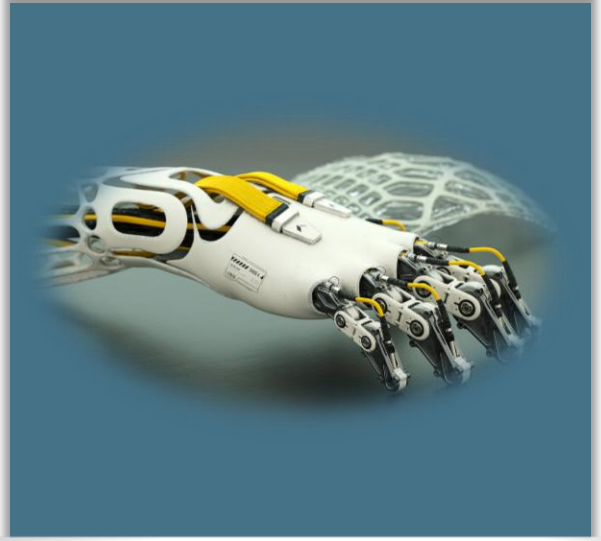
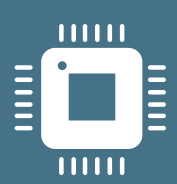
时序神经网络

王闻博
Email: wenbo_wang@kust.edu.cn
昆明理工大学 机电工程学院

2026年5月15日



时序神经网络




1. 循环神经网络模型
2. Sequence-to-Sequence模型
3. 从注意力机制到自注意力
4. 传统Transformer模型

问题的提出

- **序列数据的预测/回归问题**
  - 给定输入的实数向量序列：$\{x_1, x_2, \cdots x_T\}$；
  - 在t = 1,2, ⋯ , T个位置上，对向量x_i进行预测，给出概率分布p_i，整体产生输出的概率分布向量序列：$\{p_1, p_2, \cdots p_T\}$。
- **基于MLP的朴素处理办法**
  - 前提假设：序列数据的长度固定；
  - 序列数据情况下，MLP可能的输入输出定义：
    - 将输入的实数向量拼接为一整个输入向量，作为MLP的输入；
    - 将输出的概率向量拼接，作为MLP的输出。
- **任务数据特性和MLP固有结构的矛盾**
  - 序列数据长度可变，而MLP的输入层宽度固定；
  - 局部特征的表示和学习：序列数据通常在不同位置上有相似的局部特征，MLP的结构并不能对这些局部相似特征进行特殊化的处理。

朴素循环神经网络（Recurrent Neural Networks）的定义

- **网络的输入和输出**
  - 给定输入的实数向量序列：$\{x_1, x_2, \cdots x_T\}$；
  - 在$t = 1, \cdots,$ T个位置上，给出输出（如，概率预测）p_i，产生输出向量序列：$\{p_1, p_2, \cdots p_T\}$。
- **网络结构的复用：在每一个位置（时间步）t上，重复使用同一个网络结构**
  - 不同时间步的权重参数是共享的（从而减少了参数量）。
- **引入隐藏层和隐藏状态（隐变量）**
  - 隐藏层（中间层）以x_t 和隐变量h_{t-1}为输入，以h_t为输出，在每一个时间步更新：

$$\mathbf{h}_t = \tanh(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

将历史状态/信息压缩到隐变量，从而实现对时间依赖的学习。

- **输出层仅依赖当前的隐变量h_t**

$$\mathbf{p}_t = \sigma(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$

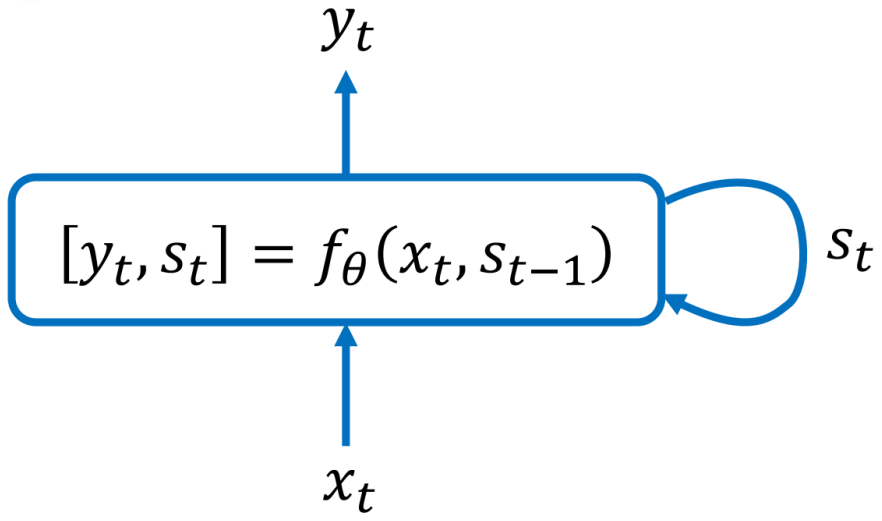
其中$\sigma(\cdot)$可以是softmax等激活函数，视任务所需输出（如分类任务）而定。

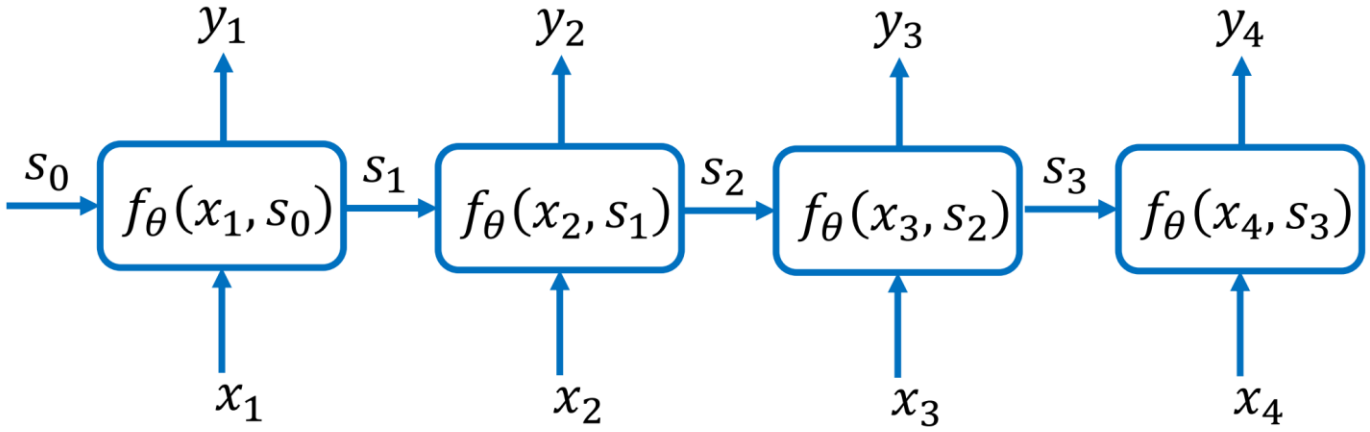
循环神经网络（RNN）图解

- **引入所谓的“隐状态” (Latent State)**
  - 输入和输出：$\mathbf{x}_t$ 和 $\mathbf{y}_t$；
  - 隐状态：$\mathbf{s}_t$；
  - 待学习参数：$\boldsymbol{\theta}$;
- **输入输出表达式和状态更新公式**
  - $\mathbf{y}_t = g_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{s}_t) = g(\mathbf{W}_{sy}\mathbf{s}_t + b_y)$，其中$b_y$是偏置量。
  - $\mathbf{s}_t = h_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{s}_t) = h(\mathbf{W}_{xs}\mathbf{x}_t + \mathbf{W}_{ss}\mathbf{s}_{t-1} + b_s)$，其中$b_s$是偏置量。
- **沿输入-输入序列（时间步）展开左上图循环**

网络特点：

- （1）参数$\boldsymbol{\theta}$（即$\mathbf{W}_{xs}$、$\mathbf{W}_{ss}$ 和$\mathbf{W}_{sy}$）为共享权重，即沿时间轴的不同时间节点（Timestamp）的输出和隐状态计算共享同一线性变换矩阵。
- （2）循环连接：当前时刻的输出不仅依赖当前输入，还依赖前一时刻的隐状态。





循环神经网络（RNN）的模型特点

- **RNN符合一般动态系统模型结构**

$$\begin{cases} s(t) = F(s(t-1), x(t)) & \longleftarrow \text{系统状态方程} \\ y(t) = G(s(t)) & \longleftarrow \text{系统输出方程} \end{cases}$$

  - 系统当前时刻的状态 $s(t)$（即 RNN 中的隐藏状态 h_t）：由上一时刻的状态 $s(t-1)$ 和当前输入 $x(t)$ 共同决定。这体现了 RNN 的“记忆”能力——历史信息通过前一状态传递到当前。
  - 系统当前的输出 $y(t)$（即 RNN 的输出 p_t）：仅由当前状态 $s(t)$ 决定，与之前的输入无直接依赖——历史信息已被压缩在状态中。
- **RNN是一种自回归（Autoregressive）模型**
  - 序列中每一个位置上的输出只依赖之前位置的信息。
- **当前位置的状态是局部特征与全局特征的共同表示**
  - **局部特征**：当前时刻的输入 x_t 提供了瞬时的、局部的信息；
  - **全局特征**：通过隐藏状态 h_{t-1} 累积的历史信息，包含了从序列起始到上一时刻的所有上下文。

循环神经网络的训练

- **循环神经网络 $\hat{y} = f(\mathrm{x}; \mathrm{W})$ 的训练样本集形态：等长同步序列任务**
  - 输入序列：$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots \boldsymbol{x}_T\}$；
  - 输出序列：$\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots \boldsymbol{y}_T\}$；
- **损失函数的设计**
  - （a）回归任务；根据时间步误差，典型如单步平方误差：

$$e_t = (\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t)^2,$$

通过累计时间步误差的形式构成损失函数$\boldsymbol{L}$；

  - （b）自回归任务：根据具体任务使用其他损失函数，如输出序列的条件概率：

$$P(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_T \mid \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T) = \prod_{t=1}^{T} P(\mathbf{y}_t \mid \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t)$$

构造（最小化交叉熵）

$$L = \sum_{t=1}^{T} L_t = -\sum_{t=1}^{T} \log P(\mathbf{y}_t \mid \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t)$$

循环神经网络的梯度反向传播

- **RNN的反向传播**
  - 特点：反向传播的误差定义与MLP相同（损失对中间层净输入的梯度），但要注意损失对当前隐状态（t时刻）的偏导同后续时间步（t+1时刻）的隐状态相关：

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial y_t}\frac{\partial y_t}{\partial s_t} + \frac{\partial L}{\partial s_{t+1}}\frac{\partial s_{t+1}}{\partial s_t}$$

  - 权重梯度的计算需累加各时间步的梯度，如对$\mathbf{W}_{ss}$（隐状态到隐状态权重）而言，有

$$\frac{\partial L}{\partial \mathbf{W}_{ss}} = \sum_{t=1}^{T}\frac{\partial L}{\partial \mathbf{z}_t^s}\frac{\partial \mathbf{z}_t^s}{\partial \mathbf{W}_{ss}} = \sum_{t=1}^{T}\boldsymbol{\delta}_t^s\,\mathbf{s}_{t-1}^{\top}$$

  - 类似地，有输入到隐状态权重$\mathbf{W}_{xs}$的梯度：

$$\frac{\partial L}{\partial W_{xs}} = \sum_{t=1}^{T}\frac{\partial L}{\partial z_t^s}\frac{\partial z_t^s}{\partial W_{xs}} = \sum_{t=1}^{T}\delta_t^s\,x_t^T$$

误差项定义：

输出层：
令 **净输入** $\mathbf{z}_t^y = \mathbf{W}_{sy}\mathbf{s}_t + \mathbf{b}_y$，则 $\mathbf{y}_t = g(\mathbf{z}_t^y)$

输出层误差项：

$$\boldsymbol{\delta}_t^y = \frac{\partial L}{\partial \mathbf{z}_t^y}$$

隐藏层：
令 **净输入** $\mathbf{z}_t^s = \mathbf{W}_{xs}\mathbf{x}_t + \mathbf{W}_{ss}\mathbf{s}_{t-1} + \mathbf{b}_s$，
则 $\boldsymbol{s}_t = h_\theta(\mathbf{z}_t^s)$

隐藏层误差项：

$$\boldsymbol{\delta}_t^s = \frac{\partial L}{\partial \mathbf{z}_t^s}$$

循环神经网络的训练中的问题

- **梯度消失和梯度爆炸**
  - 反向传播在RNN结构中需要沿时间线迭代，见

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial y_t}\frac{\partial y_t}{\partial s_t} + \frac{\partial L}{\partial s_{t+1}}\frac{\partial s_{t+1}}{\partial s_t}$$

  - 由于上式中的方框部分的时间步的累乘效应，容易出现梯度消失或梯度爆炸问题——梯度消失情况下，梯度更新将非常缓慢；
    - 上式红框中，偏导计算需要对激活函数求导：

$$\frac{\partial s_{t+1}}{\partial s_t} = W_{ss}^T \odot h'(W_{ss} \cdot s_t + W_{xs} \cdot x_{t+1} + b_s)$$

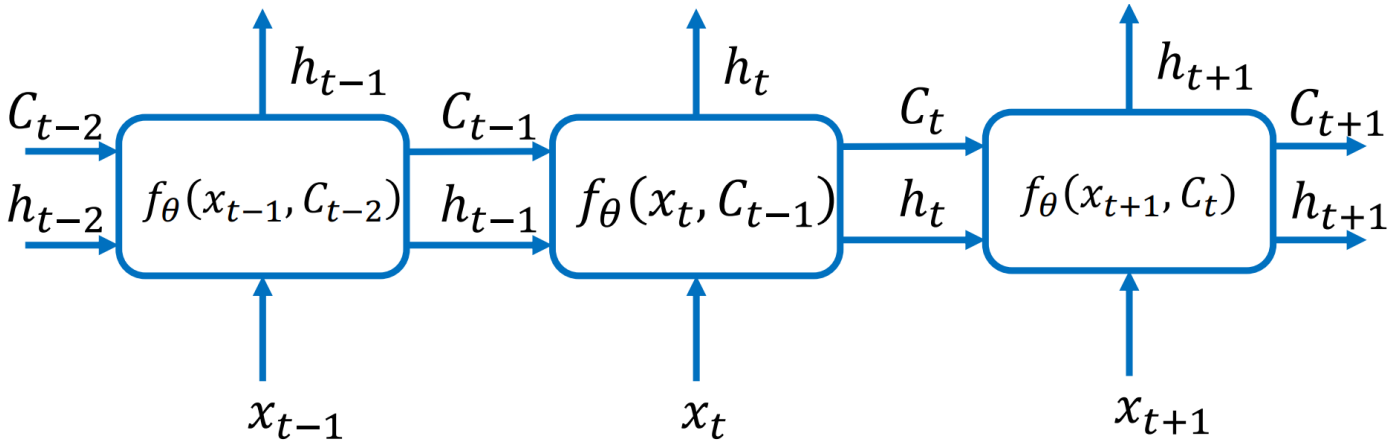
    激活函数导数$h'(\cdot)$的绝对值往往在（0,1）间，累乘将导致梯度消失。
  - 对于长时间序列的依赖性建模效果不佳。

循环神经网络的训练问题

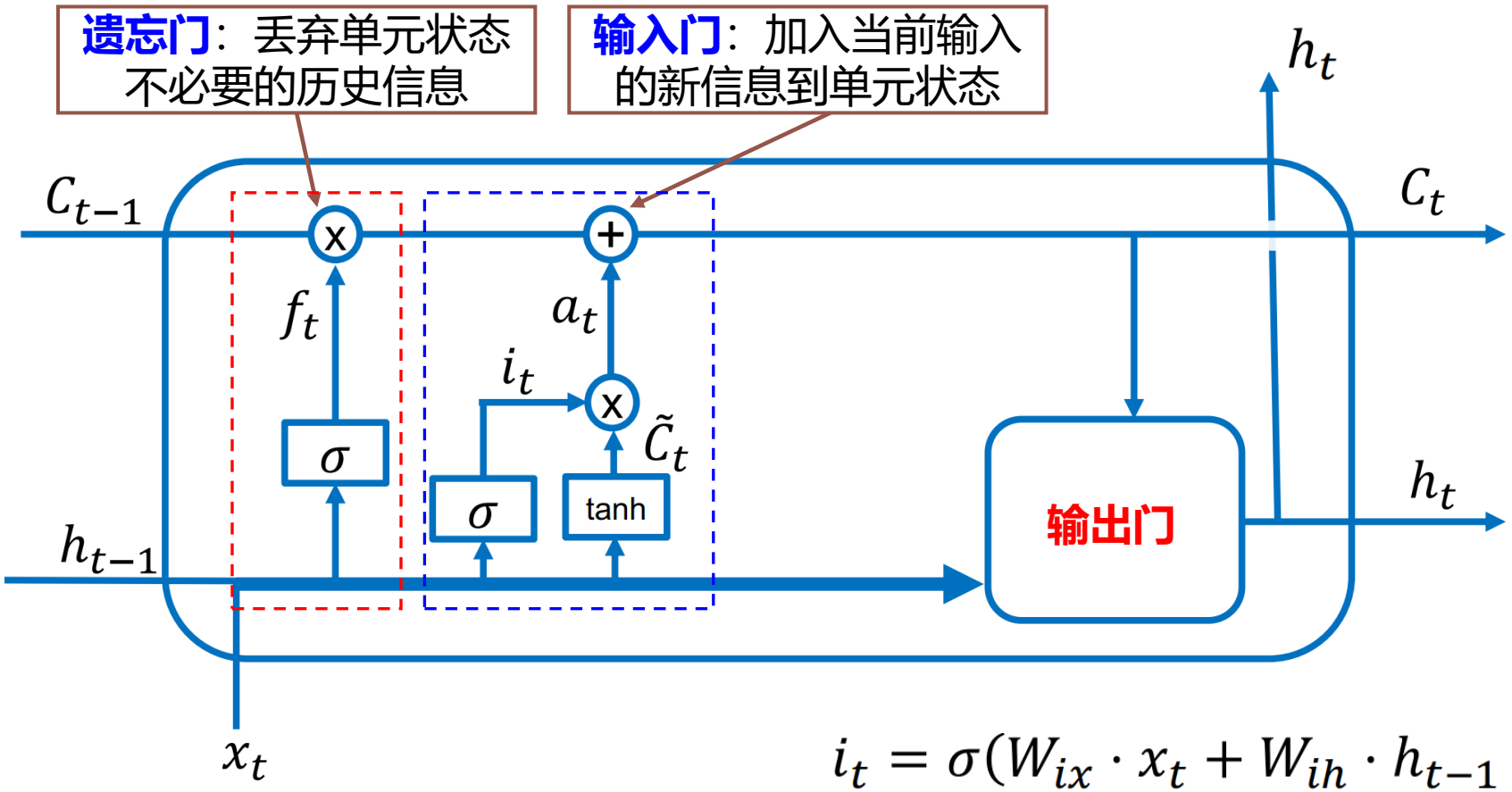
- **历史上的解决方案**
  - 使用跳跃连接；
  - 截断随时间反向传播（Truncated BPTT）：仅反向传播固定k步，而不是整个序列
  - 梯度裁剪（gradient clipping）——按范数截断梯度；
  - 使用批归一化方案；
  - 使用门控机制（控制长期历史信息对当前隐状态的贡献），缓解梯度消失问题。
  - 由此引出著名的“长短时记忆”（Long-Short Time Memory，LSTM）网络等方案。

LSTM网络结构概览

- **LSTM网络的关键结构**
  - 含有两个状态$s_t = [C_t, h_t]$;
  - C_t：单元状态（长期状态）
    - 存储从一个时间步传递到下一个时间步的信息;
    - 由单元状态引入类似跳过连接的方式，减少梯度消失问题。
  - h_t：隐状态（短期状态）
    - 通常是LSTM的输出;
    - 为得到网络最终输出y_t，通常需要其他网络层对h_t的进一步处理以产生所需的输出。



LSTM网络结构详解：单元状态



门控：Hardamard积，将门向量与待控制的特征向量相乘；
特点：门向量与特征向量维度相同，元素相乘允许每个特征维度独立地决定是否通过或衰减。

$$f_t = \sigma(W_{fx} \cdot x_t + W_{fh} \cdot h_{t-1} + b_f)$$

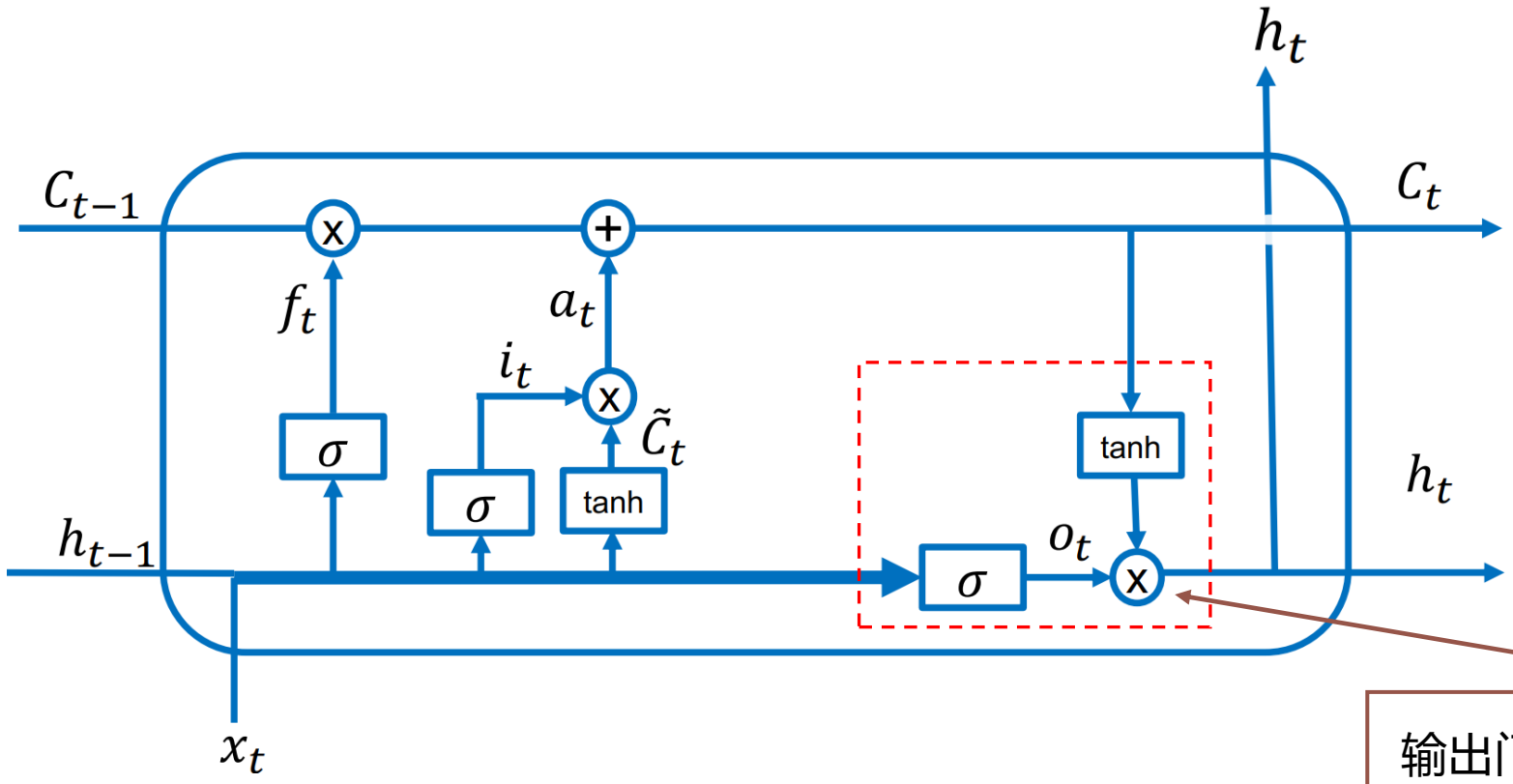
注意：遗忘门与C、h和x都相关

$$i_t = \sigma(W_{ix} \cdot x_t + W_{ih} \cdot h_{t-1} + b_i)$$ 决定哪些信息更新进C

$$\tilde{C}_t = \tanh(W_{cx} \cdot x_t + W_{ch} \cdot h_{t-1} + b_c)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

LSTM网络结构详解：隐状态

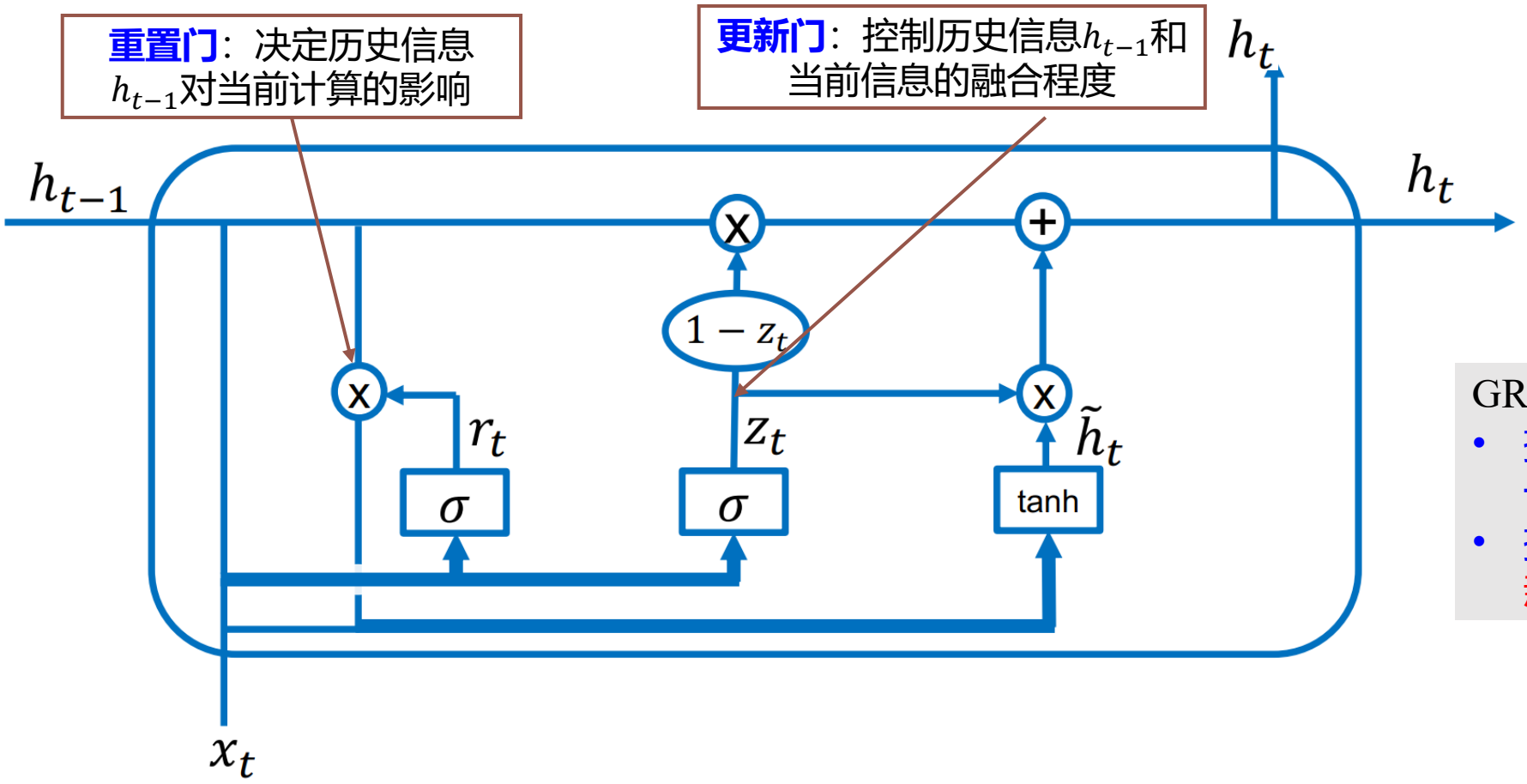


输出门：基于当前的单元状态，决定隐状态的出处内容

$$o_t = \sigma(W_{ox} \cdot x_t + W_{oh} \cdot h_{t-1} + b_o)$$

$$h_t = o_t \circ \tanh(C_t)$$

门控循环单元（Gated Recurrent Unit，GRU）:针对LSTM的进一步改进



GRU与LSTM相比：

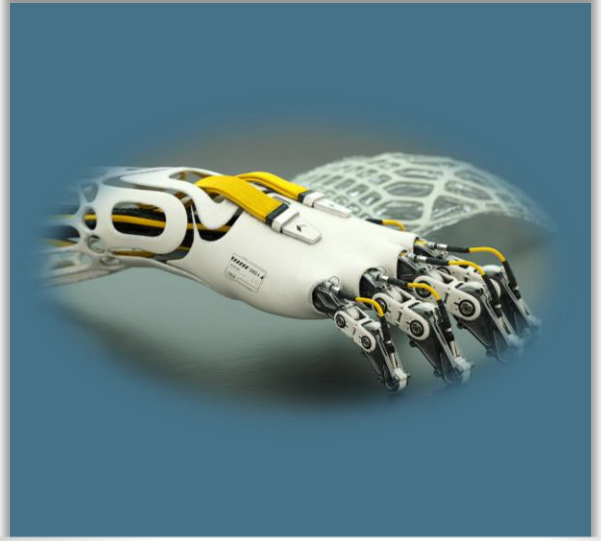
- 把单元状态和隐状态合并为一个隐状态输出；
- 把遗忘门和输入门合并入更新门z。

$$r_t = \sigma(W_{rx} \cdot x_t + W_{rh} \cdot h_{t-1})$$

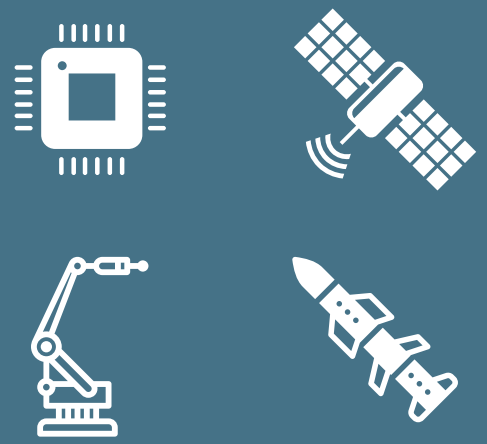
$$z_t = \sigma(W_{zx} \cdot x_t + W_{zh} \cdot h_{t-1})$$

$$\tilde{h}_t = \tanh\left(W_{hx} \cdot x_t + W_{hh} \cdot (r_t \circ h_{t-1})\right)$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$



时序神经网络

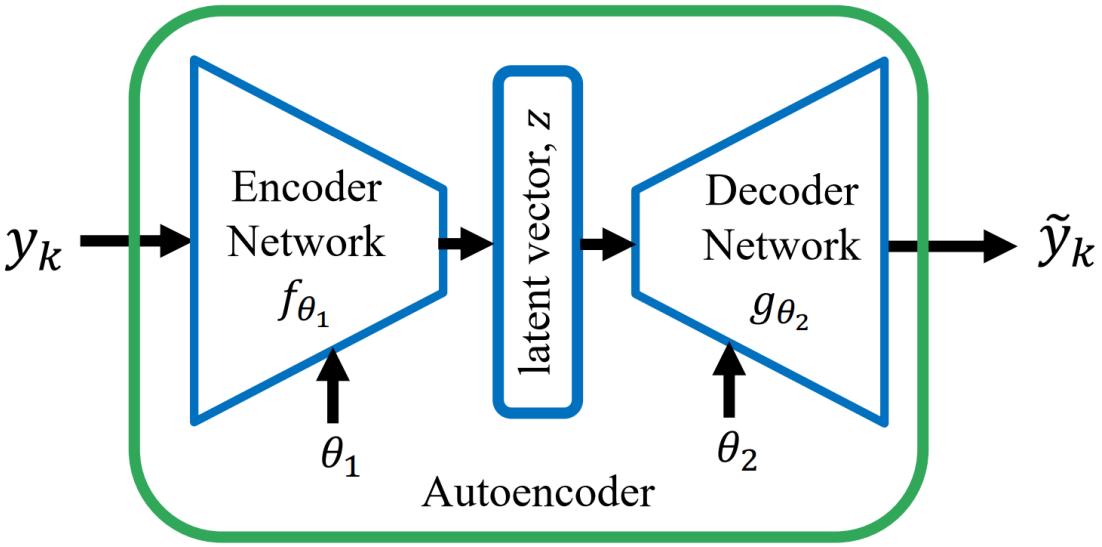


1. 循环神经网络模型
2. Sequence-to-Sequence模型
3. 从注意力机制到自注意力
4. 传统Transformer模型

Autoencoder（AE）简介

- **Autoencoder的特征和主要组成**
  - 是一种无监督学习的神经网络模型，通过“压缩-重建”机制学习数据的低维表示（潜在空间）。其核心目标是将输入数据压缩为紧凑编码（隐编码），并从中高精度地重建原始数据，无需人工标注标签；
  - 编码器（Encoder）：将高维输入 x 压缩为低维隐向量z（通常维度远小于输入）。
  - 解码器（Decoder）：从隐编码 z 重建原始变量，作为输出；
  - AE的训练：构建“输入与输出之间的”损失函数，反向传播由网络中所采取的模块特性决定

$$Loss(\theta) = \sum_{k=0}^{K-1} \left\| y_k - g_{\theta_2}\left(f_{\theta_1}(y_k)\right) \right\|^2$$



Seq2Seq框架简介

- **Autoencoder框架下的有限长序列输入输出**
  - 输入：长度为n的有序token组 $\{x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_{n-1}, x_n\}$。
  - 输出：长度为n的有序组$\{y_1, y_2, \ldots, y_{i-1}, y_i, \ldots, y_{n-1}, y_n\}$。
  - Encoder和Decoder可分别使用RNN（可以是多层）框架下的模型，即：

Decoder还产生一个输出 → y_t

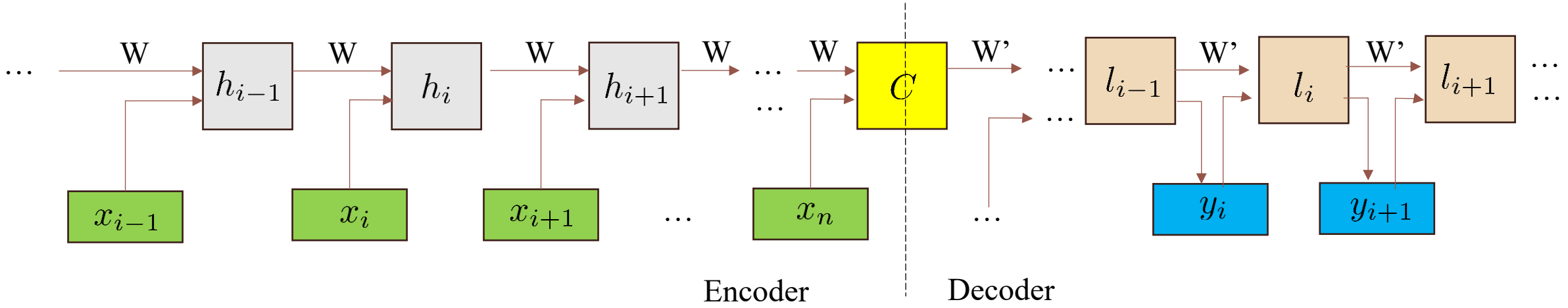
$$h_t = f(x_t, h_{t-1})$$

Hidden State → h_t

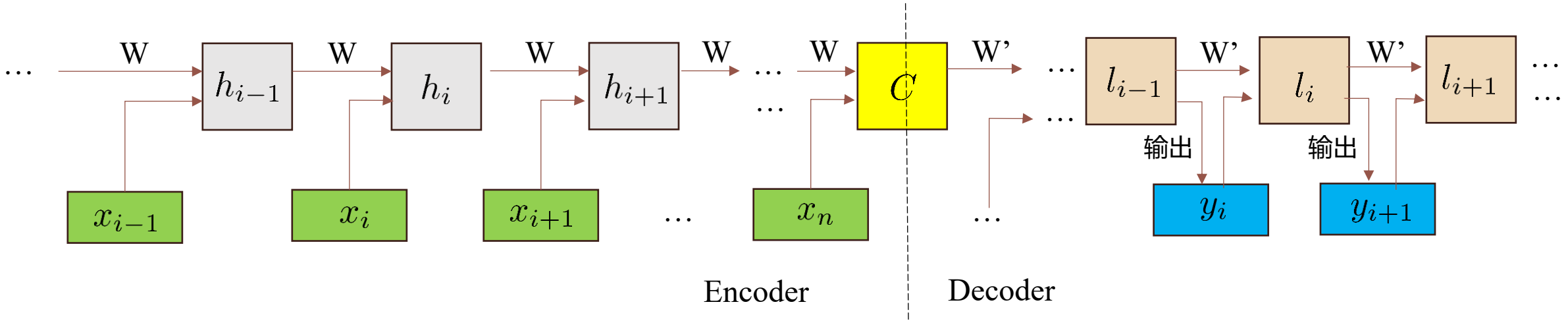
输入 → x_t

上个输入对应的Hidden State → h_{t-1}

  - 沿输入-输出序列（时间步）展开，如下图



不存在Attention时的Seq2Seq模型



- 每步输入 x_i 产生一个Hidden Vector h_i，每个Hidden Vector都经连接矩阵 W 输入给下一步的隐含向量；
- Encoder的最后一个Hidden Vector h_n 作为上下文的唯一概要表示 C（称为Context Vector），用做Decoder的初始状态输入；
- Decoder的每一步输出 y_i 作为RNN下一步更新的输入；
- Decoder的每一个潜状态向量（称为Latent Vector）经连接矩阵W′ 输入给下一潜状态向量。

Seq2Seq模型引入注意力机制（Attention）

- Decoder输出的条件概率由其本身的Latent Vector l_i 和Context Vector C 共同决定：

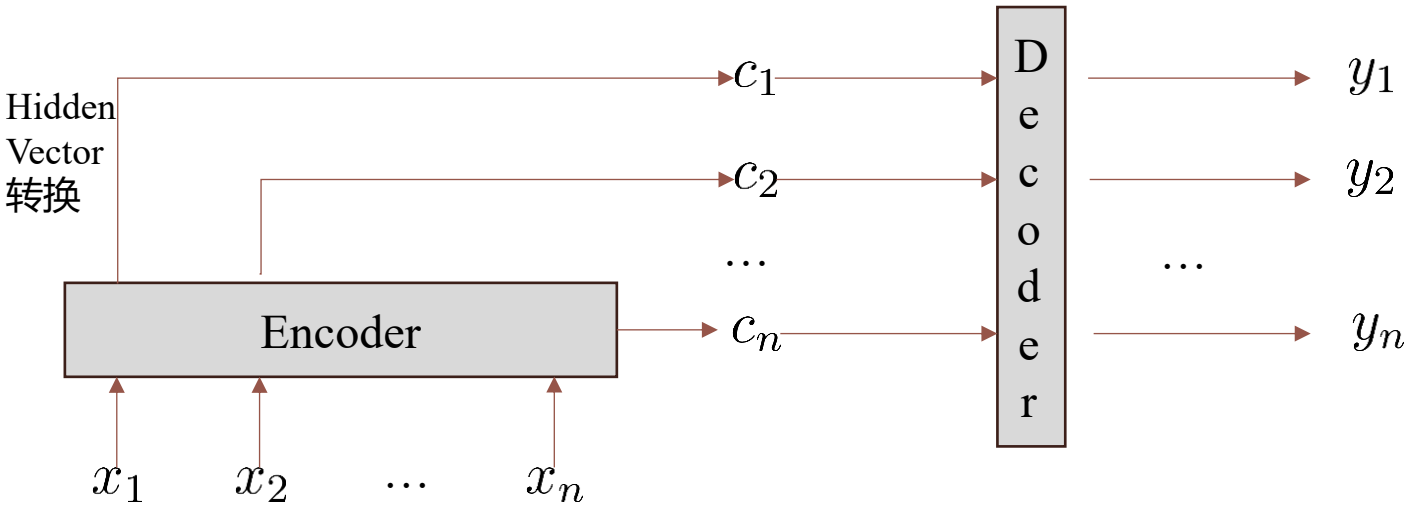
$$\Pr(y_i|y_1,\ldots,y_{i-1}) = g(y_{i-1}, l_i, C)$$

其中，$g(\cdot)$ 是由解码器决定的映射函数。

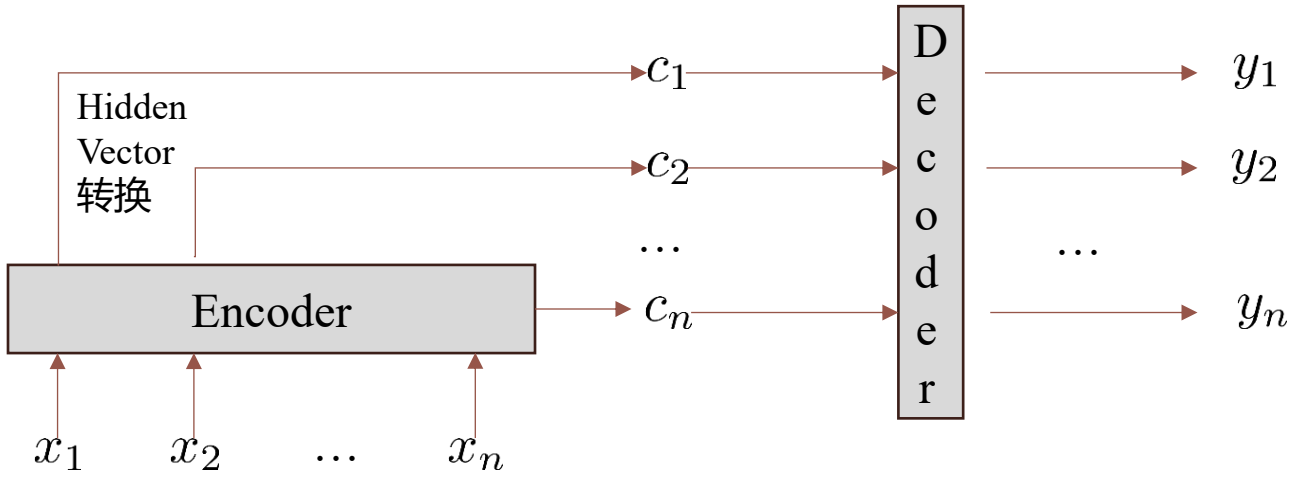
引入Attention模型的一般思路

- 扩展上下文表示的维度，对每个输入向量都产生一个对应的Context Vector C_i

$$\Pr(y_i|y_1,\ldots,y_{i-1}) = g(y_{i-1}, l_i, c_i)$$



RNN注意力机制：由加权平均生成/确定Attention的值



Bahdanau Attention

- **每个输出向量对应的Context Vector是每个输入对应的隐含状态的加权值：**

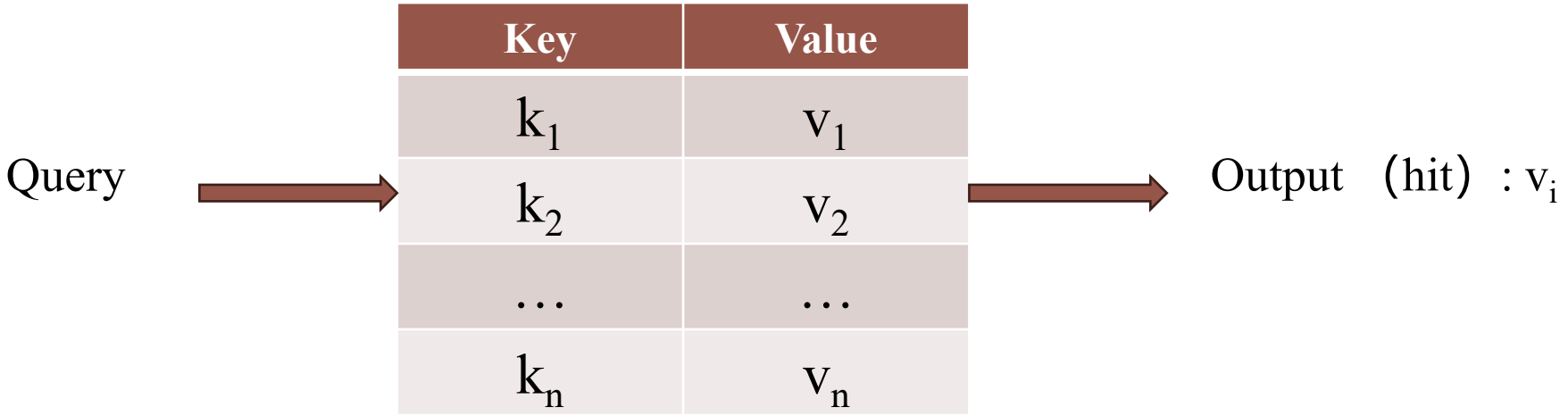
$$c_i = \sum_{j=1}^{n} a_{i,j} h_j$$

其中，$a_{i,j}$ 是第 *j* 个Hidden Vector对输出向量y_i 的可学习贡献权重（影响）。

- **由上，如何决定注意力分布矩阵 $A = [a_{i,j}]_{i,j}$，即为Seq2Seq模型待解决的问题。**

确定Attention值：软注意力机制

- **硬注意力：类比数据库系统**
- **考虑从一个数据库中的Key-Value Pair中查询-检索（Query-Retrieve）某个给定值**



- **一次关联一个key；**
- **一次检索返回一个定值value。**

软注意力机制（续）

- **软注意力：一次查询关联过的多个key值，通过计算query（q）与key（k_i）值之间的相关关系，确定哪个key与query最相关**

$$\text{attention}(q, \{k_i\}_{i=1}^n, \{v_i\}_{i=1}^n) = \sum_{i=1}^n a_i v_i$$

  - 相关性（similarity）s_i：q与k_i相似性的度量：$s_i = \text{similarity}(q, k_i)$
  - Attention分布概率：对s_i做Softmax处理后的值

$$a_i = \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_{j=1}^n e^{s_j}}$$

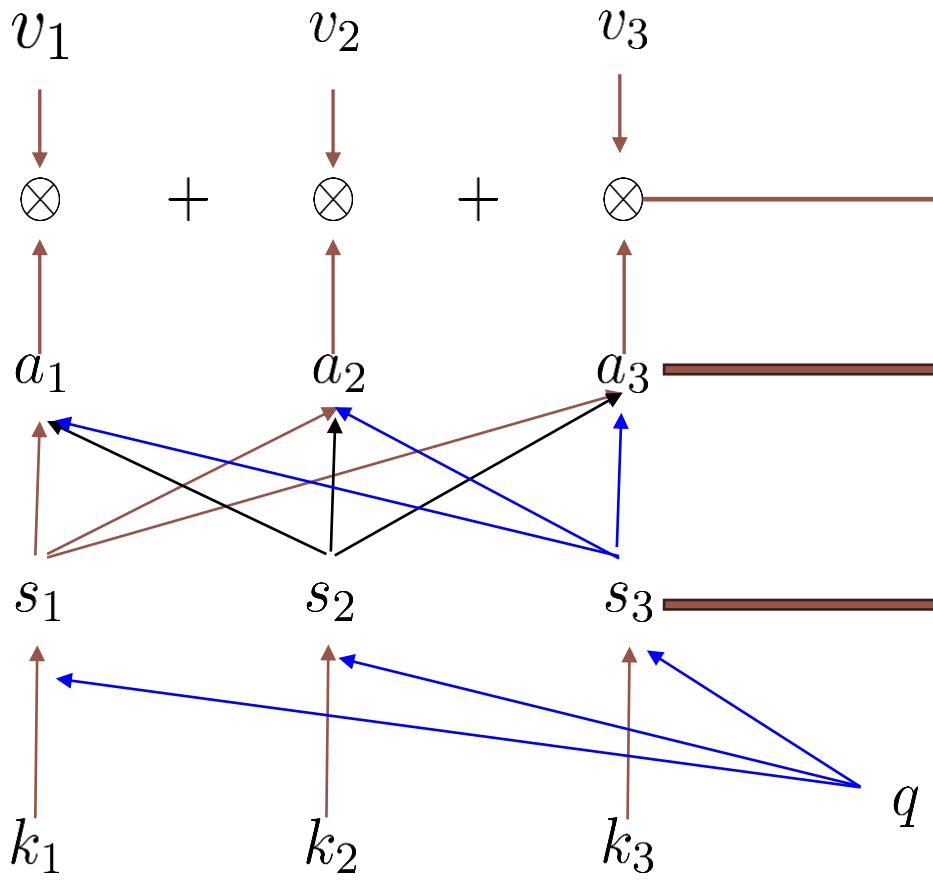
  - Attention值的计算：

$$\text{attention}(q, \{k_i\}_{i=1}^n, \{v_i\}_{i=1}^n) = \sum_{i=1}^n a_i v_i$$

Softmax相关性计算

软Query-Retrieval

- **图示化计算步骤**



$$\text{attention} = \sum_{i=1}^{3} a_i v_i$$

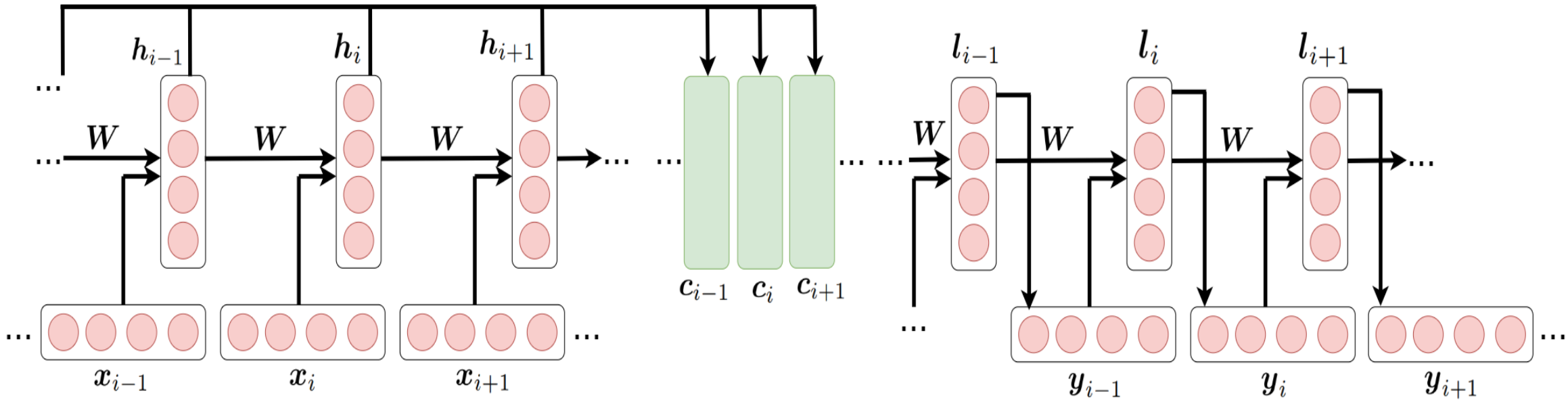
$$a_i = \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}}$$

$$s_i = \text{similarity}(q, k_i)$$

常用的Similarity计算方式：内积（或Scaled内积）$s_i = q^T k_i$

软注意力机制适配：基于RNN的Autoencoder

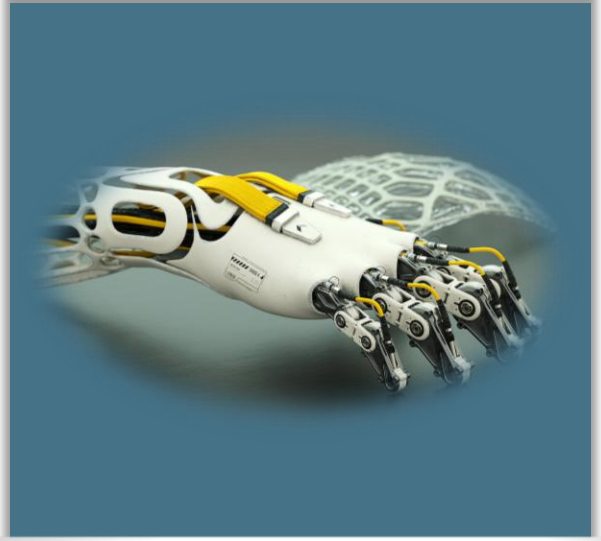
- **早期线性注意力机制**



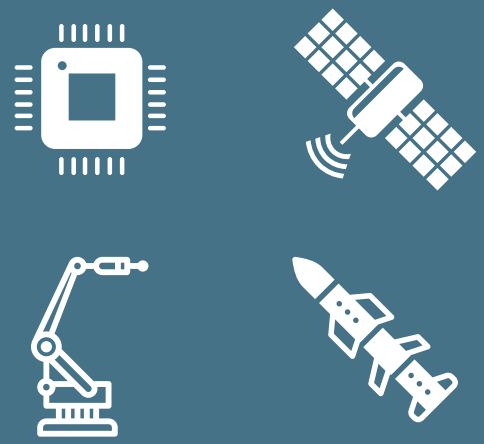
- 注意力机制计算s_{ij}：解码器的隐状态l_{i-1}与编码器的隐状态h_i做相似性的计算

$$\mathbb{R} \ni s_{ij} := \text{similarity}(\boldsymbol{l}_{i-1}, \boldsymbol{h}_j)$$

- 输出y_i由解码器的隐状态l_i决定；
- 输入隐状态h_i由输入x_i决定。



时序神经网络



1. 循环神经网络模型

2. Sequence-to-Sequence模型

3. 从注意力机制到自注意力

4.传统Transformer模型

自注意力（Self-Attention）机制

- **自注意力**：所有的query、 key和value都来自于对同一输入序列的计算。
- 举例（Toy Example）

输入：	I	am	a	student
Query：	student			
Value（相似性计算时需要做嵌入处理）：	I	am	a	-
归一化后的Similarity	0.7	0.2	0.1	-
Attention值	$0.7v_1+0.2v_2+0.1v_3$			

点积自注意力中Query、Key和Value的计算

- **向量形式（对d维输入而言）**
  - **Query：** $\mathbb{R} \ni q_i = W_Q^T x_i$
  - **Key：** $\mathbb{R} \ni k_i = W_K^T x_i$
  - **Value:** $\mathbb{R} \ni v_i = W_V^T x_i$
- **把输入、Query、Key和Value按列向量堆叠成矩阵，X，Q，K，V，注意力机制的矩阵形式如下：**

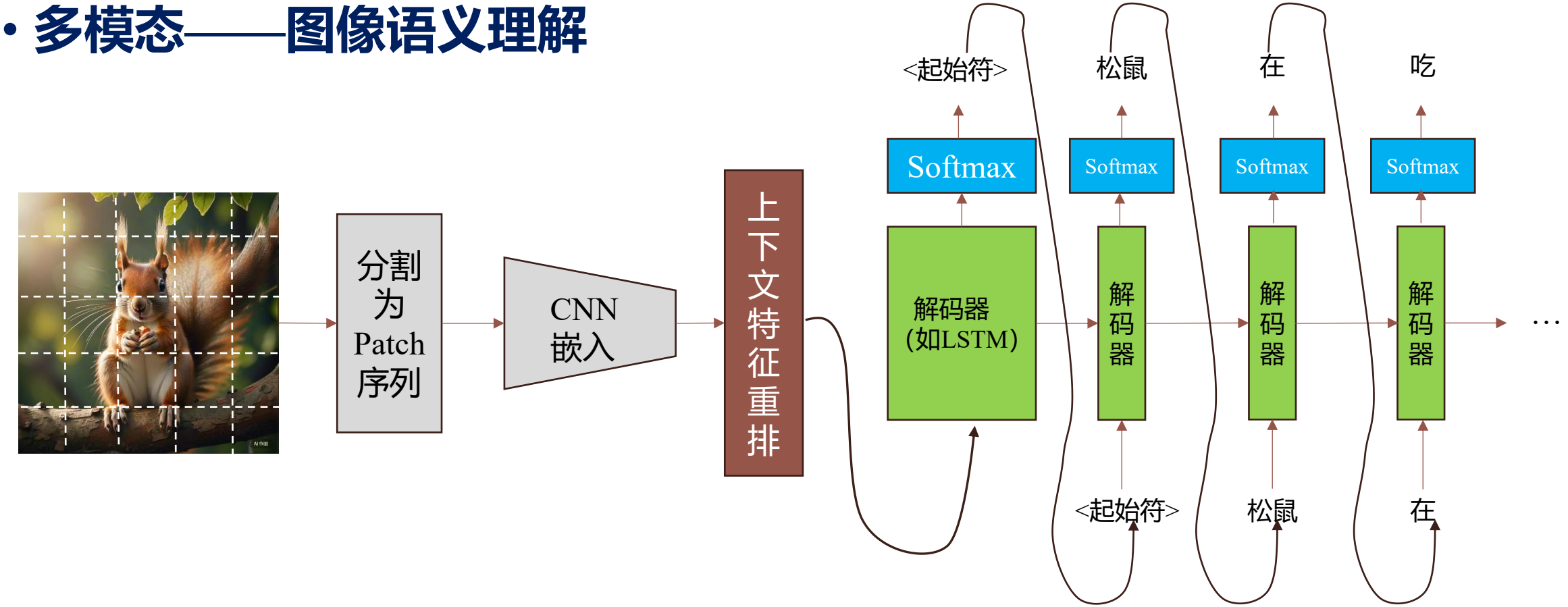
$$\begin{aligned}\text{Attentiion}(Q,K,V) &= V\text{Softmax}\left(\frac{1}{\sqrt{p}}Q^T K\right) \\ &= Z = [z_1,\ldots,z_n]\end{aligned}$$

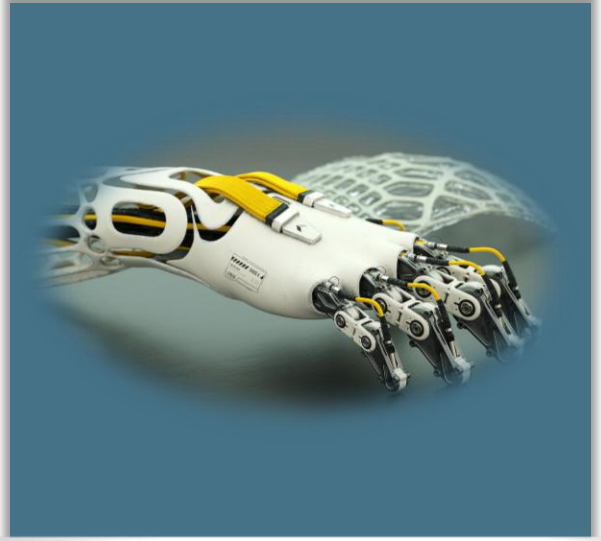
Scaled注意力机制

其中，三个矩阵$W_{*=Q,K,V}$是可学习的权重矩阵，三个向量的维度满足：$d_q = d_k = d_v$

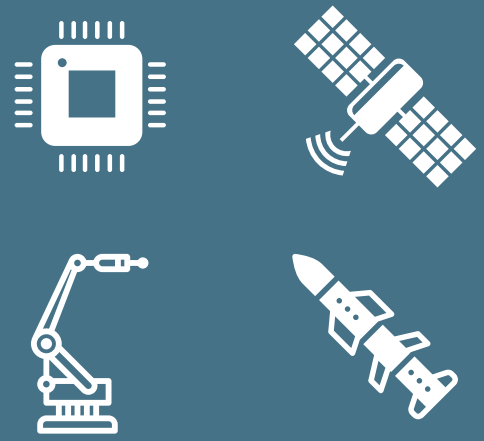
（一个可能的）带自注意力机制的一般化Encoder-Decoder例子

- 多模态——图像语义理解





时序神经网络



1. 循环神经网络模型
2. Sequence-to-Sequence模型
3. 从注意力机制到自注意力
4. 传统Transformer模型

词嵌入（Embedding）：Transformer的输入

- **词嵌入是对输入序列中Token的预处理过程**
  - 是序列数据处理任务（如自然语言处理，NLP）中将离散符号Token（如单词或子词）转化为连续数值向量的过程，为后续模型（如 Transformer）提供可计算的语义表示；
  - 最简单的embedding：独热编码。

- **词嵌入通过 “嵌入矩阵 + 语义学习” 实现**
  - **方法1**：利用平台的Embedding层学习词嵌入（如Torch平台的nn.Embedding）；
  - **方法2**：使用预训练的词嵌入，典型如Word2Vec 模型（主流方法之一）
    - 预训练方法利用在较大语料上预训练好的词嵌入或预训练模型，在当前任务中把这些词嵌入加载到待训练模型中。除word2vec外，预训练模型还有：ELMo、BERT、XLNet、ALBERT等。

Transformer中编码器部分的自注意力机制模块

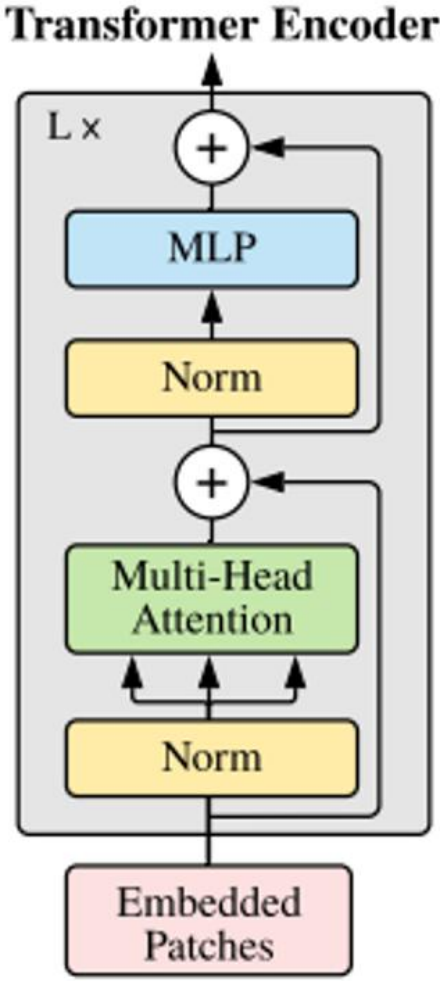
- **位置编码**
  - **（如右图）**在嵌入向量处理的过程中，需要加入位置编码，通过显式添加位置信息，使模型区分序列中各元素的顺序；
  - 正弦/余弦格式的位置编码：

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

其中，d_{model}是词向量（嵌入向量）维度；10000是调节波长超参数（用以控制正弦函数频率衰减）。

  - 位置编码和embedding编码通过逐元素相加（addition）而非拼接（concatenation）的形式组合在一起，相当于在同一语义空间中叠加两种信息。



- **层归一化（Norm）：对层的激活净输入信号进行归一化**
- **多头自注意力+残差连接机制**
- **多层感知机+残差连接机制**

Transformer中的自注意力机制模块（续）

- **位置编码**
- **层归一化（Norm）**
- **多头自注意力+残差连接机制**
  - （回忆：软注意力机制）查询（Q）、键（K）和值（V）矩阵的计算都从同一输入（Norm层的输出）x_i获得：

$$\mathbb{R} \ni q_i = W_Q^T x_i \quad \mathbb{R} \ni k_i = W_K^T x_i \quad \mathbb{R} \ni v_i = W_V^T x_i$$

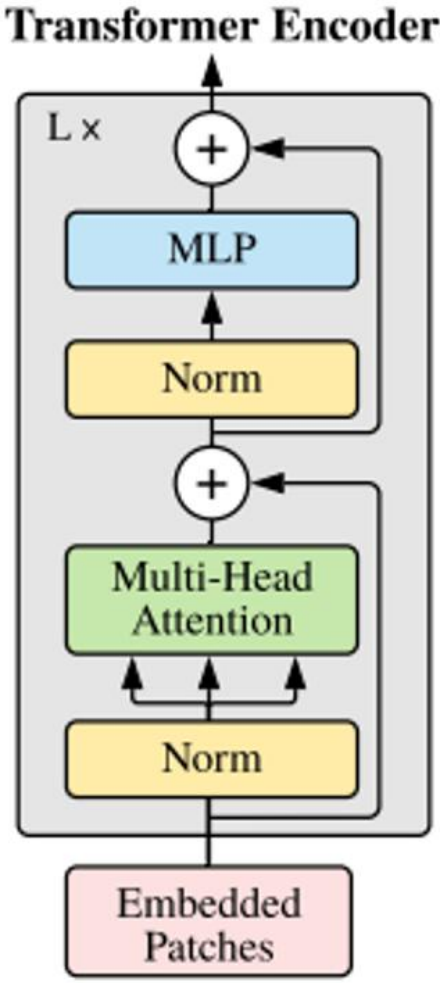
  - 多头自注意力——对同样的输入序列**X**，引入N个头（即N组Q、K、V权重矩阵）：$\{\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V\}_{h=1}^N$，每组权重矩阵独立初始化并学习，确保不同头关注输入的不同特征子空间：

$$\mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^Q, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^K, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^V$$

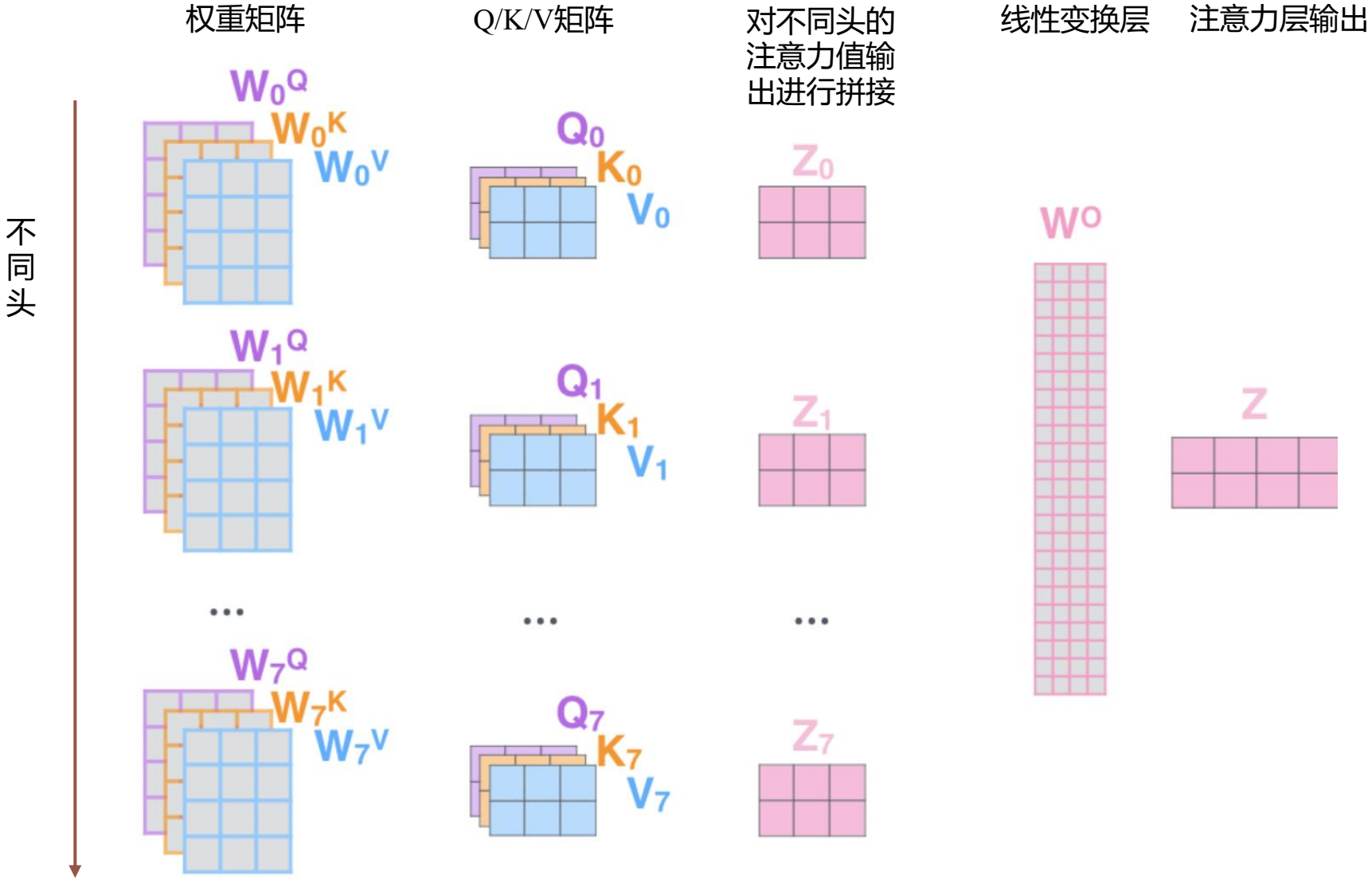
  - 每个头的注意力（缩放点积注意力）值计算（计算后不同头通过线性层拼接）：

$$\text{head}_h = \text{softmax}\left(\frac{\mathbf{Q}_h \mathbf{K}_h^\top}{\sqrt{d_k}}\right)\mathbf{V}_h$$

- **多层感知机+残差连接机制**



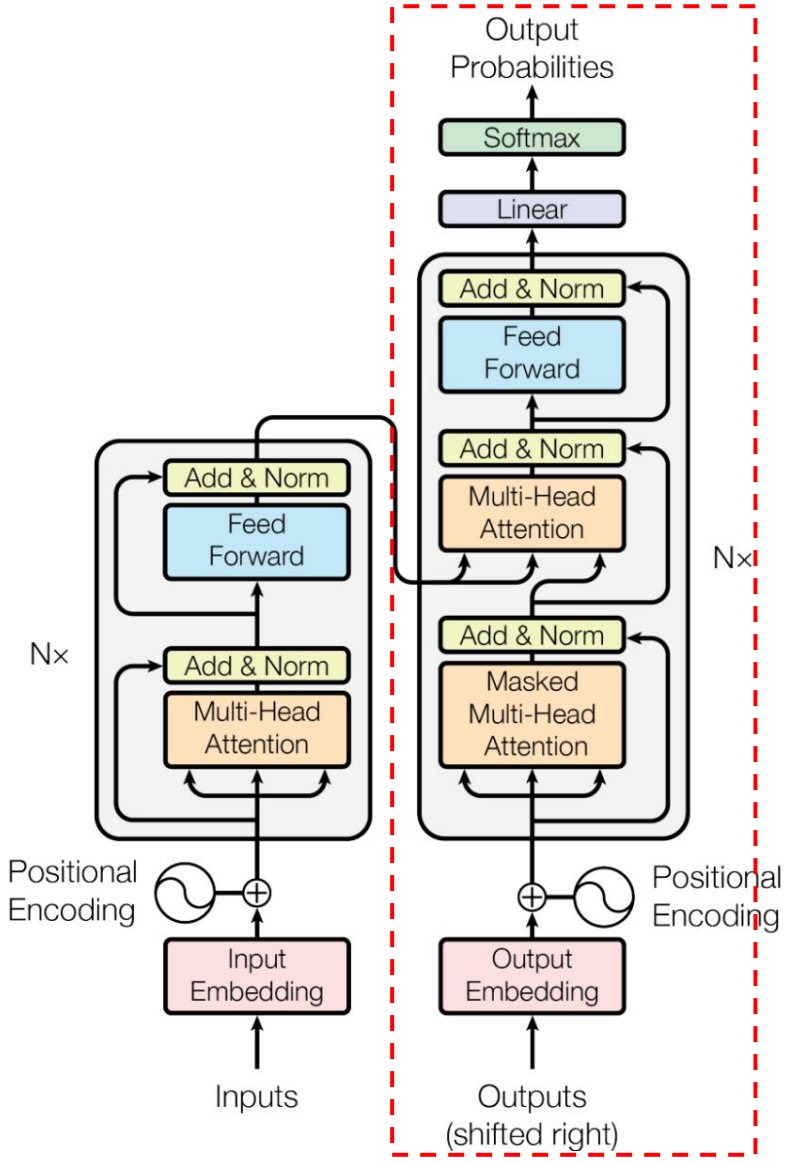
多头自注意力机制计算图示



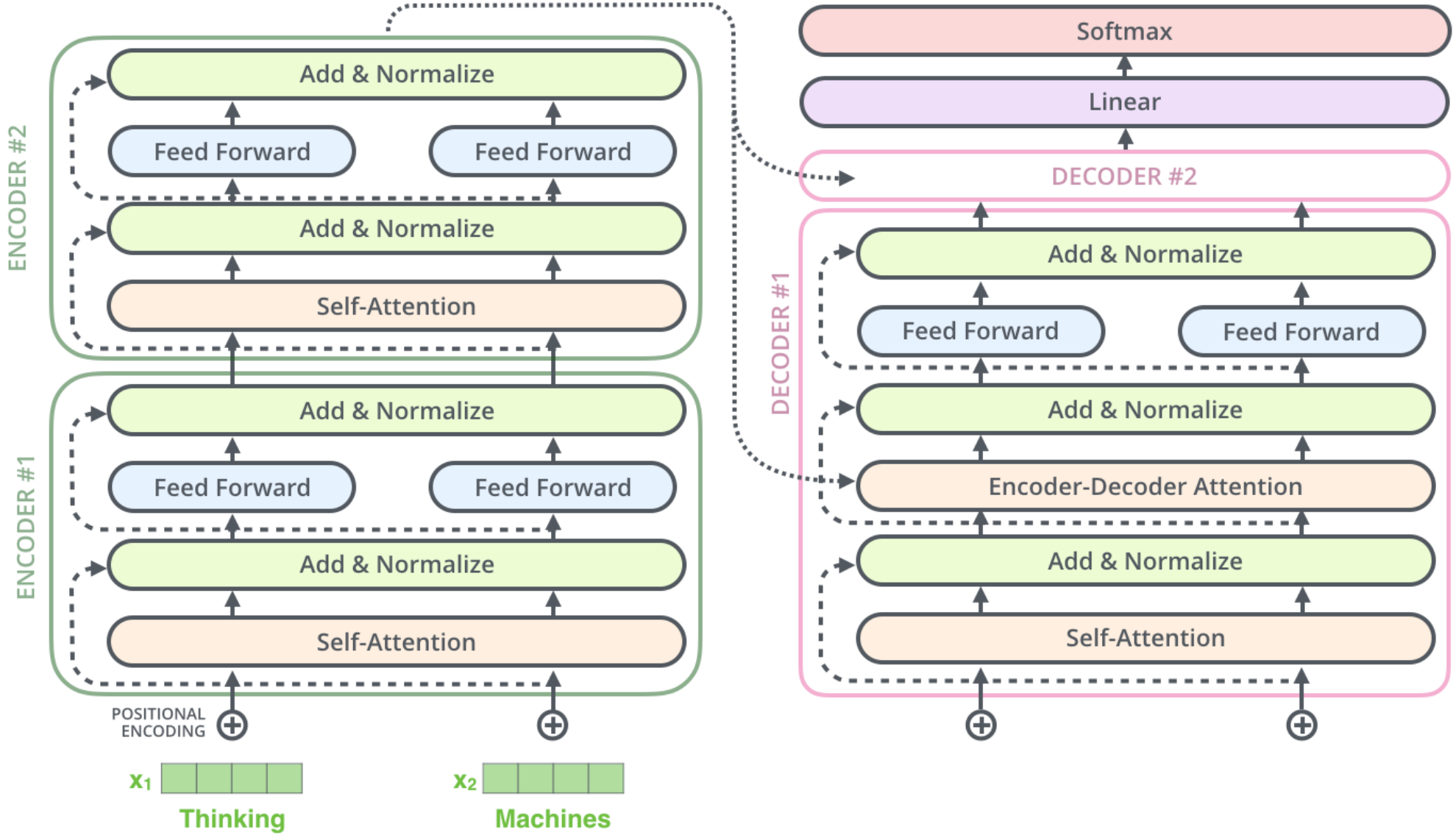
图片来源：
https://jalammar.github.io/illustrated-transformer/

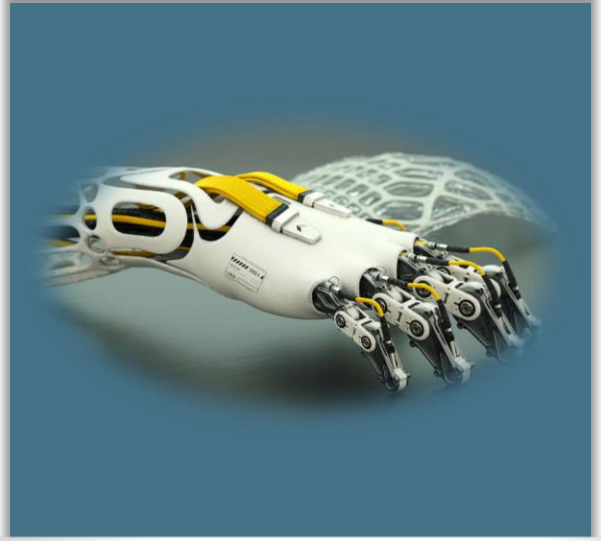
编码器-解码器构型下的Transformer

- **解码器的输入**
  - 目标序列嵌入：也就是我们期望模型生成的序列，例如机器翻译中的目标语言句子。
  - 位置t掩码：在解码器的自注意力层中，当处理第t个位置的输入时，模型会生成一个掩码（mask），使得该位置的词只能关注到自身（位置t）及之前位置（位置1到T-1）的词。
- **多头交叉注意力机制**
  - 查询（Query）向量的来源：来自于解码器的上一个输出；更准确地说，是来自同一模块解码器上一层的输出（即 Masked Multi-Head Attention + Add & Norm 之后的输出）。
  - 键（Key）和值（Value）向量的来源：来自编码器堆叠Attention层的最终输出的序列表示（即编码器最顶层 Add & Norm 后的输出）。
  - **注意：**无论解码器有多少层（右图第1,2,...,N层），其交叉注意力层接收的 Key 和 Value 输入来源是固定的（见下页），始终是整个编码器（所有堆叠层处理完毕后）的最终输出序列。

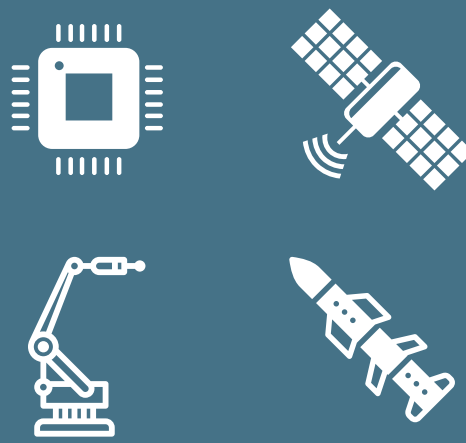


编码器-解码器构型下的Transformer图解





时序神经网络



讨论