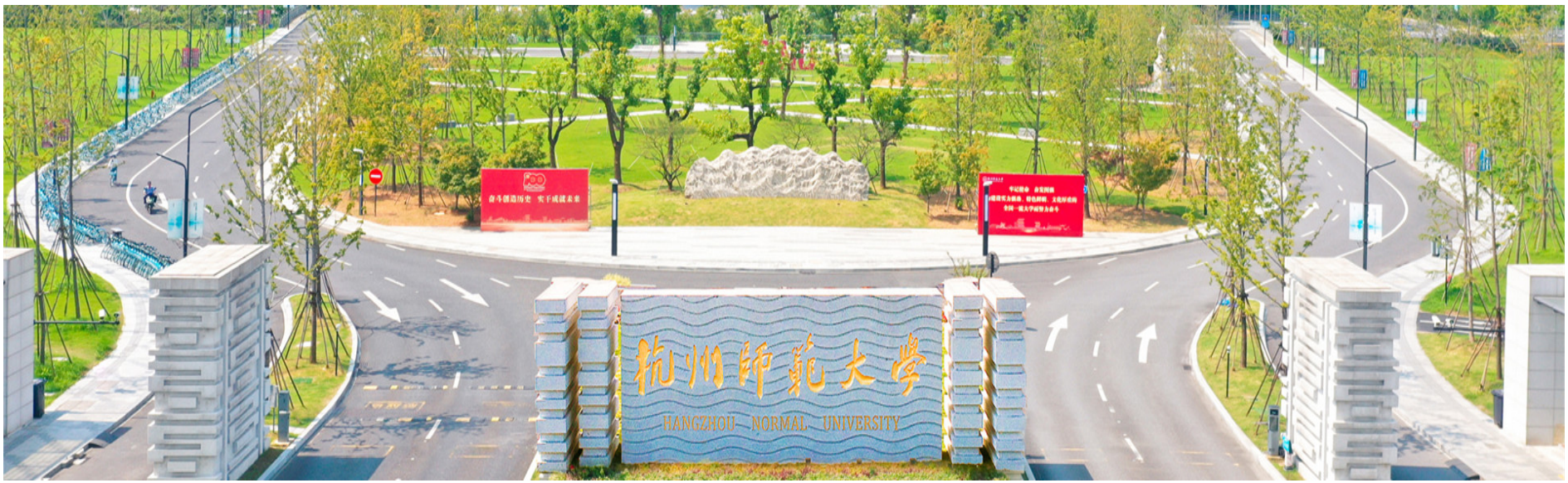
第八讲 - 注意力机制

张建章

阿里巴巴商学院
杭州师范大学

2024-03-01



Attention 机制的提出

神经网络中的注意力机制 (Attention Mechanism) 最早于 2014 年在计算机视觉领域被提出，用于理解模型在进行预测时所观察的内容。在自然语言处理领域，Bahdanau 等[1]在机器翻译任务中首次引入注意力机制，以提升编码器-解码器 (Encoder-Decoder) 架构在机器翻译任务中的性能。Encoder-Decoder 架构广泛应用于 NLP 中的序列到序列任务 (Sequence-to-Sequence, Seq2Seq)，如，机器翻译、问答系统、文本摘要等，这些任务的输入和输出均为文本序列。

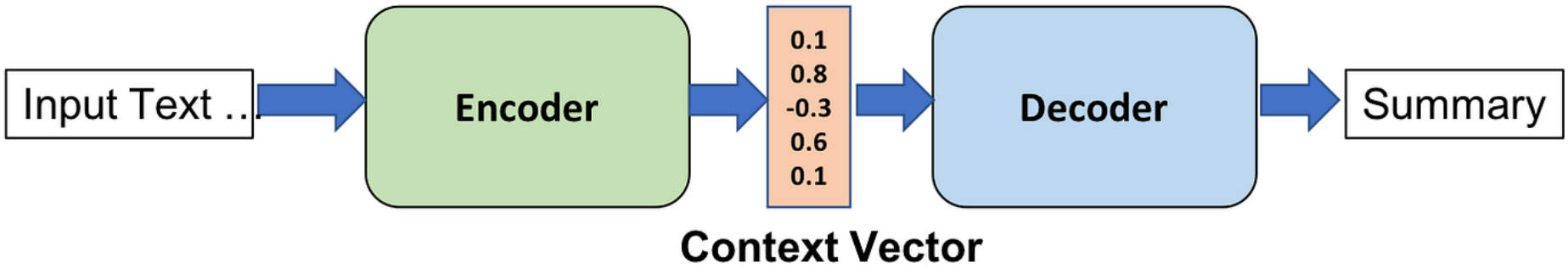


图 1: encoder-decoder 架构示意图 (以文本摘要为例)

[1]Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." 3rd International Conference on Learning Representations, ICLR 2015. 2015.

传统 Encoder-Decoder 模型的不足

Encoder 将输入文本编码为一个固定维度的上下文向量 (context vector)，该上下文向量包含了输入序列的全部信息，并用于 Decoder 的输出过程：① 对任何长度的输入文本都使用单一的固定维度的 context vector 编码，会降低模型处理长文本序列的性能；② 模型在解码输出过程中，无法考虑当前输出词语与输入文本中不同词语之间的相关性。

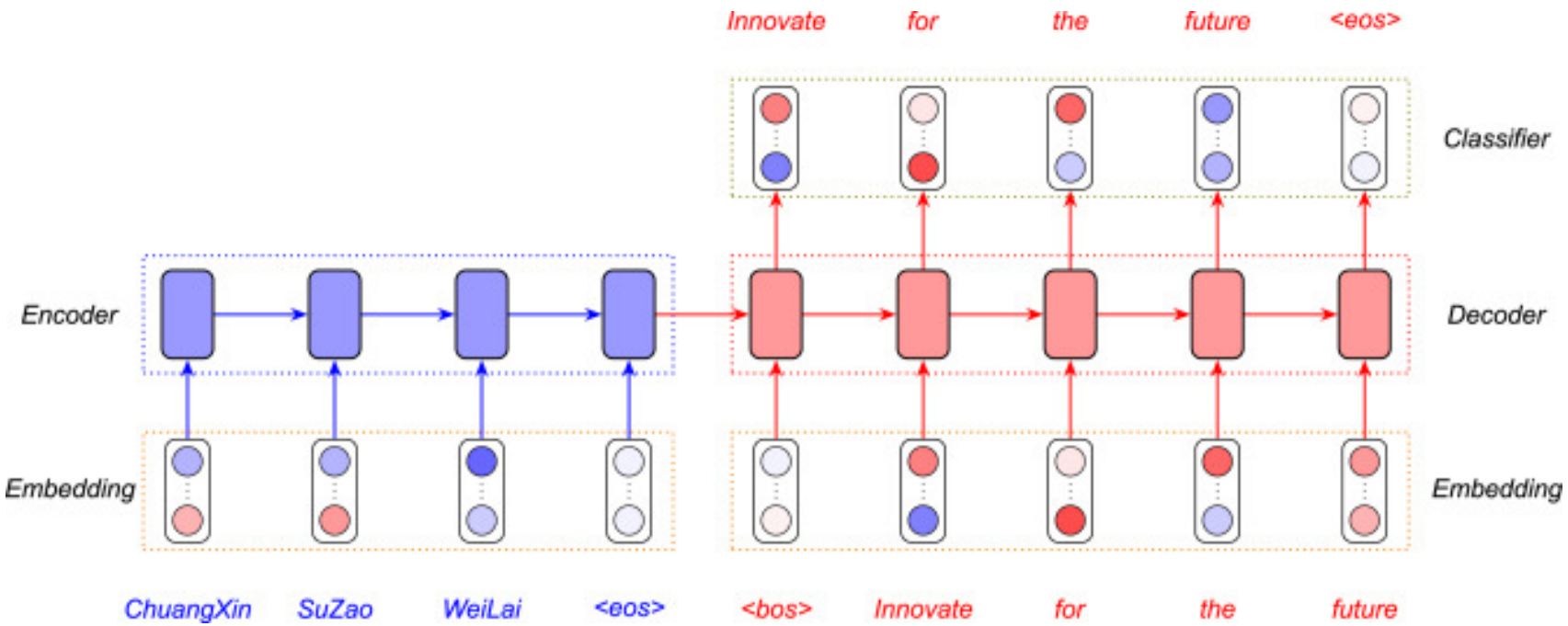


图 2: encoder-decoder 架构示意图 (以机器翻译为例)

为了解决上述不足，Bahdanau 等在 Encoder-Decoder 模型中加入了 Attention 机制，并应用于机器翻译任务，对齐源语言与目标语言。Attention 机制背后的核心思想是，输入序列中的不同部分对输出序列中的每一部分贡献不同，因此，Decoder 在解码输出不同的词语时，应更加关注输入文本序列中与当前输出最相关的那些词语，为其赋予更多权重。

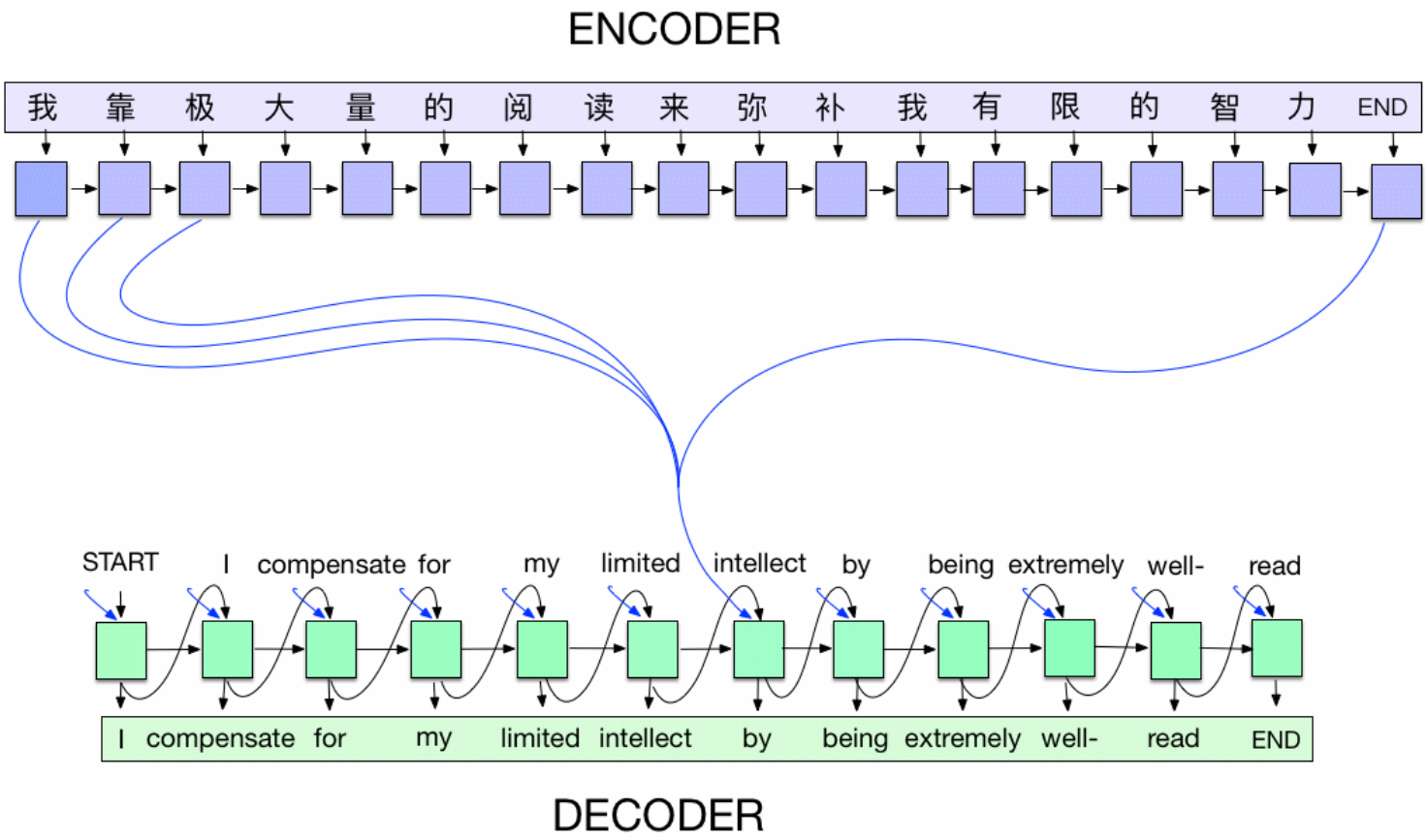


图 3: Attention 机制示意图 (以机器翻译为例, 点我查看交互动图)

Attention 初始网络架构

Bahdanau 等在论文 *Neural machine translation by jointly learning to align and translate* 中提出的 Attention 模型架构如下图所示。

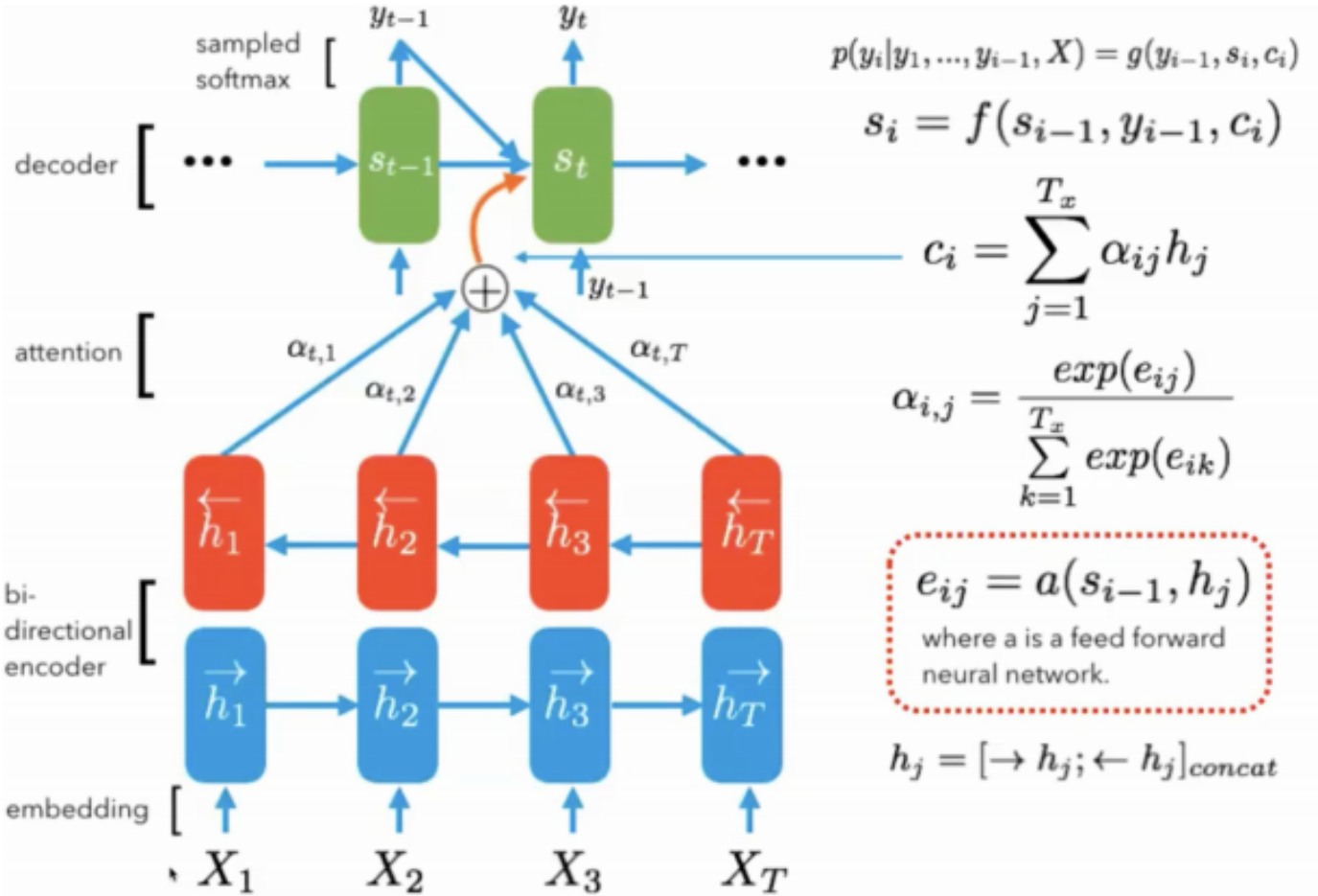


图 4: Attention 机制的初始架构

Attention 初始形式计算过程 I

应用任务为机器翻译，输入为源语言 (source language)，输出为目标语言 (target language)，Encoder 和 Decoder 均使用 RNN 单元。

(1) 输出层预测当前时间步的输出词语时使用上一个输出词语 y_{i-1}、当前时间步的隐藏状态 (hidden state)s_i、由 Attention 计算得出的上下文向量 (context vector)c_i;

$$p(y_i|y_1, y_2, ..., y_{i-1}, \vec{X}) = g(y_{i-1}, s_i, c_i)$$

(2) 当前时间步的 hidden state 由上一个时间步的 hidden state s_{i-1}、上一个时间步的输出 y_{i-1} 和当前时间步的 context vector 计算得出;

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

(3) context vector 由输入文本序列中全部词语的 hidden state 线性组合得到 (加权和)

Attention 初始形式计算过程 II

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

(4) 每个输入词语的 hidden state 由 Encoder 中的双向 RNN 的 hidden state 拼接 (concatenate) 构成，

$$h_j = [\overrightarrow{h_j}, \overleftarrow{h_j}]$$

(5) 权重值 α_{ij} 由 softmax 函数计算得出，表示输入文本中第 j 个词对输出序列中第 i 个词的重要性权重

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

Attention 初始形式计算过程 III

(6) 能量 (energy) 值 e_{ij} (也称作评分, score) 由一个前馈神经网络 a 使用前一个时间步的 hidden state$s_i - 1$ 和第 j 个输入词的 hidden state 计算得出:

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

总结：Attention 机制的初始形式旨在实现机器翻译中目标语言与源语言之间的软对齐 (soft alignment)，即，在输出目标语言的词语时，通过权重值对目标语言中不同的词语施加不同的注意力，以反映输入语言中不同词语对翻译输出目标语言当前词语的重要性不同。

Attention 统一化描述

将 Attention 从 Encoder-Decoder 框架中抽取出来，借助数据库检索中的相关概念，可对 Attention 机制的一般形式进行统一描述。Attention 机制的一般形式包括三个主要部分: (1) 查询 (Query); (2) 键 (Key); (3) 值 (Value);

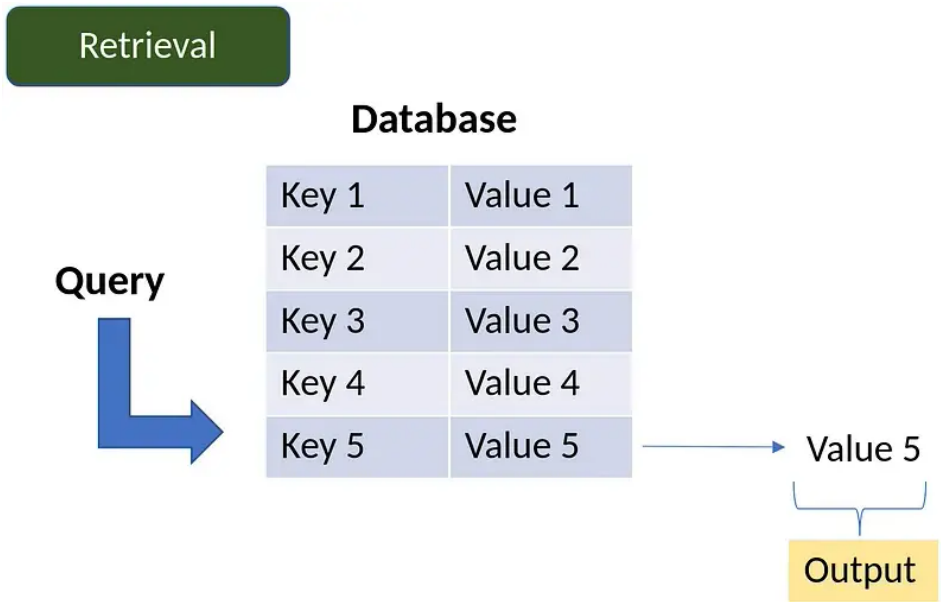


图 5: Attention 机制一般形式示意图 (类比数据库检索)

Attention 的计算过程类似数据库检索过程：使用给定的 query，计算数据库中与 query 相似的 key，取出相似的 key 对应的 value。

Attention 一般形式的计算过程

令 q、k、v 分别表示查询向量、key 向量、值向量，Attention 一般形式的计算过程可表示如下：

$$attention(q,k,v) = \sum_i similarity(q,k) * v$$

上述计算过程可进一步拆分为三个阶段:

(1) 计算 query 与 key 向量的相似度/相关性评分，常用的评分函数有点积、缩放的点积；

(2) 将评分归一化为概率，表示权重;

(3) 依权重对 value 向量进行加权求和。

思考: Attention 初始形式中的 query、key、value、评分函数分别是什么?

Attention 一般形式的计算过程

上述三阶段计算过程可图示化表示如下：

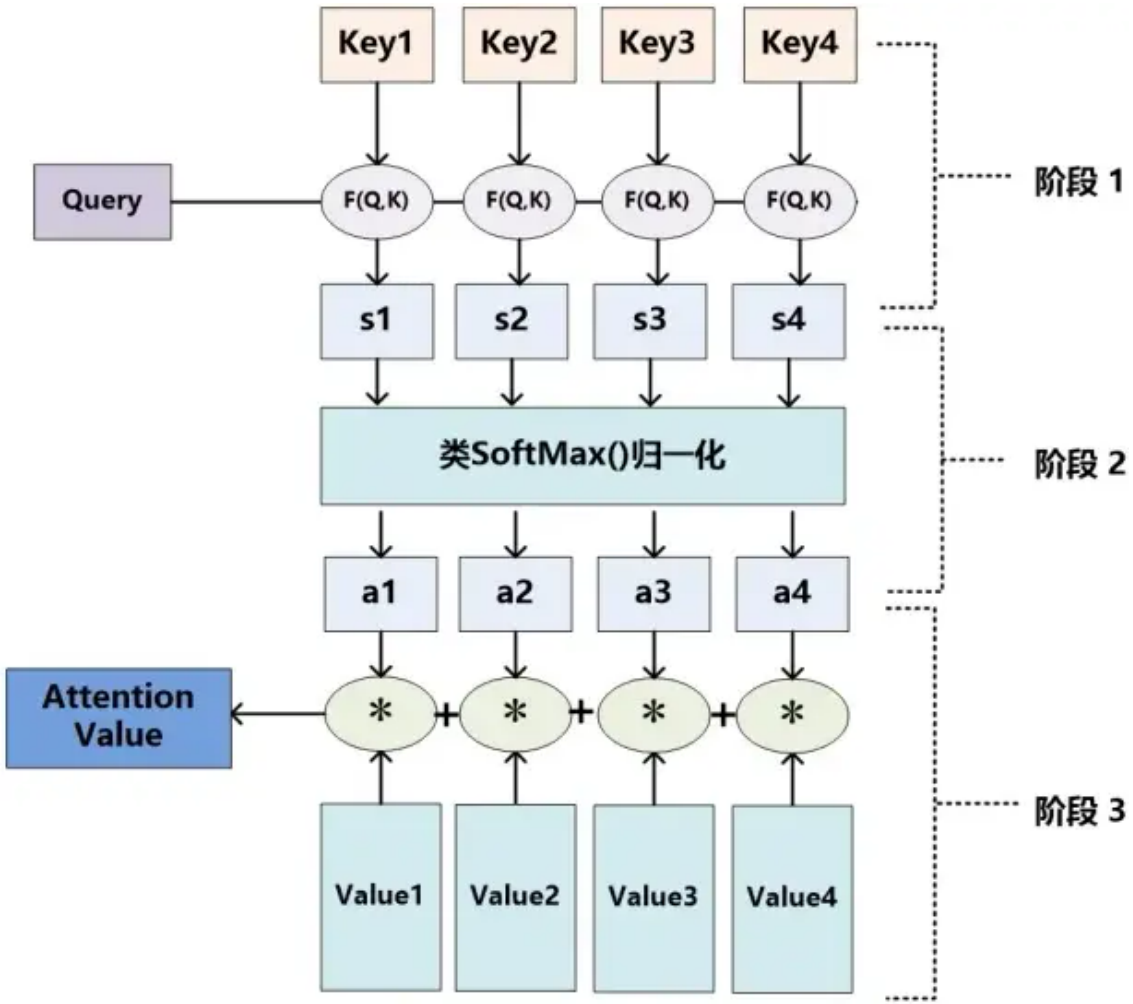


图 6: Attention 一般形式的计算过程示意图

传统的 Attention 机制是输出对输入中的不同部分赋予不同的权重(施加不同的注意力)。例如，在机器翻译中输入是源语言，输出是目标语言。

自注意力机制 (self attention) 是输入中的某一部分对输入中的其他部分施加注意力，即，self attention 计算输入中不同部分之间的相关性。例如，对于一条输入语句，自注意力发生在某个词语与句子中其他词语之间，建模词语与其上下文之间的相关性。

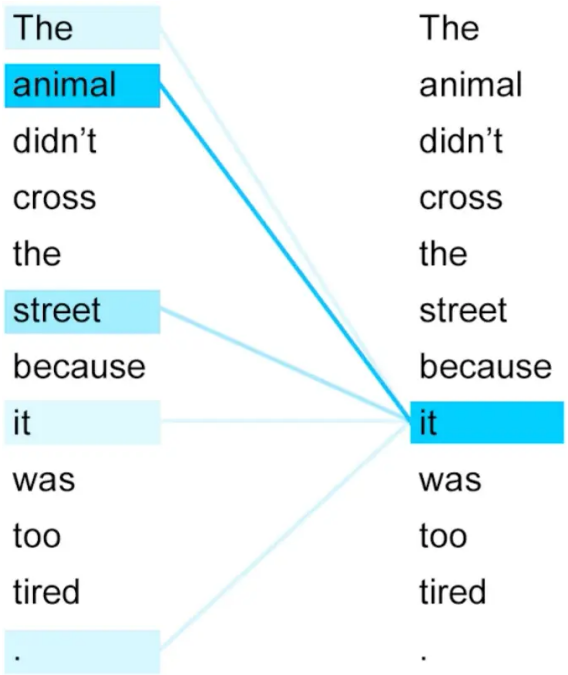


图 7: self attention 示例

Self Attention 的输入和输出

Self Attention 的输入是 N 个词向量，输出也是 N 个词向量。例如，在下图中有四个输入向量 $\boldsymbol{a}$，对于每个输入向量都有一个对应的输出向量 $\boldsymbol{b}$，蓝色圆角矩形表示 Self Attention 计算过程，输出向量 $\boldsymbol{b}^1$ 是综合考虑了每一个输入向量 $(\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3, \boldsymbol{a}^4)$ 与 $\boldsymbol{a}^1$ 的相关性后计算得出的。因此，Self Attention 输出的词向量也称为上下文词向量 (contextual word embedding)。

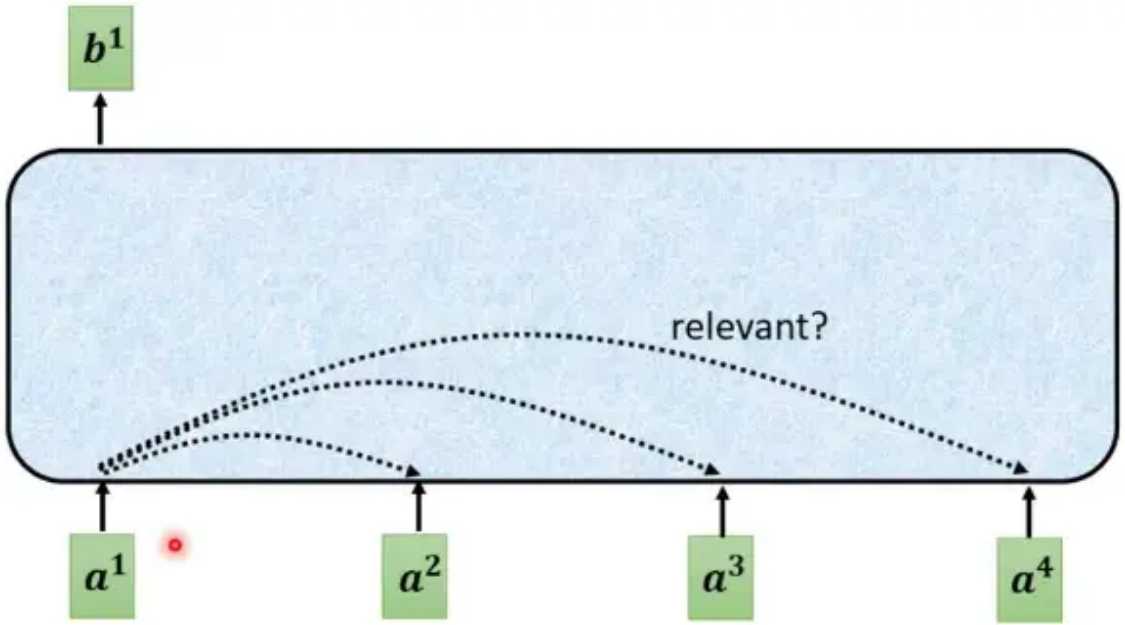


图 8: self attention 的输入和输出示意图

Self Attention 的计算过程

Self Attention 的计算过程在 Attention 一般形式的三阶段计算过程的基础上增加了一个计算 query 向量、key 向量和 value 向量的阶段:

(1) 使用输入向量 $\boldsymbol{a}$ 乘以三个权重矩阵 W^q, W^k, W^v 得到向量 $\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$;

(2) 计算 query 向量与 key 向量点积，表示两者的相关性分数;

(3) 将相关性分数归一化，表示权重;

(4) 依权重对 value 向量进行加权求和;

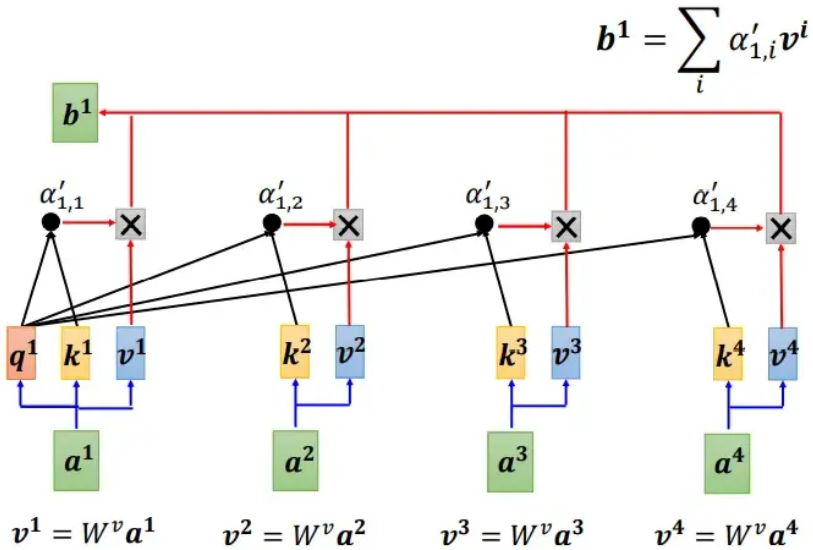


图 9: self attention 计算过程示例 (计算上图中的 b^1)

Self Attention 的计算过程的矩阵表示

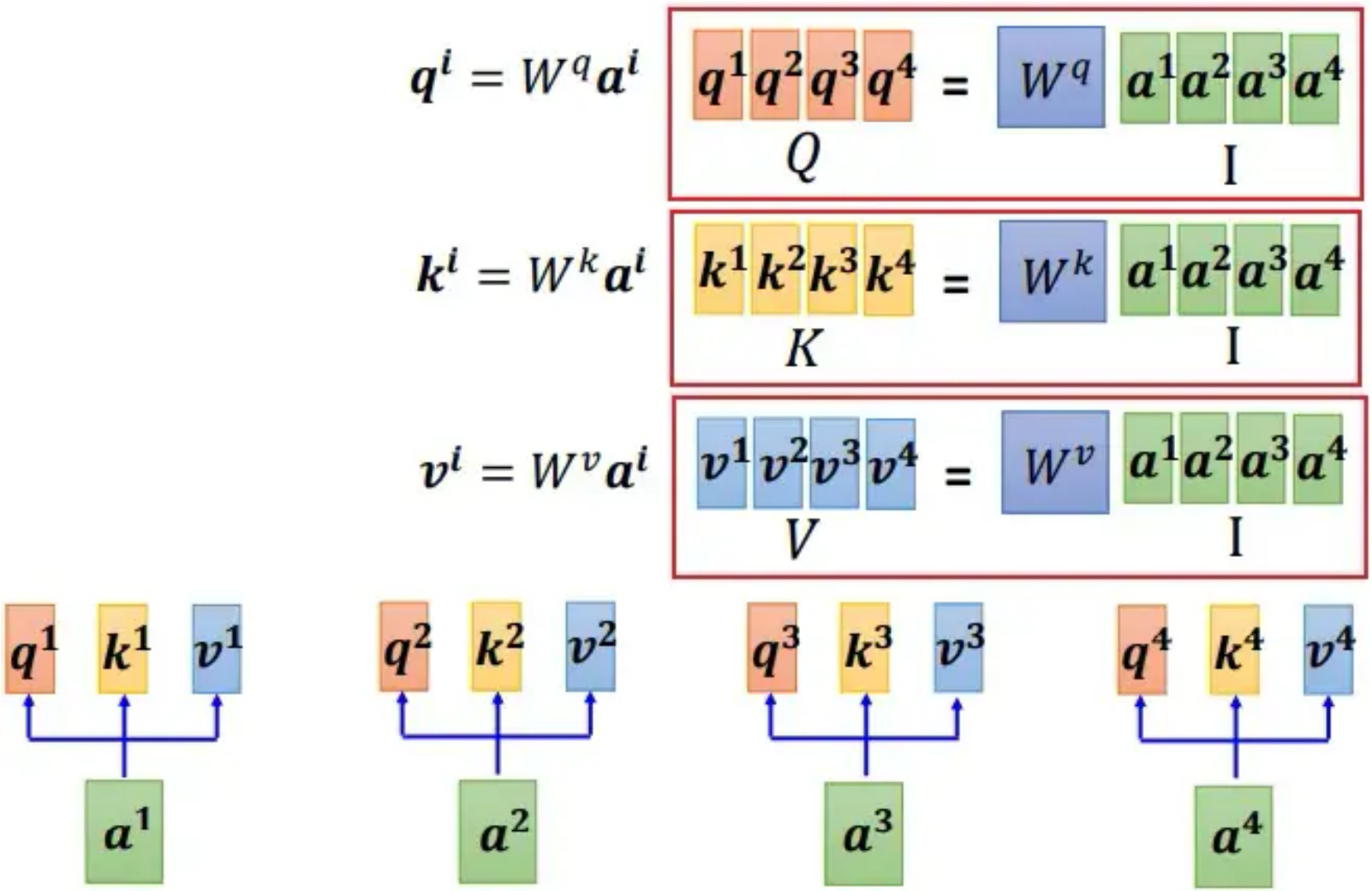


图 10: self attention 计算过程 - query、key、value 向量的计算

Self Attention 的计算过程的矩阵表示

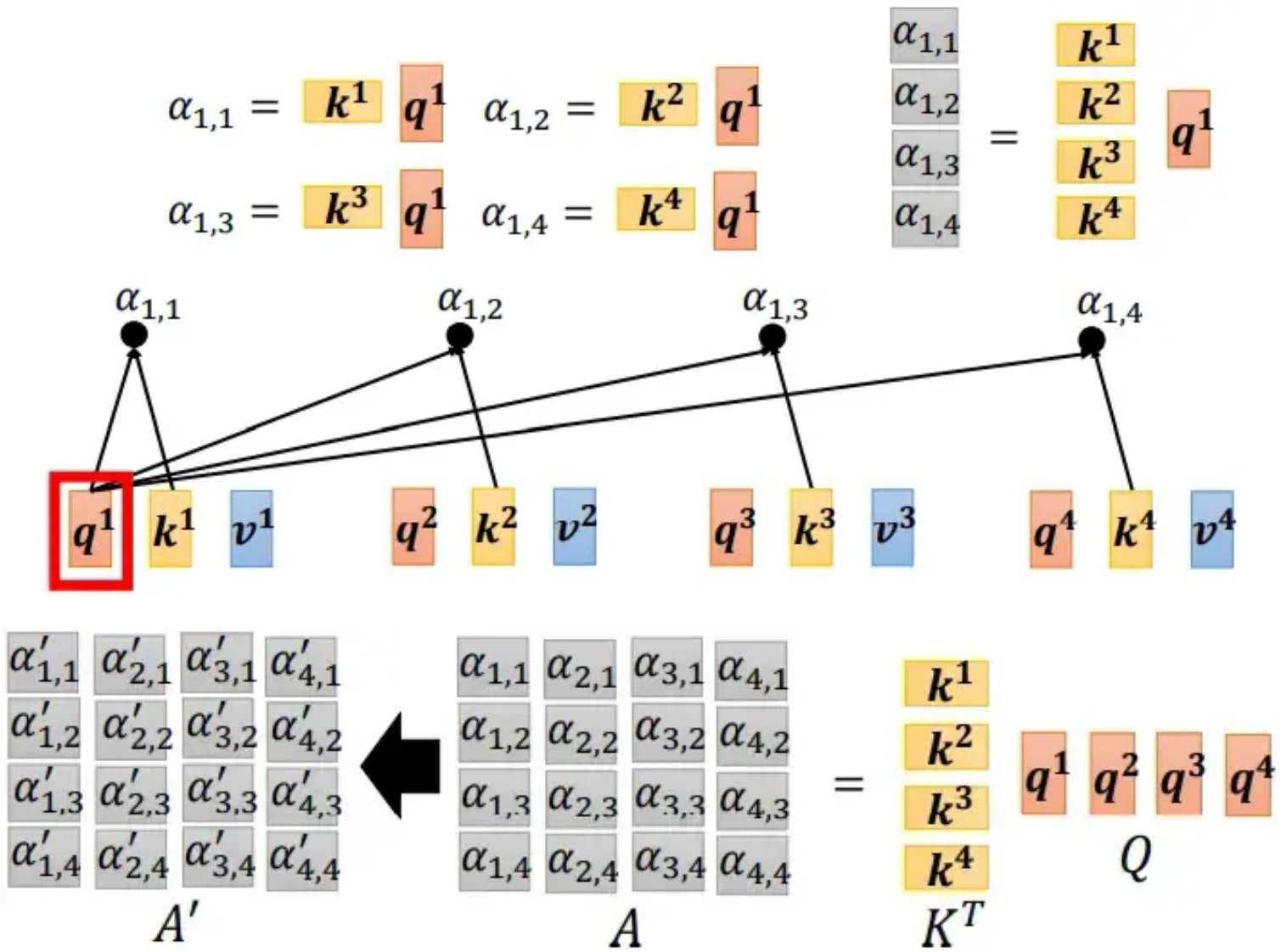


图 11: self attention 计算过程 - 相关性分数计算

Self Attention 的计算过程的矩阵表示

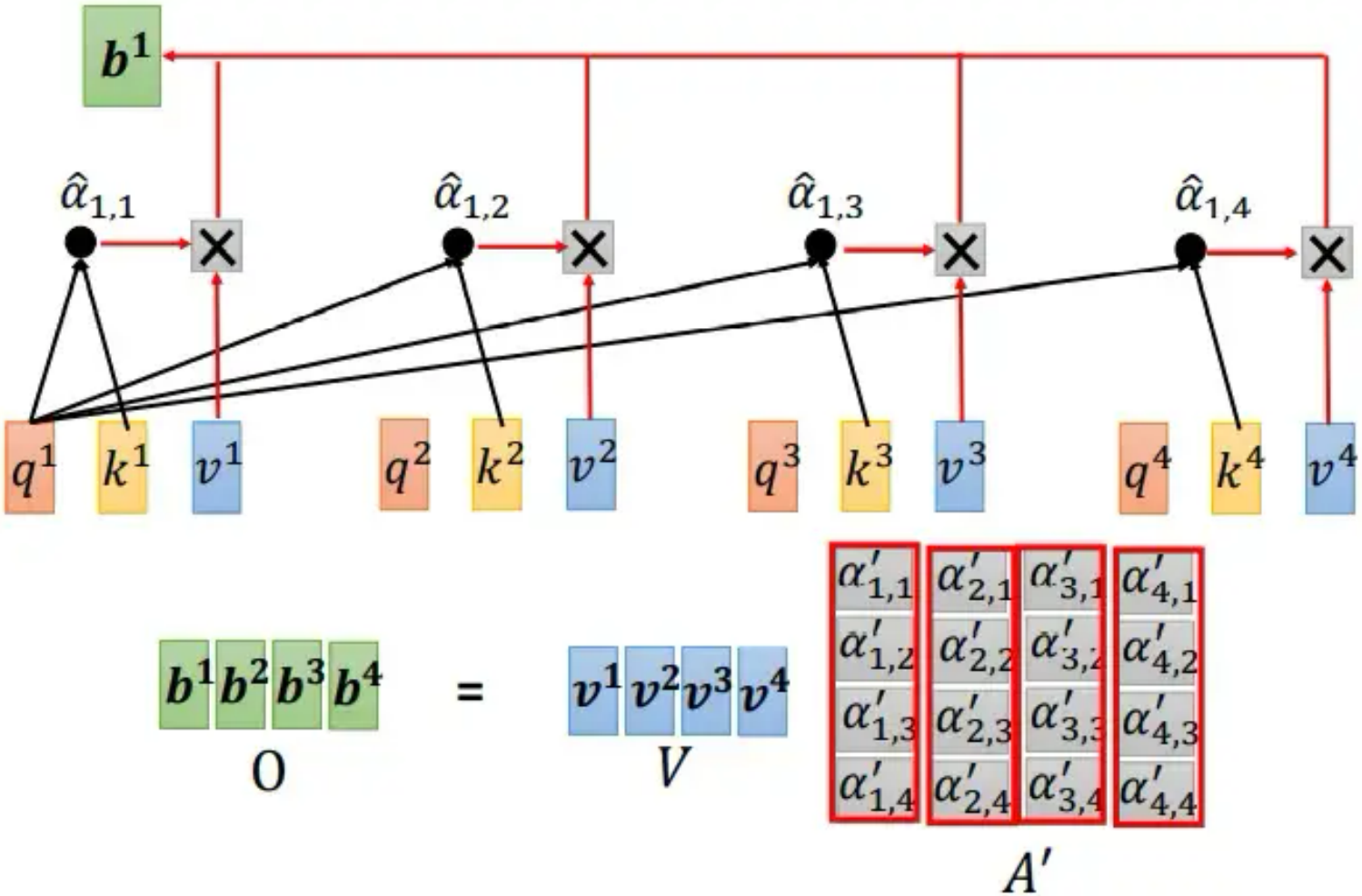


图 12: self attention 计算过程 - 输出向量计算

Multi-head Self Attention 是多个 Self Attention 的叠加，一个 Self Attention 建模输入文本中不同词语之间一种类型的关系，Multi-head Self Attention 使用多个 Self Attention 独立计算多个输出，并对多个输出进行拼接，同时建模多种类型的关系。

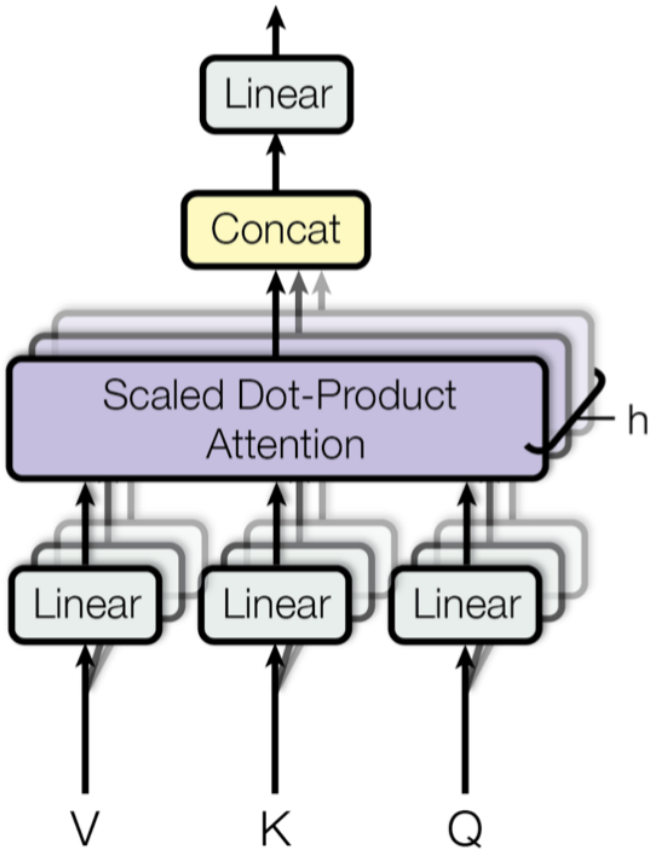


图 13: Multi-head Self Attention 示意图

计算过程： ① 计算出 $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$ 向量后，乘以 h 个权重矩阵，得到 h 个 Self Attention 计算所使用的 query、key 和 value 向量; ② 以并行 (parallel) 方式独立计算每个 Self Attention 的输出; ③ 拼接多个输出，并乘以权重矩阵 W^o 得到最后输出。

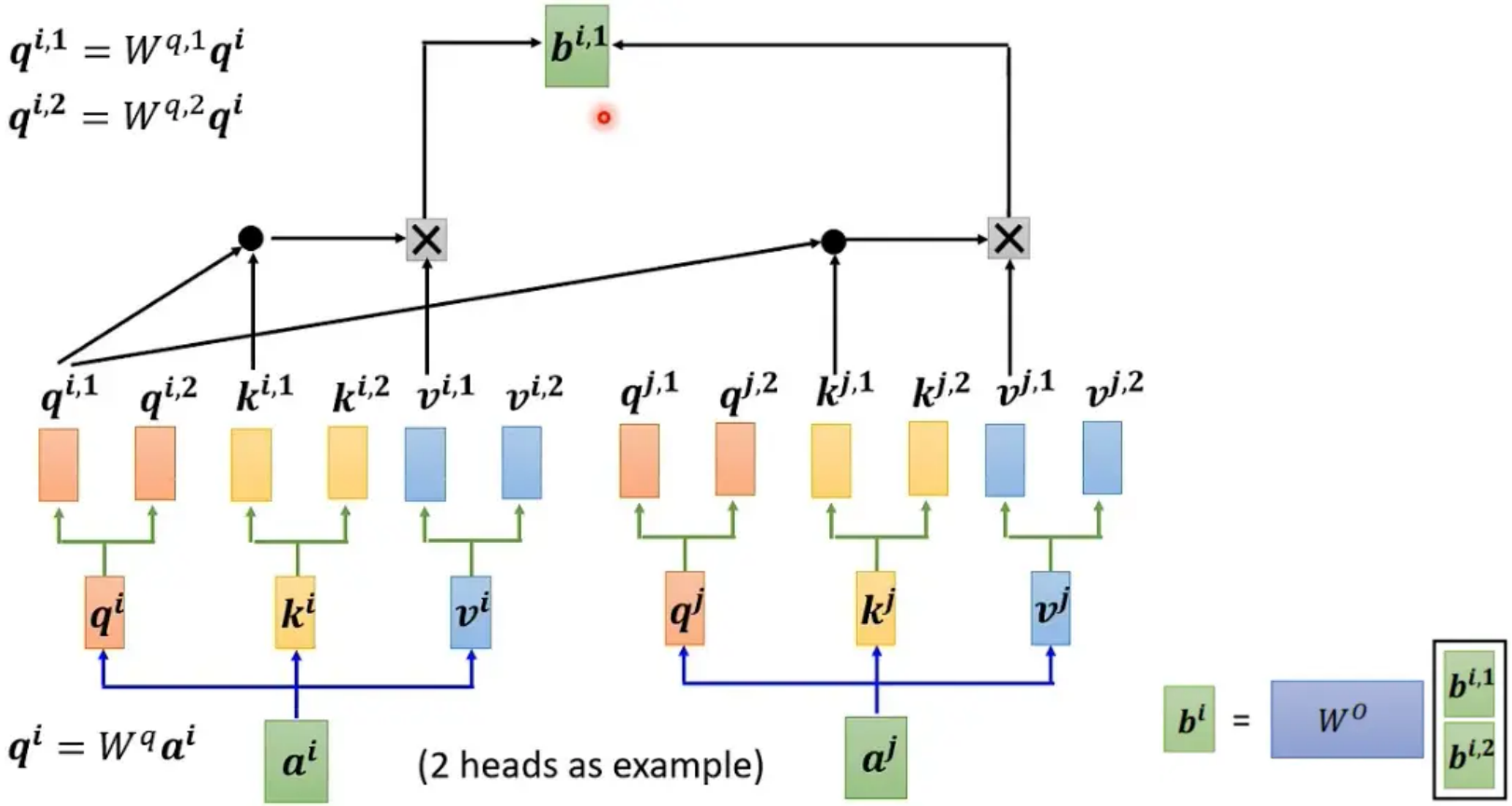


图 14: Multi-head Self Attention 计算示例 (2 heads)

目前的主流的大语言模型 (Large Language Models, LLMs)，如 BERT 和 GPT，都是基于 Google 在 2017 年提出的 Transformer 神经网络架构。Multi-head Self Attention 是 Transformer 架构的核心部件 (component)。

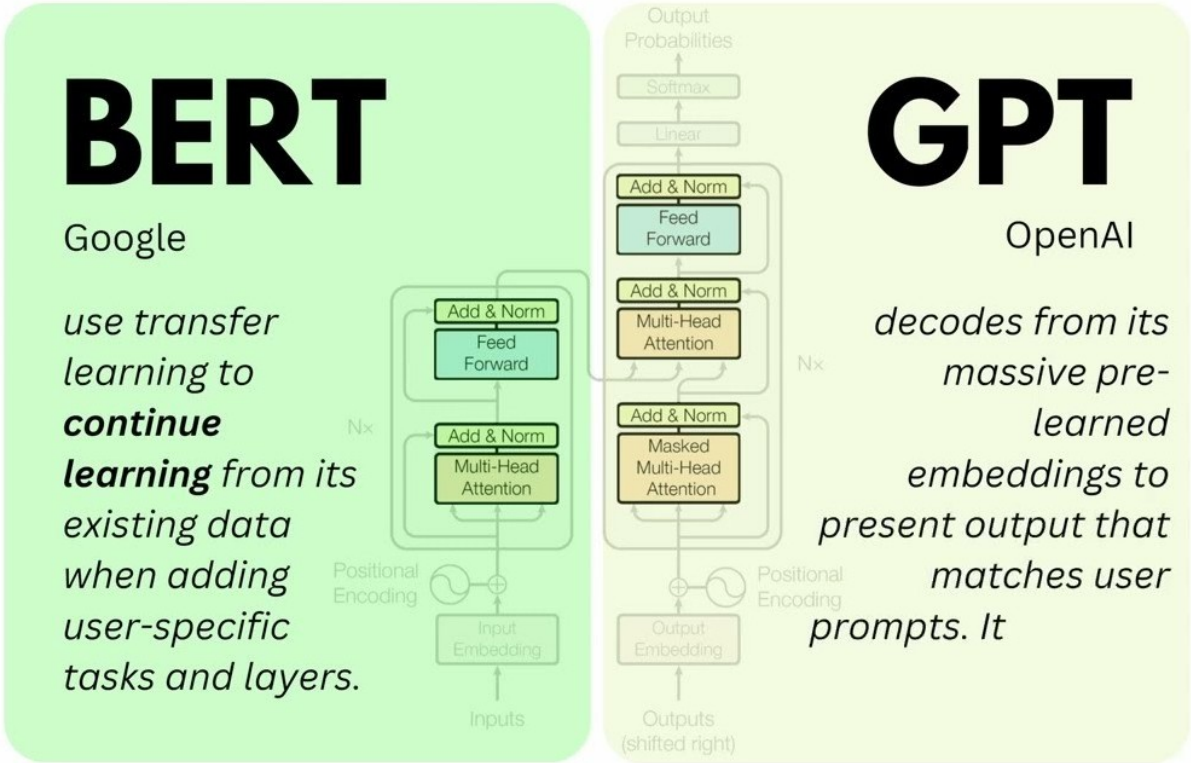


图 15: Transformer 与 BERT、GPT 的关系

BERT 使用 Transformer 编码器，双向建模，适合需要深度理解上下文的任务。GPT 使用 Transformer 解码器，单向生成，适合文本生成任务。

THE END