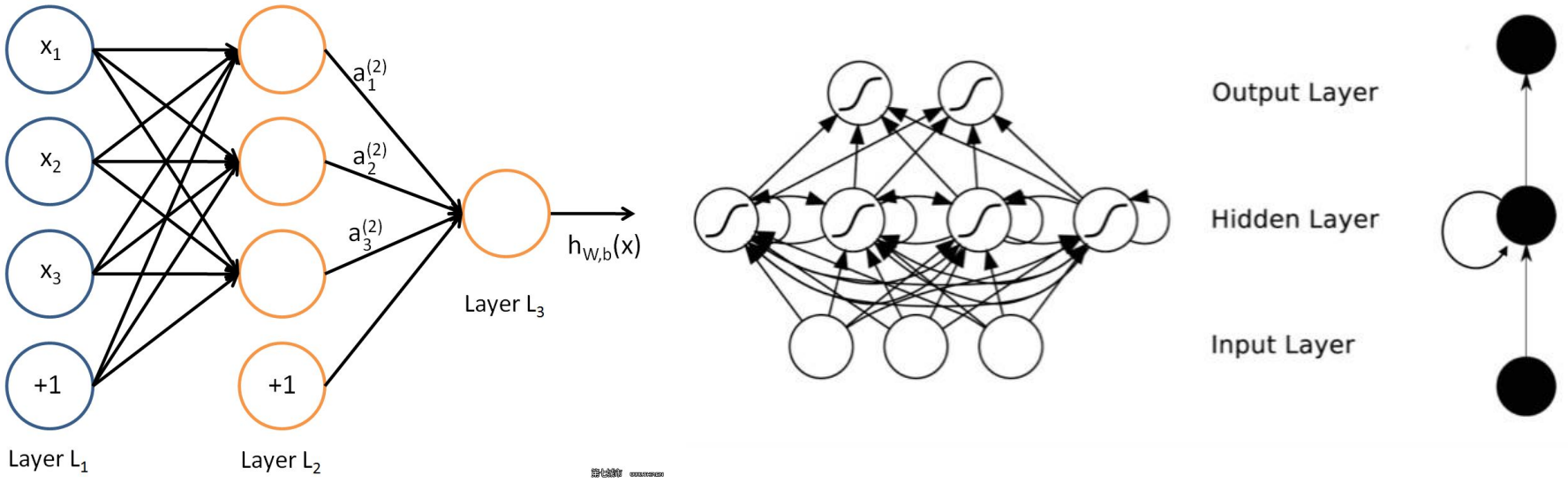
目录

一、Recurrent Neural Networks(RNN)基本原理

1. 典型的RNN结构

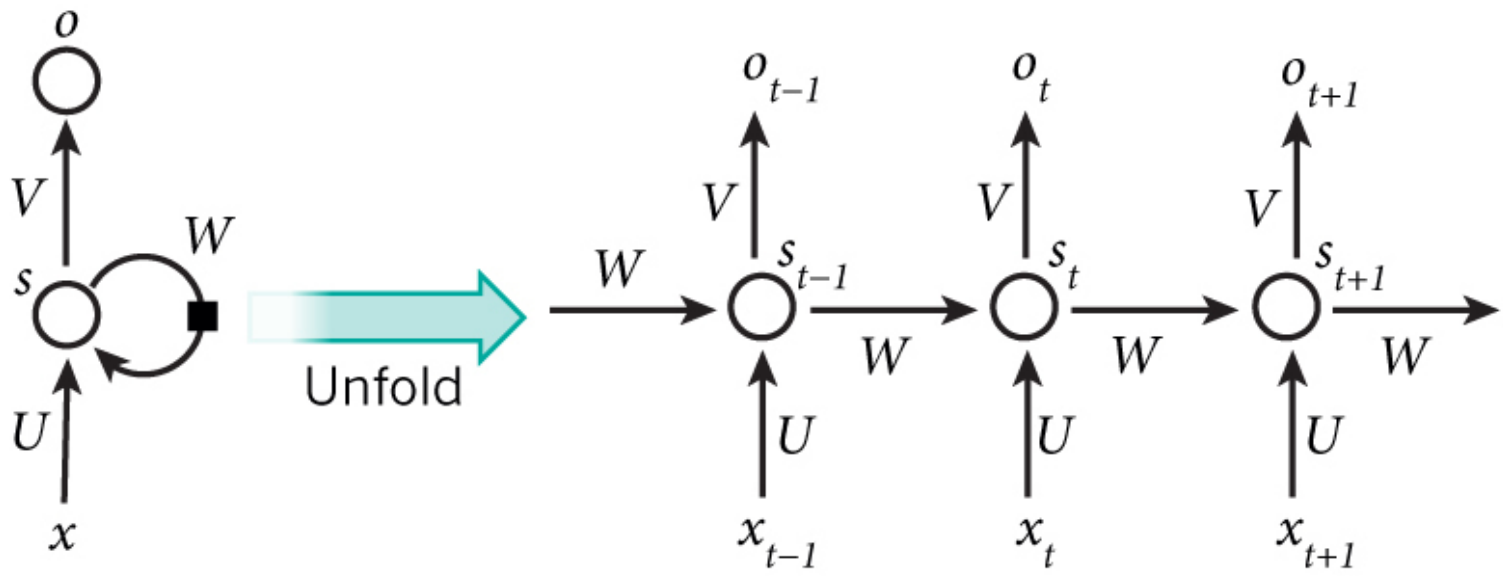
2. RNN的前向传导算法

1.典型的RNN结构



CNN（左）和RNN（右）结构的比较

CNN 是通过权值共享来做到相邻区域的特征提取，那么 RNN 是如何做到提取特征的呢？关键就在于“状态（State）”的引入。



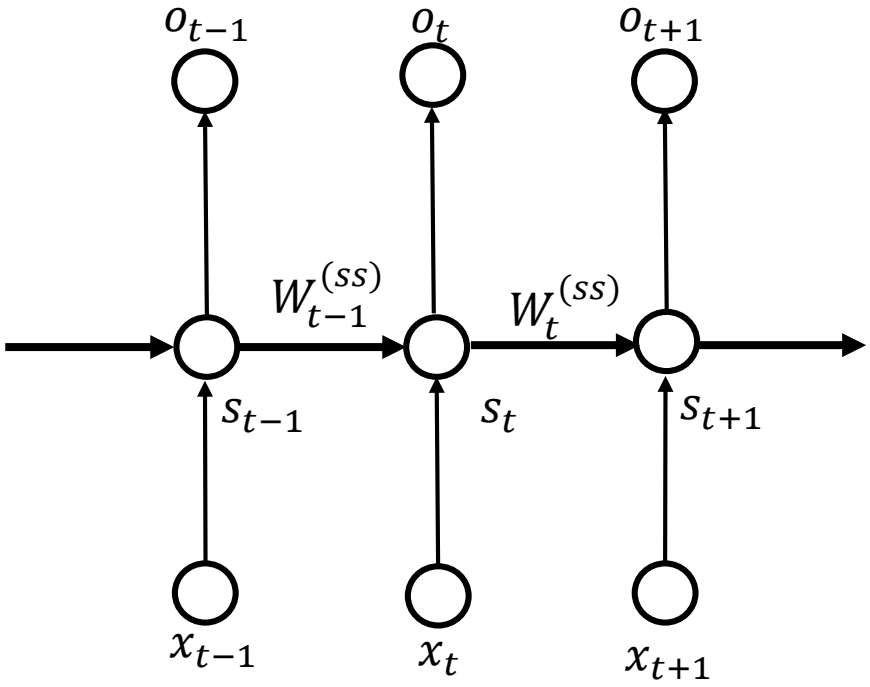
将RNN循环结构展开

2.RNN 的“前向传导算法”

$\mathrm{x_{t-1}}$， $\mathrm{x_t}$， $\mathrm{x_{t+1}}$

$\mathrm{o_{t-1}}$， o_{t}， $o_{\mathrm{t+1}}$

$\mathrm{S_{t-1}}$， $\mathrm{S_t}$， $\mathrm{S_{t+1}}$

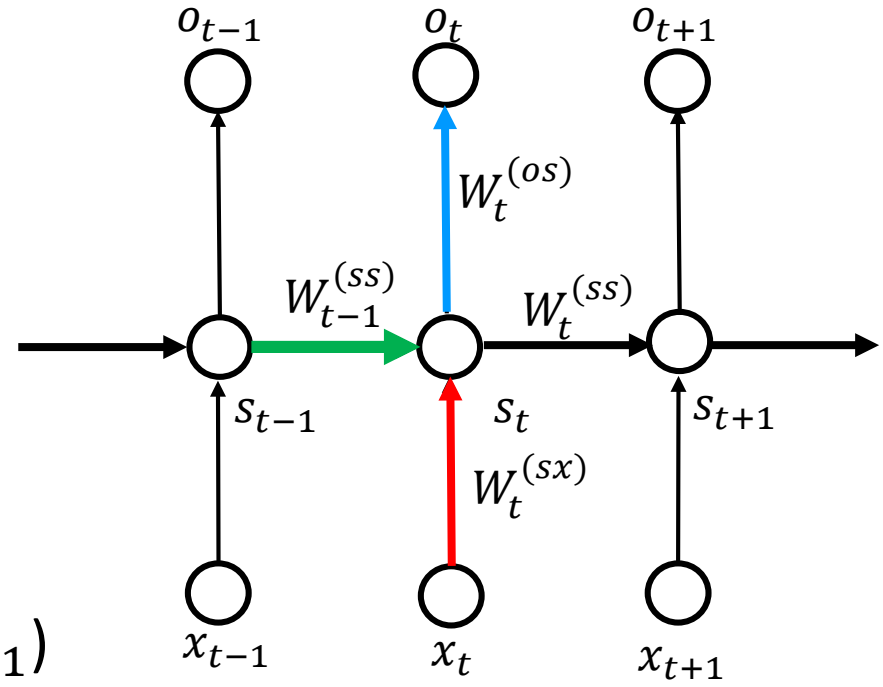


可以分别视为第
t-1,t,t+1“时刻”的输入、
输出与隐藏层的状态

- $s_t = W_t^{(sx)} x_t + W_{t-1}^{(ss)} s_{t-1}$
- $o_t = W_t^{(os)} s_t$

将 s_t 代入可得 o_t 如下：

- $o_t = W_t^{(os)}(W_t^{(sx)} x_t + W_{t-1}^{(ss)} s_{t-1})$



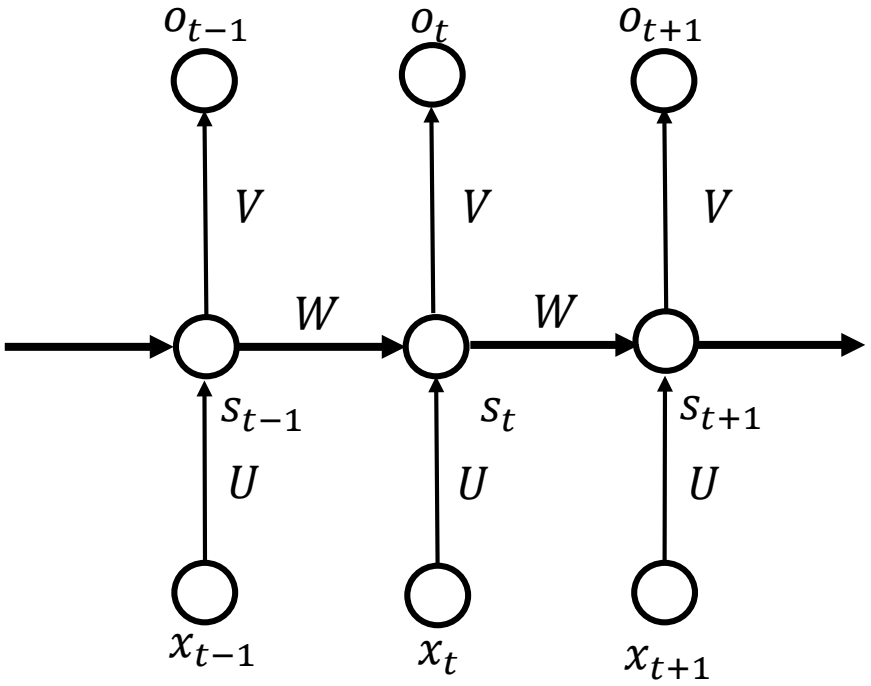
由于RNN的参数是共享的，于是有如下假定：

- $W_1^{os} = \cdots = W_t^{os} = V$
- $W_1^{sx} = \cdots = W_t^{sx} = U$
- $W_1^{ss} = \cdots = W_t^{ss} = W$

于是可得：

- $s_t = Ux_t + Ws_{t-1}$
- $o_t = Vs_t = V(Ux_t + Ws_{t-1})$

并且一般认为：$s_0 = (0,0,\ldots,0)^T$



- NN 中有激活函数的概念，将它应用到RNN中是非常自然的（视s为隐藏层、o为输出层即可）

$$s_t = \phi_t^{(sx)}(Ws_{t-1} + Ux_t)$$

$$o_{\mathrm{t}} = \phi_{\mathrm{t}}^{(os)}(Vs_t) = \phi_t^{(os)}\left(V\phi_t^{(sx)}(Ws_{t-1} + Ux_t)\right)$$

- 其中，由于参数共享，各个$\phi_i^{(os)}$、$\phi_t^{(sx)}$通常都会取为相同的函数

$$\phi_1^{(os)} = \cdots = \phi_t^{(os)} = \cdots = f$$

$$\phi_1^{(sx)} = \cdots = \phi_t^{(sx)} = \cdots = g$$

- 那么最后就能得到等式：

$$s_t = g(Ws_{t-1} + Ux_t)$$

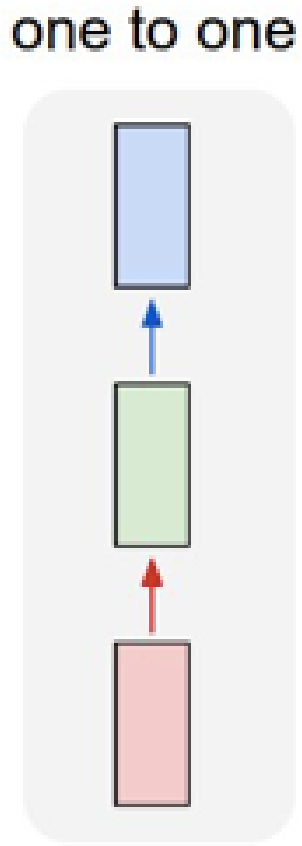
$$o_t = f(Vs_t) = f(Vg(Ws_{t-1} + Ux_t))$$

二、RNN的结构

1. 几种经典的RNN结构
2. Bidirectional RNN
3. 深层RNN

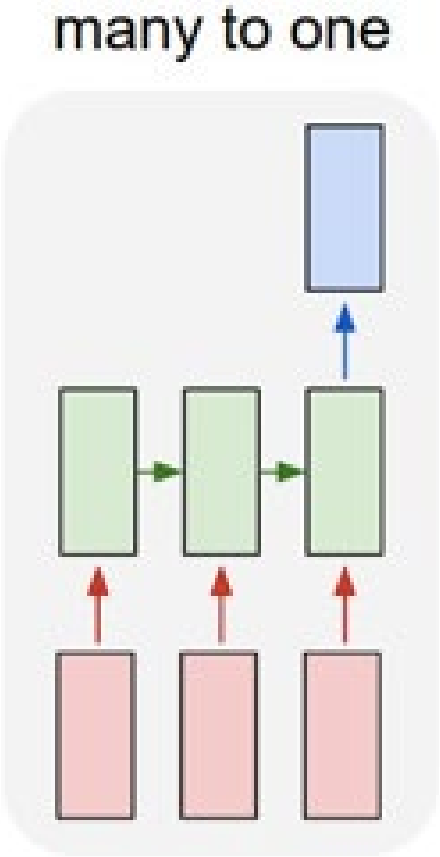
1.几种典型RNN结构

- One to One
最朴素的RNN结构，例如：对某单一输入进行分类(图像分类)。



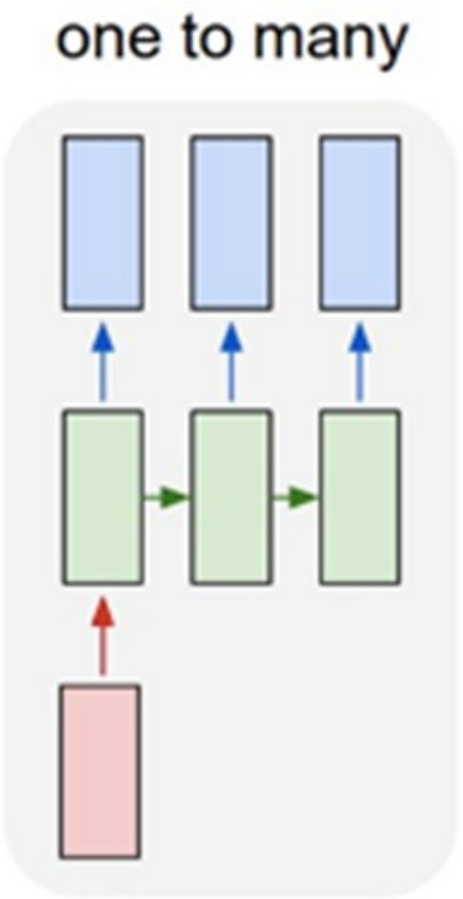
- Many to One

例如：对文本或者视频进行分类，输入一段文本或者视频，输出 单个标签（情感分析）。



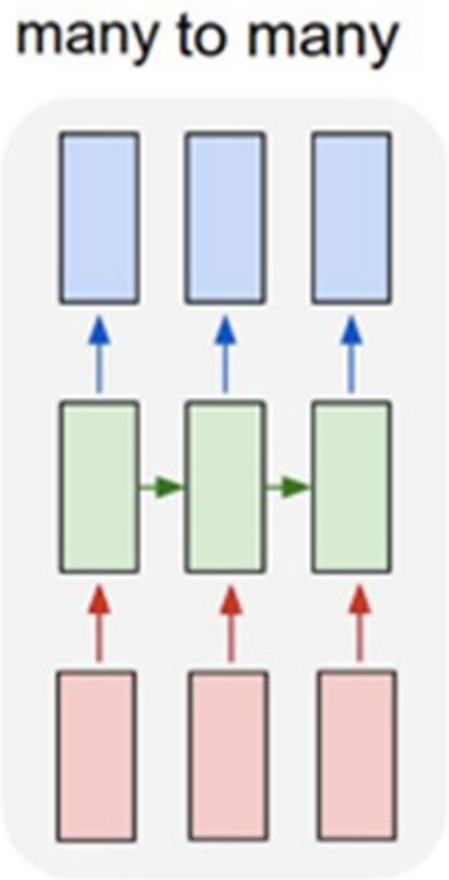
- One to Many

例如：输入一张图片，输出一个句子序列对图片进行描述（看图说话）



- Many to Many

例如：机器翻译，输入某种语言的文本，输出另一个语言的文本。

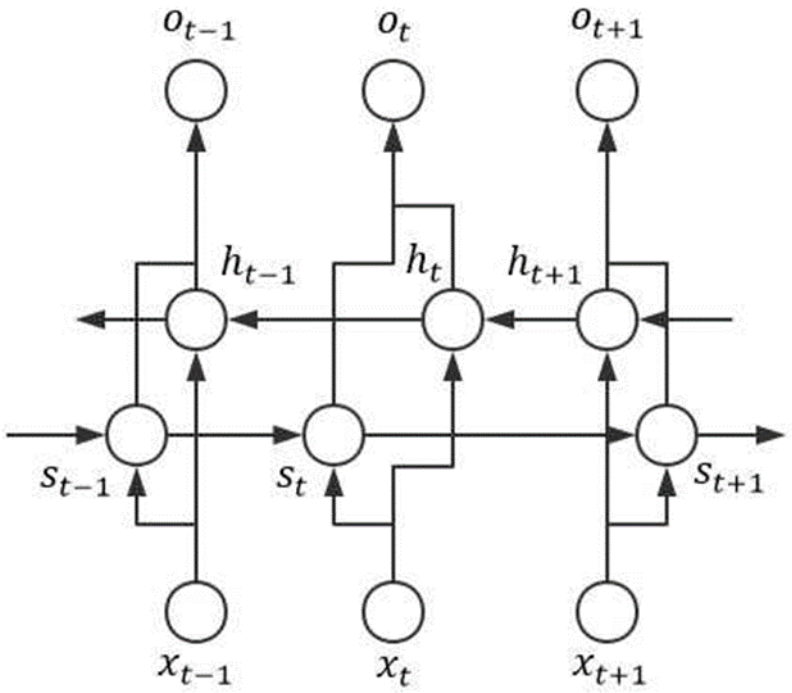


2.Bidirectional RNN（可译为“双向 RNN”）

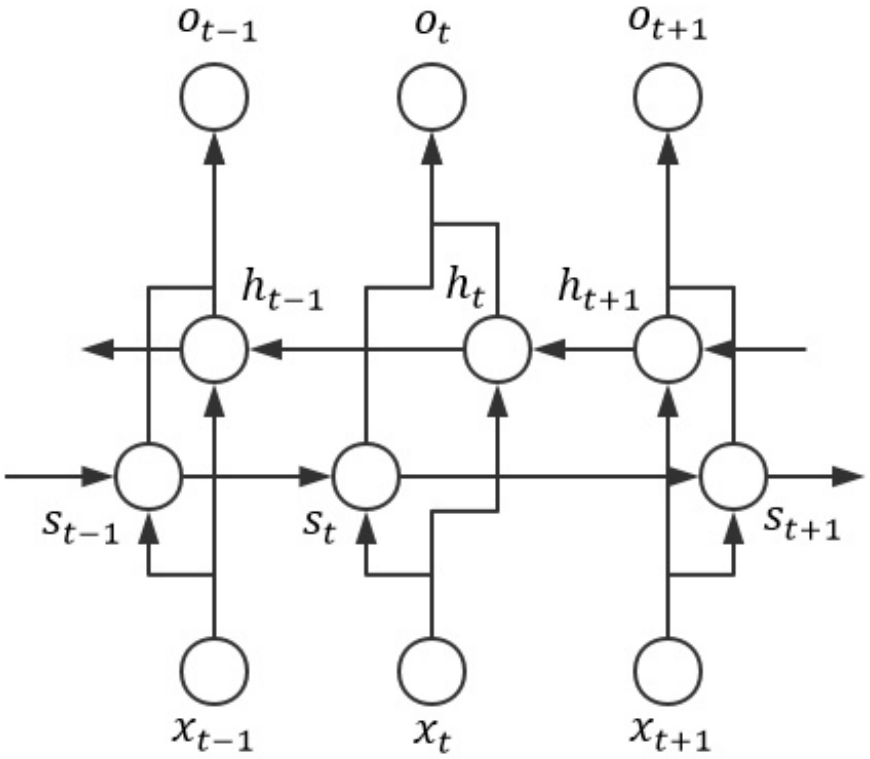
- 目前为止讨论过的 RNN 的 State 的传递都是单方向的线性传递，那么是否能改进这种传递方式呢？

- 答案自然是肯定的。

单向传递虽然确实能够利用上“历史信息”，但它却无法利用“未来信息”。以翻译为例，我们常常会需要“联系上下文”来得出某句话的含义，单向传递允许我们“联系上文”、但“联系下文”却无法做到。



双向RNN的结构



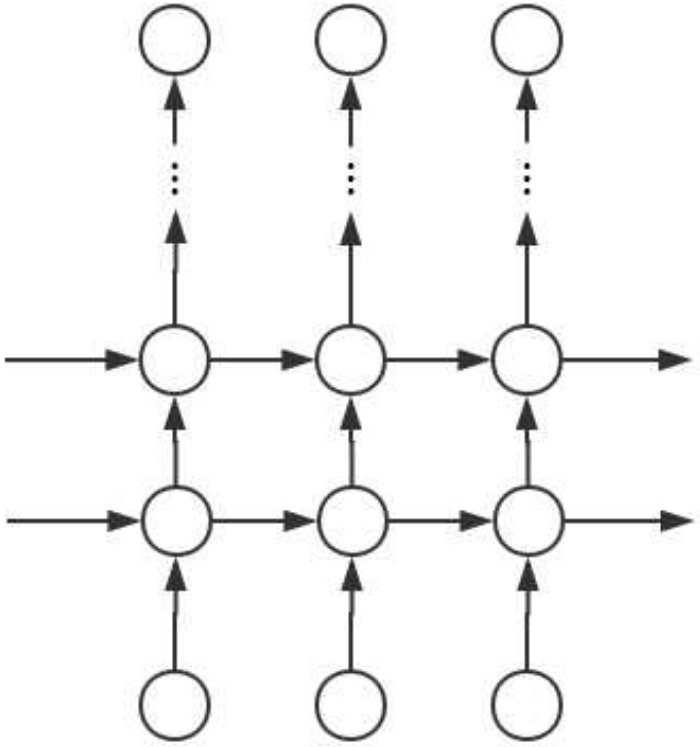
$$o_t = W_t^{(os)} s_t + W_t^{(oh)} h_t$$

$$= W_t^{(os)} \left(W_{t-1}^{(ss)} s_{t-1} + W_{t-1}^{(sx)} x_t \right) + W_t^{(oh)} (W_t^{(hh)} h_{t+1} + W_t^{(hx)} x_t)$$

3.深层 RNN

- 前面介绍的所有 RNN 都有一个共同的“缺点”——看上去太“浅”了（每一步只有一层 State 隐藏层），很自然的联想到加深网络的层数。

- 深层RNN每一步的输入有多层网络，因此该网络便具有更加强大的表达能力和学习 能力，但是复杂性也提高了，同时需要训练更多的数据。

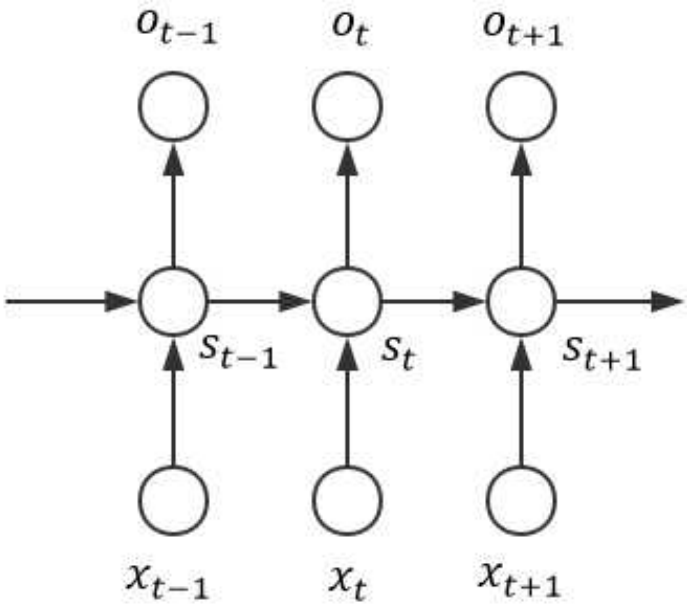


深层RNN

三、BPTT (Back propagation Through Time) 算法

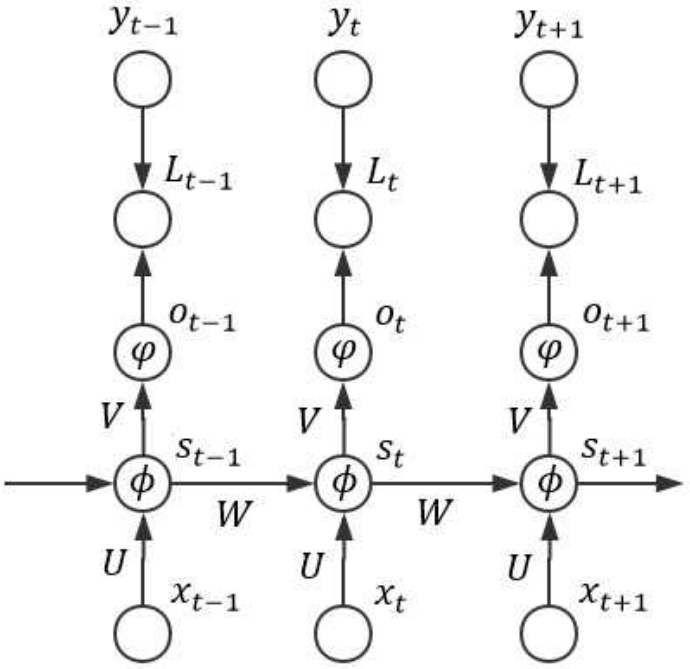
- 取ϕ作为隐藏层的激活函数
- 取φ作为输出层的变换函数
- 取 $L_t = L_t(o_t, y_t)$ 作为模型的损失函数，其中标签y_t是一个 one-hot 向量；由于 RNN 处理的通常是序列数据、所以在接受完序列中所有样本后再统一计算损失是合理的，此时模型的总损失可以表示为（假设输入序列长度为n）：

$$L = \sum_{t=1}^{n} L_t$$



$o_t = \varphi(Vs_t) = \varphi(V\phi(Ux_t + Ws_{t-1}))$,
其中$s_0 = (0,0,0..0)^T$

- 令$o_t^* = Vs_t$，$s_t^* = Ux_t + Ws_{t-1}$
- 有$o_t = \varphi(o_t^*)$，$s_t = \phi(s_t^*)$



$L_t = L_t(o_t, y_t)$

- 从而（注：统一使用“*”表示 element wise 乘法，使用“×”表示矩阵乘法）

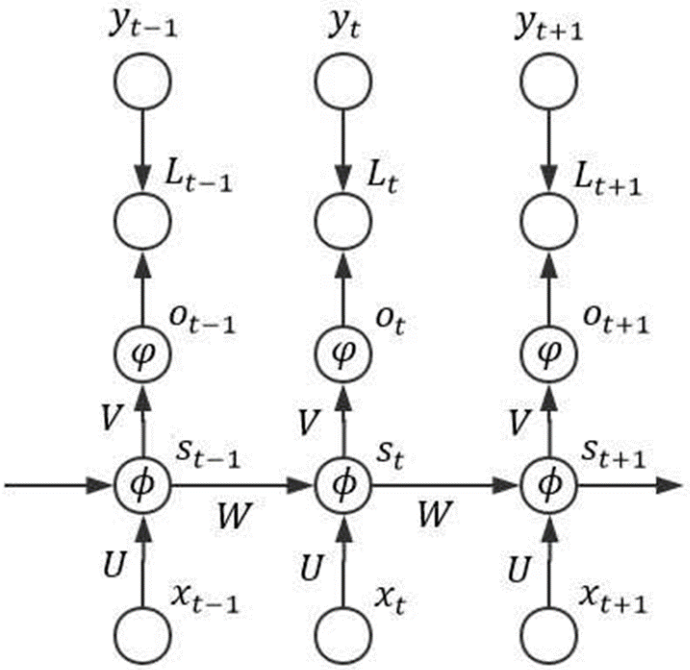
$$\frac{\partial L_t}{\partial o_t^*} = \frac{\partial L_t}{\partial o_t} * \frac{\partial o_t}{\partial o_t^*} = \frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*)$$

$$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial Vs_t} \times \frac{\partial Vs_t}{\partial V} = \left(\frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*)\right) \times s_t^T$$

- 由$L = \sum_{t=1}^{n} L_t$得总梯度为

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{n} \left(\frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*)\right) \times s_t^T$$

- 事实上，RNN 的 BP 算法的主要难点在于它 State 之间的通信，亦即梯度除了按照空间结构($o_t \rightarrow s_t \rightarrow x_t$)传播以外，还得沿着时间通道传播($s_t \rightarrow s_{t-1} \rightarrow \cdots \rightarrow s_1$)，这导致我们比较难将相应 RNN 的 BP 算法写成一个统一的形式（回想之前的“前向传导算法”）。为此，我们可以采用“循环”的方法来计算各个梯度



• 计算时间通道上的“局部梯度”:

$$\frac{\partial L_t}{\partial s_t^*} = \frac{\partial s_t}{\partial s_t^*} * \frac{\partial L_t}{\partial s_t}$$

$$= \frac{\partial s_t}{\partial s_t^*} * \left(\frac{\partial s_t^T V^T}{\partial s_t} \times \frac{\partial L_t}{\partial V s_t} \right)$$

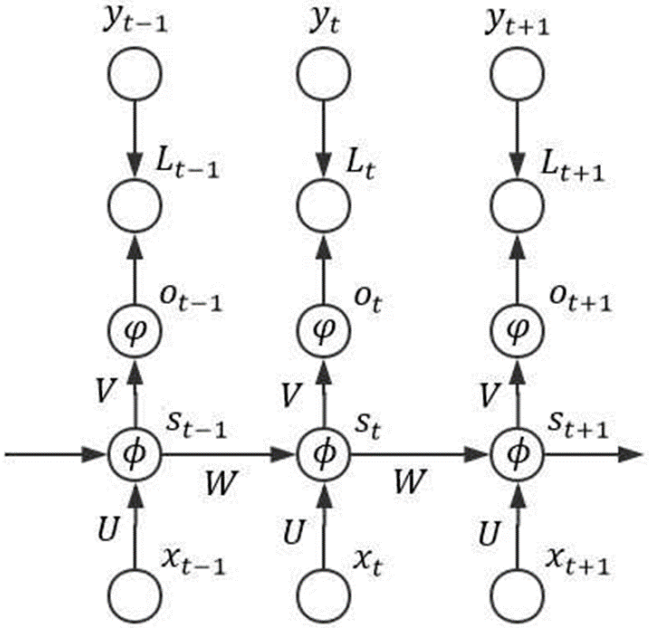
$$= \phi'(s_t^*) * \left[V^T \times \left(\frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*) \right) \right]$$

$$o_t = \varphi(o_t^*)$$
$$s_t = \phi(s_t^*)$$
$$o_t^* = V s_t$$
$$s_t^* = U x_t + W s_{t-1}$$

$$\frac{\partial L_t}{\partial s_{t-1}^*} = \frac{\partial s_t^*}{\partial s_{t-1}^*} \times \frac{\partial L_t}{\partial s_t^*}$$

$$= \frac{\partial s_{t-1}}{\partial s_{t-1}^*} \times \frac{\partial s_t^*}{\partial s_{t-1}} \times \frac{\partial L_t}{\partial s_t^*}$$

$$= \phi'(s_{t-1}^*) * (W^T \times \frac{\partial L_t}{\partial s_t^*})$$



- 利用时间通道上的“局部梯度”计算U和W的梯度：

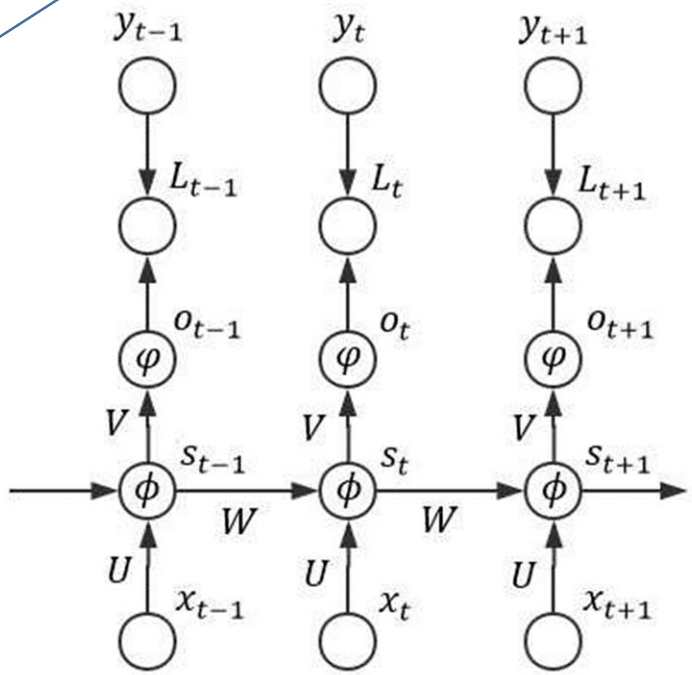
$$\frac{\partial L_t}{\partial U} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_t^*} \times \frac{\partial s_t^*}{\partial U}$$

$$= \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_t^*} \times x_t$$

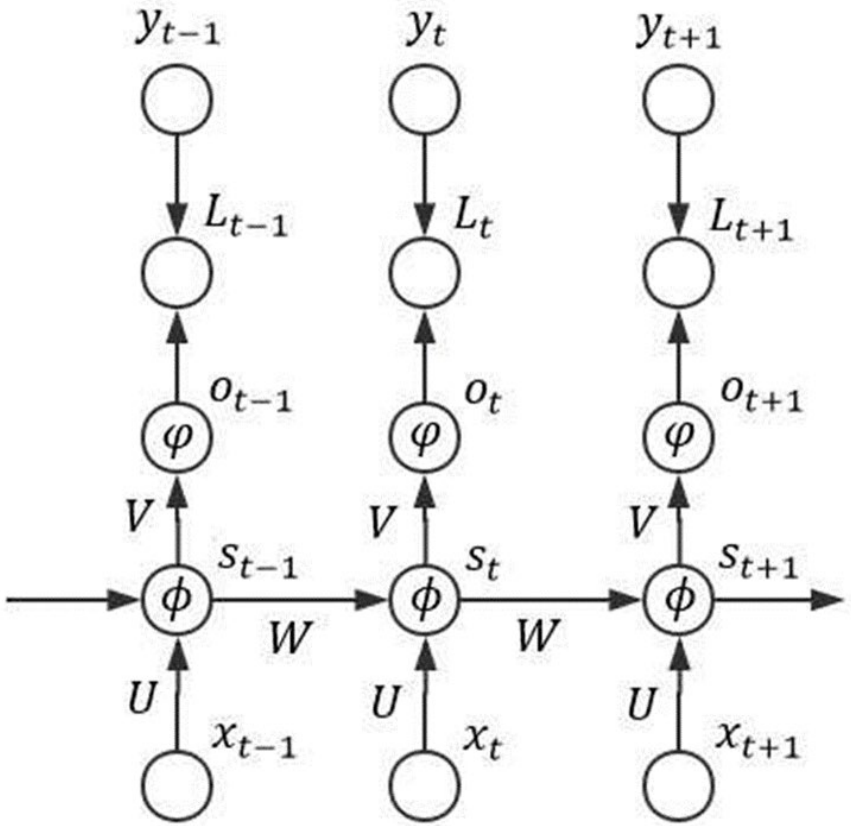
$$\frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_t^*} \times \frac{\partial s_t^*}{\partial W}$$

$$= \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_t^*} \times s_{t-1}$$

$s_t^* = Ux_t + Ws_{t-1}$



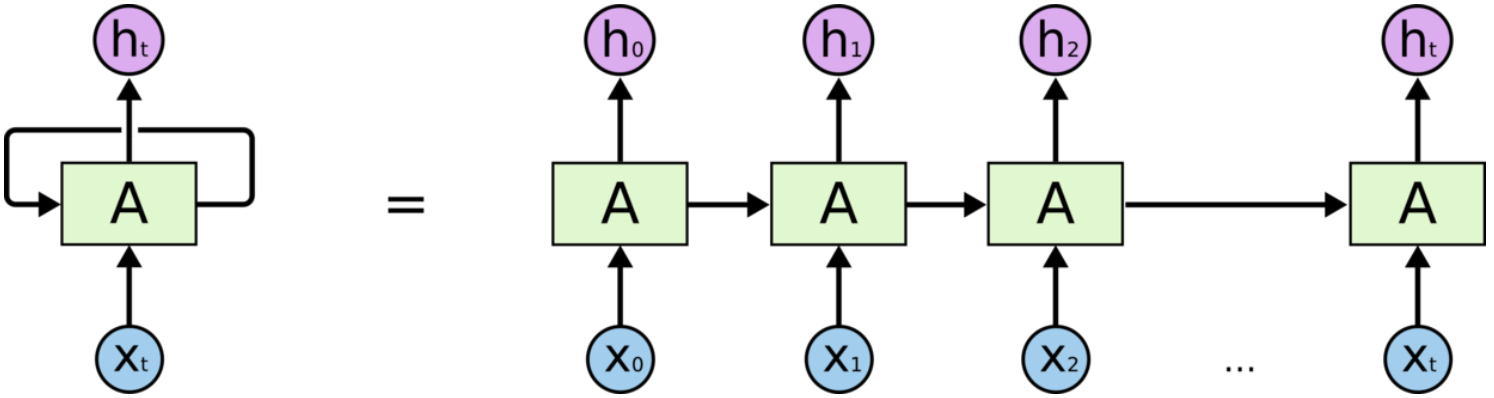
- 以上即为 RNN 反向传播算法的所有推导，它比 NN 的 BP 算法要繁复不少。事实上，像这种需要把梯度沿时间通道传播的 BP 算法是有一个专门的名词来描述的——Back Propagation Through Time（常简称为 BPTT，可译为“时序反向传播算法”）



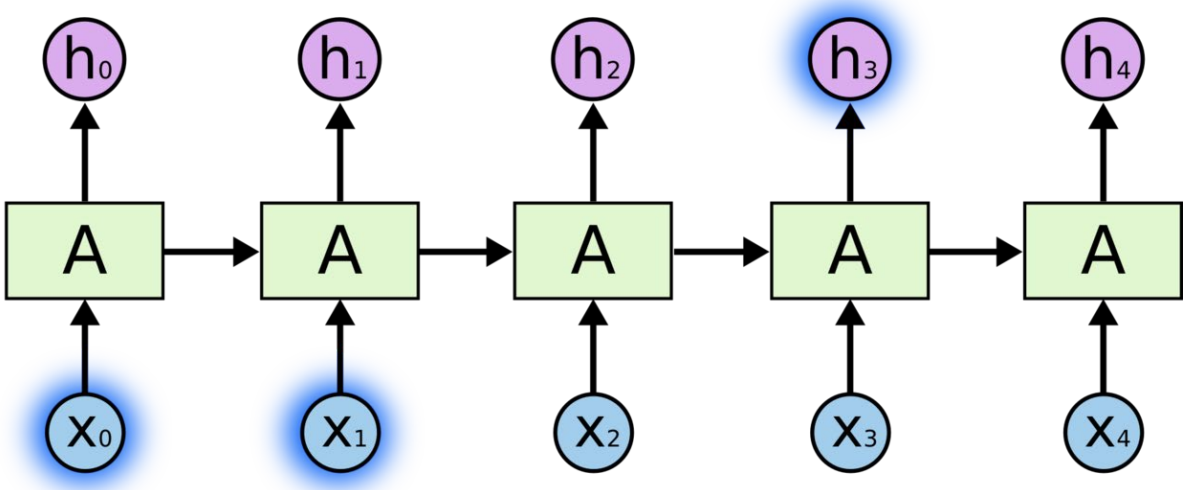
四、Long-short term memory (LSTM) networks

1. RNN的不足
2. LSTM简介
3. LSTM的结构
4. LSTM的变体

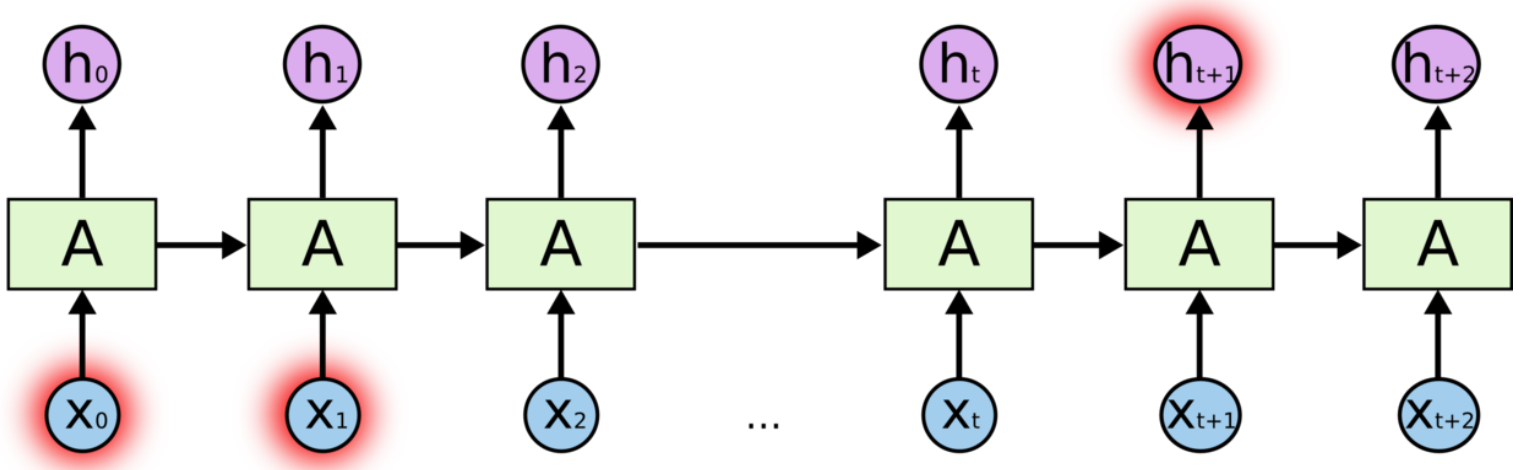
1.RNN的不足



- 上图所示左边是RNN的结构图。神经网络的模块，A，正在读取某个输入 x_t，并输出一个值 h_t。循环可以使得信息可以从当前步传递到下一步。
- RNN可以被看做是同一神经网络的多次复制，每个神经网络模块会把消息传递给下一个。



- RNN 的关键点之一就是他们可以用来连接先前的信息到当前的任务上，例如使用过去的视频段来推测对当前段的理解。
- 有时候，我们仅仅需要知道先前的信息来执行当前的任务。例如，我们有一个语言模型用来基于先前的词来预测下一个词。如果我们试着预测 “the clouds are in the sky” 最后的词，我们并不需要任何其他的上下文 —— 因此下一个词很显然就应该是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔非常小的，RNN 可以学会使用先前的信息。



- 但是同样会有一些更加复杂的场景。假设我们试着去预测“I grew up in France... I speak fluent French”最后的词。当前的信息建议下一个词可能是一种语言的名字，但是如果我们需要弄清楚是什么语言，我们是需要先前提到的离当前位置很远的 France 的上下文的。这说明相关信息和当前预测位置之间的间隔就肯定变得相当的大。
- 不幸的是，在这个间隔不断增大时，RNN 会丧失学习到连接如此远信息的能力。

梯度消失和梯度爆炸

- 实际应用中，会出现梯度消失和梯度爆炸。回忆之前的公式

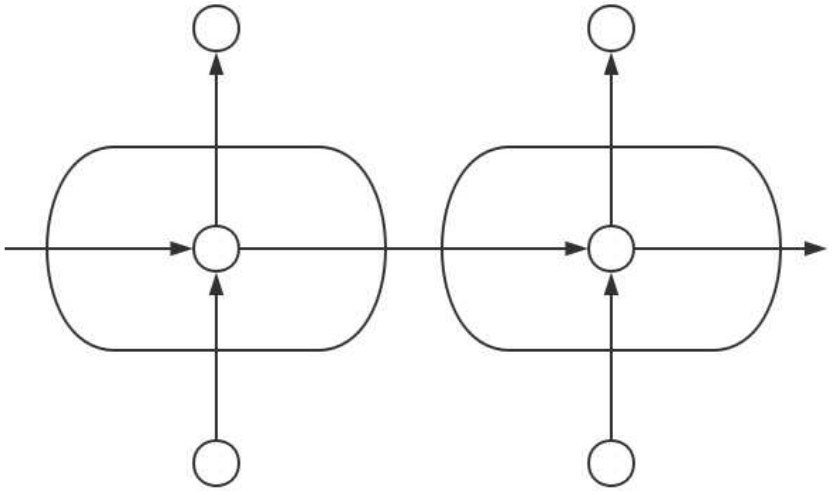
$$
\begin{aligned}
s_t &= g(Ws_{t-1} + Ux_t) \\
&= g(Wg(Ws_{t-2} + Ux_{t-1}) + Ux_t) \\
&= \cdots \\
&= g(Wg(Wg(\ldots Wg(Ws_0 + Ux_1)\ldots) + Ux_{t-1}) + Ux_t)
\end{aligned}
$$

- 在指数函数$y = a^x$,如果0<a<1,则当x非常大的时候，y趋于0；如果a>1,y趋于无穷大。这里联系上述公式，如果0<W<1,由于指数幂的关系，一开始的信息可能会消失，这就是梯度消失；如果W>1,由于指数幂的关系，一开始的信息会非常的大，又会出现另一种现象梯度爆炸。
- 解决上述问题需要对RNN进行改进，其中非常经典的就是LSTM。

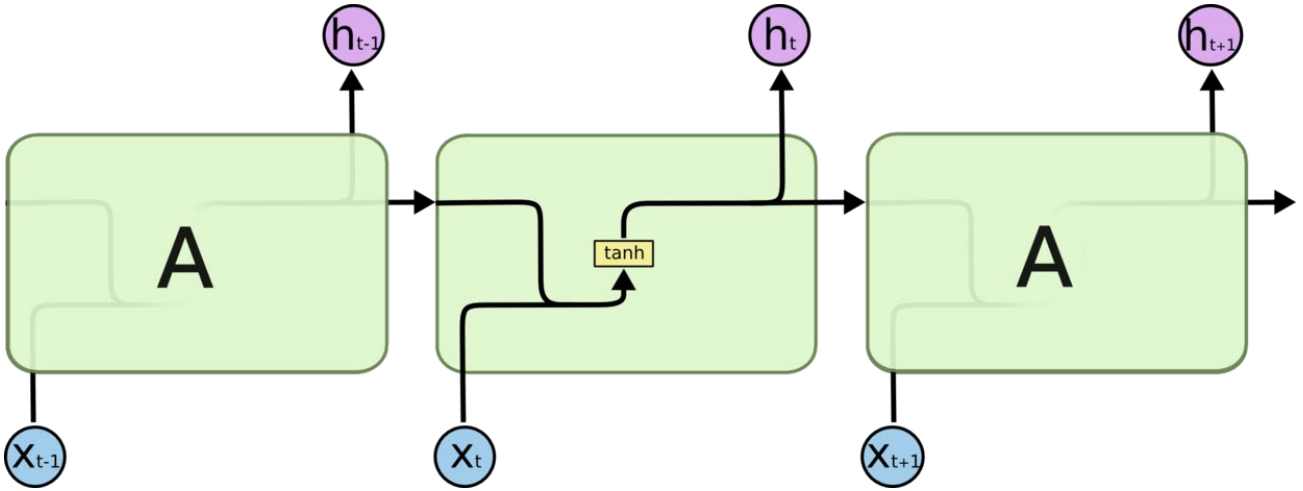
2.LSTM简介

- Long-short term memory (LSTM) networks——是一种 RNN 特殊的类型，可以学习长期依赖信息。

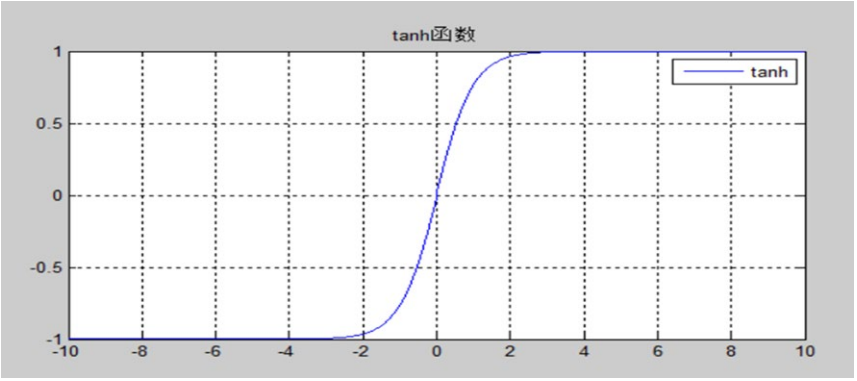
- LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力!

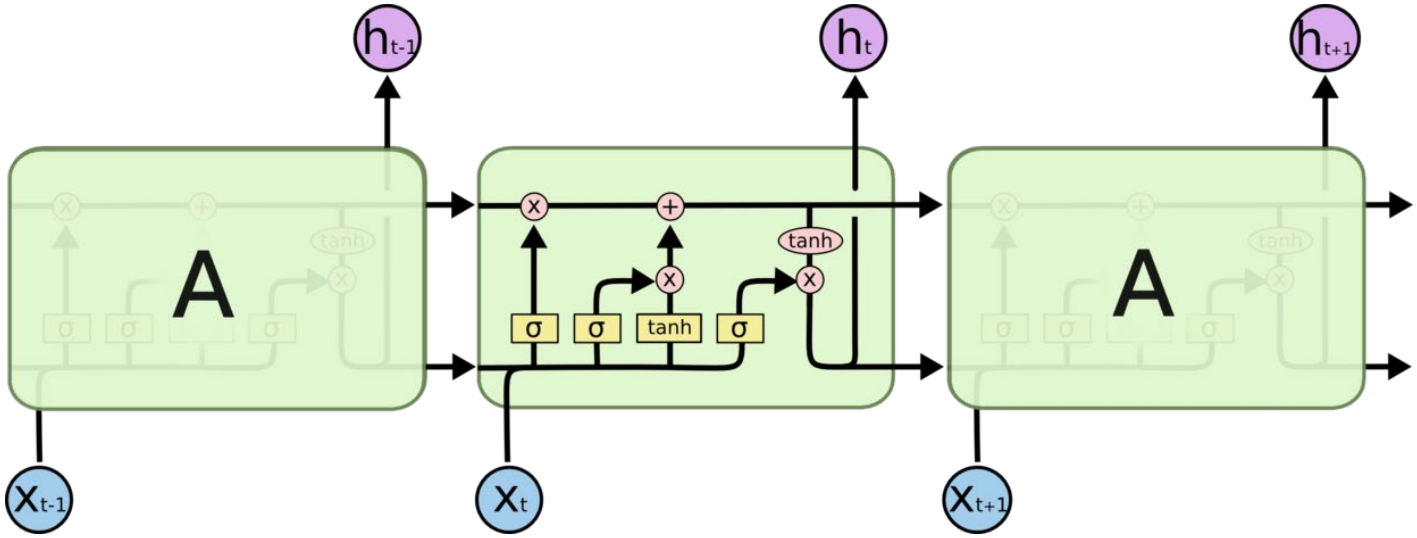


- 完全可以把 State 看作是一个“单元（cell）”，这个单元中可以仅包含单一的向量、也可以包含诸多错综复杂的东西在把 State 视为 cell 后，我们就可以在 cell 中任性地设计各种结构以改进 State 了



在标准的 RNN 中，cell模块只有一个非常简单的结构，例如一个 tanh 层。

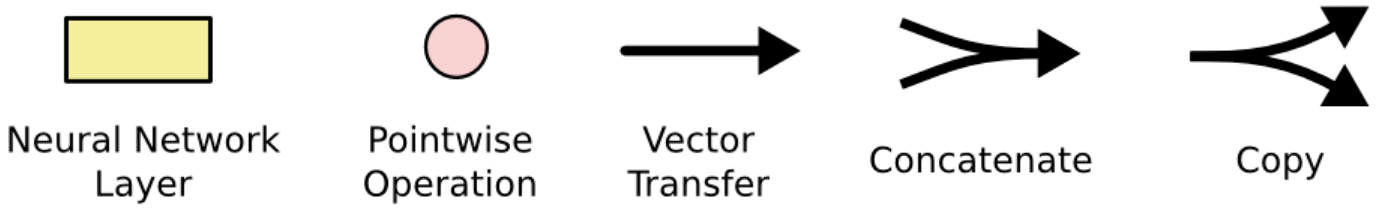




LSTM 同样是这样的结构，但是cell拥有一个不同的结构。不同于单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互。

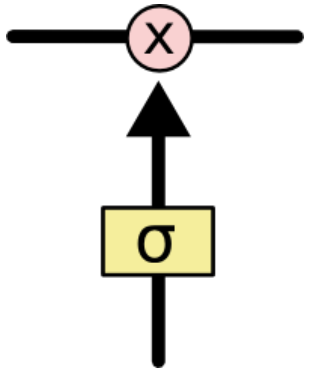
3.LSTM的结构

分解LSTM

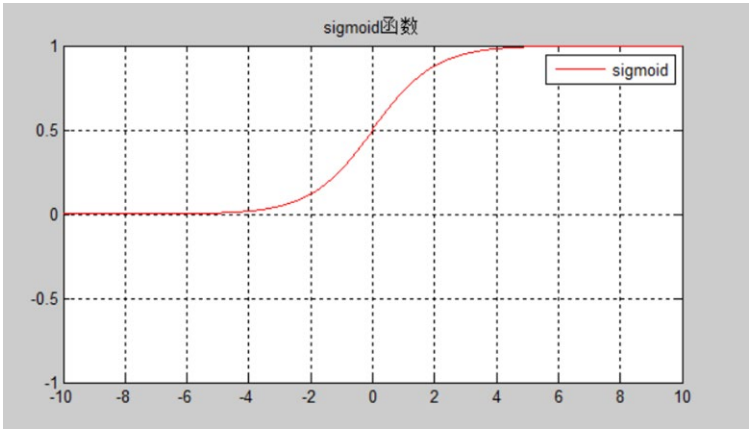


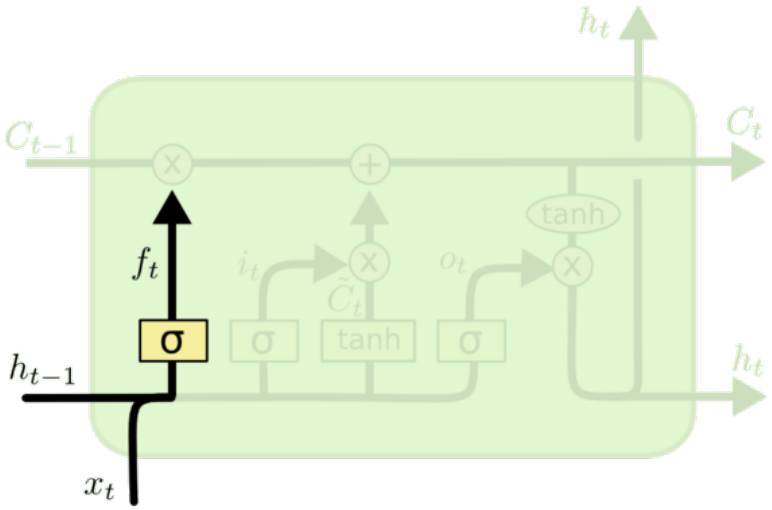
图中使用的各种元素的图标。

- LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。



- Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”！

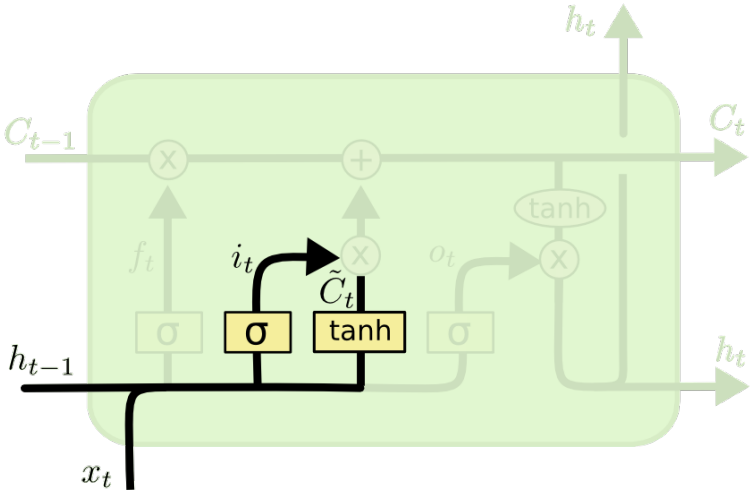




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- 在 LSTM 中的第一步是决定从细胞状态中丢弃什么信息。这个决定是 通过一个称为忘记门层完成的。

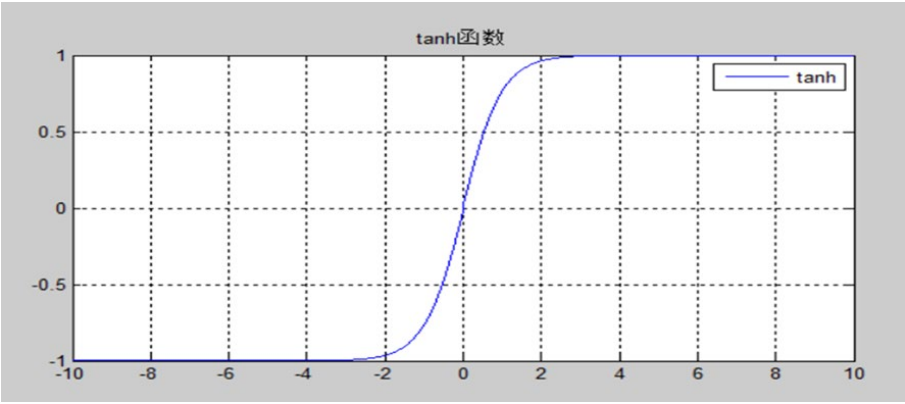
- 该门会读取 h_{t-1}和 x_t，输出一个在 0 到 1 之间的数值与细胞状态 C_{t-1}进行点乘。1 表示“完全保留”，0 表示“完全舍弃”。

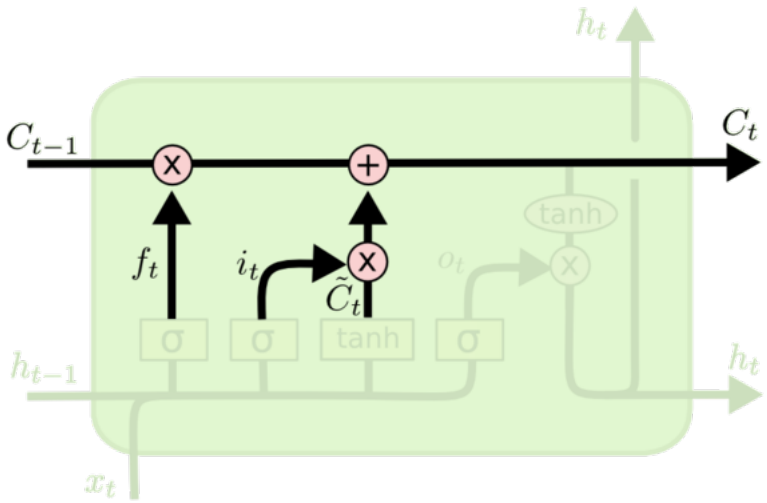


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

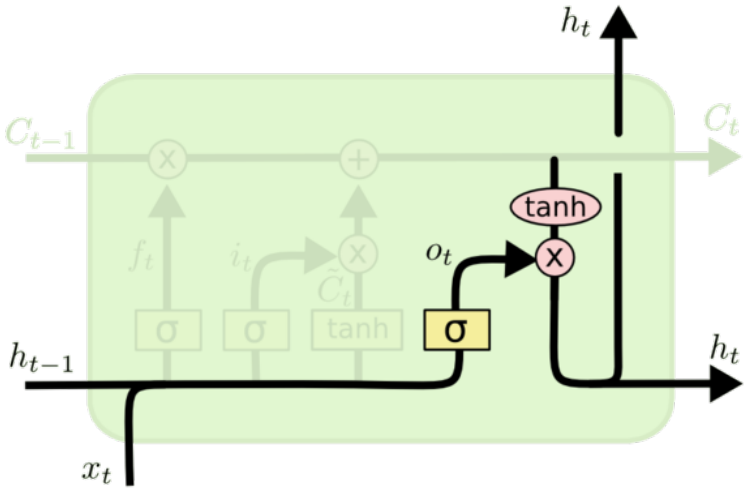
- 下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 层称“输入门层”决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量$\tilde{C}_t$，会被加入到状态中。





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

- 将旧状态与 f_t相乘，丢弃掉确定需要丢弃的信息。接着加上 $i_t * \widetilde{C}_t$。这就是新的候选值C_t，根据我们决定更新每个状态的程度进行变化。



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

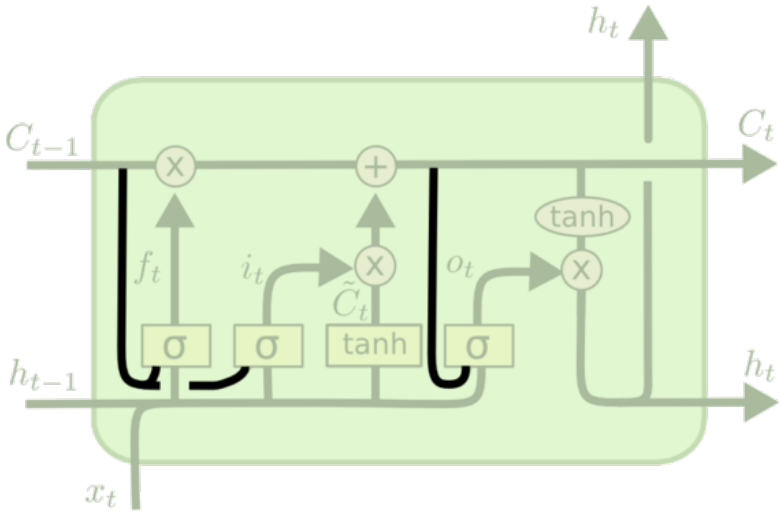
现在，需要确定输出什么值。这个输出将会基于细胞状态，但也是一个过滤后的版本。

- 首先，运行一个 sigmoid 层来确定细胞状态的哪个部分将输出。

- 接着，将细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘。

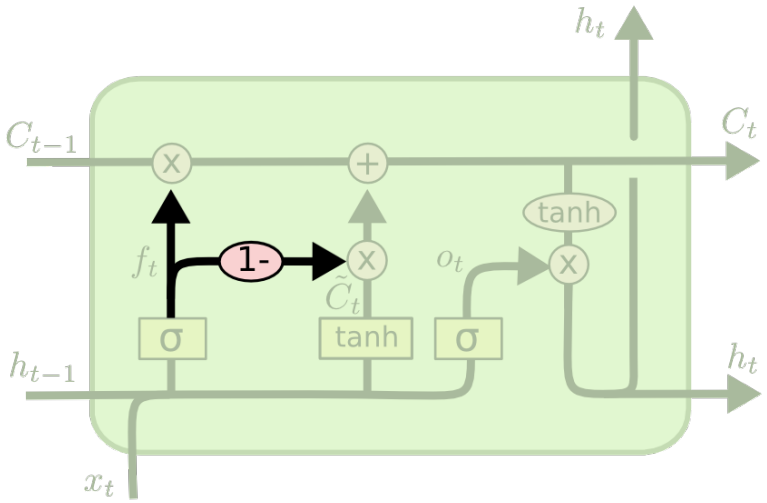
- 最后仅仅会输出我们确定输出的那部分，即h_t。

4.LSTM 的变体



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

- 增加peephole 到每个门上，但是许多论文会加入部分的 peephole 而非所有都加。



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

- 另一个变体是通过使用 coupled 忘记和输入门。不同于之前是分开确定什么忘记和需要添加什么新的信息，这里是使用同一个门一同做出决定。此处$i_t = 1 - f_t$.