NLP的大多数任务中的数据都是字符、词、短语、行或句子的序列，可以将这些任务看作是给序列中的每一项(item)作标记(lable)，如词性标注、分词、分类、句子分割等

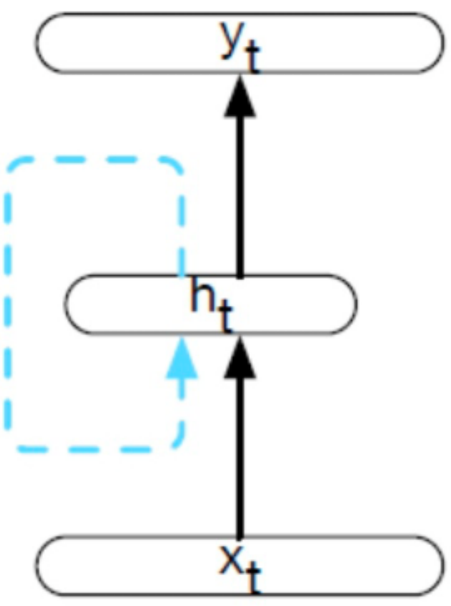
在之前的学习中，处理这种任务可以用到两种模型，一种是基于Markov假设的概率图模型(HMM、HEMM、CRF)，一种是基于滑动窗口序列分类的分类模型（Naive贝叶斯，神经网络）

但它们都各有问题，对于第一种模型，它基Markov平稳假设，即窗口内状态有依赖关系，这会导致①上下文受限 ②大窗口参数复杂 ③Markov假设在文字序列结构变化部分不成立，对于第二种模型，会出现①上下文范围受限 ②词组分离(将不该拆的词拆开，如bookworm → book/worm) ③符号接地问题(the ground)即模型只能学到文本中的抽象符号，但无法联系现实世界

循环神经网络(RNN)

它含一个环的神经网络模型 它含：

输入x_t 隐藏状态h_t 输出y_t



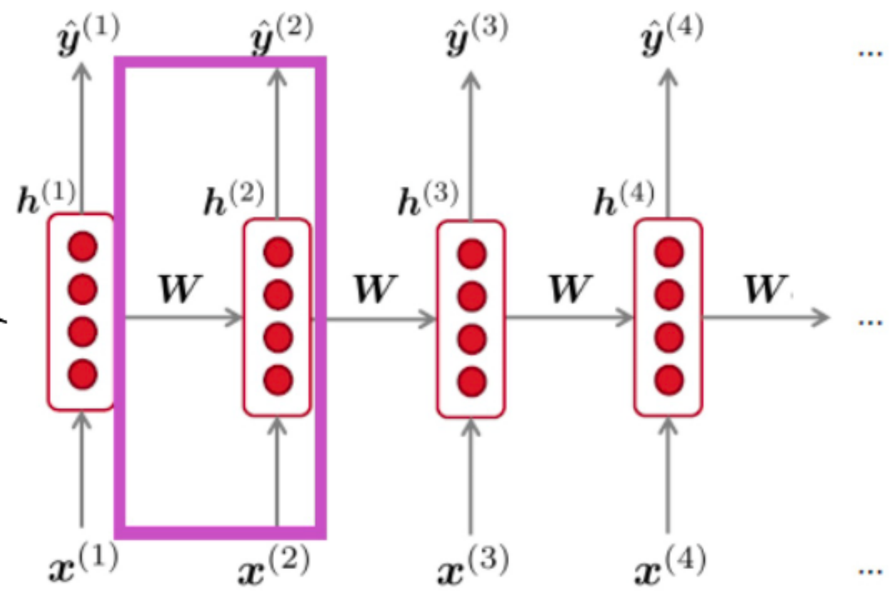
每循环一次，就输出一次，但一般只取最后一次输出

特点：输出与前一时刻的输出有关

前一时刻的输出为当前时刻输入

优点：当前时刻输出只与当前时刻输入有关

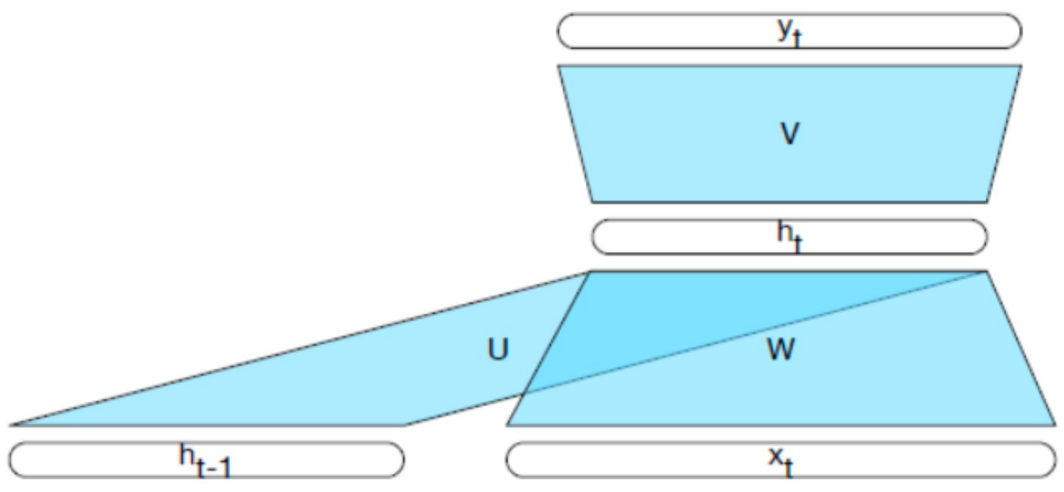
隐藏状态存储上下文信息



前同算法：

$$h_t = g(Uh_{t-1} + Wx_t)$$

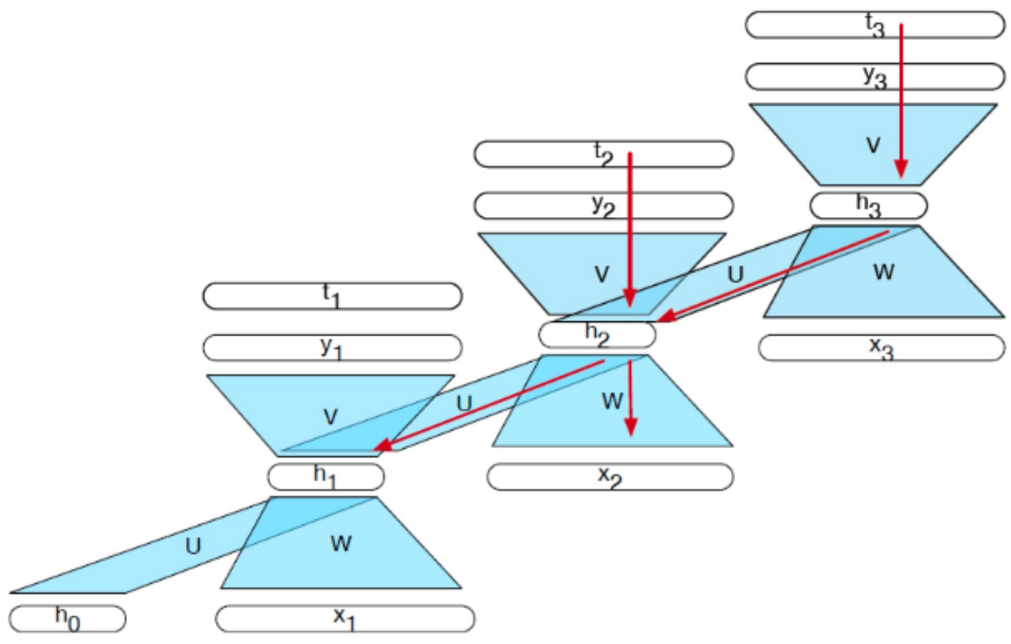
$$y_t = Softmax(Vh_t)$$



function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\,h_{i-1} + W\,x_i)$
  $y_i \leftarrow f(V\;h_i)$
return y

后向传播



窗口：理论上说，RNN能无限展开，但实际训练中是会限制其展开窗口长度，让其循环展开。如窗口长度为n，那么梯度就只回传n层，模型也只记住最近n个词

RNN应用

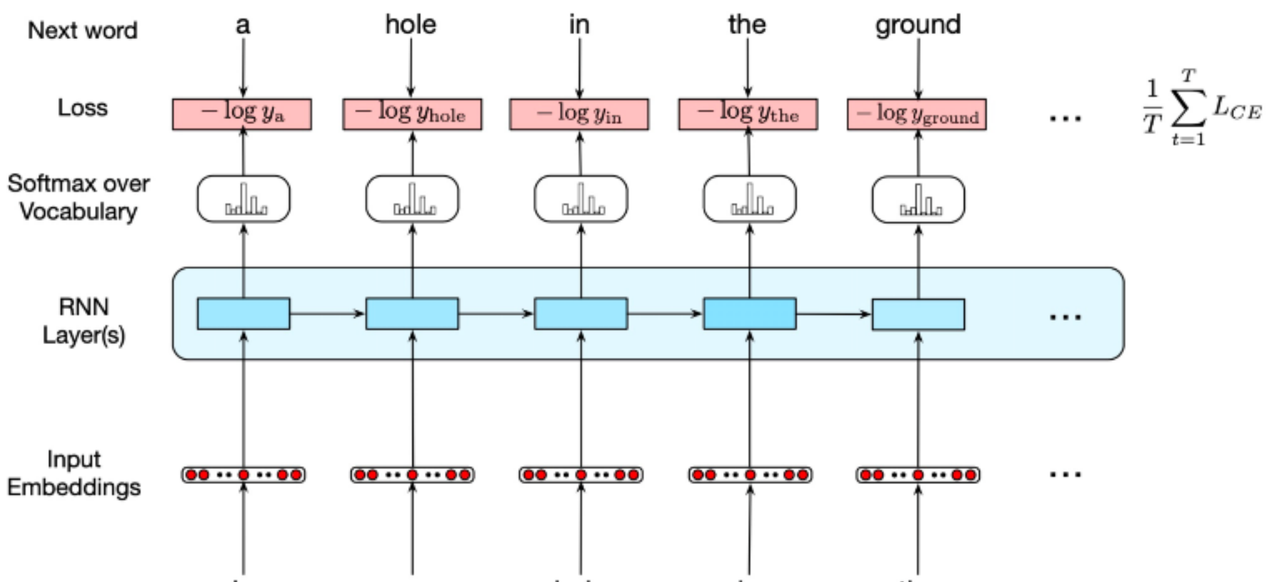
1. 语言模型

$$P(W_n|W_1^{n-1}) = y_n = Softmax(Vh_n)$$

$$P(W_1^n) = \prod_{i=1}^{n} P(W_k|W_1^{k-1}) = \prod_{i=1}^{n} y_k$$

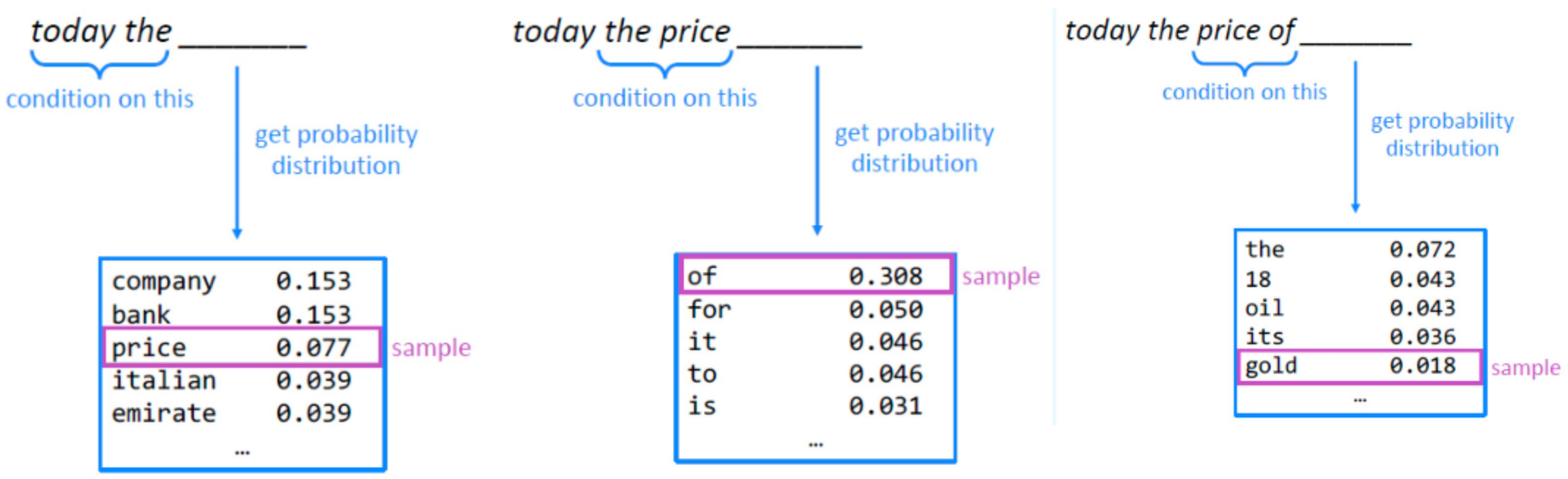
损失函数：$L_{CE}(\hat{y}, y) = -\log y_i = -\log \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

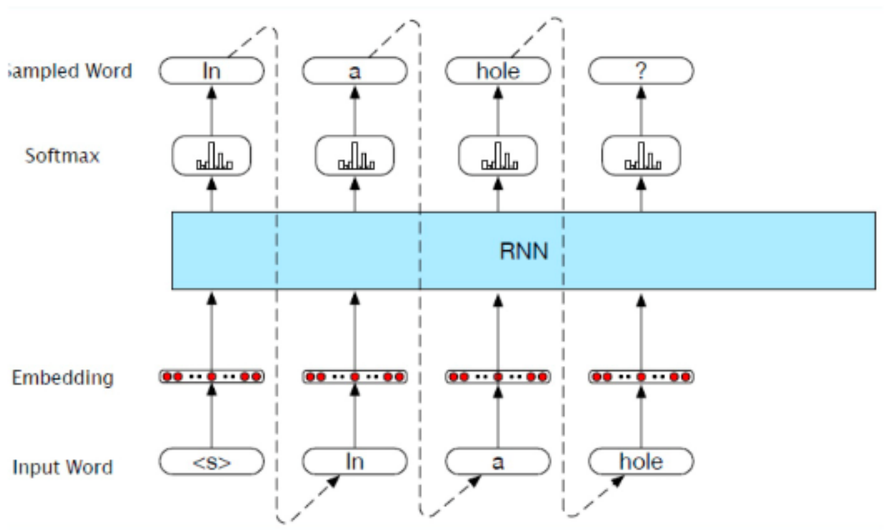
表示第t时刻的损失



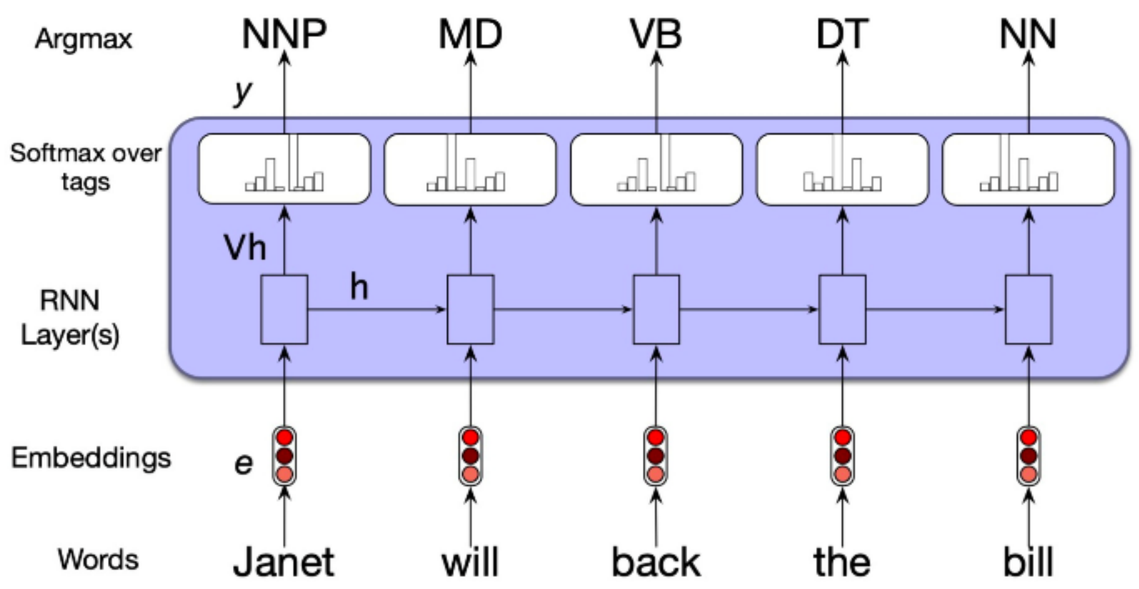
基于语言模型的文本生成：

困惑度：$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$





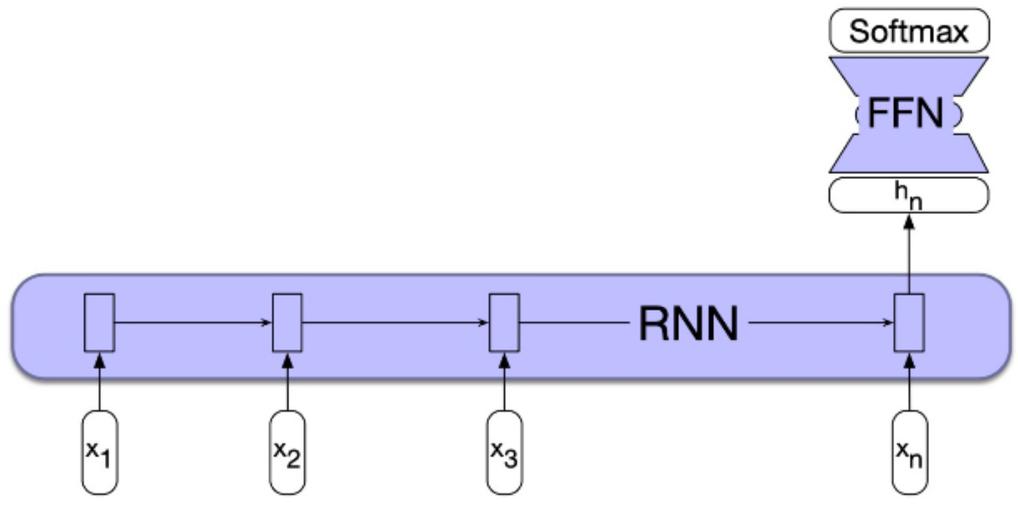
2、序列标注



精细标注：RNN + CRF（条件随机场）

RNN只输出概率最大的结果而不管是否通顺，CRF是一个看"当前词与前后标签"而判断标签是否合理的分类器，辅助修正RNN的结果

句子分类模型：RNN + FFN（前馈神经网络）+ Softmax



x_t：句子中每个分词的词向量

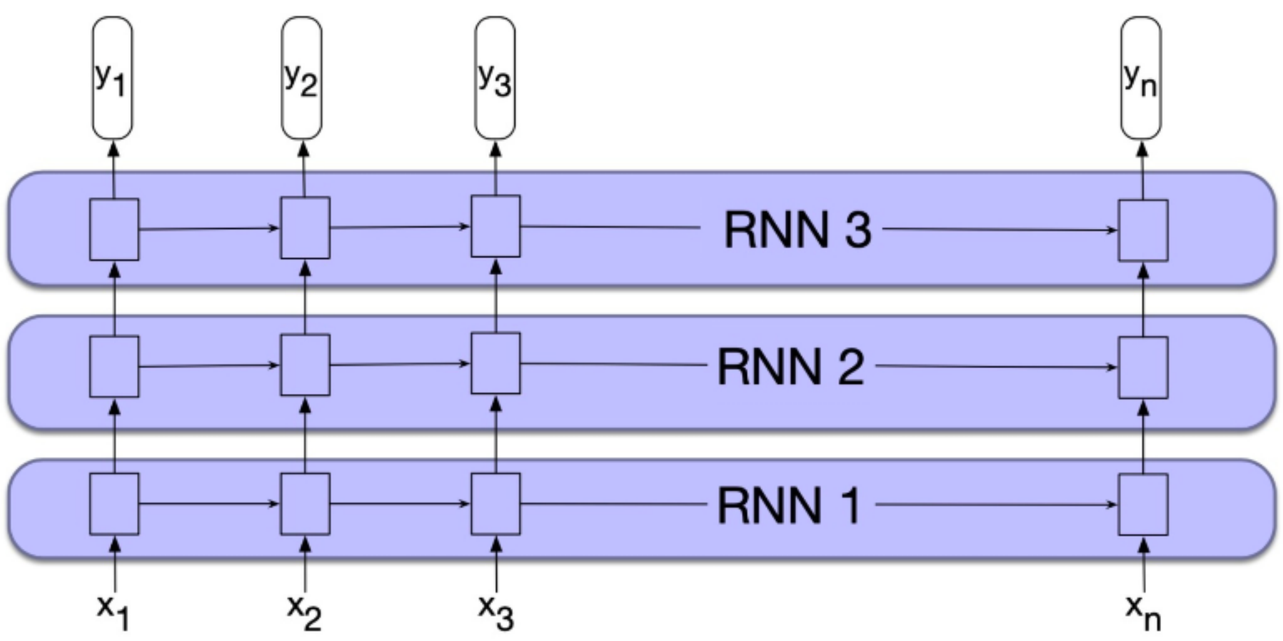
h_n：RNN读完整句话后压缩得到的"整句语义向量"
它含 $x_{1\sim n}$ 的所有上下文信息

FNN：对 h_n 做非线性特征变换，将语义向量映射到更适合分类的空间

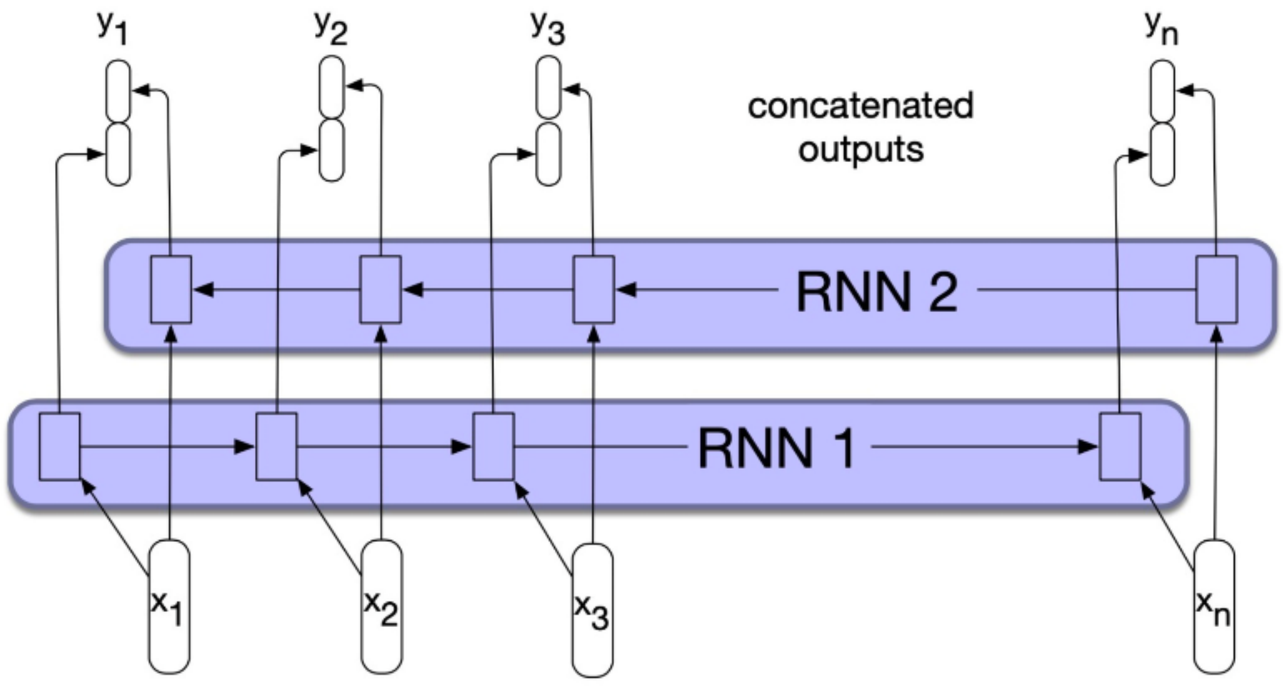
Softmax：将FNN的输出转化为概率分布，取最大概率对应类别输出

多层和双向循环神经网络

多层：将单层RNN输出作为下一层输入



多层双向



RNN1：从左到右处理序列

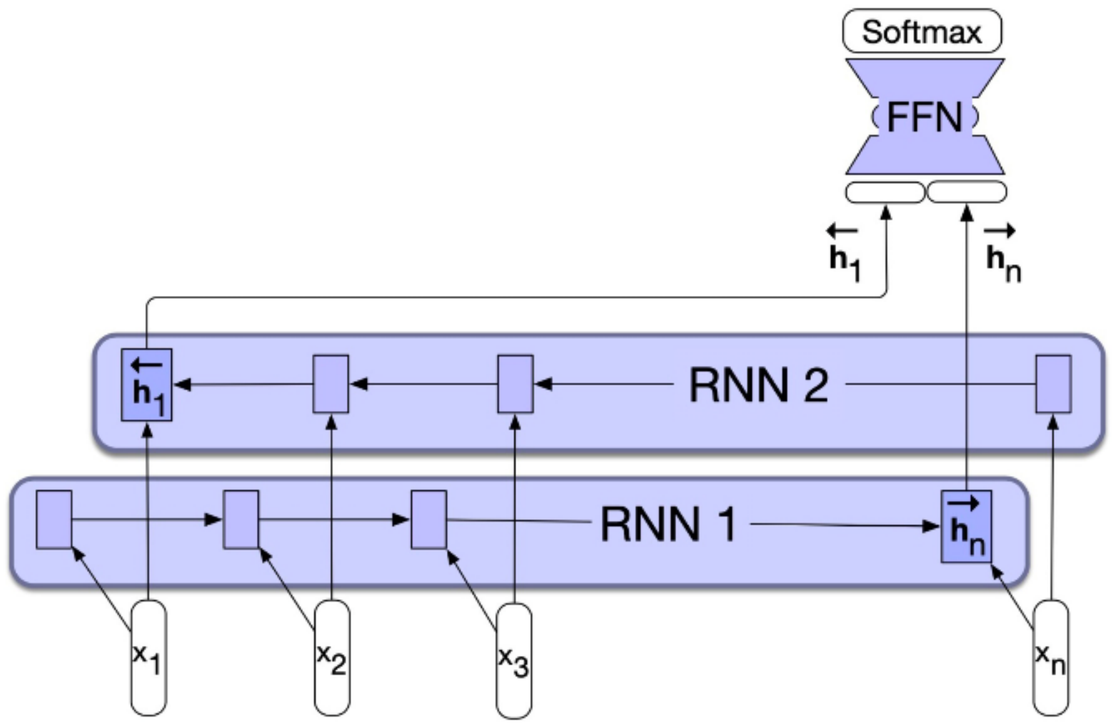
RNN2：从右到左处理序列

优势：对于每个时间步 t，能将 $h_t^{forward}$ 与 $h_t^{backword}$ 拼接，获得完整上下文特征而不只是前文

前向： $h_t^f = RNN_{forward}(X_t^t)$

后向： $h_t^b = RNN_{backword}(X_t^n)$

双向： $h_t = h_t^f \oplus h_t^b$



长短期记忆网络（LSTM）

RNN两个问题：①长时记忆问题：RNN单元记忆信息随着距离增加而降低

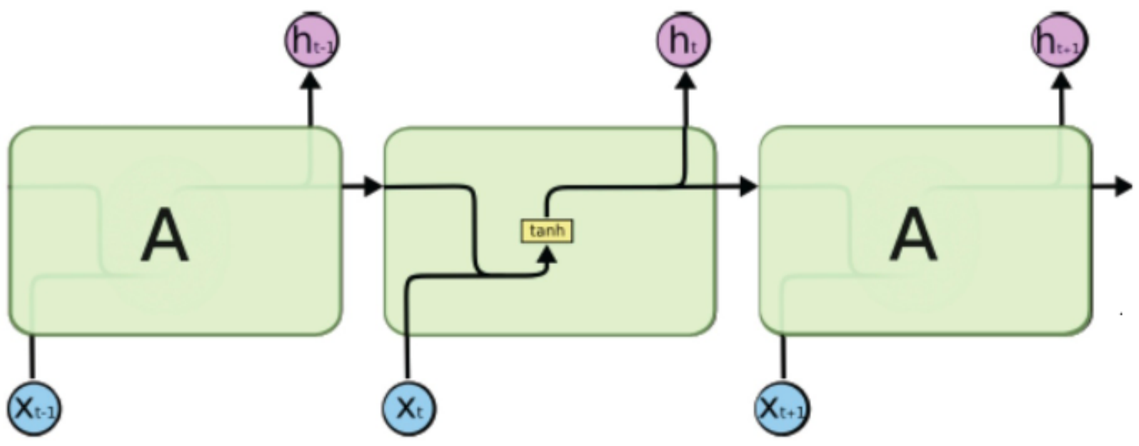
② 梯度消失问题：反向传播梯度逐渐消失。对于标准RNN激活函数一般是Tanh，$\partial h(t)/\partial h(t-1)$绝对值通常小于1，导致时间步很长时梯度会指数级衰减至0

$$\frac{\partial J(t)}{\partial h(1)} = \frac{\partial J(t)}{\partial h(t-1)} \times \frac{\partial h(t-1)}{\partial h(t-2)} \times \cdots \times \frac{\partial h(2)}{\partial h(1)} \approx 0$$

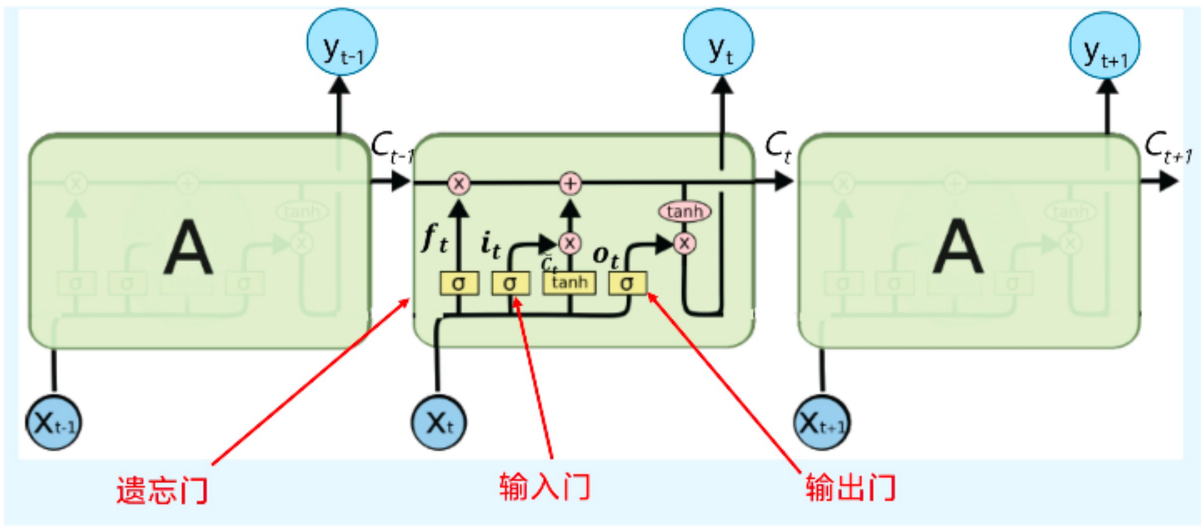
$J(t)$为t时刻的输出

长短期记忆网络：移除不需要的信息，增加可能需要的信息

标准RNN中的重复模块只包含单层神经网络（通常是一个Tanh激活的全连接层）



LSTM中的重复模块包含四个相互作用的神经网络层.



遗忘门：决定扔掉哪些旧信息

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f)$$

输入门：决定存入哪些新信息

$$i_t = \sigma(W_t[h_{t-1}, X_t] + b_i)$$

细胞状态更新层：生成新的候选信息

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, X_t] + b_c)$$

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

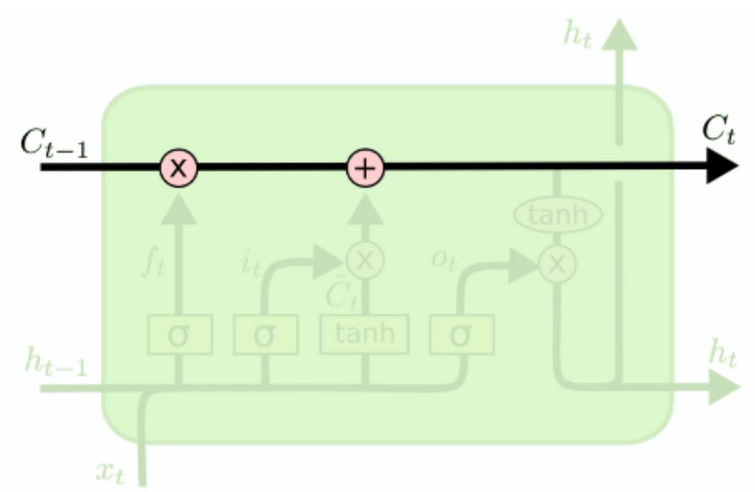
有时会把该层归入输入门中

输出门：决定输出哪些信息

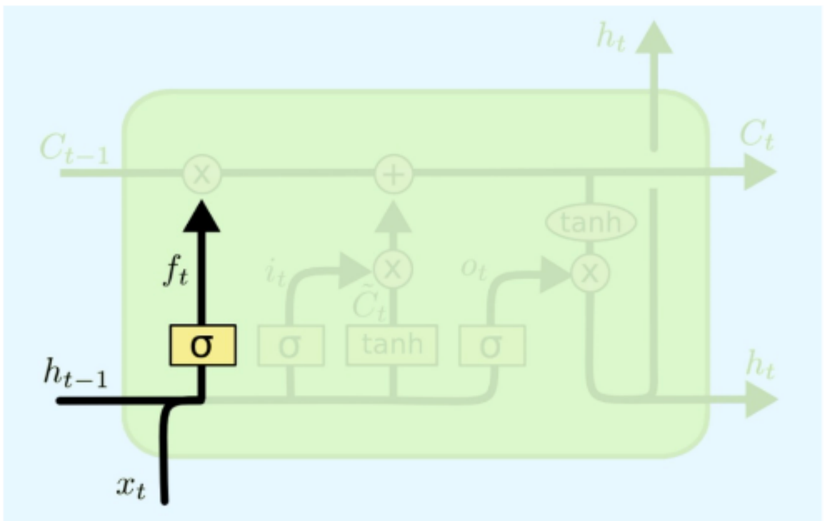
$$O_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o)$$

$$y_t = O_t * \tanh(C_t)$$

步骤：0、细胞状态即记忆，RNN里没有这个概念，或者说RNN里隐藏状态 h_t 既包含记忆又包含输出，每到一个新时间步 h_t 就会进行一次状态更新非线性运算，h_{t-1} 被 h_t 覆盖。而为了保留记忆，LSTM引入了细胞状态概念，将记忆与输出分开，它只进行线性运算 $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$，相比非线性，它丢失信息更少

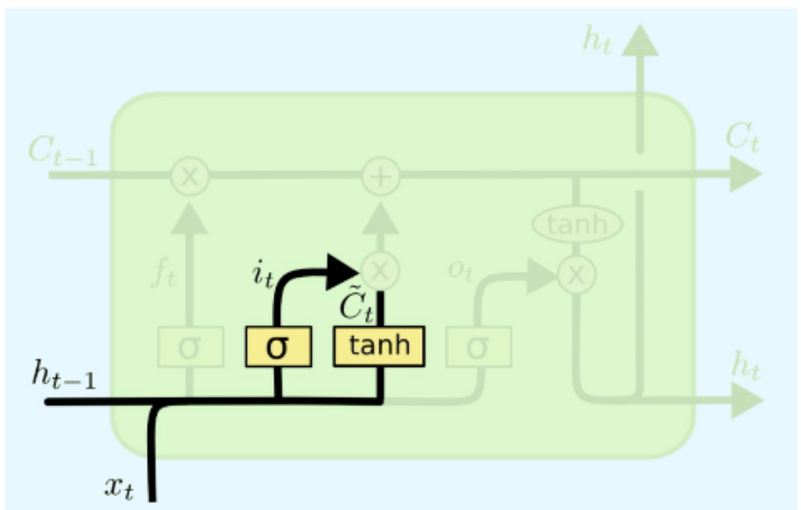


1、遗忘门读取h_{t-1}和x_t，输出一个每一位的值在0~1之间的向量f_t，用于处理C_{t-1}



$$f_t = \sigma(W_t \cdot [h_{t-1}, x_t] + b_f$$

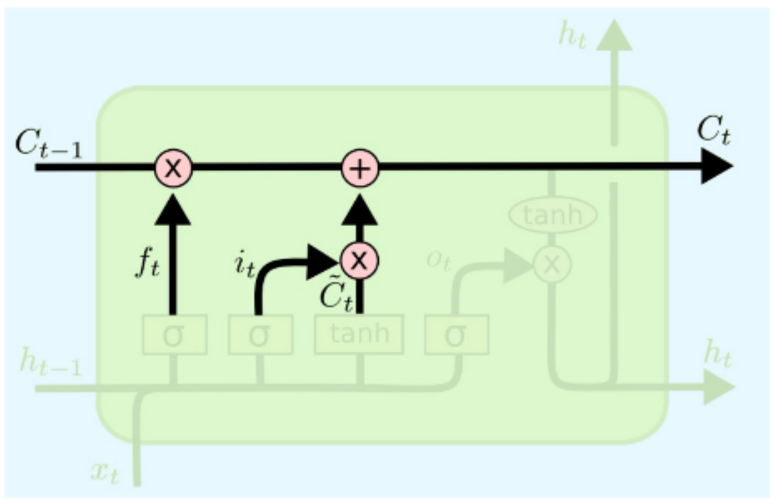
2、输入门层决定什么值要进行更新，tanh层创建一个新的候选值向量$\tilde{C}_t$加入C_t中



$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

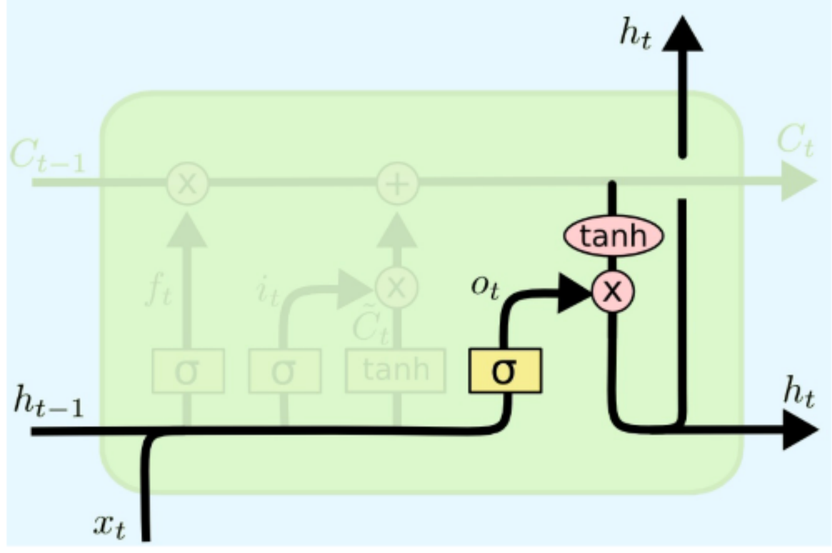
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c$$

3、旧状态C_{t-1}与f_t相乘丢弃掉部分旧信息，加上新信息$i_t * \tilde{C}_t$更新细胞状态C_t



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

4、运行一个sigmoid层来处理h_{t-1}与x_t，一个tanh层处理C_t（值域[-1,1]），并将两门的输出相乘，最终输出我们确定输出的那部分



$$O_t = \sigma(W_o[h_{t-1}, X_t] + b_o)$$

$$h_t = O_t * \tanh(C_t)$$

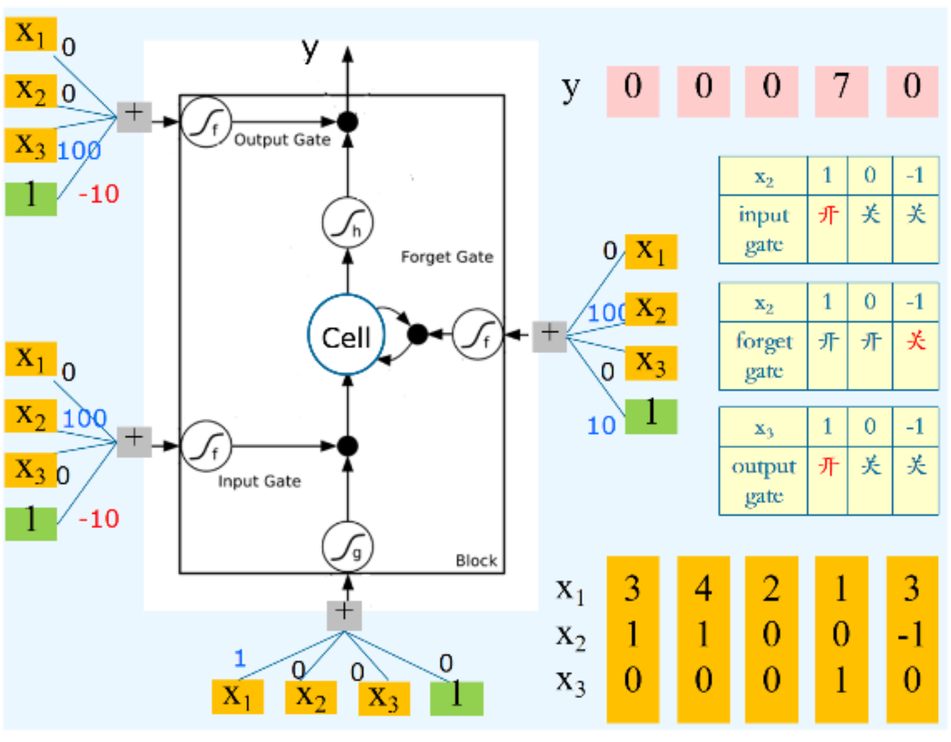
LSTM极简演示

摘自李宏毅博士的Deep Learning Tutorial

C	0	0	3	3	7	7	7	0	6
x_1	1	3	2	4	2	1	3	6	1
x_2	0	1	0	1	0	0	-1	1	0
x_3	0	0	0	0	0	1	0	0	1
y	0	0	0	0	0	7	0	0	6

x_2= 1时，在memory中累加上输入中的x_1的值
x_2=-1时，重置memory中的值
x_3= 1时，输出memory中的值

输入3个特征：X_1 X_2 X_3



y 0 0 0 7 0

x_2	1	0	-1
input gate	开	关	关

x_2	1	0	-1
forget gate	开	开	关

x_3	1	0	-1
output gate	开	关	关

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0

LSTM被简化为开关模式

开：数值≈1　　关：数值≈0

遗忘门开：保留旧记忆

遗忘门关：清空旧记忆

输出门开：输出记忆

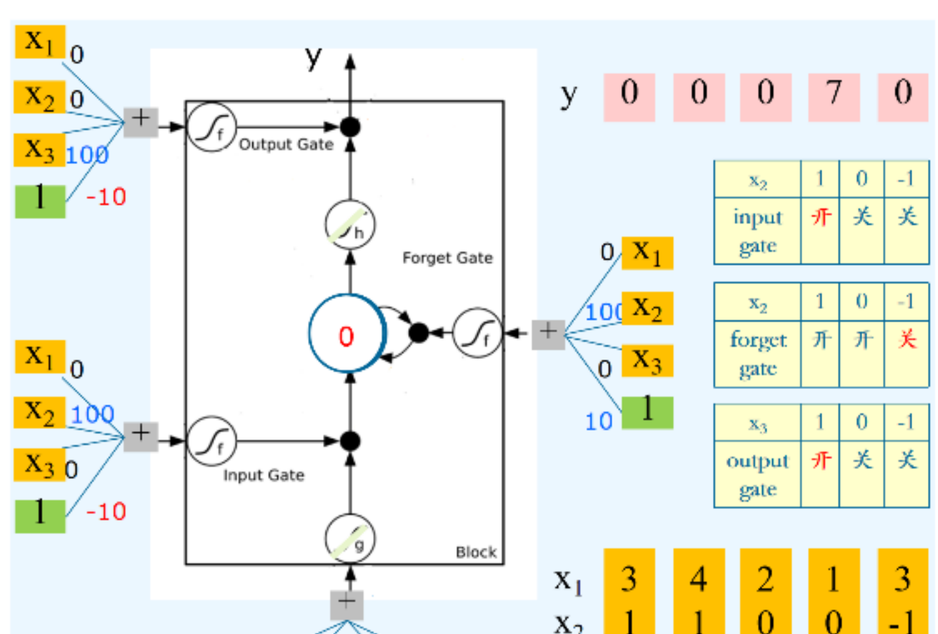
输出门关：不输出

即 $X_2=1$：保留旧记忆，写入新记忆

$X_2=0$：只保留旧记忆

$X_2=-1$：重置记忆（清空，不写入）

$X_3=1$：输出



y 0 0 0 7 0

x_2	1	0	-1
input gate	开	关	关

x_2	1	0	-1
forget gate	开	开	关

x_3	1	0	-1
output gate	开	关	关

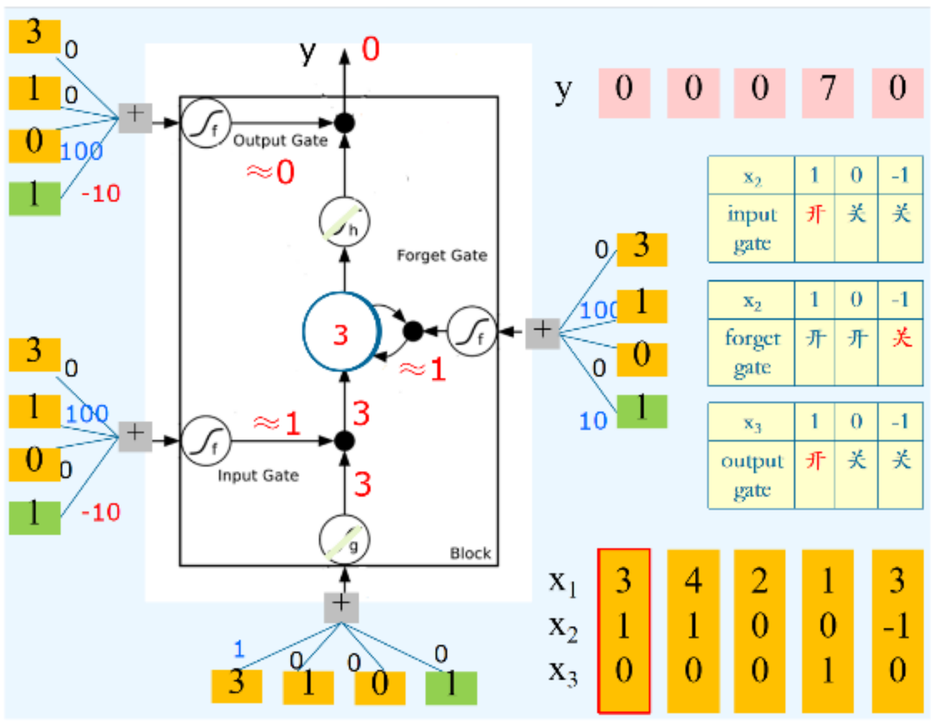
x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0

时间步1：$X=[1,0,0]$

门控：输入门关 遗忘门开 输出门关

Cell：$C_0=0$ —保留→ $C_1=0$

输出：不输出

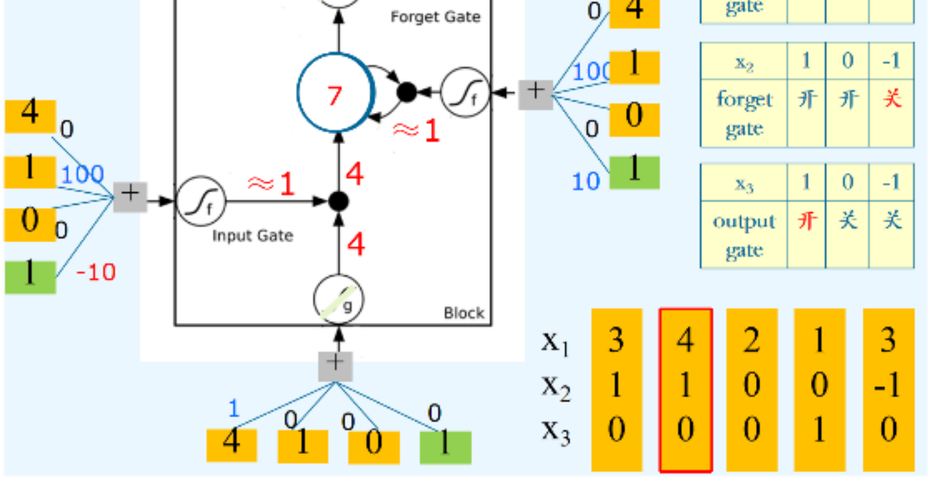


时间步2：$x=[3,1,0]$

门控：输入门开 遗忘门开 输出门关

Cell：$C_1=0 \xrightarrow{\text{写入}} C_2=C_1+x_1=3$

输出：不输出

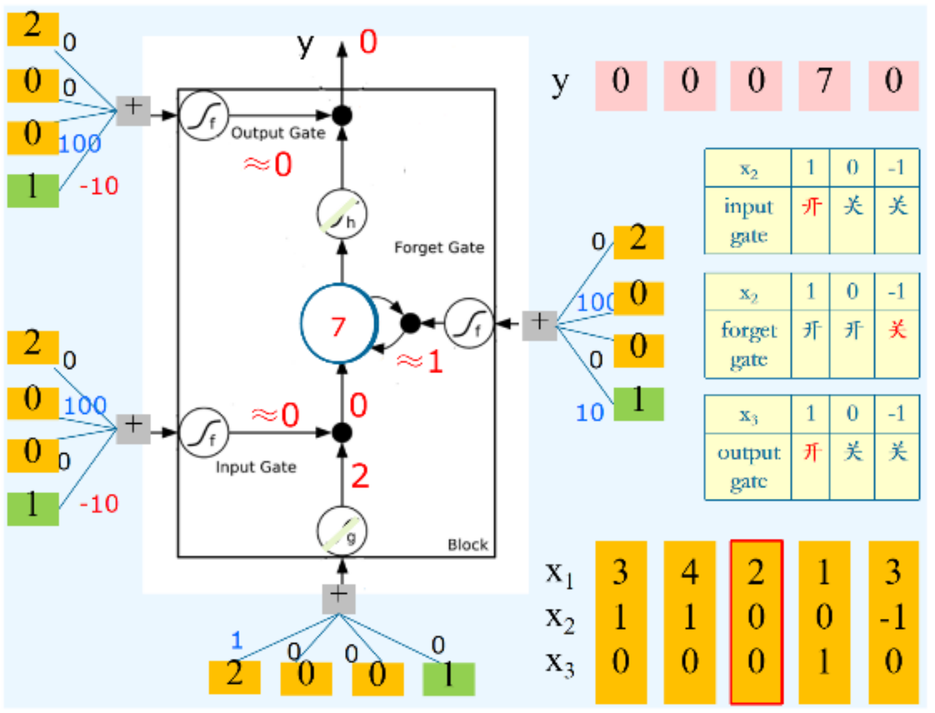


时间步3：$x=[4,1,0]$

门控：输入门开 遗忘门开 输出门关

Cell：$C_2=3 \xrightarrow{\text{写入}} C_3=C_2+x_1=7$

输出：不输出

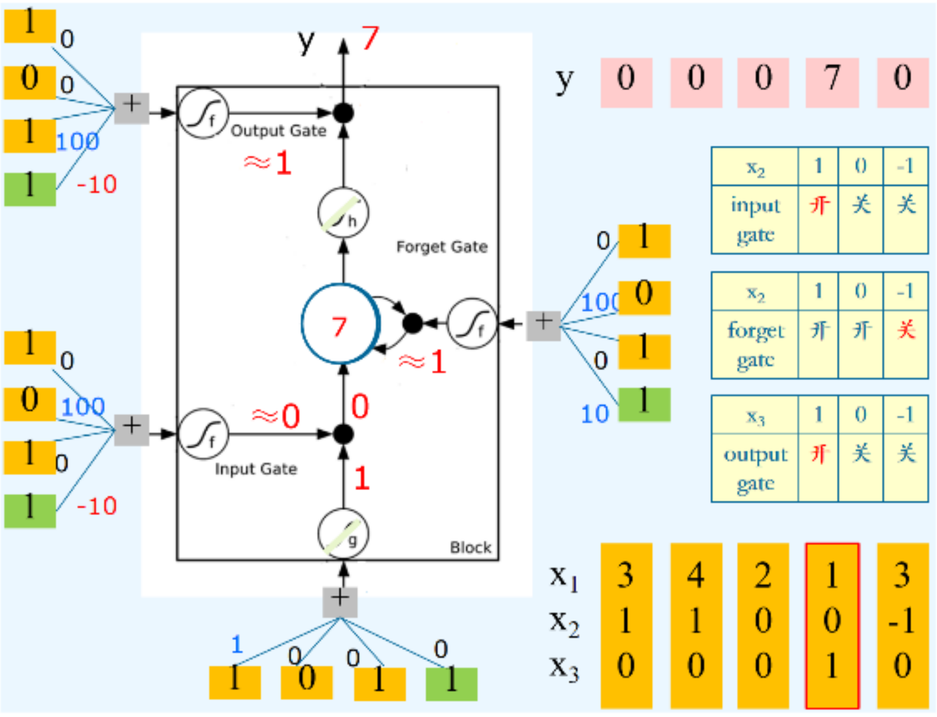


时间步4：$x=[2,0,0]$

门控：输入门关 遗忘门开 输出门关

Cell：$C_4=7 \xrightarrow{\text{保留}} C_4=7$

输出：不输出

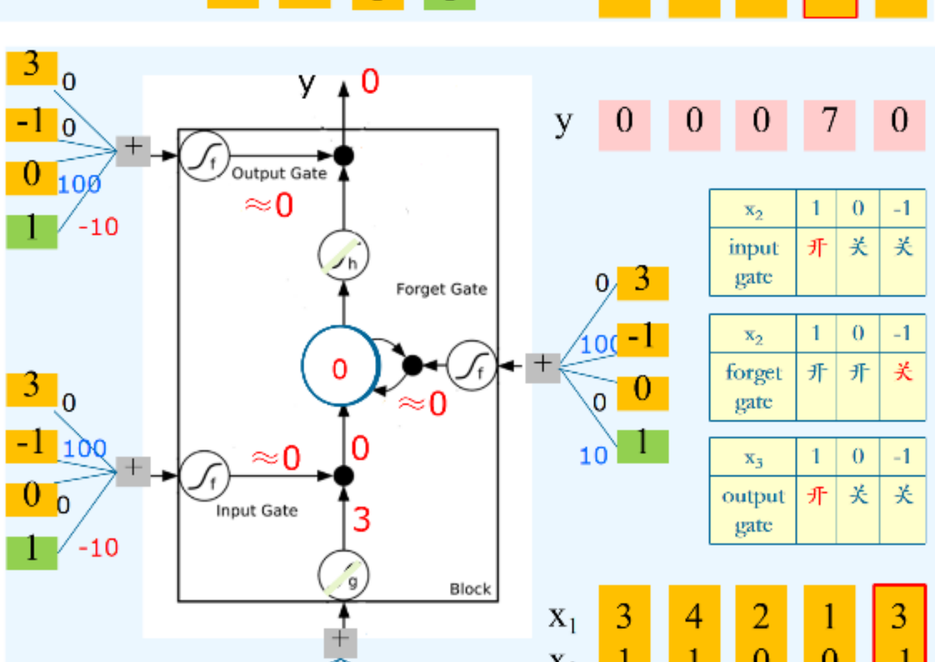


时间步5：$x=[1,0,1]$

门控：输入门关 遗忘门开 输出门开

Cell：$C_4=7 \xrightarrow{\text{保留}} C_5=7$

输出：7



时间步6：$x=[3,-1,0]$

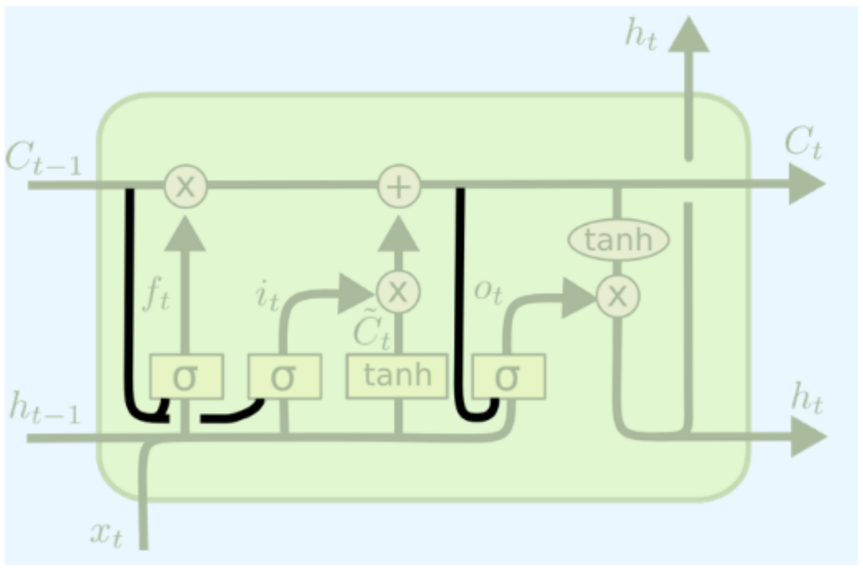
门控：输入门关 遗忘门关 输出门关

Cell：$C_5=7 \xrightarrow{\text{重置}} C_6=0$

输出：不输出

LSTM 变体

1、加入 Peephole Connection（窥视孔连接）



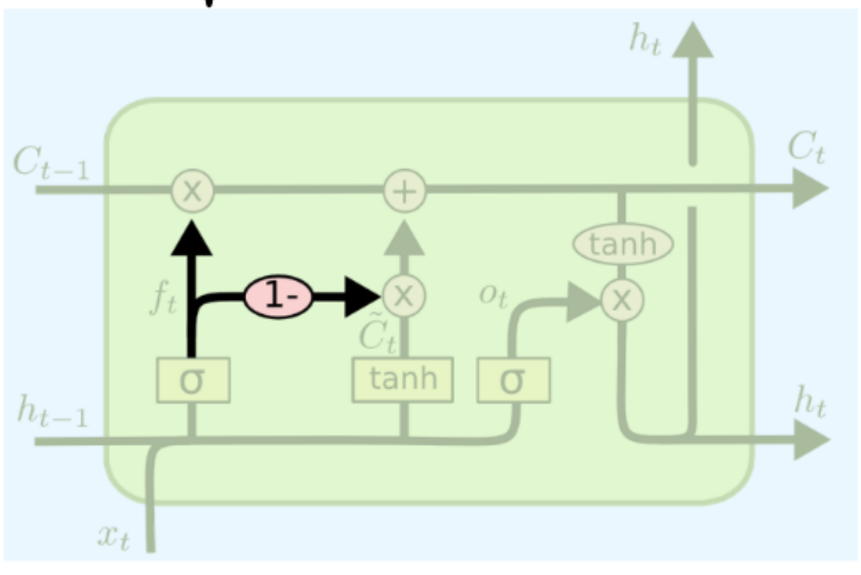
给三个门控加入能够直接读取细胞状态的通路

遗忘、输入门：读入 C_{t-1}

输出门：读入 C_t

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

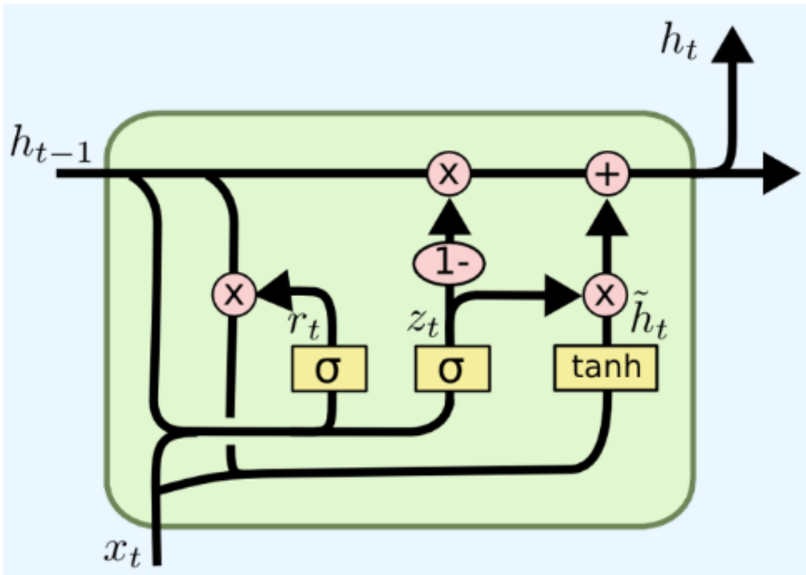
2、合并遗忘与输入为"更新门"



将遗忘与输入合并为"更新门"，只在添加新信息的位置遗忘旧信息（$f_t + 1 - f_t = 1$）

$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

综合以上改动，便得到了 GRU（门控循环单元）



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

重置门：第一个 sigmoid 层，用于控制"保留多少 h_{t-1}"，用 $r_t \times h_{t-1}$ 得到过滤后的历史状态，若 $r_t \approx 1$ 则说明 h_{t-1} 几乎被全部保留，$r_t \approx 0$ 说明全部遗忘。

更新门：第二个 sigmoid 层，用于耦合遗忘与写入。其输出 z_t，一方面经过变换得到 $(1 - z_t)$ 用于给 h_{t-1} 加权，控制保留多少旧信息；另一方面给候选新状态 $\tilde{h}_t$ 加权，控制写入多少新信息

候选隐藏状态：tanh层，输入 x_t 与重置门过滤后的旧状态 h_{t-1}，生成新候选状态 $\tilde{h}_t$

最终状态：$h_t = (1-z_t)h_{t-1} + z_t \tilde{h}_t$

重置门与更新门的核心区别

虽然它们都有"决定保留多少旧状态 h_{t-1}"的任务，但目标不同。重置门保留的 h_{t-1} 是为了参与新候选状态的运算，即"新信息要不要参考旧信息、要参考多少"，本质上是在处理新旧状态的短期关联。而更新门是处理旧状态 h_{t-1} 要不要保留到下一时刻，保留多少，本质上是在维护长期记忆。它们是相互独立的。

完全有可能出现"重置门将 h_{t-1} 全部舍弃，而更新门 h_{t-1} 全部保留"或反之的情况，但并不冲突

为什么 GRU 舍弃了细胞状态 C_t？

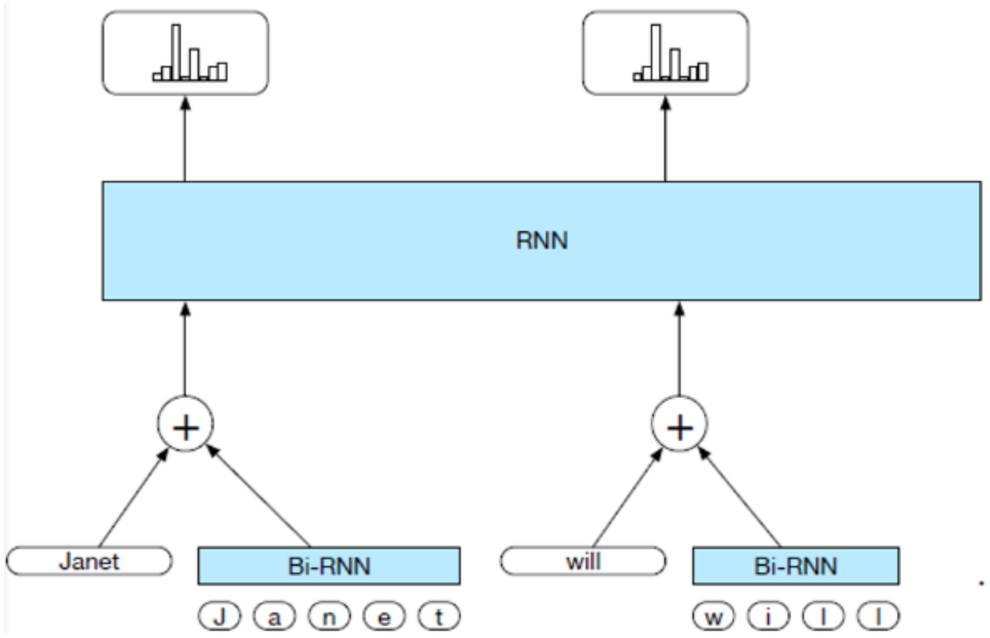
GRU 设计者认为 h_t 中本就包含 C_t 信息，于是又将记忆与输出合并为 h_t，但令 h_t 保持类似 C_t 的更新方式。舍弃 C_t 意味着 GRU 将面临长时记忆失效的问题，但GRU通过重置门实现了 $h_t = (1-z_t)h_{t-1} + z_t \tilde{h}_t$ 来传递长期记忆。虽然比不上RNN的 C_t 记忆完善，但效率显著上升，开销也显著下降

表格

维度	LSTM (Long Short-Term Memory)	GRU (Gated Recurrent Unit)
结构复杂度	**高** (3 个门：遗忘、输入、输出)	**低** (2 个门：更新、重置)
状态分离	**强分离** (C_t 专存记忆，h_t 专输出)	**融合** (h_t 既存记忆又输出)
核心逻辑	门控精细调节信息流，路径清晰	用更新门 z_t 直接平衡旧状态与新状态
性能代价	参数多，训练慢，容易过拟合	参数少 (约少 1/3)，训练快，更轻量
适用场景	数据量大、序列极长、追求极致精度（如机器翻译）	数据有限、实时性要求高、轻量级部署（如文本分类、推荐）

LSTM应用

1、字符级词嵌入



… 输出

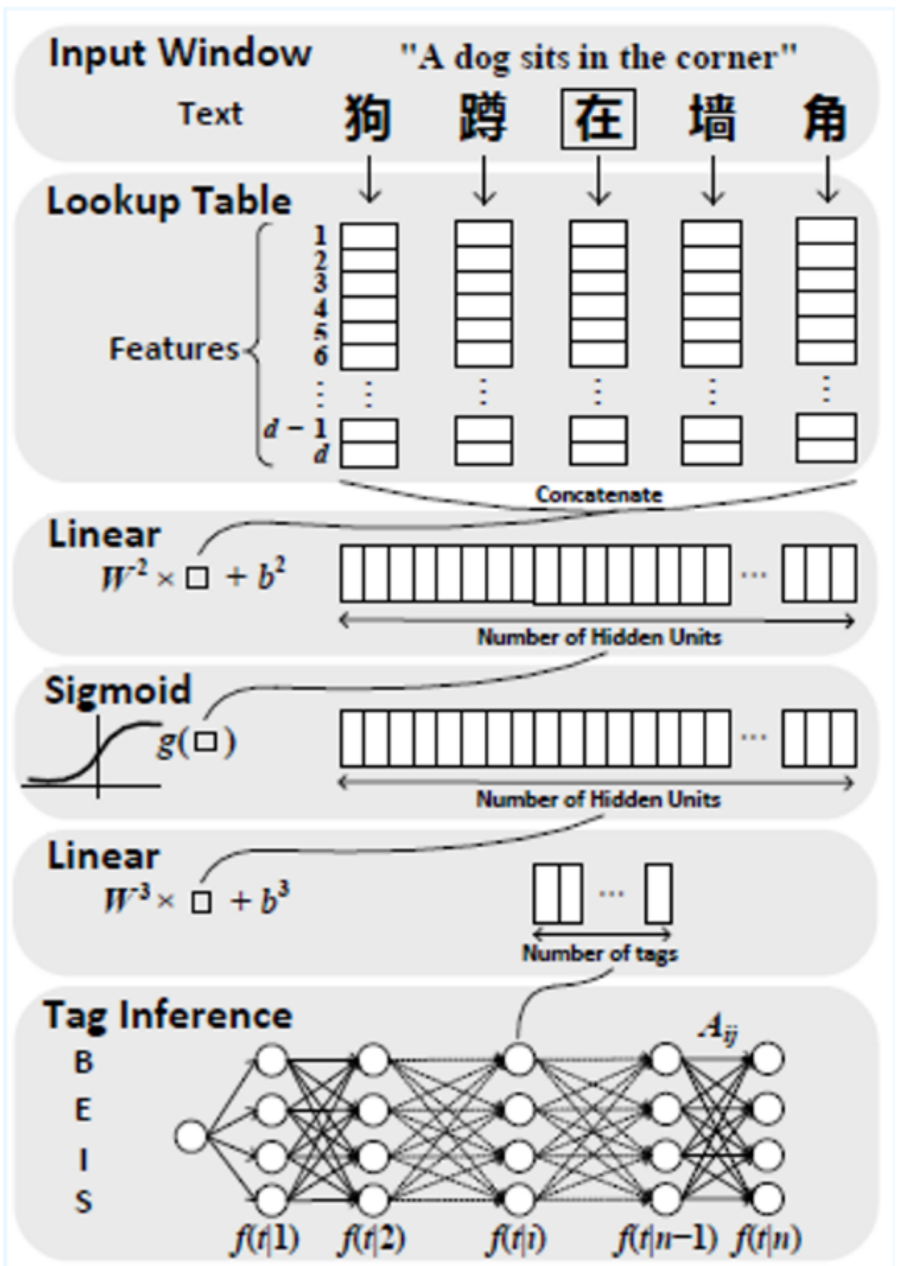
LSTM

特征提取层

输入层

特征提取层：每个单词的字符序列输入一个双向RNN（Bi-RNN），提取字符级特征（前后缀等），并将词本身的嵌入和Bi-RNN输出的融合，输入LSTM

2、序列标注



第零层：输入窗口层

第一层：Embedding层（嵌入）

第二层：隐藏层

第三层：神经网络输出层

第四层：Tag Inference层（解码）

1、Embedding层

提取每个字的特征，输入每个字的编号，通过查询（lookup）操作将编号转化为该字对应的特征向量，生成特征矩阵 $M \in R^{d \times |D|}$（d：向量空间维度，|D|：长度固定的字表）

本层节点数：wd （w：窗口长度）

将所有向量拼接起来送入隐藏层

2、隐藏层

拼接后的高维向量经过一个全连接层W^2和一个sigmoid函数.

实现特征交叉与降维

本层节点数：H（隐藏层神经元数量）

系数矩阵 $W^2 \in R^{H \times (wd)}$

3、输出层

给定标记集合T，为句子$C_{[1\cdots n]}$中每个位置上的数字Ci输出一个长度为|T|的向量，用来表示Ci对于T中每个标记的分值.

节点数：|T|

系数矩阵：$W^3 \in R^{|T| \times H}$

4、解码层

通过解码算法（图中为viterbi算法）输出最终的标记序列

$f(t|i)$：Ci对应标记t的分值

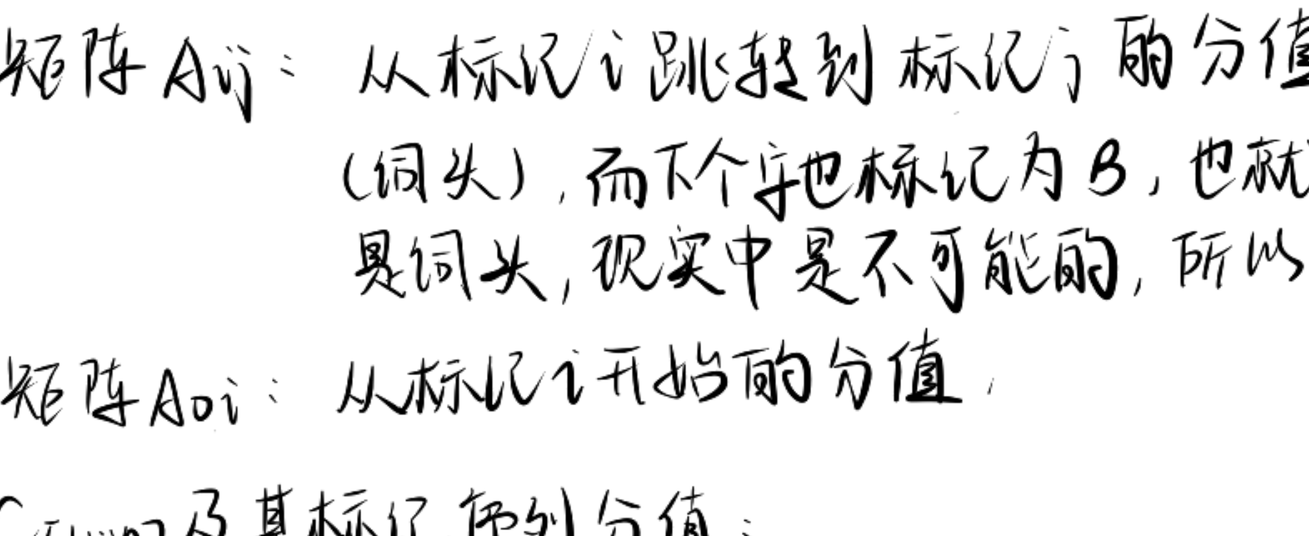
转移矩阵Aij：从标记i跳转到标记j的分值，如一个字标记为B（词头），而下个字也标记为B，也就是说两个相邻的字都是词头，现实中是不可能的，所以对应的ABB分值很低

初始矩阵Aoi：从标记i开始的分值.

句子$C_{[1\cdots n]}$及其标记序列分值：

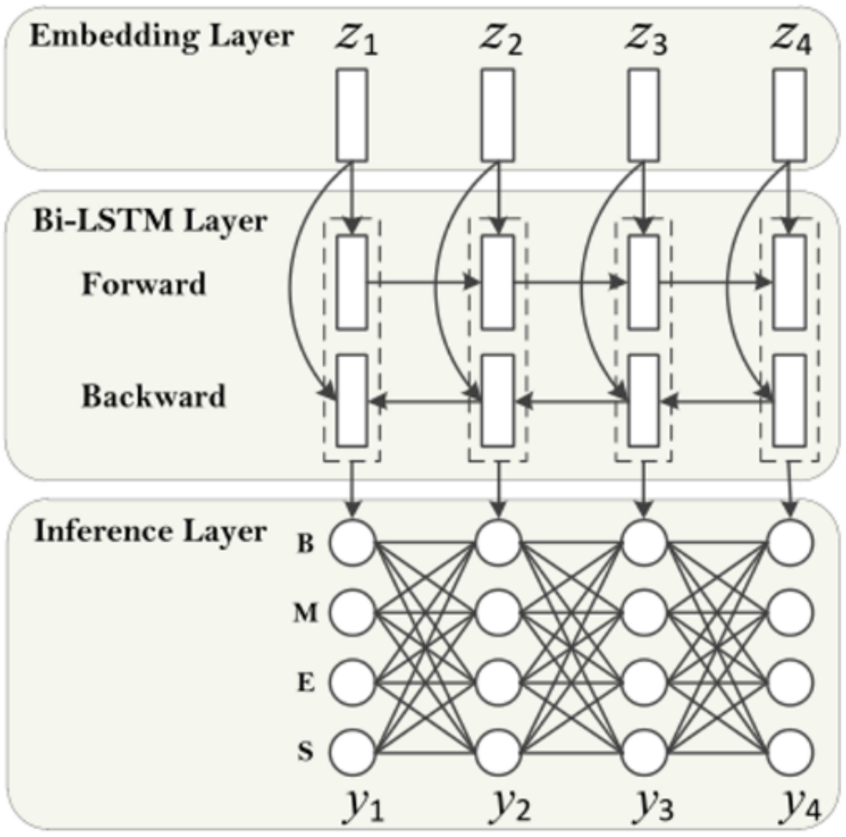
$$S(C_{[1\cdots n]}, t_{[1\cdots n]}, \theta) = \sum_{i=1}^{n} [A_{t_{i-1}t_i} + f(t_i|i)]$$

最佳标记路径：$t^*_{[1\cdots n]} = \arg\max_{\forall t'_{[1\cdots n]}} S(C_{[1\cdots n]}, t_{[1\cdots n]}, \theta)$



3、更先进的序列标注

用双向LSTM替代窗口+前馈网络，用Uni-gram和Bi-gram嵌入增强输入特征.



输入：序列 $X = \{x_1, \cdots, x_n\}$
x_i 的 i 代表其在序列中的位置

嵌入：两种嵌入方式：

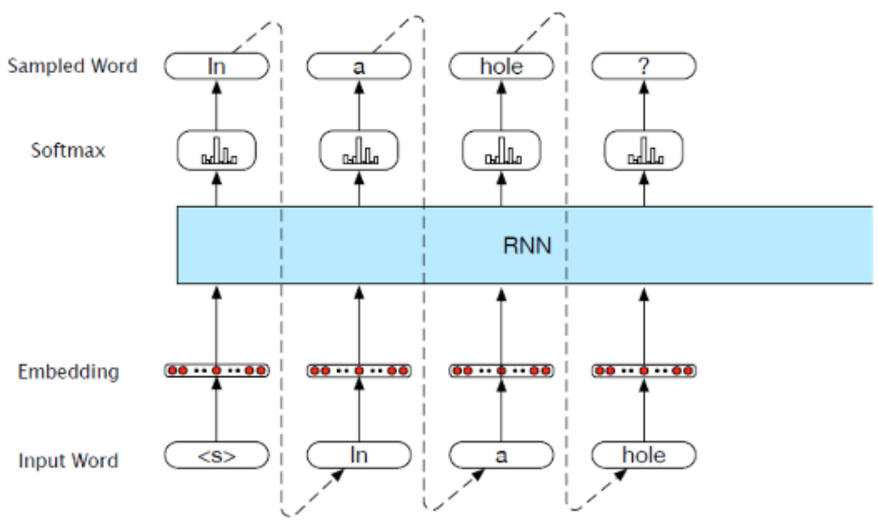
- Uni-gram：$z_i = e_{x_i}$，直接查嵌入表
  将 x_i 映射为 z_i
- Bi-gram：
  $z_i = e_{x_i} \oplus e_{(x_{i-1}, x_i)} \oplus e_{(x_{i+1}, x_i)}$
  $\oplus$ 表示拼接
  这里指嵌入时融入前后字的信息

Bi-LSTM 层：用双向传播，提取"前文"和"后文"的特征，最终每个位置的输出为正/反向 LSTM 隐藏状态的拼接
$h_i = \overrightarrow{h_i} \oplus \overleftarrow{h_i}$

Inference 层：类似 Tag Inference

编码序列模型

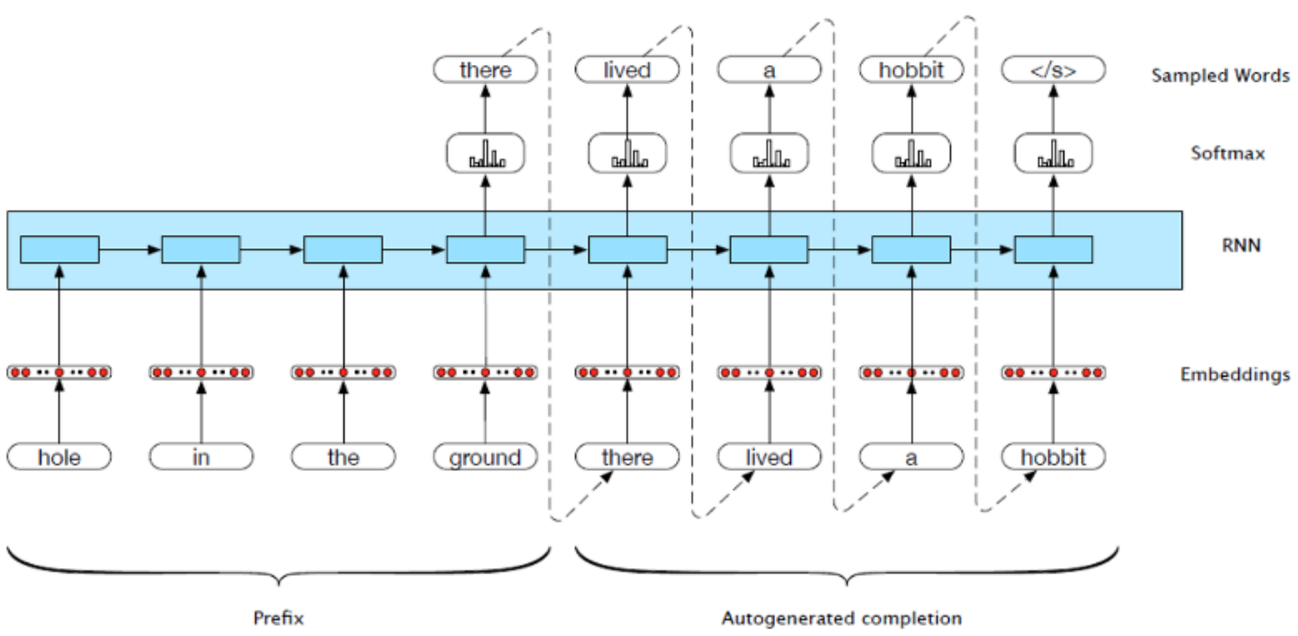
在本章 RNN 部分我们提到过基于 RNN 的语言模型



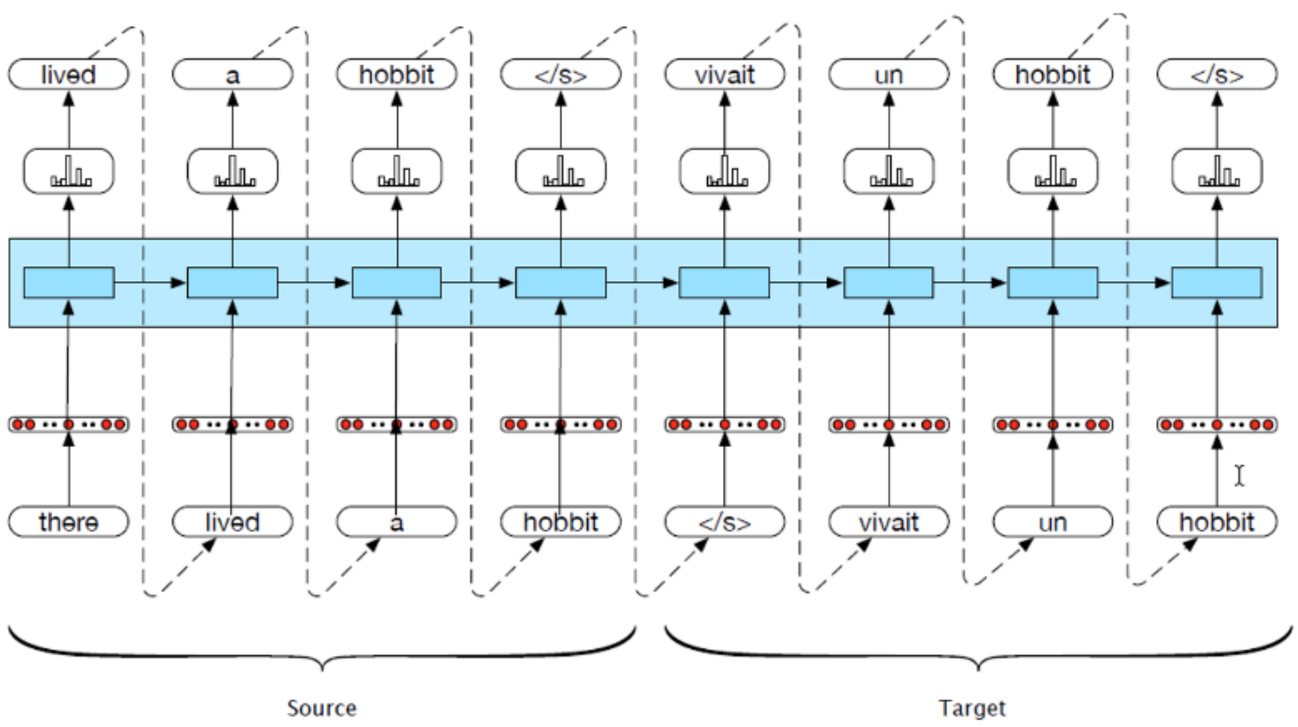
$$P(w_n | w_1^{n-1}) = y_n$$
$$= \text{Softmax}(Vh_n)$$

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

基于这个模型，我们可以做一种任务：预输入一段话，机器自动生成后续

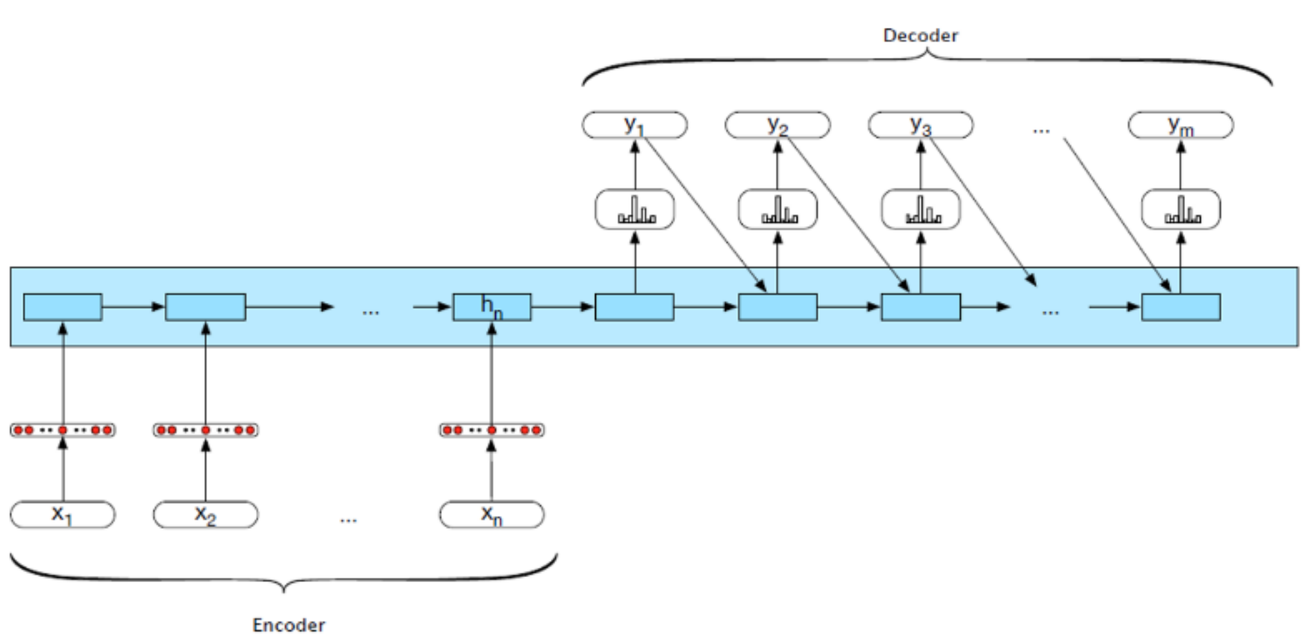


更深入一层，我们输入一段源文本，让机器输出一段我们想要的目标文本，即"翻译"



我们把翻译任务抽象为：输入一段序列，输出目标序列，即Seq2Seq问题(Sequence to sequence)，这个问题难点在于两个序列长短不一定相同，而之前我们接触的算法都要求输入输出长度一致

- Encoder-Decoder(seq2seq)基础架构

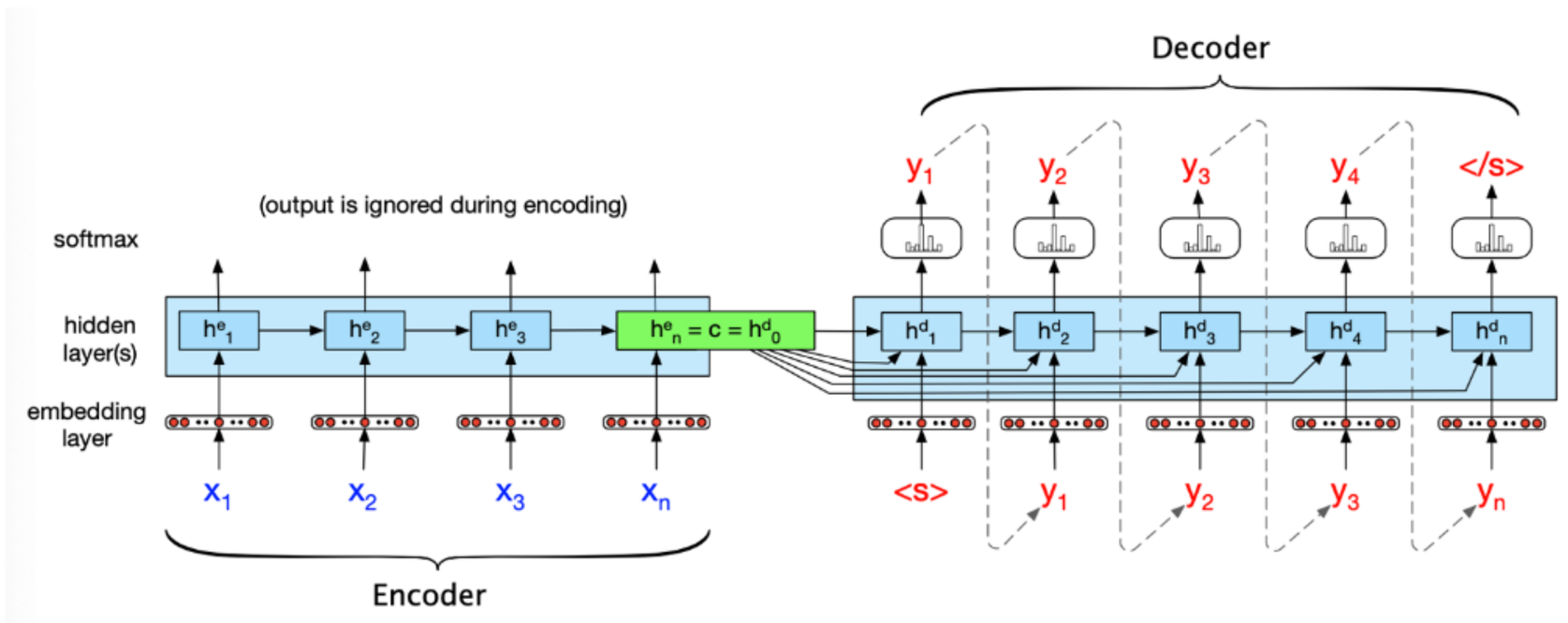


这是Seq2Seq模型物理结构，由两个RNN组成：

1. Encoder(编码器)：将源语言序列 $x_1, \cdots, x_n$ 压缩为一个上下文向量 $C = h_n$(最后一个时刻隐藏状态)

2. Decoder(解码器)：以C为初始状态，自回归生成目标语言序列 $y_1, \cdots, y_m$

$$\text{公式}\begin{cases} h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d) \\ z_t = f(h_t^d) \\ y_t = softmax(z_t) \end{cases}$$



Encoder-Decoder架构为什么能处理变长输入输出？

Encoder：把任意长度的序列压缩成固定维向量

Encoder的隐藏状态 h_i^e 维度是设定好的（如512维）

最终输出 $C = h_n^e$ 包含了整个输入的语义信息

Decoder：将固定维向量还原为变长输出。

Decoder的生成过程是自回归的，即将自己过去的输出当作当前输入的一部分，因此模型可以自行决定什么时候生成序列结束符 </s>，也即是变长输出。

问题：C只输入Decoder第一时刻，后续每一步只依赖上一个生成词 $\hat{y}_{t-1}$ 与状态 h_t^d，源序列信息会逐渐稀释

优化：法一、将C加入Decoder状态更新

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, C)$$

法二、将C加入softmax层

$$y_t = \text{softmax}(\hat{y}_{t-1}, z_t, C)$$

应用：机器翻译（A语言－B语言）　自动摘要（长文本-短文本）

对话生成（问句－答句）　代码生成（自然语言－代码）

如：机器翻译

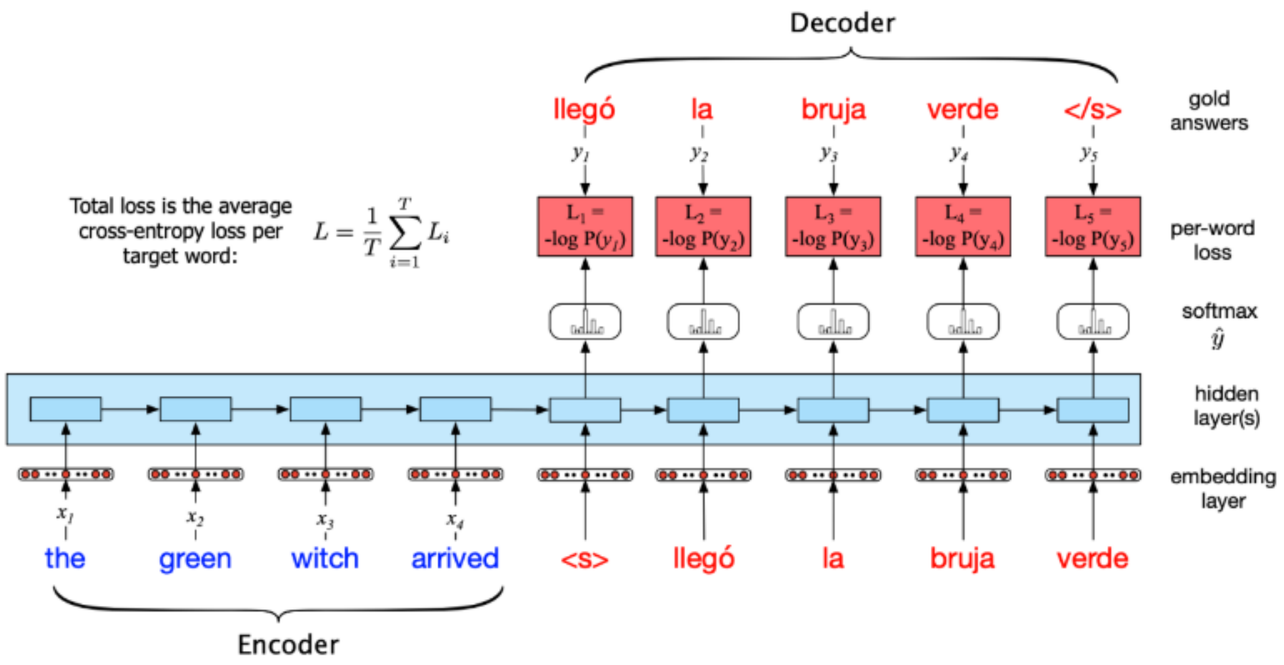
$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)\ldots P(y_m|y_1,\ldots,y_{m-1},x)$$

把句子x翻译为句子y的概率拆解为自回归的条件概率乘积

（对应RNN中 y_t 依赖源序列x和前t-1个词）

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})$$

每一步词表V中选概率最大的词作为该步的输出



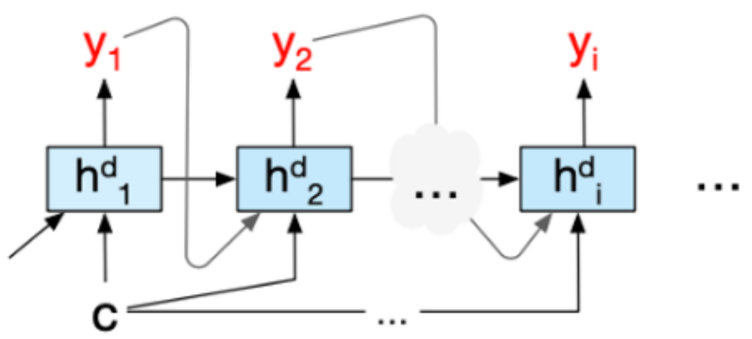
注意力机制 (Attention)

为了使Decoder长期记忆源序列，我们给译码每一时刻增加 context输入

i时刻context输入与i时刻偏码端输入有关，这意味着：

- 不再用单一的 $h_n^e = c = h_0^d$，而是让Decoder时刻都可以回看Encoder的所有时刻的隐藏状态
- 每一步的上下文 C_i 会根据Decoder当前状态动态加权，即根据其当前状态给Encoder不同位置的状态分配不同权重，生成"当前步需要的专属上下文"

即： $h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$ C_i是当前步的动态上下文



注意力分数计算：计算Decoder当前状态与Encoder每个状态的相似度

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

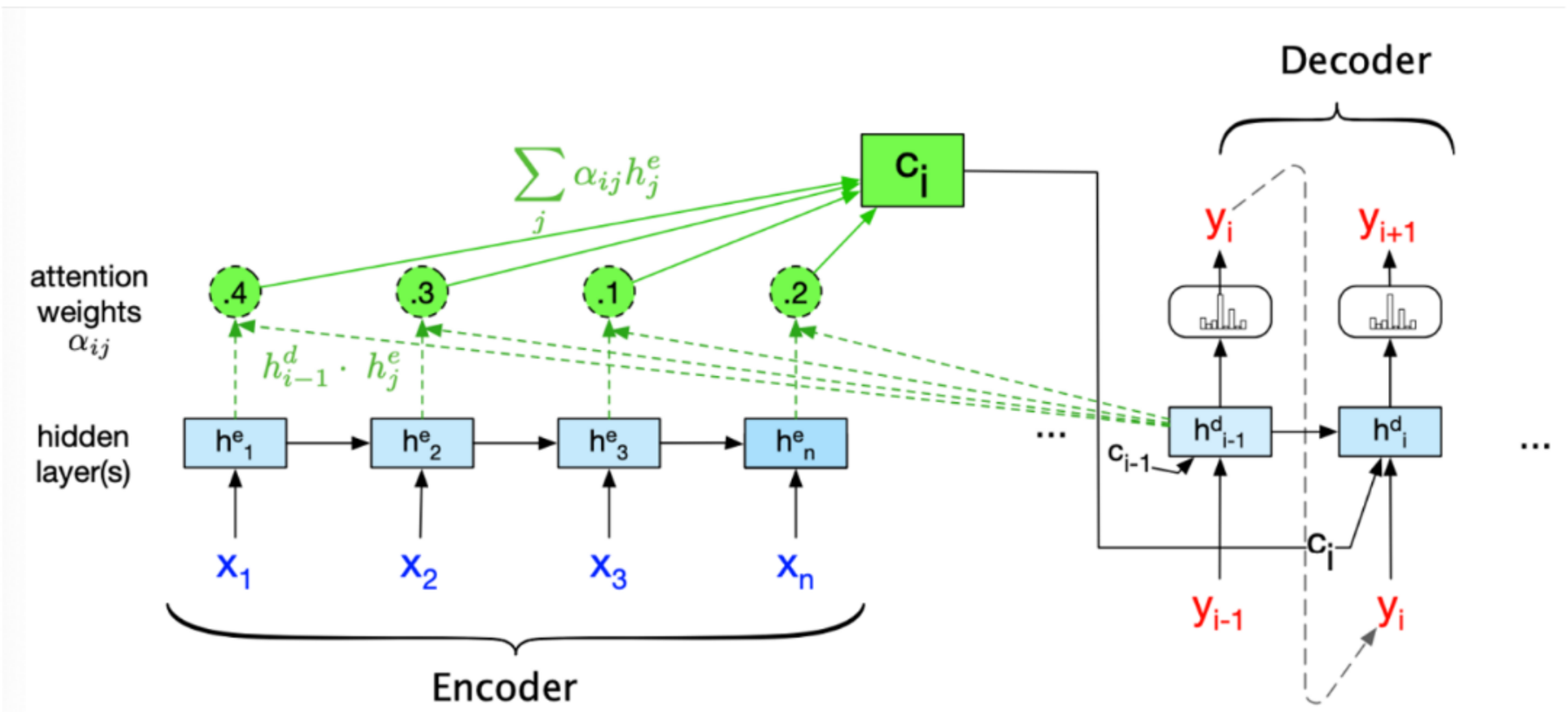
对所有score做softmax，得到注意力权重 α_{ij}

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

$$= \frac{exp(score(h_{i-1}^d, h_j^e)}{\sum_k exp(score(h_{i-1}^d, h_k^e))}$$

再对 h_j^e 加权求和得 c_i

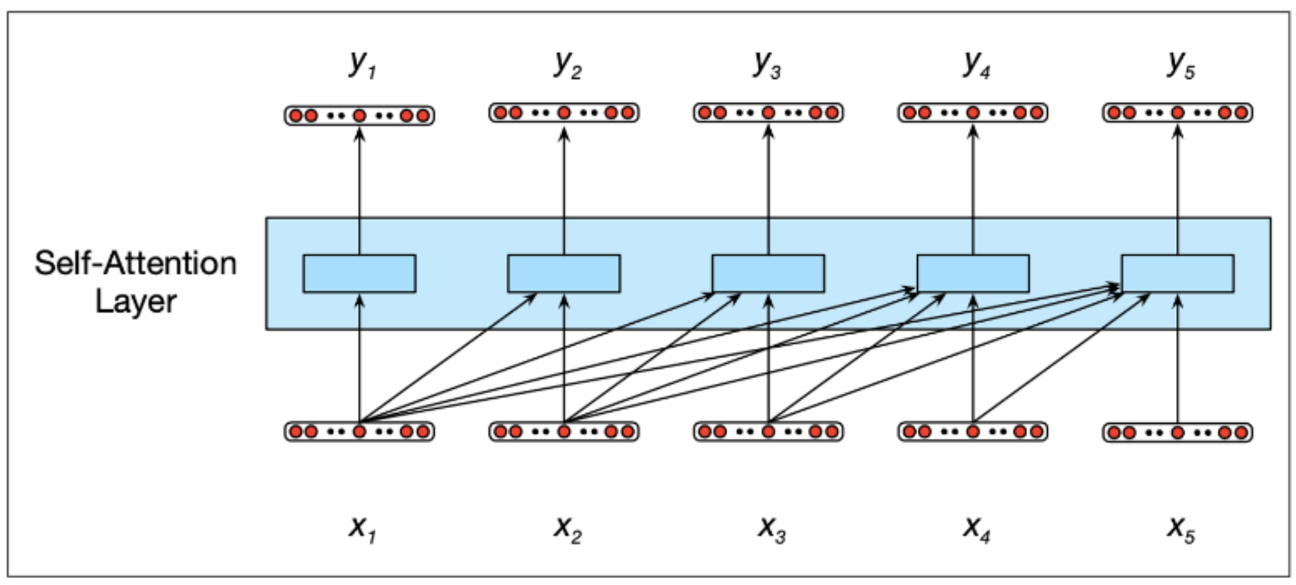
$$c_i = \sum_j \alpha_{ij} h_j^e$$



自注意力机制（Self-Attention）

在同一序列内部，让每个位置都能与其他位置计算相似度、加权求和，实现"序列内部的全局上下文融合"

Encoder和Decoder不再用两个RNN网络，而是复用一个自注意力层。



Encoder层内部

全局交互：图中做了简化，实际上所有 x_i 都会流向所有位置。

每个位置输出都包含全局上下文信息。

每个 y_i，都是所有输入 x_j 的加权求和。

$$score(x_i, x_j) = x_i \cdot x_j$$

$$\alpha_{ij} = \text{softmax}(score(x_i, x_j)) \ \forall j \le i$$
$$= \frac{\exp(score(x_i, x_j))}{\sum_{k=1}^{i} \exp(score(x_i, x_k))} \ \forall j \le i$$

$$y_i = \sum_{j \le i} \alpha_{ij} x_j$$

对比维度	普通注意力（Cross-Attention）	自注意力（Self-Attention）
参与序列	**两个独立序列**（源序列 + 目标序列）	**同一个序列**（输入序列自己）
核心目标	序列间对齐（比如翻译的词对齐）	序列内全局上下文融合
Q/K/V 来源	Q 来自目标序列，K/V 来自源序列	Q/K/V 全部来自同一个输入序列
输入输出长度	输入两个长度不等的序列，输出长度等于目标序列	输入一个序列，输出长度和输入完全相同
经典应用	机器翻译、Seq2Seq 任务	Transformer Encoder、BERT、GPT、文本分类
信息流动	单向（源序列 → 目标序列）	双向（序列内所有位置互相连接）
解决的问题	基础 Seq2Seq 长句信息丢失	RNN 长距离依赖、串行计算效率低

Transformer

一种基于Encoder-Decoder结构模型

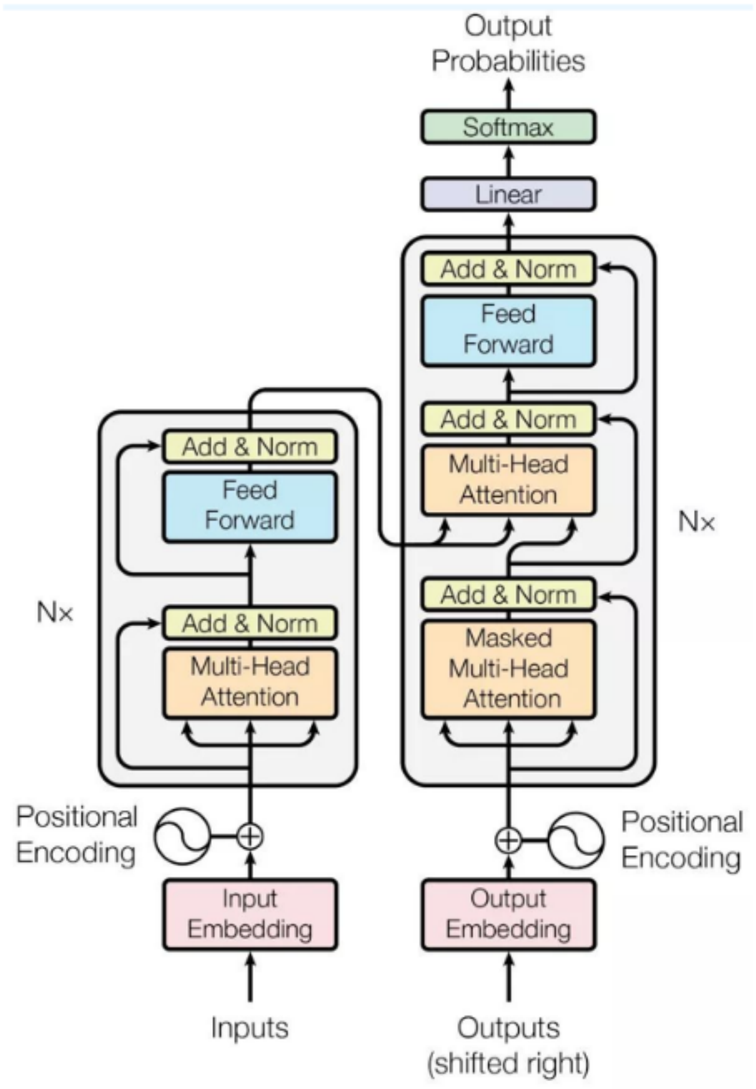


Figure 1: The Transformer - model architecture.

Encoder：

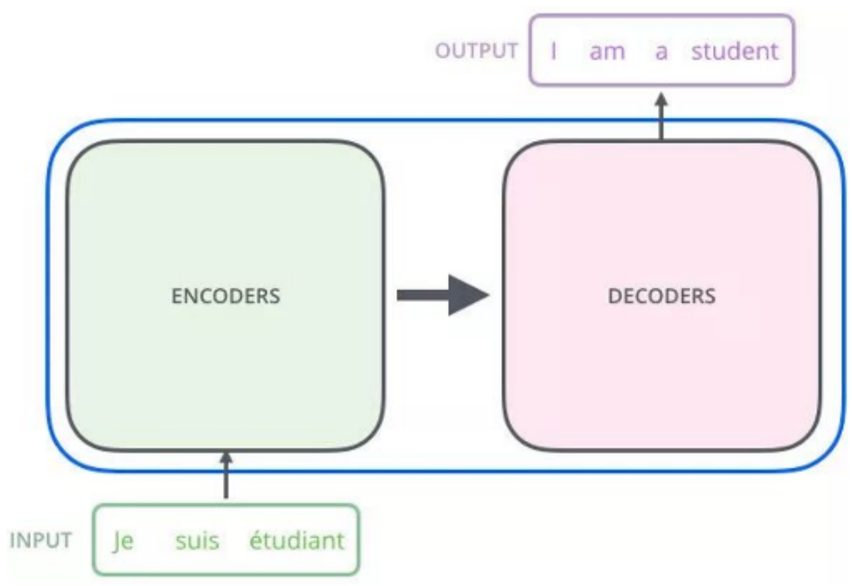
1.Input经过embedding后，要做positional encodings
2.然后是Multi-head attention，
3.再经过position-wise Feed Forward，
4.每个子层之间有残差连接。

Decoder：

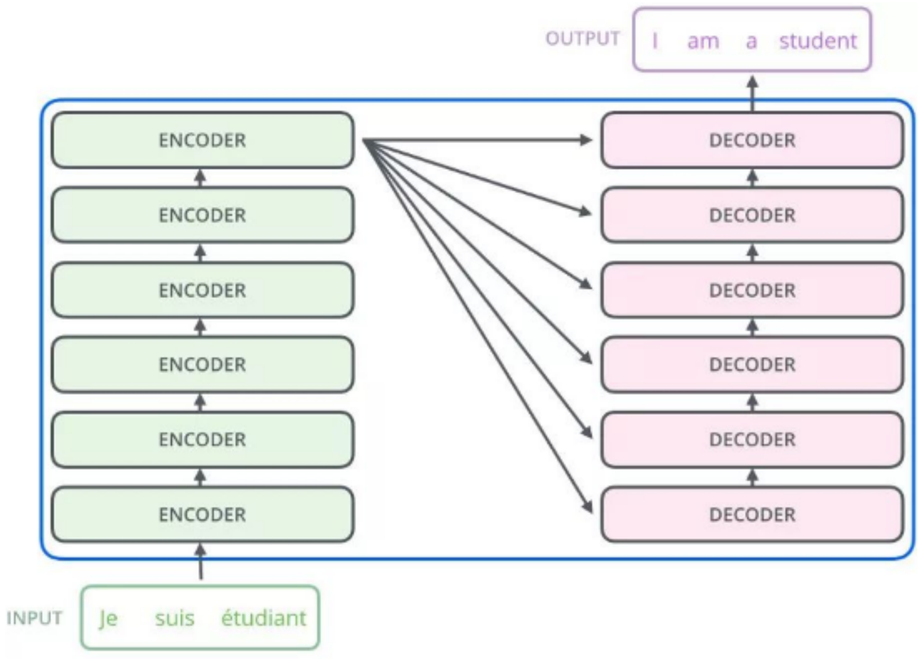
1.如图所示，也有positional encodings，Multi-head attention和FFN，子层之间也要做残差连接，
2.但比encoder多了一个Masked Multi-head attention，
3.最后要经过Linear和softmax输出概率。

Encoder与Decoder都由N层Self-Attention组成
但Decoder比Encoder多一层Multi-Head Attention
(掩码多头注意力层)

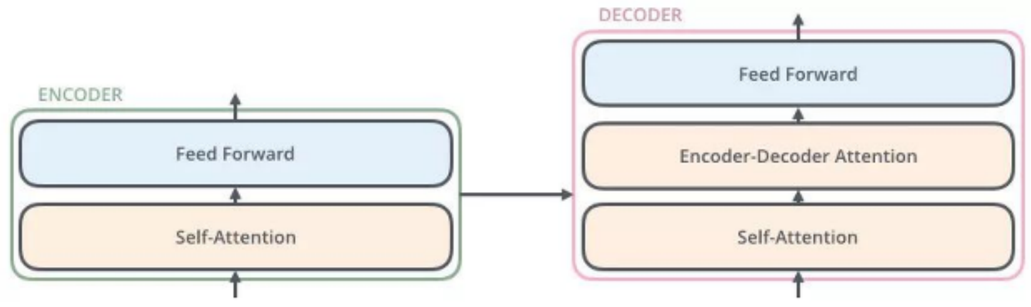
结构简化：



拆解



拆解

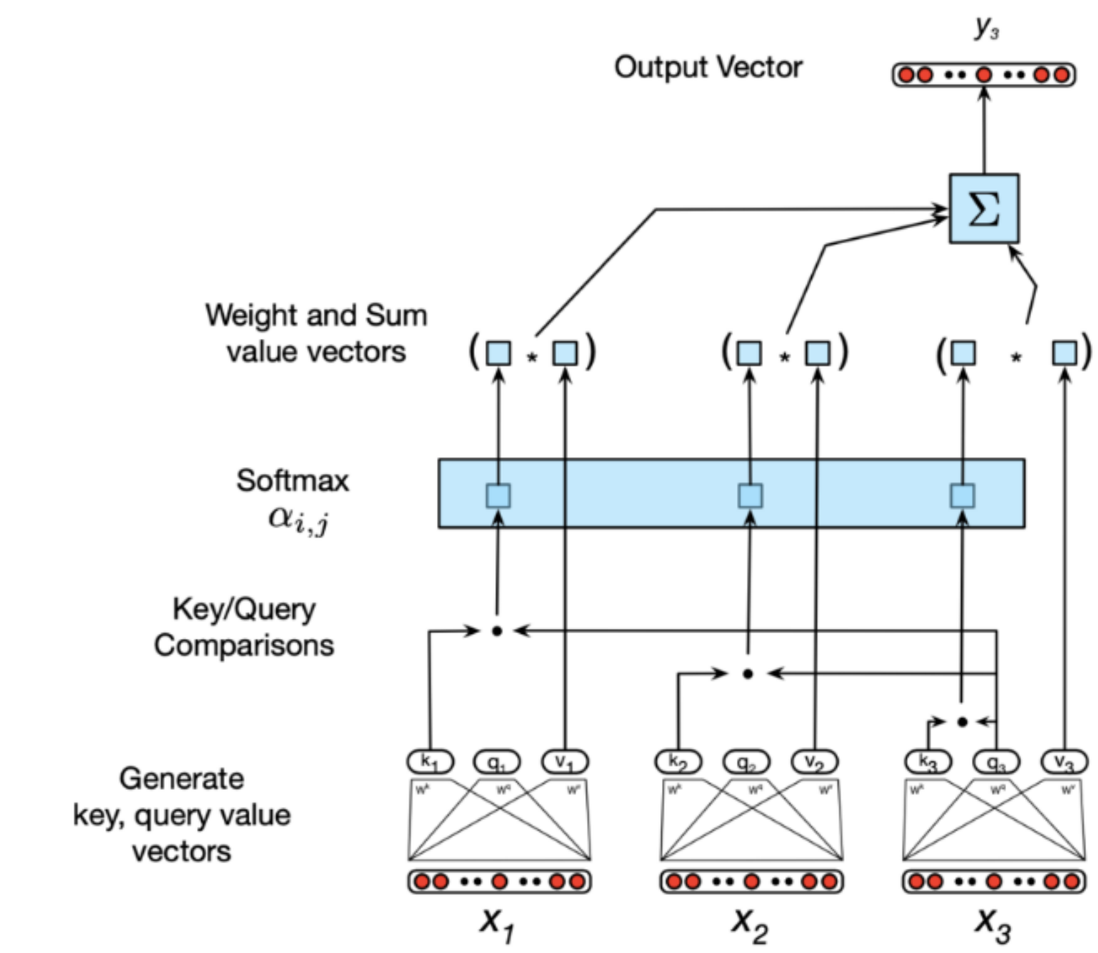


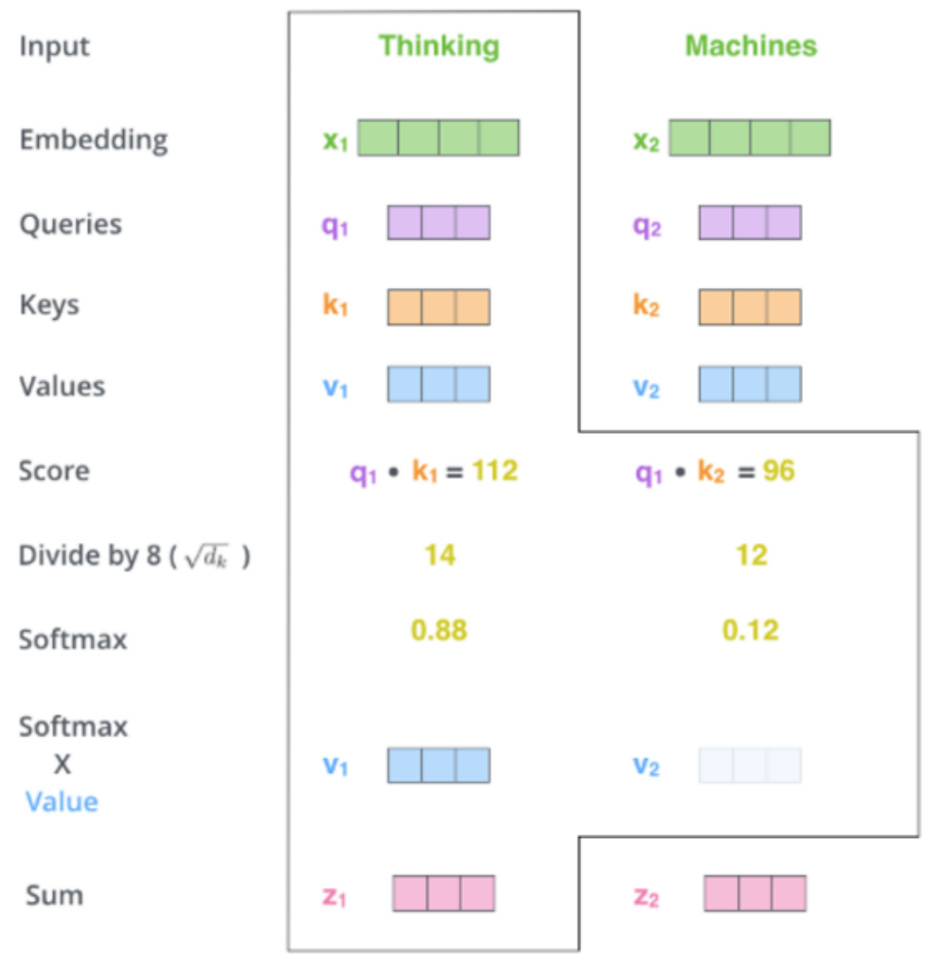
自注意力（Q/K/V）的完整计算流程

核心：$score(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}}$

q_i：第 i 个词的 Query（查询）向量　k_j：第 j 个词的 Key（键）向量

d_k：Q/K 向量的维度





流程拆解：

最底层：每个输入词 x_i 通过三个独立权重矩阵 W^Q W^K W^V 生成自己的 q_i、k_i、v_i

$$q_i = W^Q x_i \qquad k_i = W^K x_i \qquad v_i = W^V x_i$$

第二层：Q-K 匹配，计算 $score(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}}$

第三层：对每个 q_i 对应的所有 score，做 Softmax 归一化，把分数转换成权重 a_{ij}，代表 x_i 对 x_j 的注意力占比

$\sum_j a_{ij} = 1$

第四层：对所有 v_j 作加权求和：$y_i = \sum_j a_{ij} v_j$，即把所有词的信息按注意力权重揉进当前词

Q：查询，我要关注谁，用来发起查询，与K做匹配。

K：键，我是谁，提供匹配标签，被Q匹配

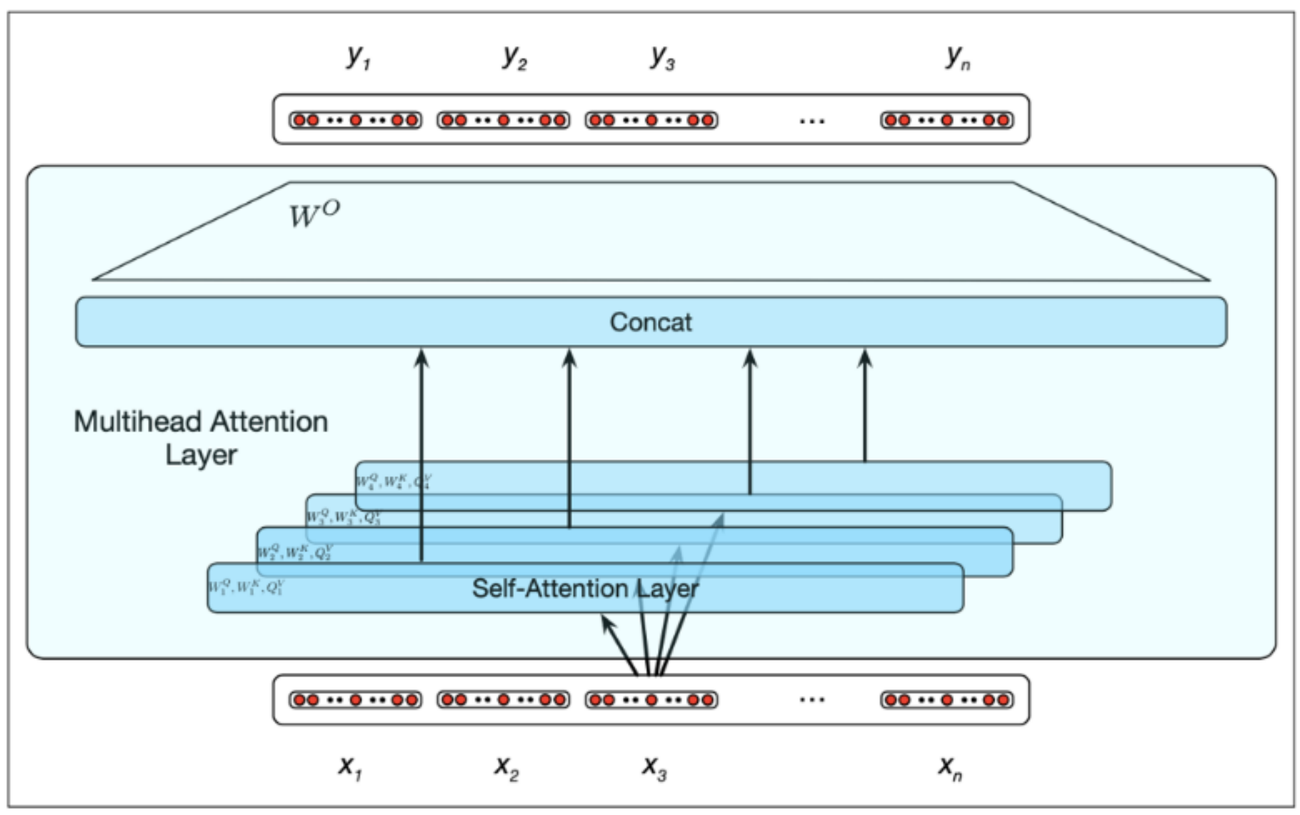
V：值，我有什么信息，用来被加权融合，提供原始语义信息

W^Q、W^K、W^V都是模型需要训练的参数。

Multi-Head Attention（多头注意力）

Self-Attention的升级版，将Self-Attention过程同时做h次，最后将每次的输出$Z_1 \cdots Z_h$合起来得到一个更长的向量Z。

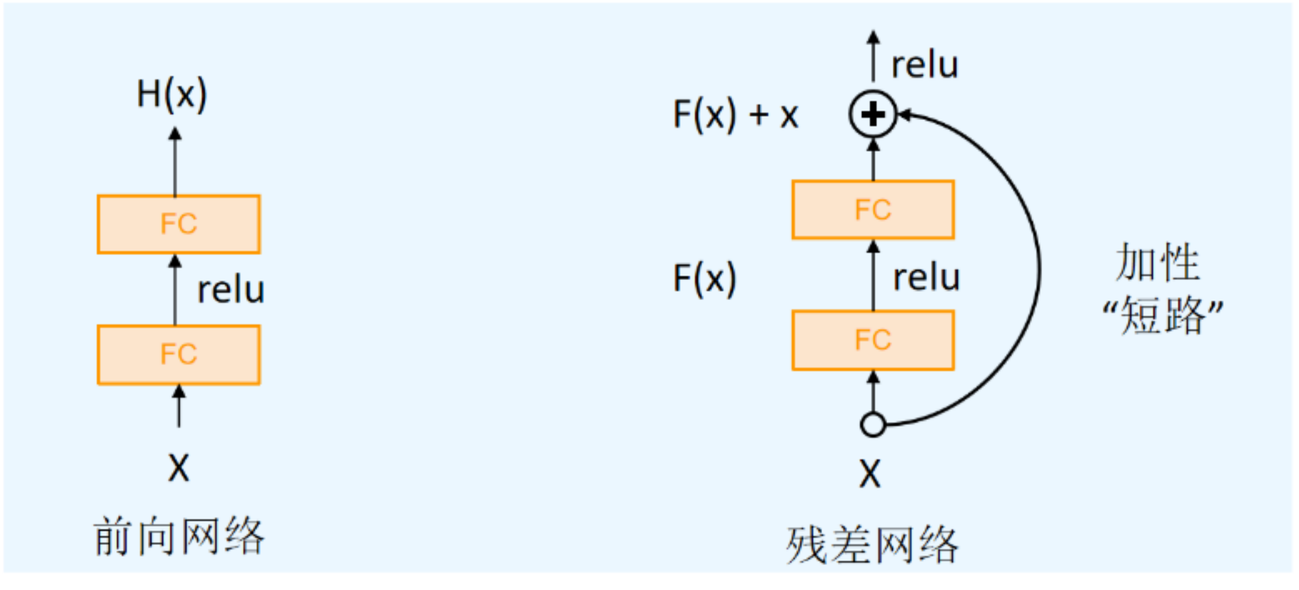
再将用一个输出权重矩阵W^O对Z作线性变换得到输出$y_1 \cdots y_n$。



为什么要多头注意力？因为单头只能有一种注意力模式，无法同时捕捉多种关系。比如在一个句中，一个词可能需要关注主、谓、宾、定、状、补、语法、语义、长距离依赖等多种关系。

残差结构

深度网络的问题之一便是后向传播时梯度逐渐消失，而残差结构可以解决这个问题，这是Transformer能堆几十层的基础。



残差网络：输入x分两路，主路正常运算（学习残差），短路不对x作任何运算，最终将F(x)+x送入ReLU后输出

$H(x) = F(x) - x$

让网络学习残差 $F(x) = H(x) - x$，而非直接学完整映射

优点：$\frac{\partial loss}{\partial x} = \frac{\partial loss}{\partial y} \cdot \left(\frac{\partial F}{\partial x} + 1\right)$

梯度可沿短路直接回传，即就算 $\frac{\partial F}{\partial x} \approx 0$

梯度也可以通过"+1"这条路传下去

层归一化（Layer Nomalization，LN）

深层网络训练还有两个问题：

1、内部协同量偏移：网络每一层分布都随着前层参数更新不断变化，如第一层参数更新，第二层输入数据就变了，第二层再更新，第三层又变了，这导致模型学习效率极低，训练极不稳定

2、激活函数饱和区：对于sigmoid、Tanh这类函数，输入数值过大/过小时，函数梯度接近0

LN技术可解决上述问题，是Transformer稳定训练、加速收敛的关键

公式：$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$

$x^{(k)}$：第k个样本原始特征向量

$E[x^{(k)}]$：对$x^{(k)}$所有特征维度求均值，可记为u

$Var[x^{(k)}]$：对$x^{(k)}$所有特征维度求方差，可记为σ^2

$\hat{x}^{(k)}$：归一化后的$x^{(k)}$

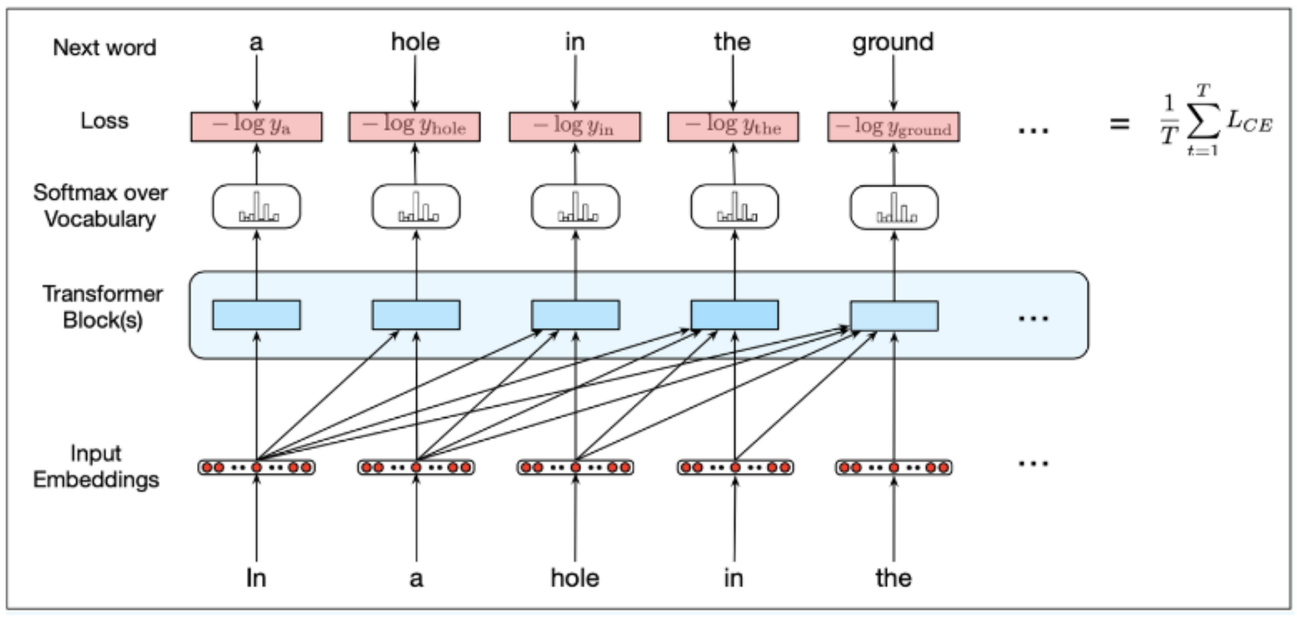
原理：让每一层的输入强制拉回标准正态分布，不论参数如何更新，输入都只服从标准正态分布，极端数据出现概率

大幅下降

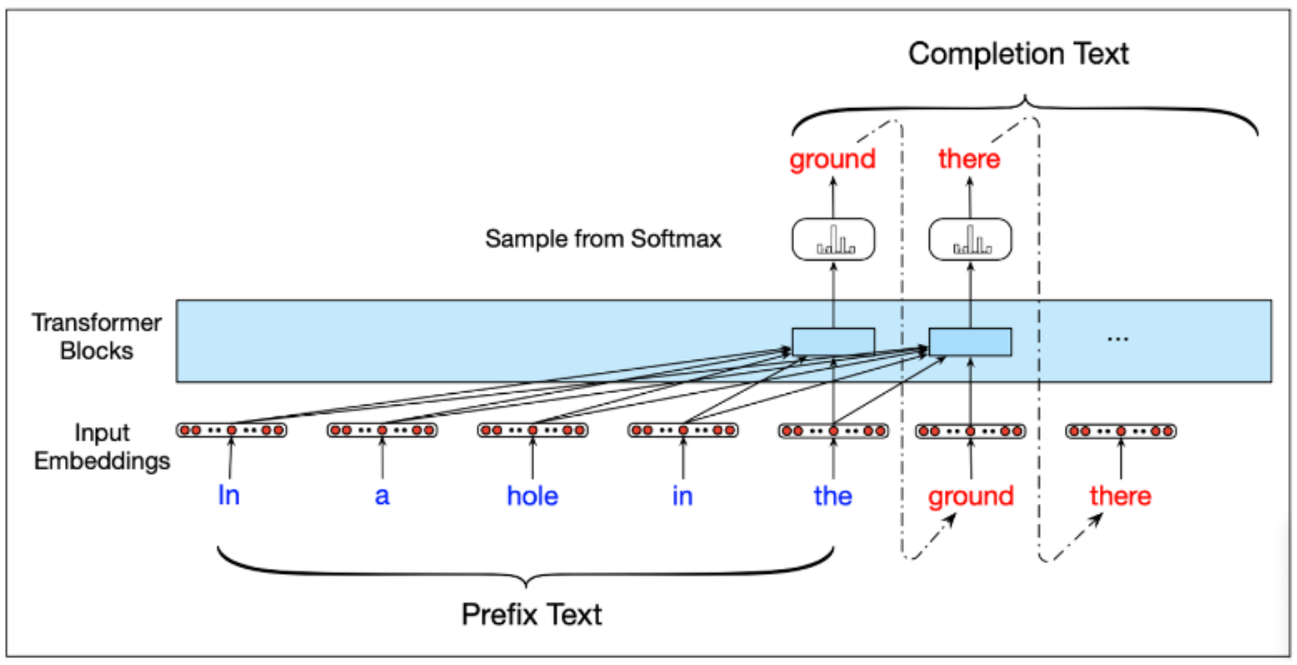
类似的技术还有面向计算机视觉的BN、面向图像生成的IN

Transformer应用

语言模型

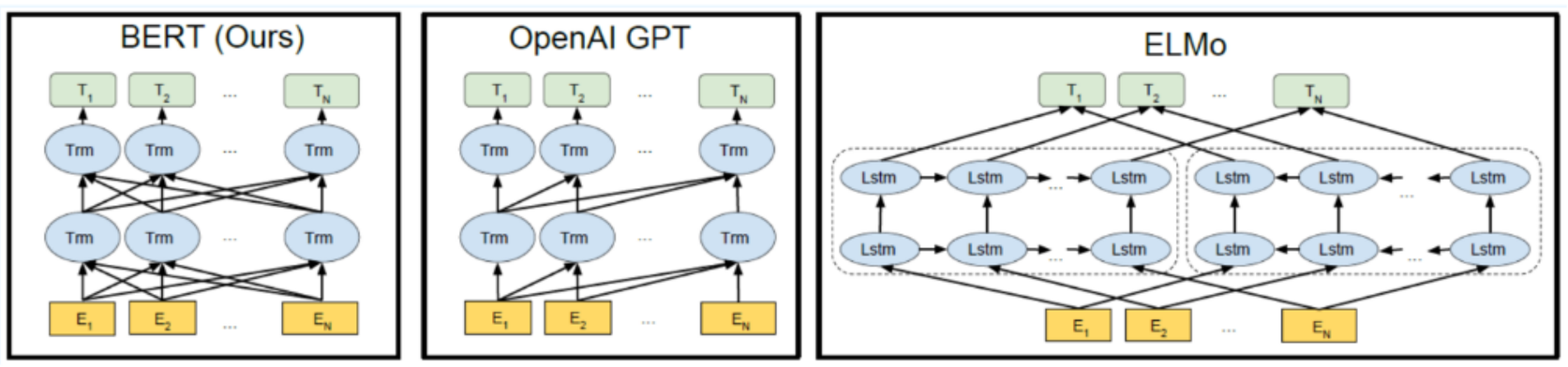


文本生成与摘要



BERT模型

一种基于Transformer架构的预训练语言模型，核心是多层双向Transformer编码器（注：GPT是多层单向，ELMO是双向LSTM）



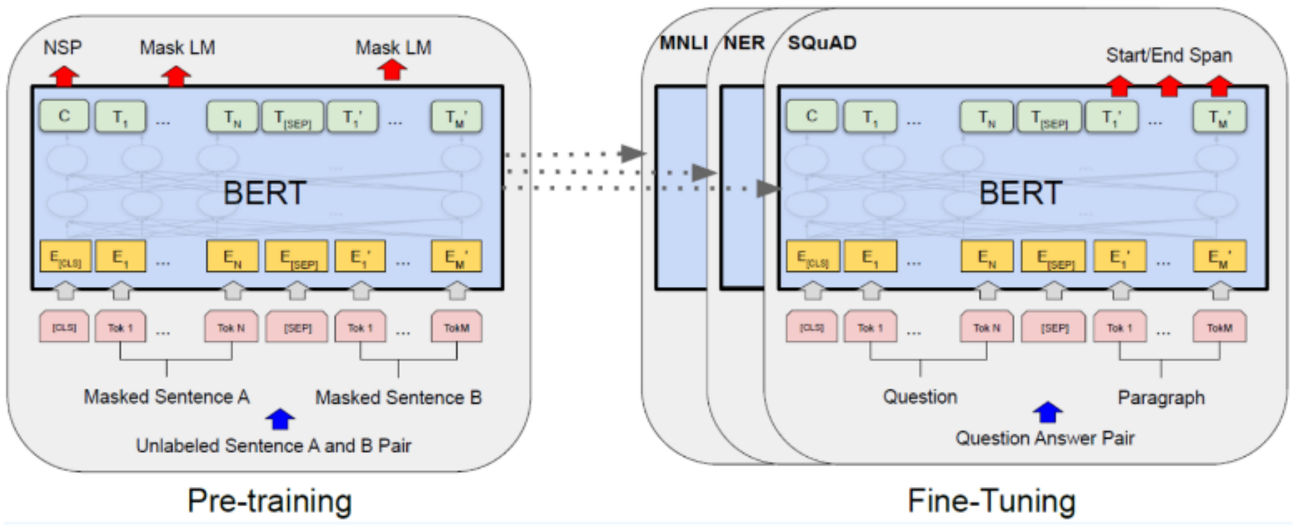
核心训练逻辑：预训练－微调

预训练：输入无标注大规模文本

两个训练任务：

- Masked LM（掩码语言模型）：随机用[MASK]遮住部分单词，让模型预测，目的是学习双向上下文
- NSP（下一句预测）：输入句子A、B，让模型预测B是否是A的下一句，目的是学习学习句子间关系

微调：输入下游任务标注数据（分类，问答等）
在预训练好的BERT上，加少量任务层，用标注数据微调参数，目的是让其将所学到的通用知识应用到具体的下游任务上。



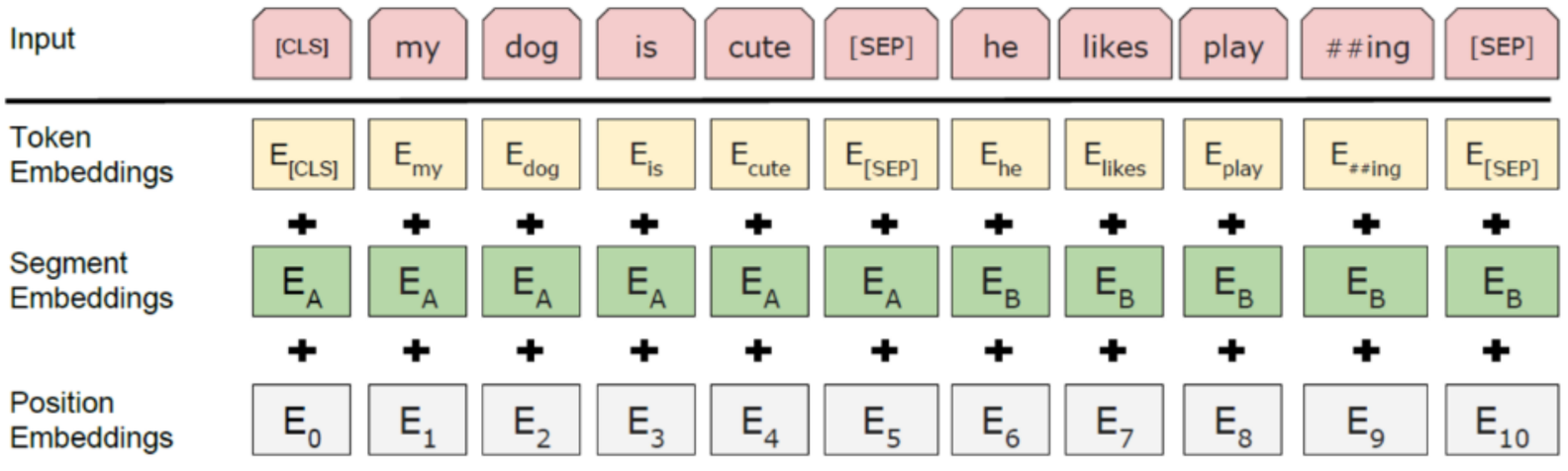
三种嵌入与三种标记

BERT的输入是三种嵌入的相加

Token Embeddings（词嵌入）：每个单词 / 子词的基础语义向量，包含特殊标记的嵌入
Segment Embeddings（段嵌入）：区分句子A和句子B，句子A用E_A，句子B用E_B，让模型知道句子边界
Position Embeddings（位置嵌入）：给每个位置添加位置信息，让模型学习序列顺序（Transformer的位置编码）

同时引入三种标记

[CLS]：放在句子 / 句子对的最开头，它的最终输出向量C，专门用于分类任务（如句子分类、NSP）
[SEP]：分隔符，用于分隔两个句子（如句子A和B），同时标记句子结束
[MASK]：掩码标记，用于Masked LM预训练任务，遮盖单词让模型预测



下游任务适配：

a、句子对分类：自然语言推理、问句相似度、句子复述等

输入：句子A + [SEP] + 句子B + [SEP]

输出：用[CLS]的向量做分类

b、单句子分析：情感分析、语法正确性判断等

输入：[CLS]+单句+[SEP]

输出：用[CLS]的向量作分类

c、问答任务：斯坦福问答数据集等

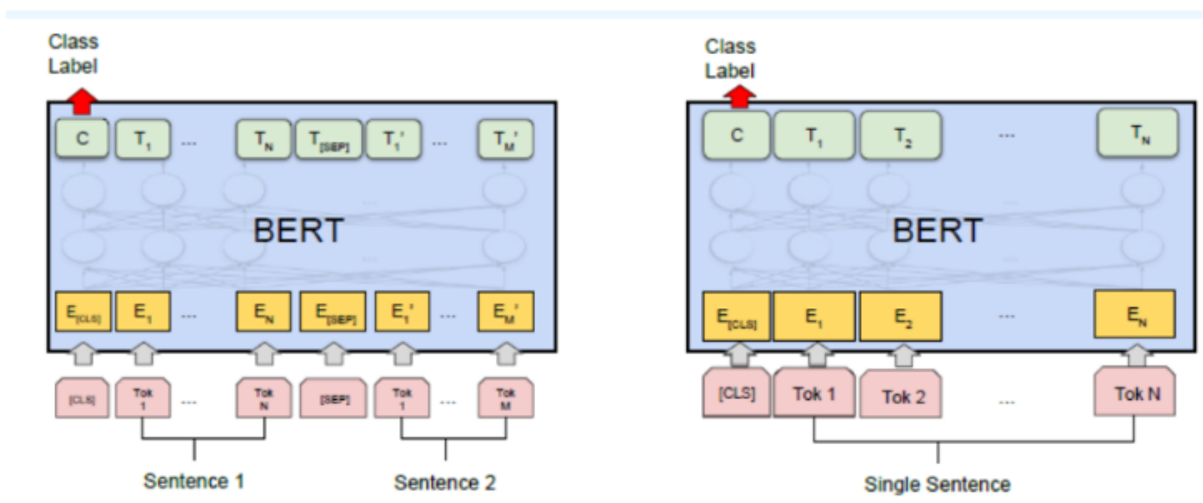
输入：问题+[SEP]+段落+[SEP]

输出：预测答案在段落的起始结束位置

d、序列标注任务：命名实体识别等

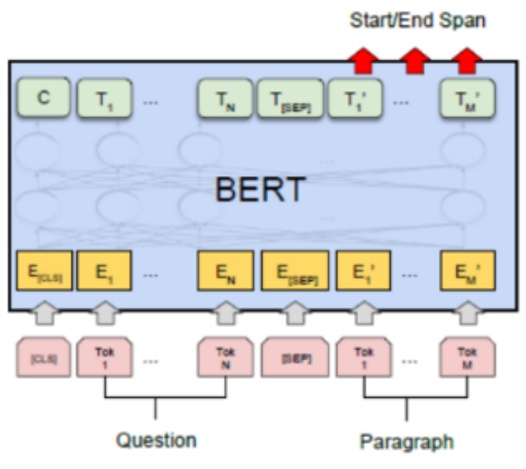
输入：[CLS]+单句+[SEP]

输出：对每个词的输出向量做分类、标类型

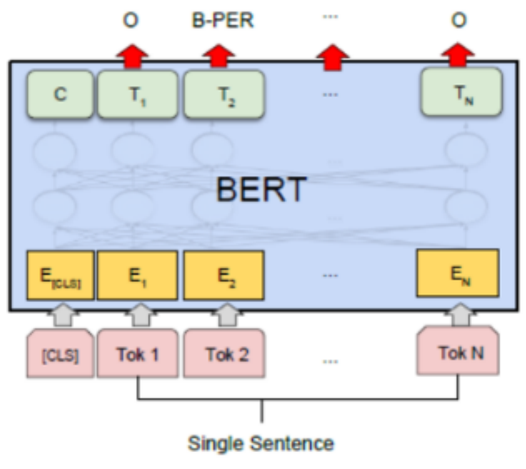


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER