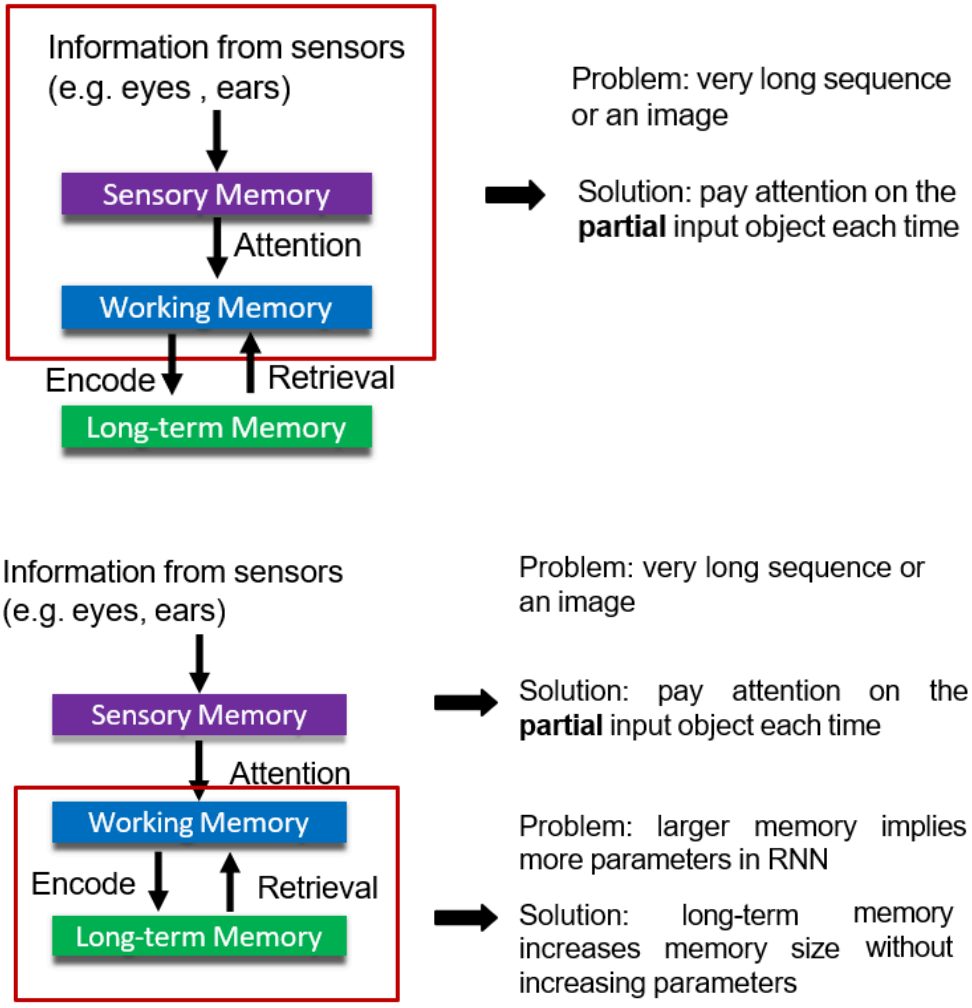
5.Attention & Transformer

Attention

Attention and Memory

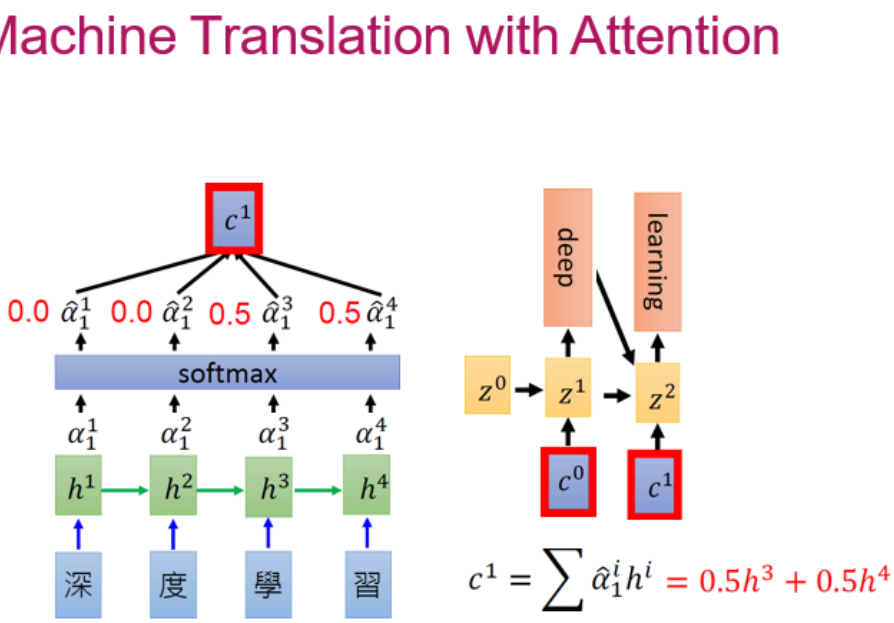


- 图中内容展示了注意力和记忆在深度学习中的三个关键组成部分：感觉记忆（Sensory Memory）、工作记忆（Working Memory）和长期记忆（Long-term Memory）。
- 感觉记忆（Sensory Memory）指的是对当前感官输入的直接感知和处理。在深度学习中，这通常与模型的输入层相关，负责接收原始数据，如图像、声音或文本，并进行初步处理。感觉记忆是短暂的，主要涉及对数据的即时处理和编码。
- 工作记忆（Working Memory）涉及到对信息的临时存储和操作，以便执行复杂的认知任务。在深度学习模型中，这与模型的中间层有关，这些层负责执行特征提取、变换和组合等操作。工作记忆对于模型进行决策和问题解决至关重要，因为它允许模型在处理数据时保持对先前信息的访问。

- 长期记忆（Long-term Memory）指的是对信息的长期存储，这些信息可以是显式的（如事实和事件）或隐式的（如技能和习惯）。在深度学习中，长期记忆通常与模型的权重和参数相关，这些权重和参数在训练过程中不断更新，从而允许模型存储和回忆学到的知识。

Attention on Sensory Information

Machine Translation with Attention

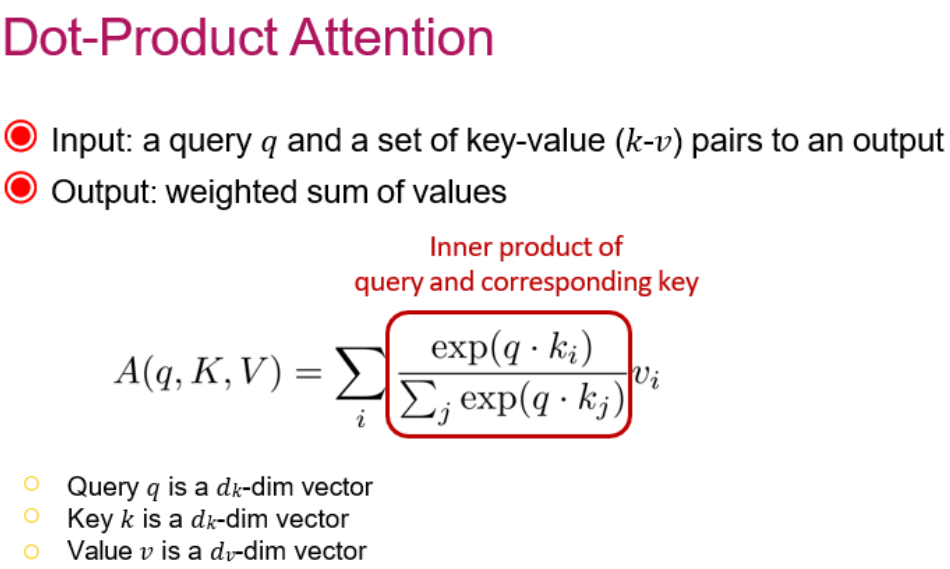


-
- 上述图片是带有注意力机制的机器翻译模型，过程如下：
- 1.编码器（Encoder）处理输入序列（例如，中文句子），生成一系列隐藏状态 h^1, h^2, h^3, h^4 。这些隐藏状态包含了输入序列的信息。
- 2.注意力权重（Attention Weights）通过计算每个隐藏状态 h_i对当前解码步骤的重要性，模型生成一组注意力权重 $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. 这些权重通过 softmax 函数计算得出，确保所有权重加起来等于 1
- 3.解码器（Decoder）使用上下文向量 c1 和先前生成的输出 z0 (其实就是query)来生成新的隐藏状态 z1,z2 和最终的输出。这些隐藏状态随后用于生成翻译的下一个词。
- 4.上下文向量（Context Vector）使用注意力权重，模型计算上下文向量 c0，这是一个加权和的隐藏状态，表示整个输入序列的信息。公式：$c^0 = \sum \alpha_i h^i$
- 总结：模型参数通过训练过程中的梯度下降方法进行学习，以最小化损失函数。注意力权重 α 通过训练学习得到，它们决定了解码器在生成每个词时对编码器隐藏状态的关注度。训练过程中，模型学习如何根据输入序列生成正确的输出序列。注意力机制使模型能够关注输入序列中最相关的部分，从而提高翻译质量。

- 个人理解核心步骤为：用 Query 和 Key （就是h）计算匹配分数（Score），比如通过点积或其他函数。将分数通过 Softmax 归一化，得到注意力权重 α 。用注意力权重对 Value 加权求和，生成上下文向量（如图中的 c ）

Attention Application

Dot-Product Attention



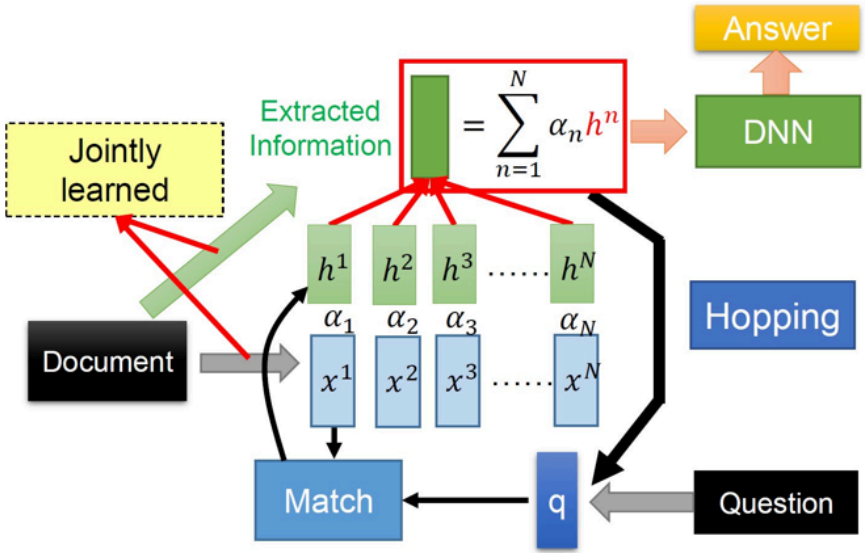
- 定义
  点积注意力机制通过计算查询（query）向量和键（key）向量之间的点积（内积）来确定注意力权重。这种机制在处理序列数据时特别有用，因为它可以捕捉序列中元素之间的关系。
- 工作原理
  流程同刚刚的翻译，不同的是这里计算匹配分数用的是点积。

Image(video) Captioning with Attention

- (1) **Image feature extraction**: Use CNN to extract local region features of the image.
- (2) **Attention weighting**: Dynamically calculate the attention weights of different image regions when generating each word.
- (3) **Context vector generation**: Weighted sum of regional features based on weights to obtain the context vector.
- (4) **Language generation**: Combining context vectors with the current state, generate descriptive words through a decoder.
  基本就是CNN+RNN，RNN带了个注意力机制。

Comprehension of reading

Reading Comprehension



Memory Network

- Muti-hop performance analysis

Story (16: basic induction)	Support	Hop 1	Hop 2	Hop 3
Brian is a frog.	yes	0.00	0.98	0.00
Lily is gray.		0.07	0.00	0.00
Brian is yellow.	yes	0.07	0.00	1.00
Julius is green.		0.06	0.00	0.00
Greg is a frog.	yes	0.76	0.02	0.00
What color is Greg? Answer: yellow		**Prediction: yellow**		

Conversational QA – CoQA, QuAC

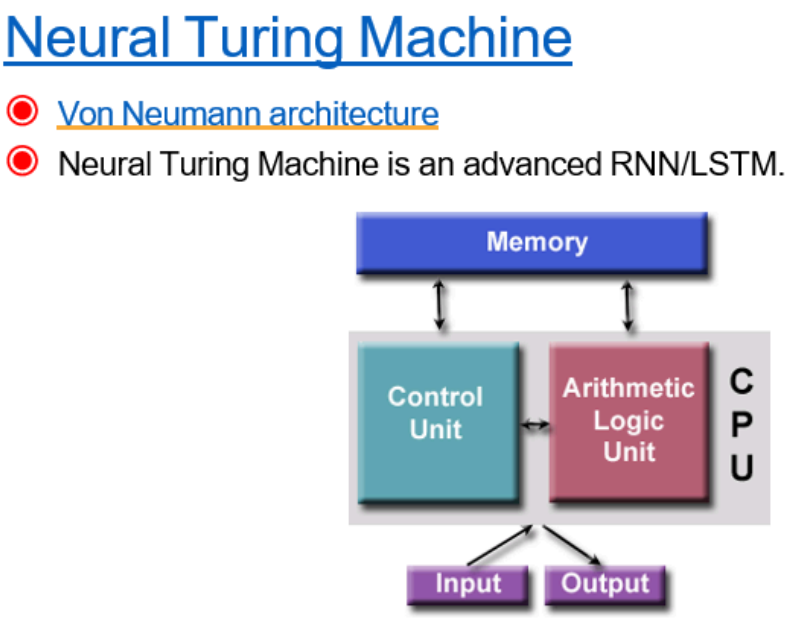
Jessica went to sit in her rocking chair. Today was her birthday and she was turning 80. Her granddaughter Annie was coming over in the afternoon and Jessica was very excited to see her. Her daughter Melanie and Melanie's husband Josh were coming as well. Jessica had . . .

- **The QA pairs are conversational**
  - Q1: Who had a birthday?
  - A1: Jessica
  - Q2: How old would she be?
  - A2: 80
  - Q3: Did she plan to have any visitors?
  - A3: Yes
  - Q4: How many?
  - A4: Three
  - Q5: Who?
  - A5: Annie, Melanie, and Josh

- 结合DNN，注意力机制

Attention on Memory

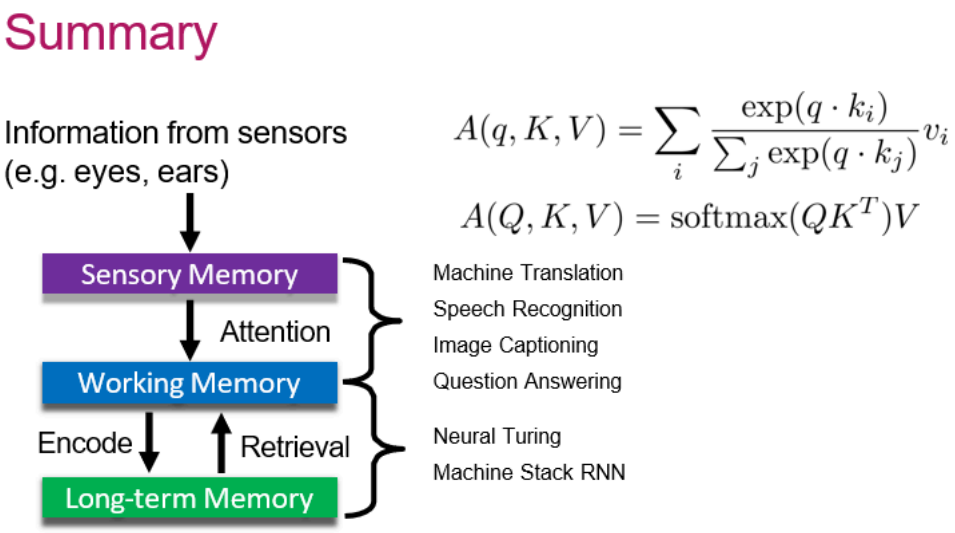
Neural Turing Machines



Zhang et al., "Structured Memory for Neural Turing Machines," arXiv, 2015.

- Neural Turing Machines essentially entail a form of neural network architecture equipped with an external memory bank, further enhancing its ability to process and manipulate data. 神经图灵机本质上需要一种配备外部内存库的神经网络架构，进一步增强了其处理和作数据的能力。
- By utilizing attention mechanisms, the neural network can navigate and interact with the external memory, allowing for read and write operations.通过利用注意力机制，神经网络可以导航并与外部存储器交互，从而允许进行读取和写入作。
- This architecture enables the machine to learn algorithms, manage sequential data, and access historical information, thus broadening the spectrum of tasks that can be undertaken by neural networks.这种架构使机器能够学习算法、管理顺序数据并访问历史信息，从而拓宽了神经网络可以承担的任务范围。
- The management and optimization of external memory in Neural Turing Machines may introduce complexities related to memory allocation and access efficiency. 神经图灵机中外部内存的管理和优化可能会引入与内存分配和访问效率相关的复杂性。

Summary



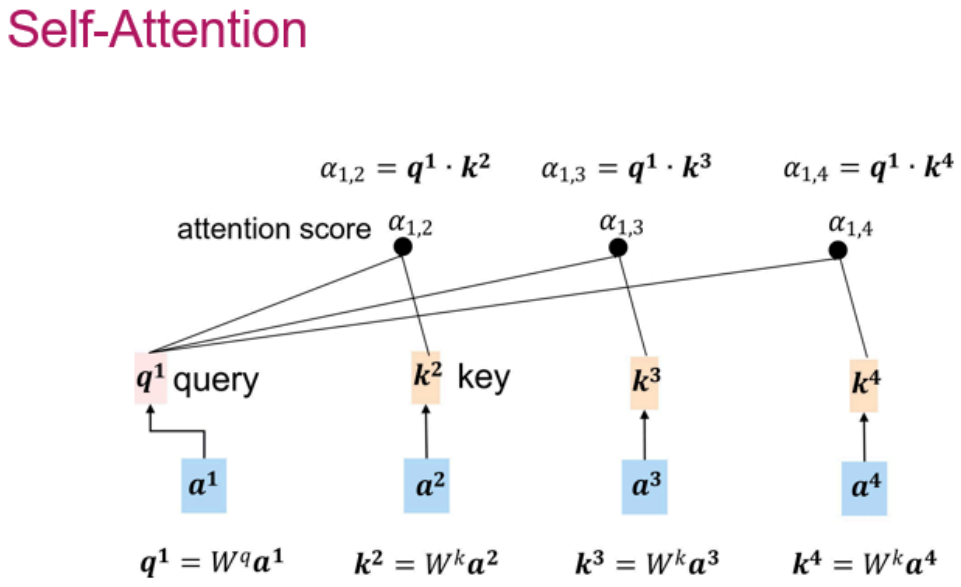
Transformer

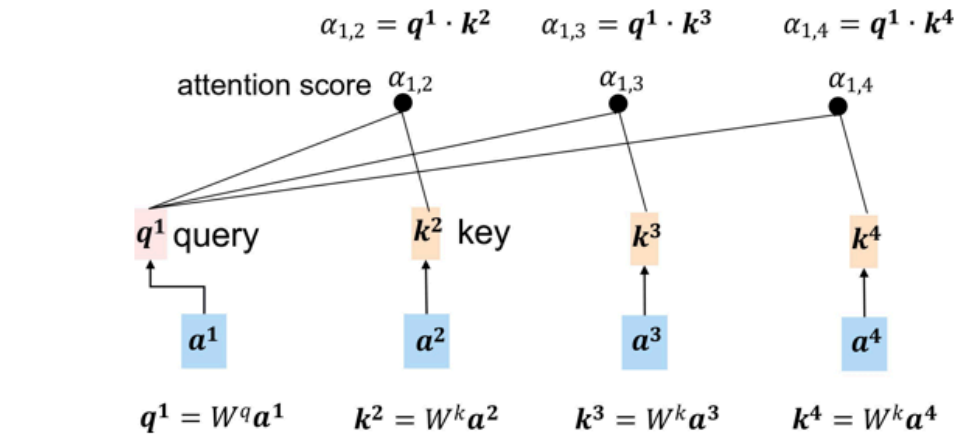
Quiz

- **QUESTION1**:What problems do RNN architectures without attention have for sequence-to-sequence problems?
  Answer:
  1.It is hard to learn the long-distance dependencies in RNNs.
  2.We cannot compute the future hidden states until the past RNN hidden states have already been computed.
  3.Lack of parallelizability.
- **QUESTION2**:In the original Transformer model, how are attention weights calculated for a given token?
  Answer:
  As the dot product between the token's embeddings and those of all other tokens, followed by a SoftMax operation.
- **QUESTION3**:What is the primary advantage of using the Transformer architecture over traditional RNNs for sequence tasks?
  Answer:
  Ability to capture long-range dependencies without the problem of vanishing gradients
- **QUESTION4**:What is one of the main disadvantages of transformer models for language modelling?
  Answer: Large memory requirements

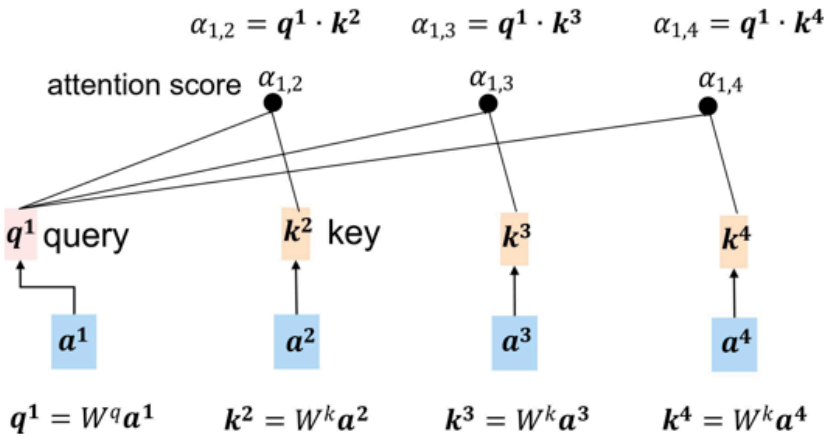
Self-Attention

- 原理：每个词通过 Query、Key 和 Value 计算对其他词的注意力权重。注意力分数通过点积计算：$Score = \frac{Q*K^T}{\sqrt{d_k}}$,其中 d_K是 Key 的维度(用于缩放防止过大)。Softmax 归一化后与 Value 加权求和，得到新的表示。

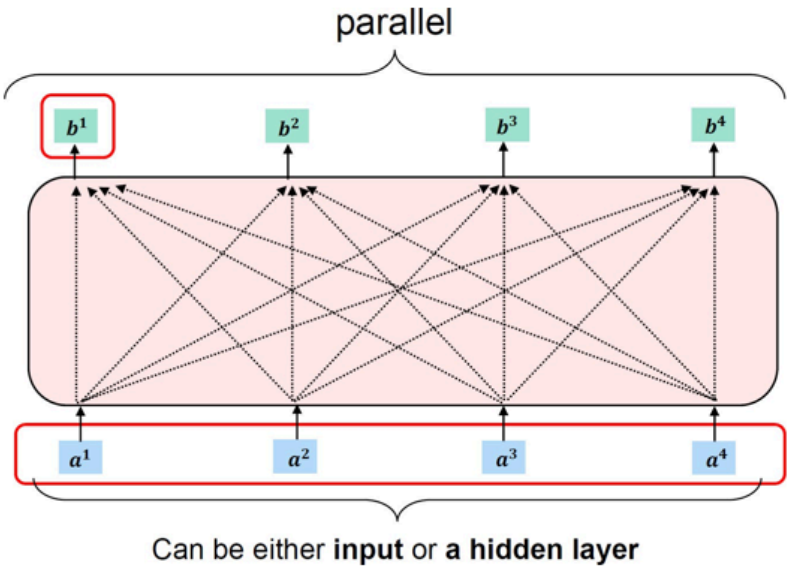




Self-Attention

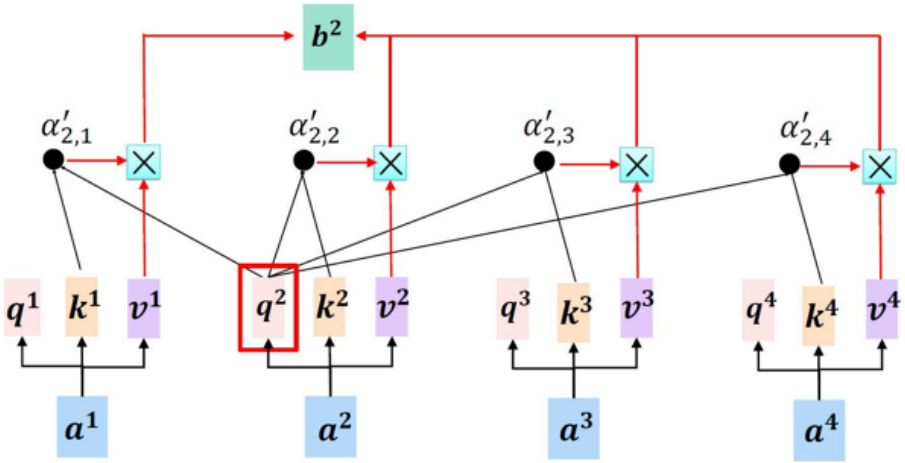


Self-Attention



Self-Attention

$$b^2 = \sum_i \alpha'_{2,i} v^i$$

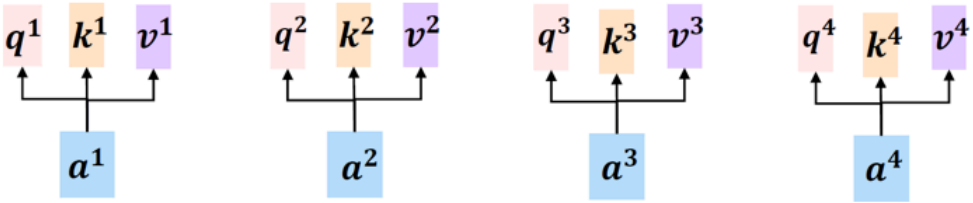


Self-Attention

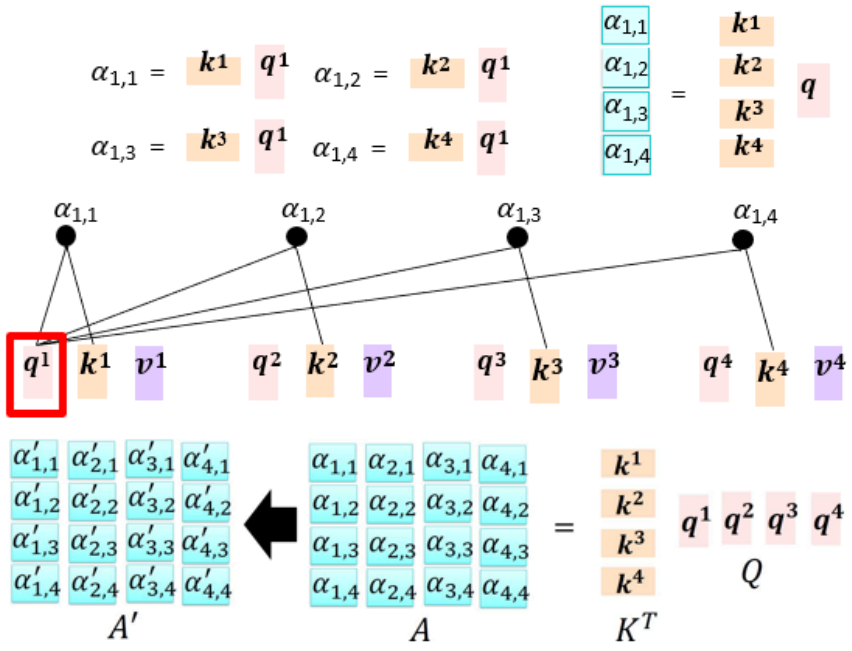
$$q^i = W^q a^i \quad q^1\, q^2\, q^3\, q^4 = W^q\, a^1\, a^2\, a^3\, a^4$$
$$Q \qquad\qquad I$$

$$k^i = W^k a^i \quad k^1\, k^2\, k^3\, k^4 = W^k\, a^1\, a^2\, a^3\, a^4$$
$$K \qquad\qquad I$$

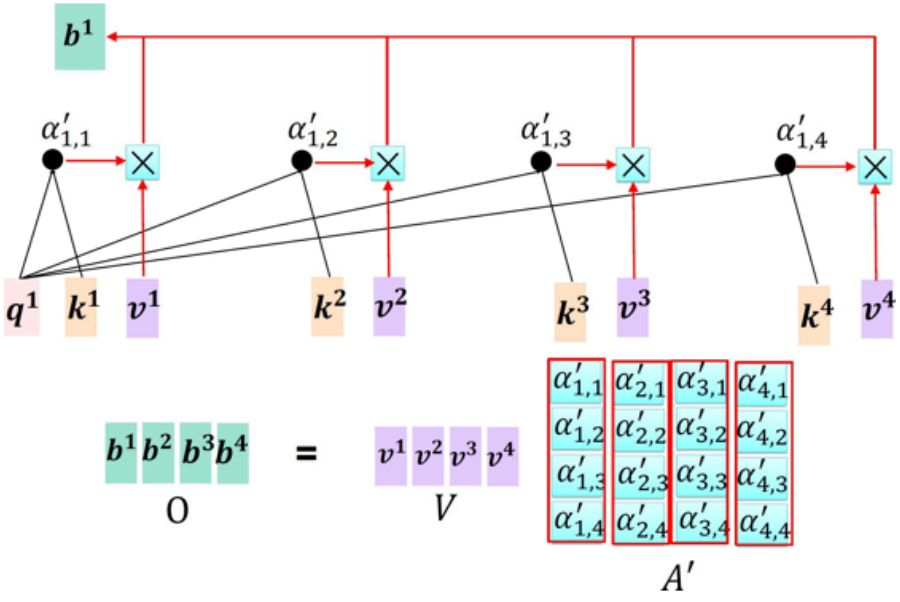
$$v^i = W^v a^i \quad v^1\, v^2\, v^3\, v^4 = W^v\, a^1\, a^2\, a^3\, a^4$$
$$V \qquad\qquad I$$

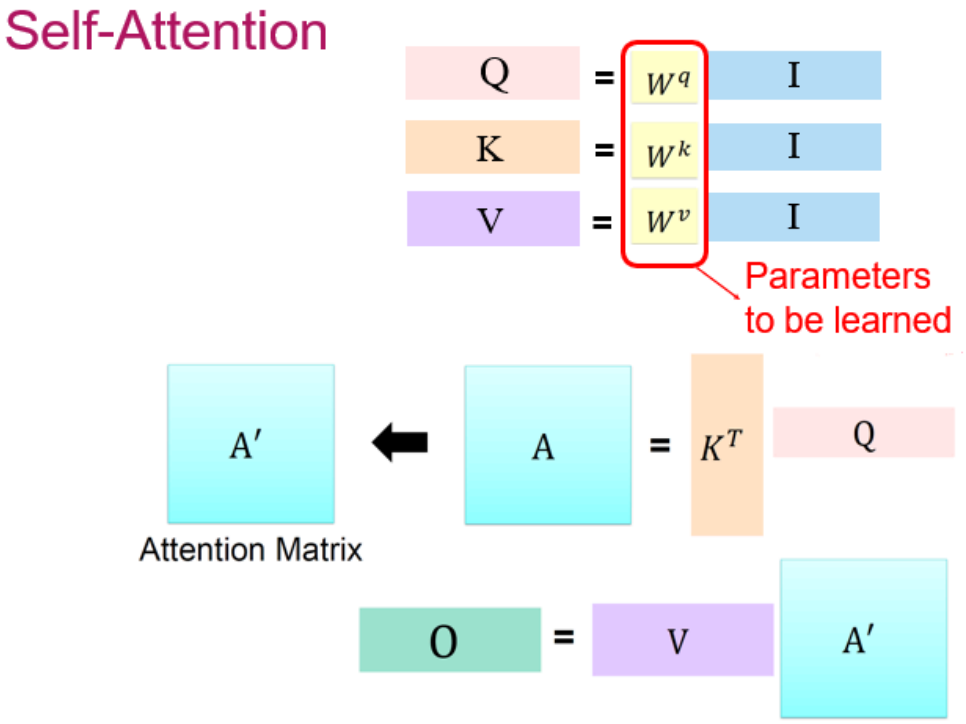


Self-Attention



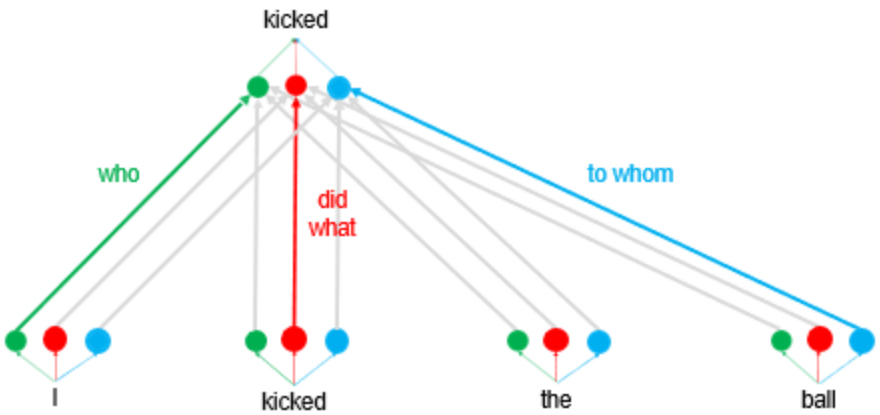
Self-Attention





Multi-head Attention

将输入分成多个子空间，分别计算注意力，然后拼接结果，增强模型捕捉不同关系的能力。（简单描述就是从整体注意片段转化为多方面的观察)

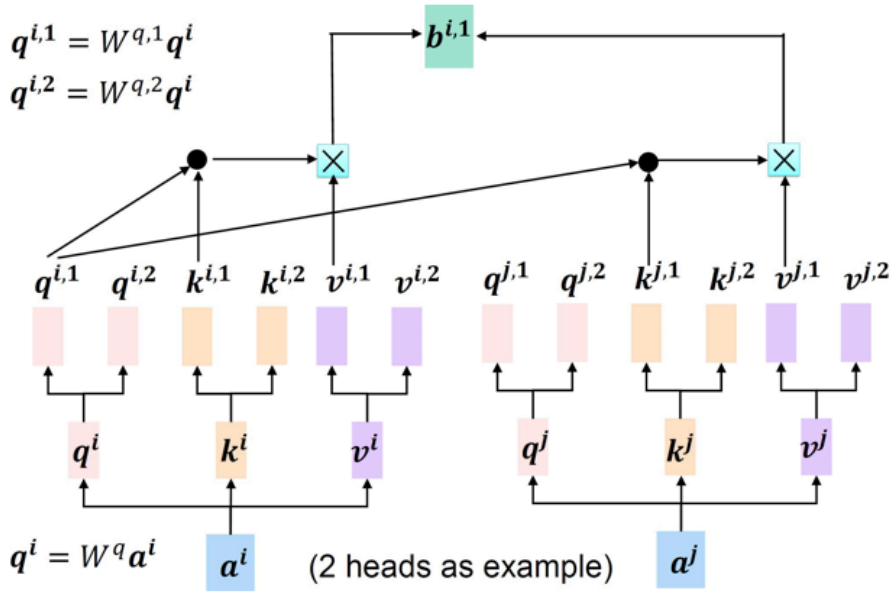


- 输入的 Query（Q）、Key（K）和 Value（V）分别通过不同的线性变换（权重矩阵 W_i^Q , W_i^K,W_i^V）投影到 h 个子空间。
  其中 h 是注意力头的数量（通常为 8 或 16）
  每个头的维度通常是

$$\frac{d_{model}}{h}$$

，确保总维度保持一致.

- 再就是独立计算每个头的注意力，$Head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.
- 最后将所有头的输出拼接起来(Concatenate)，再通过一个线性层（权重矩阵W^O）进行变换得到最终输出
- 以下是2头的示例（好怪）：



Transformer!

关键组件如下：

- **自注意力机制（Self-Attention）**：计算序列中每个元素与其他元素的关联权重，捕捉长距离依赖关系。
  Query vectors capture the global context
  Key vectors determine the importance of specific elements
  Value vectors store the intermediate feature representations

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

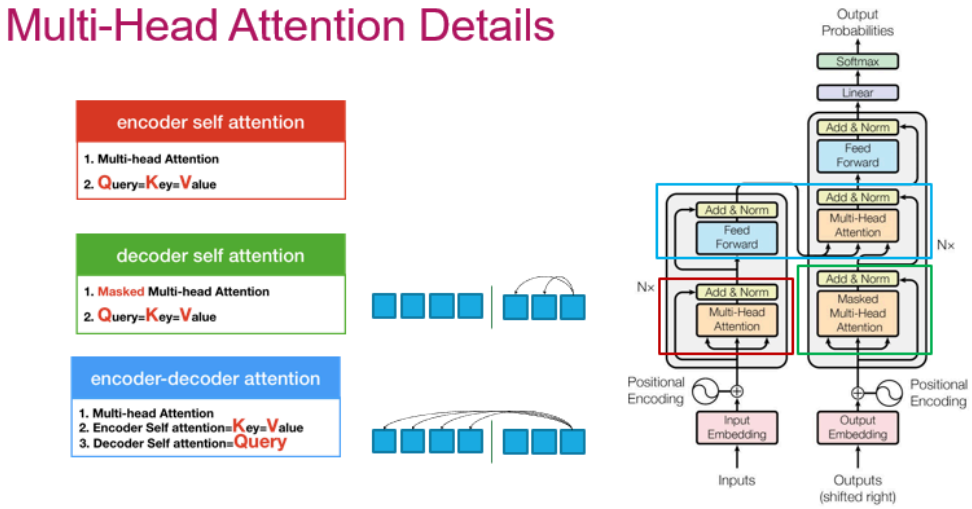
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

- **多头注意力（Multi-Head Attention）**：将注意力机制拆分为多个“头”，分别学习不同的特征表示，增强模型表达能力。
- **位置编码（Positional Encoding）**：通过添加位置信息，弥补模型对输入序列顺序的感知（因为 Transformer 本身不包含循环或卷积结构）。用于解决： temporal information is missing

- **残差连接（Residual Connection）** 和 **层归一化（Layer Normalization）**：缓解梯度消失问题，加速训练。
- **缩放点积注意力（Scaled Dot-Product Attention）**

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

- **编码器模块（Encoder Block）**：
  组成：多头注意力，2 层前馈神经网络 (NN)（带 ReLU）
  特性：残差连接，层归一化（LayerNorm）
  通过 LayerNorm(x + sublayer(x)) 使输入每层和每个训练点均值为 0、方差为 1
- 如图，是**Transformer的各个模块**



- 详解Link点我

优点：

- **长距离依赖建模**，Self-Attention 能直接捕捉序列中任意两个元素的关系，避免了LSTM和RNN梯度消失的问题
- 支持并行化计算

缺点：

- 计算资源需求大
- **编码器输入**缺少时间信息------->解决方案就是引入位置编码(positional encoding)

Positional Encoding

- 就记着偶数列用sin,奇数列用cos

Sinusoidal Positional Encoding

- Let us look at an example, and the base is 10000 here:

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

- Suppose $d_{model} = 4$.
- Calculation For pos = 0:
  - For the 0th dimension (even, $i = 0$):
    $PE(0,0) = \sin\left(\frac{0}{10000^{0/4}}\right) = \sin(0) = 0$
  - For the 1st dimension (odd, $i = 0$):
    $PE(0,1) = \cos\left(\frac{0}{10000^{0/4}}\right) = \cos(0) = 1$
  - For the 2nd dimension (even, $i = 1$):
    $PE(0,2) = \sin\left(\frac{0}{10000^{2/4}}\right) = \sin(0) = 0$
  - For the 3rd dimension (odd, $i = 1$):
    $PE(0,3) = \cos\left(\frac{0}{10000^{2/4}}\right) = \cos(0) = 1$
- Therefore, positional encoding is [0, 1, 0, 1]

Sinusoidal Positional Encoding

- Let us look at an example, and the base is 10000 here:

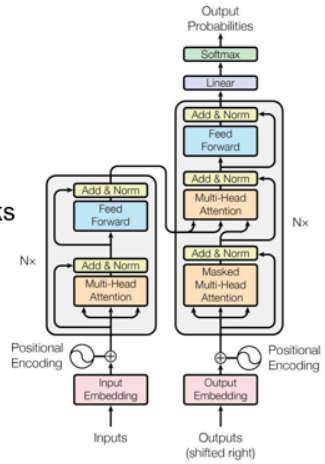
$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{\text{model}}})$$

- Suppose $d_{model} = 4$.
- Calculation For pos = 1:
  - For the 0th dimension (even, $i = 0$):
    $PE(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1)$
  - For the 1st dimension (odd, $i = 0$):
    $PE(1,1) = \cos\left(\frac{1}{10000^{0/4}}\right) = \cos(1)$
  - For the 2nd dimension (even, $i = 1$):
    $PE(1,2) = \sin\left(\frac{1}{10000^{2/4}}\right) = \sin\left(\frac{1}{100}\right)$
  - For the 3rd dimension (odd, $i = 1$):
    $PE(1,3) = \cos\left(\frac{1}{10000^{2/4}}\right) = \cos\left(\frac{1}{100}\right)$
- Therefore, the positional encoding is [0.84147,0.54030,0.00999,0.99995]

Summary

Summary

- **Non-recurrence** model is easy to parallelize
- **Multi-head attention** captures different aspects by interacting between words
- **Positional encoding** captures location information
- Each transformer block can be applied to diverse tasks



QUESTION:

- Which of the following properties will a good position encoding ideally have
- ***Answer:***

  1.Unique for all positions

  2.Relative distances are independent of absolute sequence position

  3.Well-defined for arbitrary sequence lengths
-