T]$

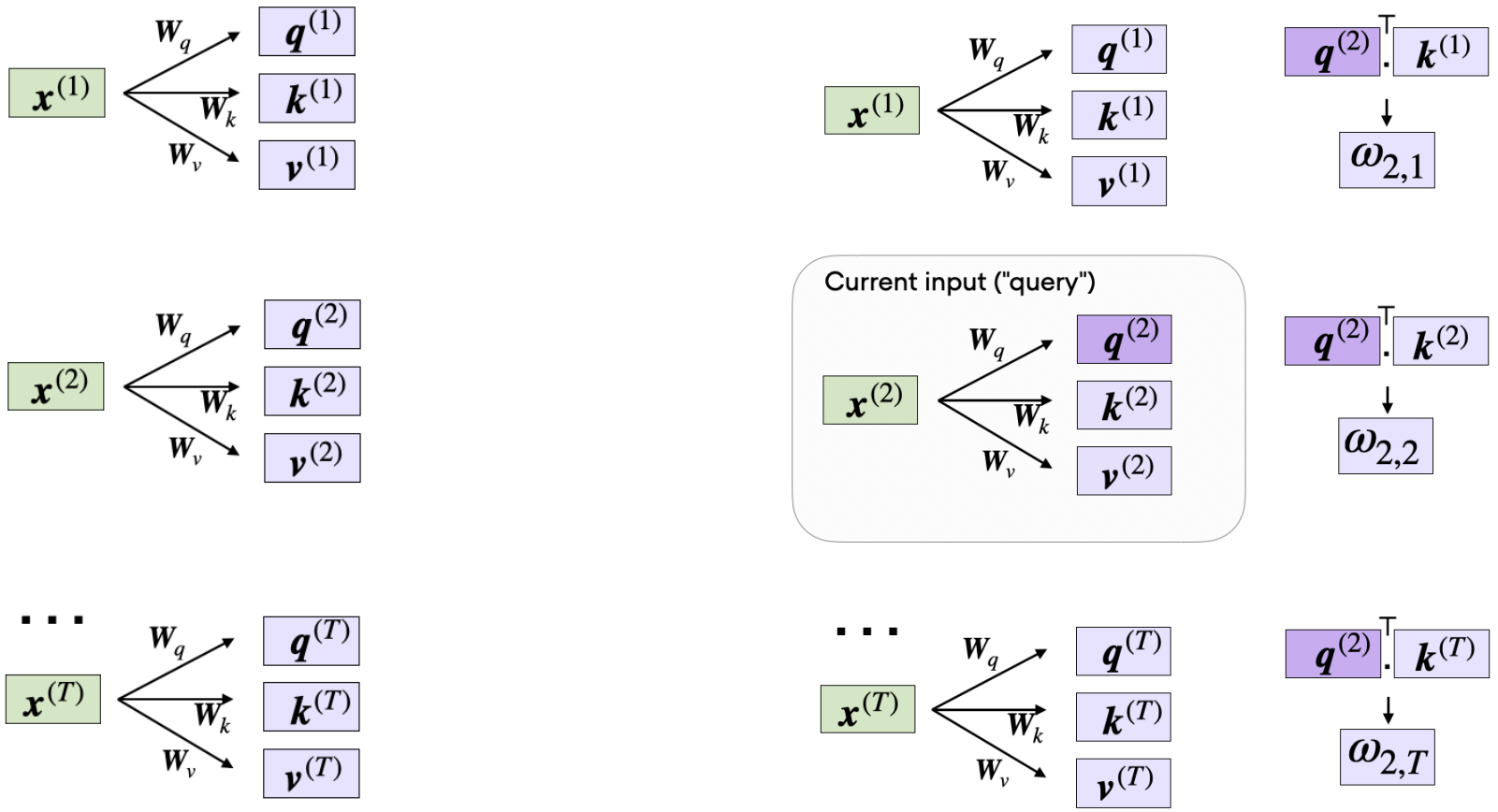






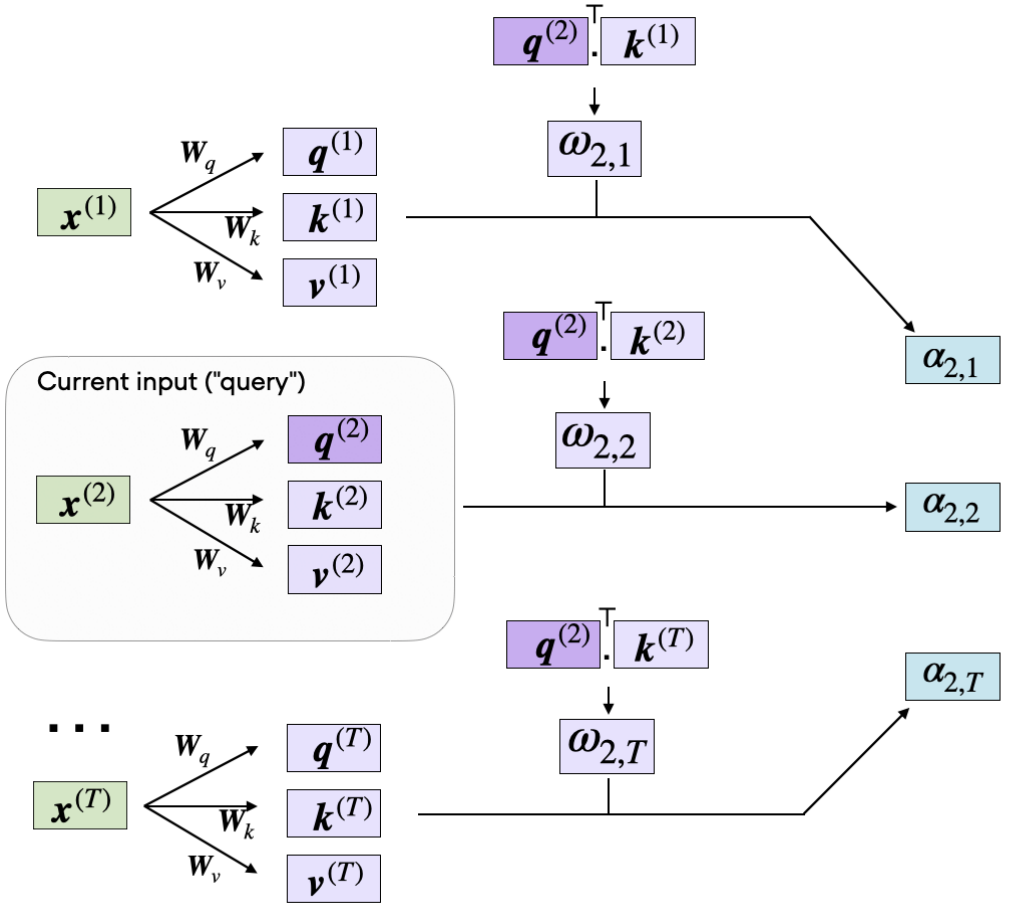
Step2： Computing Unnormalized Attention Weights

- 编码单词的查询向量（Query）与句子中每一个单词的键向量（Key）分别进行点积计算。



Step3： Computing Unnormalized Attention Weights

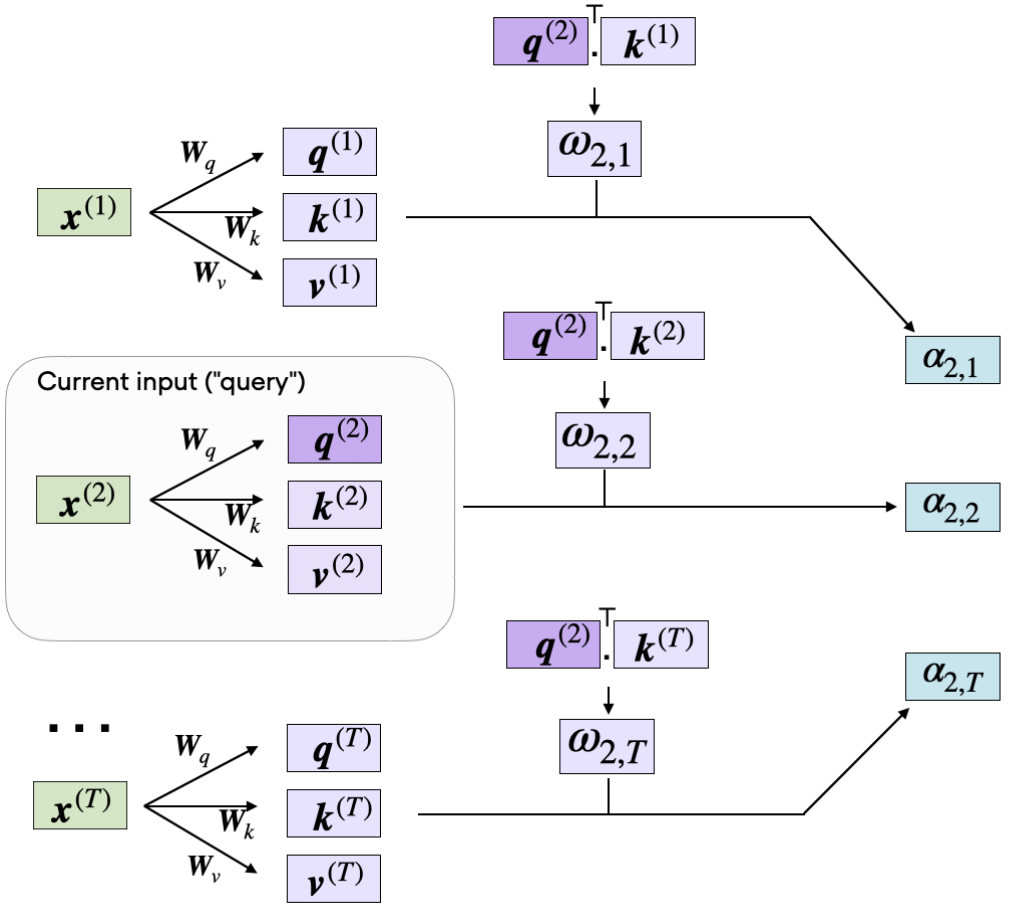
- 除于一个缩放系数 $\sqrt{d}$（d 是指 q 和 k 的维度），内积后 α 会随着维度增大而增大，除 $\sqrt{d}$ 相当于标准化输出的数据。



where $\alpha_{2,i} = \text{softmax}\left(\frac{\omega_{2,i}}{\sqrt{d_k}}\right)$

Step4：Computing the Attention Scores

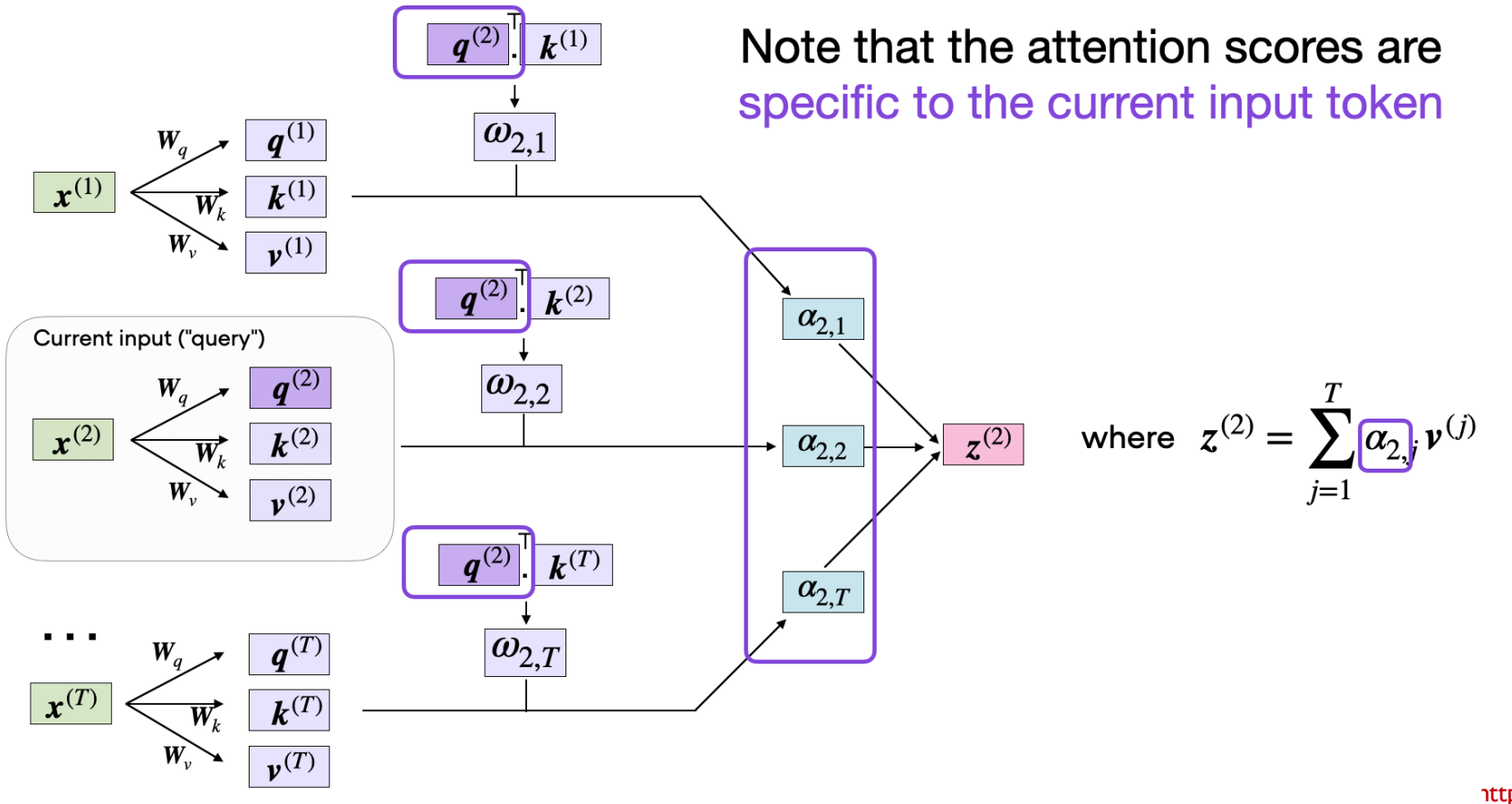
- to obtain the normalized attention weights, α, by applying the softmax function.



where $\alpha_{2,i} = \text{softmax}\left(\frac{\omega_{2,i}}{\sqrt{d_k}}\right)$

Step5： Computing the Attention Scores

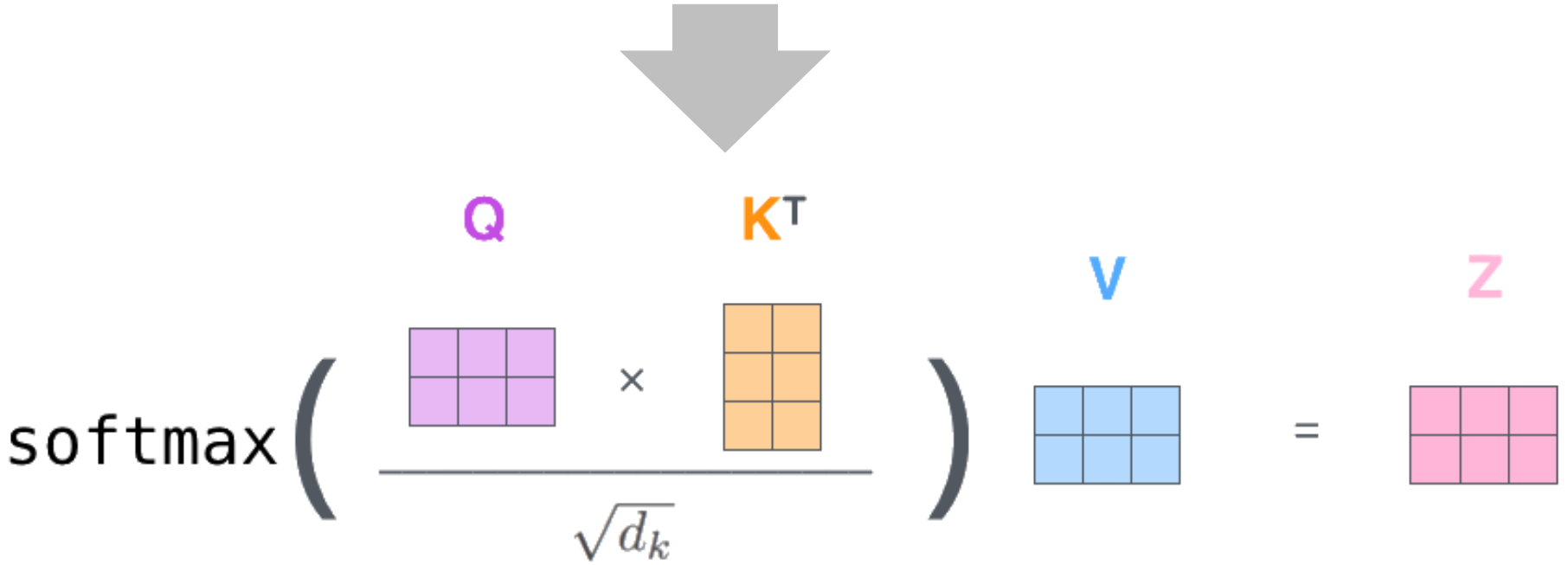
- compute the context vector $z(2)$, which is an attention-weighted version of our original query inpu t $x(2)$, including all the other input elements as its context via the attention weights:



Self-Attention

- Self-Attention 计算公式：

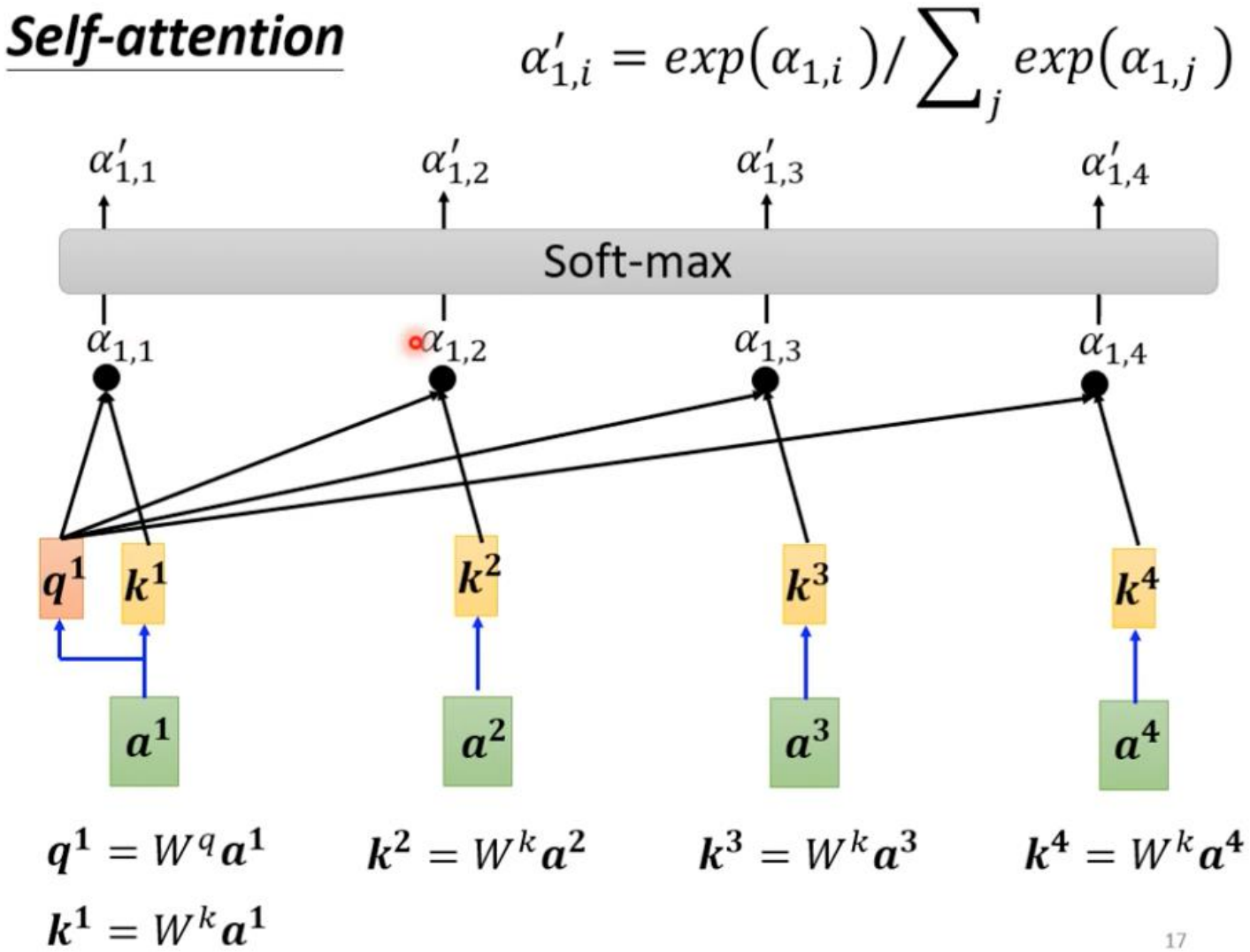
$$\text{SelfAttention}(X) = \text{softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_k}}\right)(XW_V)$$



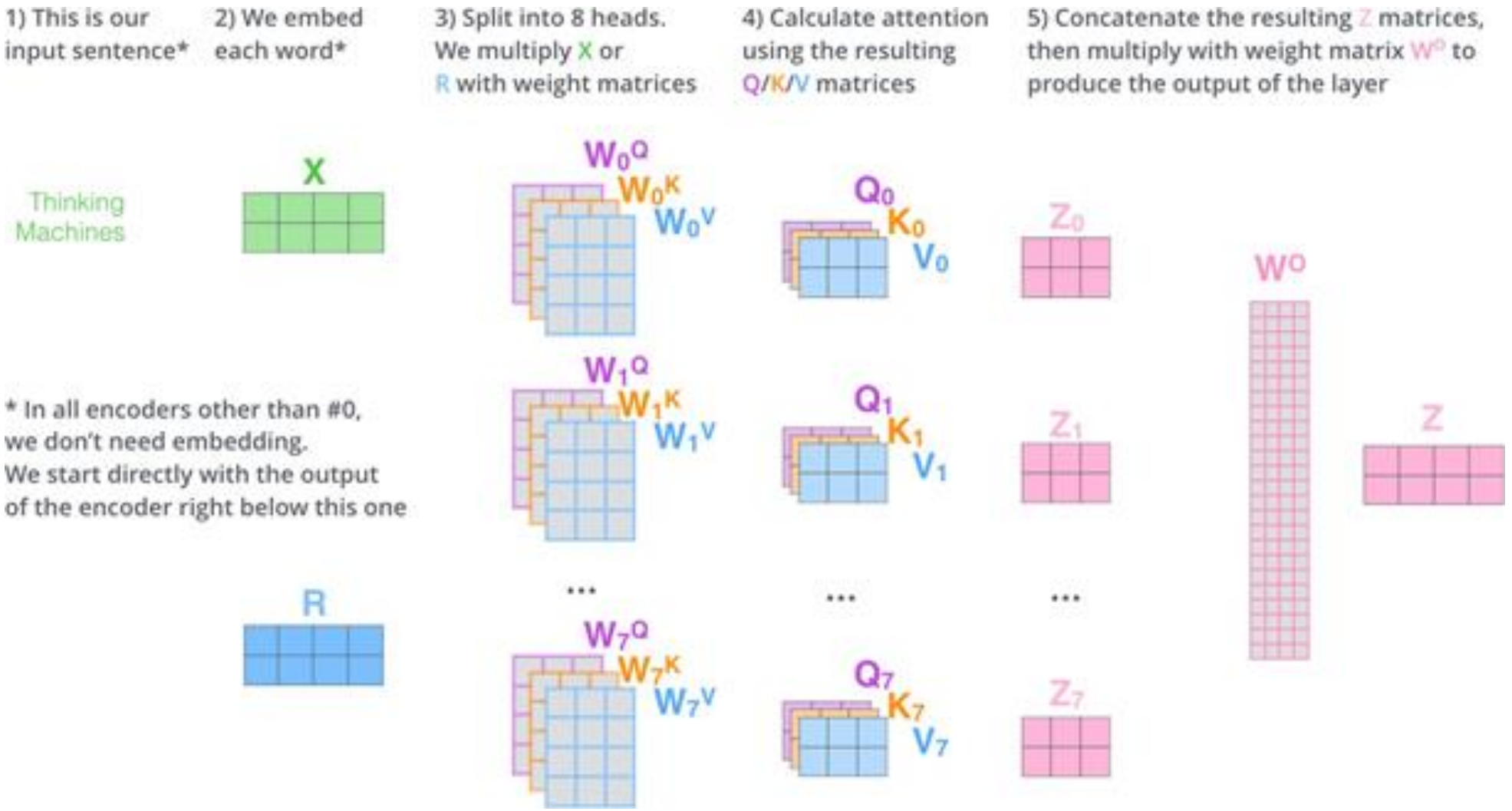
Attention == Self-Attention?

	Attention	Self-Attention
数据来源	Q 与 K/V 通常不同源（如编码器与解码器）	Q、K、V 均来自同一输入序列
功能目标	跨序列关联（如翻译中源语言→目标语言）	单序列内部依赖建模（如指代消解）
复 杂 度	O(n*m)（n、m为两序列长度）	$O(n^2)$（单序列长度n）
位置编码	需显式编码（如RNN隐藏状态）	需额外位置编码（如Sinusoidal/RoPE）
典型场景	Seq2Seq 模型、跨模态任务	Transformer、BERT 等单模态模型

网上的不同版本——李宏毅


Self-attention
$\alpha'_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$
$\alpha'_{1,1}$
$\alpha'_{1,2}$
$\alpha'_{1,3}$
$\alpha'_{1,4}$
Soft-max
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q^1
k^1
k^2
k^3
k^4
a^1
a^2
a^3
a^4
$q^1 = W^q a^1$
$k^2 = W^k a^2$
$k^3 = W^k a^3$
$k^4 = W^k a^4$
$k^1 = W^k a^1$
17

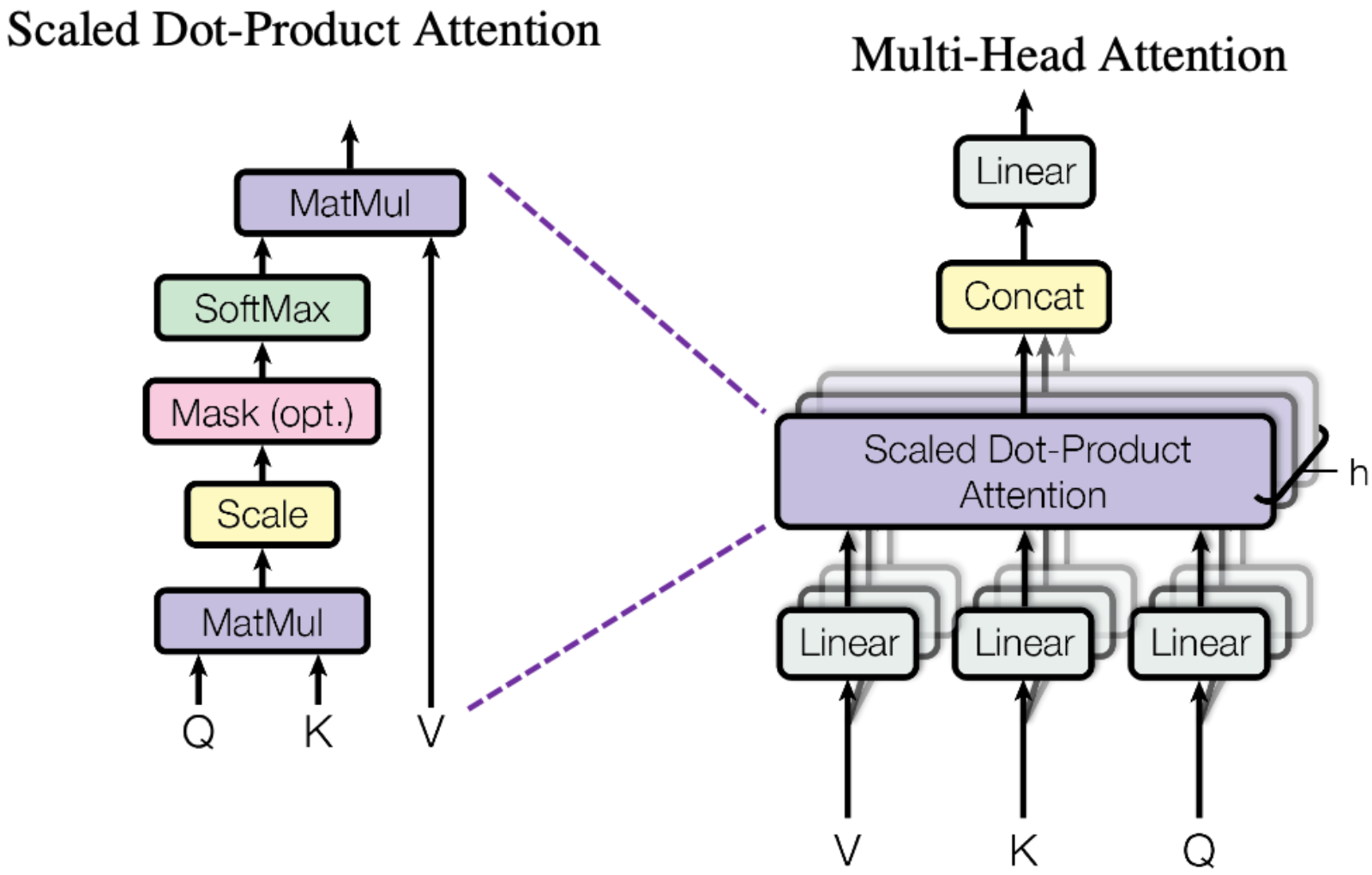
网上的不同版本——李宏毅


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

04

Multi-Head Self Attention（MHA）

Multi-Head Self Attention



Multi-Head Self Attention

- 多头自注意力是Self Attention的扩展，通过并行运行多个独立的注意力头（Heads），分别关注输入的不同特征子空间，最后将结果拼接并通过线性层融合。公式如下：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

- 其中每个头的计算为：

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

Multi-Head Self Attention

维度	Self Attention	Multi-Head Self Attention
注意力头数	单头（Single Head）	多头（Multiple Heads，如8或16）
参数共享	所有位置共享同一组Q,K,V权重	每个头拥有独立的Q,K,V权重
特征捕获	仅能捕捉单一维度的特征（如语法或语义）	可同时捕捉多维度特征（如语法、语义、位置等）
计算复杂度	$O(n^2)$ （n为序列长度）	$O(h \cdot n^2)$（h为头的数量，但单头维度降低）
模型容量	表达能力有限	增强模型表达能力，适合复杂任务
实际应用	简单任务	复杂任务

05

Mask-Self Attention

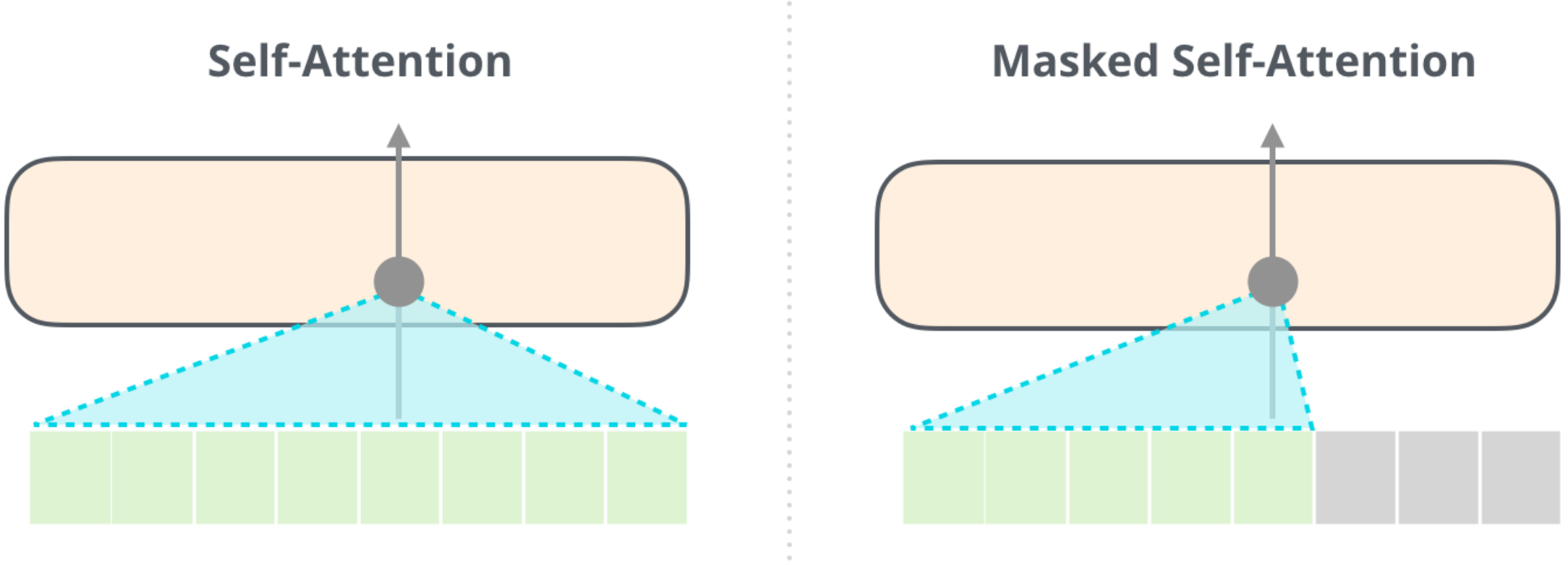
Mask MHA与 MHA 区别

- **Self Attention：**
  - 其特点是双向全序列可见，允许每个位置的token与序列中所有其他位置的token交互。
  - 适用于编码器（Encoder）中对上下文信息的全局建模。

- **Mask Self Attention：**
  - 在标准多头自注意力的基础上引入掩码机制（Masking），限制某些位置可见性，实现特定约束。
  - 应用于解码器（Decoder），确保模型在生成当前输出时仅依赖已生成的前序信息。
  - 避免“偷看”未来的token，保证自回归生成的因果性（Causality）。

masked-Self Attention

- 在解码器里，Self Attention 层只允许关注到输出序列中早于当前位置之前的单词。



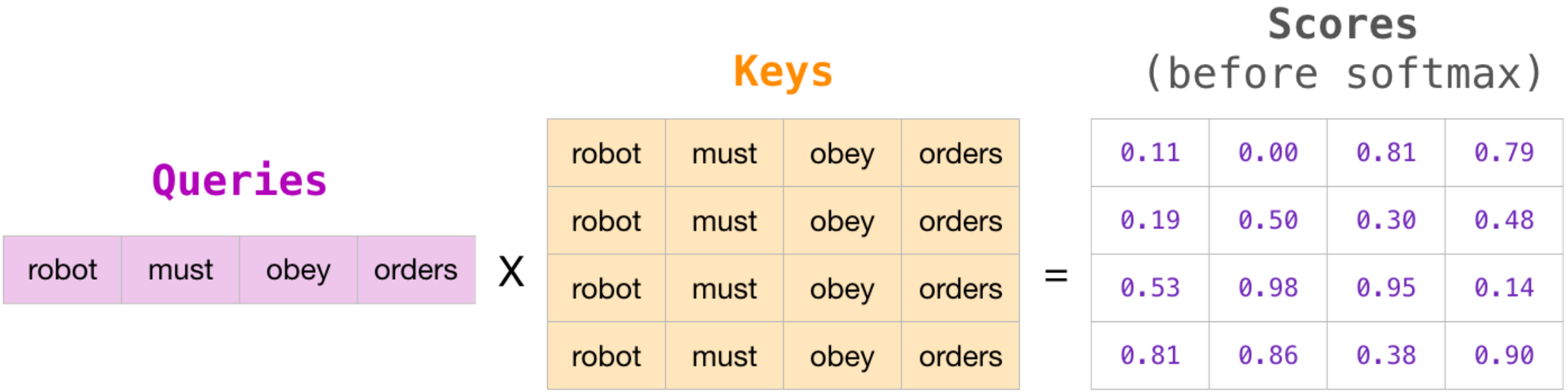
masked-Self Attention

- 在一个语言场景中，序列分为 4 个步骤处理：每步处理一个 Token。由于大模型以 batch size 工作，可以假设模型 batch size 为 4，将序列作为一个 batch 处理。

	Features				Labels
position:	1	2	3	4	
Example: 1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

masked-Self Attention

- 在矩阵的形式中，把 Query 矩阵和 Key 矩阵相乘来计算分数。
- 这里不使用单词，而是使用与格子中单词对应的矩阵。



masked-Self Attention

- 执行完乘法之后，加上下三角形的 attention mask矩阵。它将想要屏蔽的单元格设置为负无穷大或者一个非常大的负数：

Scores
(before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

masked-Self Attention

- 然后对每一行应用 SoftMax，会产生实际的分数，会将这些分数用于 Self Attention。

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax
(along rows)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

Mask MHA与 MHA 区别

特征	Self Attention	Masked-Self Attention
可见范围	全序列双向可见	仅当前及前序位置可见（单向）
掩码类型	可选Padding Mask	必选Padding Mask + Causal Mask
核心应用	编码器或跨序列交互	解码器的自回归生成
实现关键	无额外掩码操作	注意力分数计算时应用Causal Mask
典型模型	BERT编码器、Cross-Attention	GPT、Transformer解码器

总结与思考

总结

- **Self-Attention 是 Transformer 的核心机制：**
  - 通过计算输入序列中不同位置之间的相关性，实现全局依赖建模；
  - Self-Attention 是 Transformer 模型强大表达能力的关键。

- **为了提升效率与效果，Attention 架构持续演进：**
  - MQA（Multi-Query Attention）：共享多个查询头的键和值，降低计算开销。
  - GQA（Grouped Query Attention）：MQA 与 MHA 的折中方案，兼顾性能与效果。
  - MLA（Multi-head Linear Attention）：探索更高效的注意力计算方式，适应长序列处理。

总结

- **未来趋势：高效、可扩展、适合长上下文**
  - 减少复杂度：随着大模型发展，通过优化 Attention 计算复杂度提出 Linear Attention 等。
  - 长序列建模：结合稀疏注意力与动态路由，进一步压缩KV Cache。
  - 多模态扩展：探索跨模态注意力交互，如视觉-语言联合表征。

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

GitHub https://github.com/chenzomi12/AIInfra

引用与参考

- https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers
- https://shangzhih.github.io/jian-shu-attentionji-zhi.html
- https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html
- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- https://www.gnn.club/?p=2729
- http://www.myhz0606.com/article/kv-cache
- https://spaces.ac.cn/archives/10091

- PPT 开源在：https://github.com/chenzomi12/AIInfra