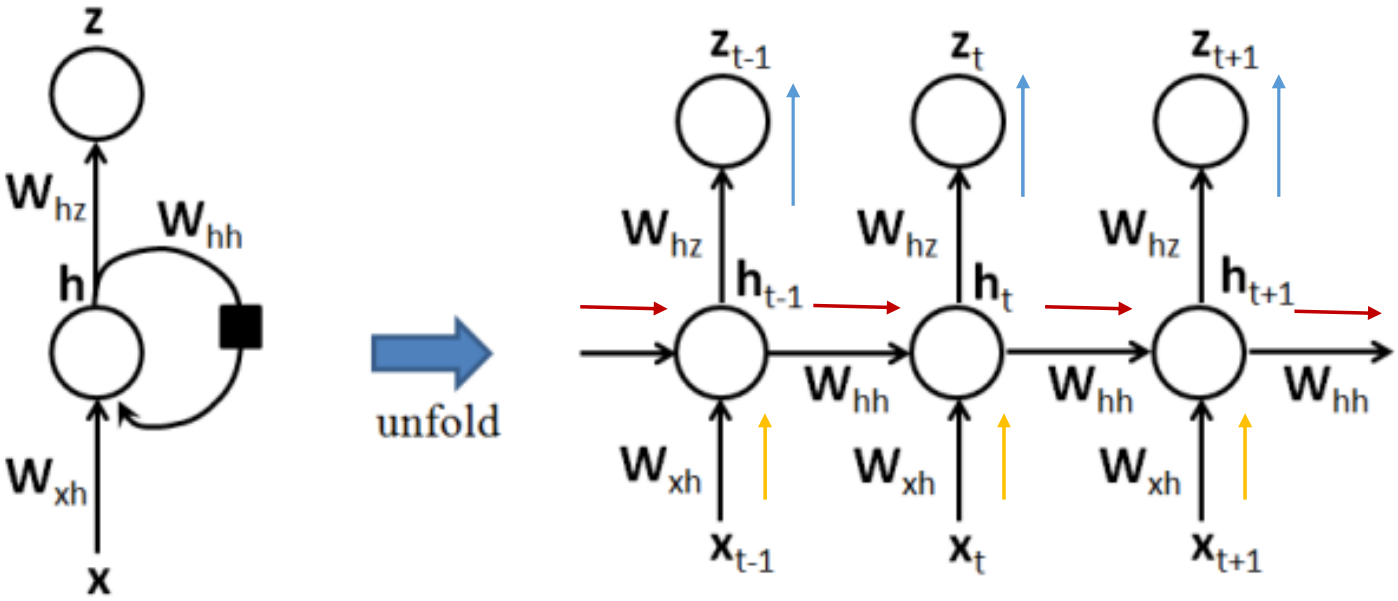
循环神经网络
Recurrent Neural Network

林立晖
2019.10.18

Recurrent Neural Network

时序扩展



$$\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$
$$\mathbf{z}_t = \mathrm{softmax}(\mathbf{W}_{hz}\mathbf{h}_t + \mathbf{b}_z)$$

Recurrent Neural Network

RNN是一类**扩展**的人工神经网络，它是为了对序列数据进行建模而产生的。

- 针对对象：序列数据。例如文本，是字母和词汇的序列；语音，是音节的序列；视频，是图像的序列；气象观测数据，股票交易数据等等，也都是序列数据。

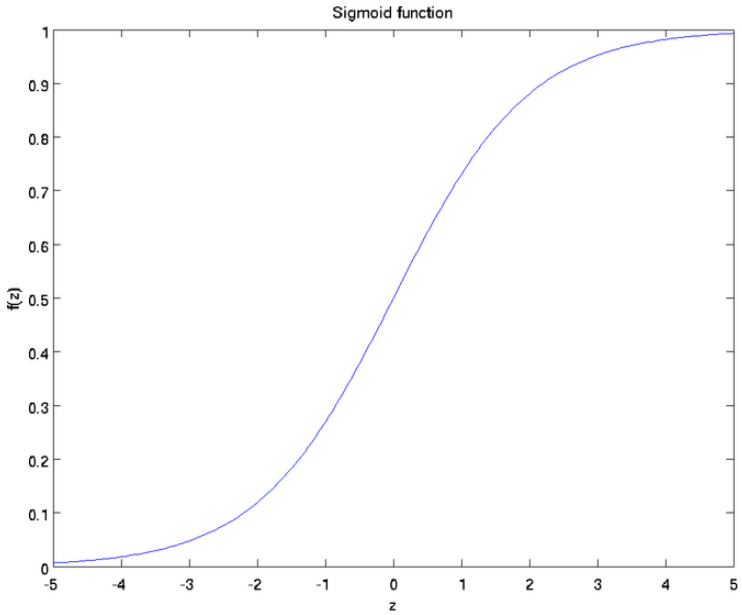
- 核心思想：样本间存在顺序关系，每个样本和它之前的样本存在关联。通过神经网络在时序上的展开，我们能够找到样本之间的序列相关性。

$$h_t = \mathcal{H}(W_{ih}x_t + W_{hh}h_{t-1} + b_h)$$

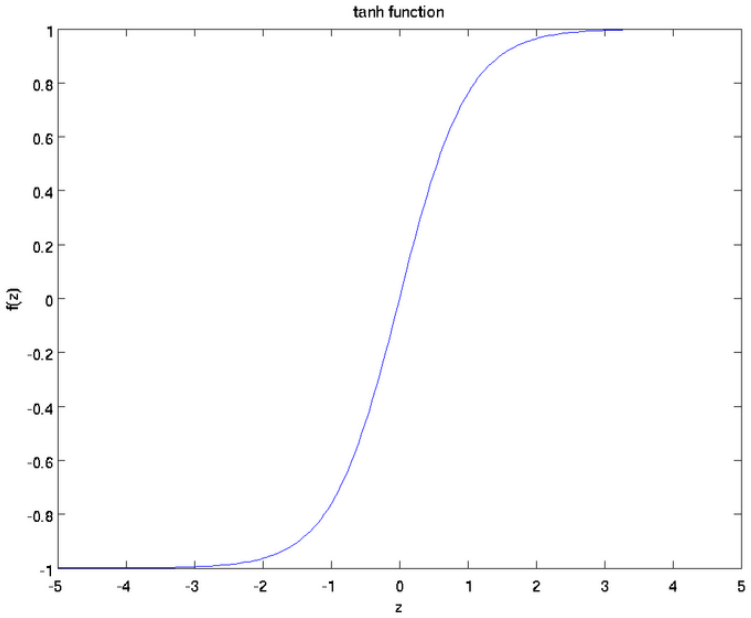
Recurrent Neural Network

activation

- RNN常用的激活函数是tanh和sigmoid。



$$f(z) = \frac{1}{1 + \exp(-z)}$$



$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Recurrent Neural Network

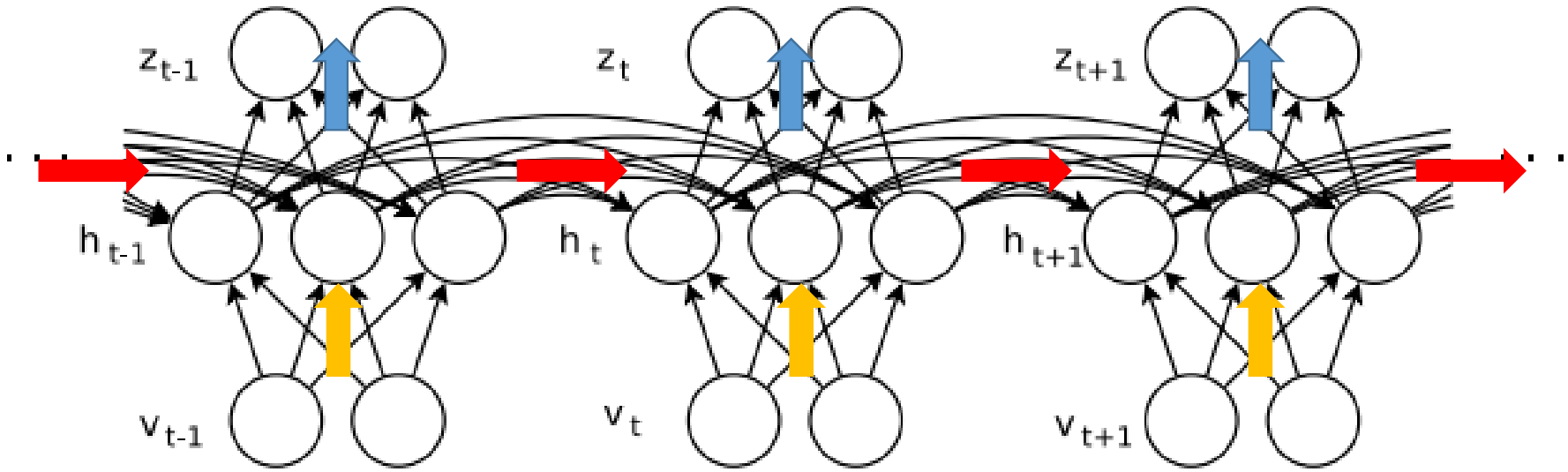
softmax

- Softmax函数是sigmoid函数的一个变种，通常我们将其用在多分类任务的输出层，将输入转化成标签的概率。

$$h_\theta\left(x^{(i)}\right) = \begin{bmatrix} p\left(y^{(i)}=1|x^{(i)};\theta\right) \\ p\left(y^{(i)}=2|x^{(i)};\theta\right) \\ \vdots \\ p\left(y^{(i)}=k|x^{(i)};\theta\right) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Recurrent Neural Network

Basic Structure



神经元在时域上**共享权重**

Recurrent Neural Network

使用BP训练RNN Backpropagation through time (BPTT)

BP回顾：通过链式法则求损失函数 E 对网络权重的偏导，沿梯度的反方向更新权重。

RNN使用的BPTT就是BP加入了时序演化过程。定义权重矩阵 U, V, W

每个时间步的网络隐藏状态和输出如下：

$$h_t = \tanh(Ux_t + Wh_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vh_t)$$

时序的总损失函数:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$
$$= -\sum_t y_t \log \hat{y}_t$$

由于是将整个序列作为一个样本，所以需要对每个时刻的误差进行求和。

Recurrent Neural Network

时序反向传播

目前的任务是求 E 对 U, V, W 的梯度。

E对$\boldsymbol{W}$在所有时刻的总梯度(其余同理):

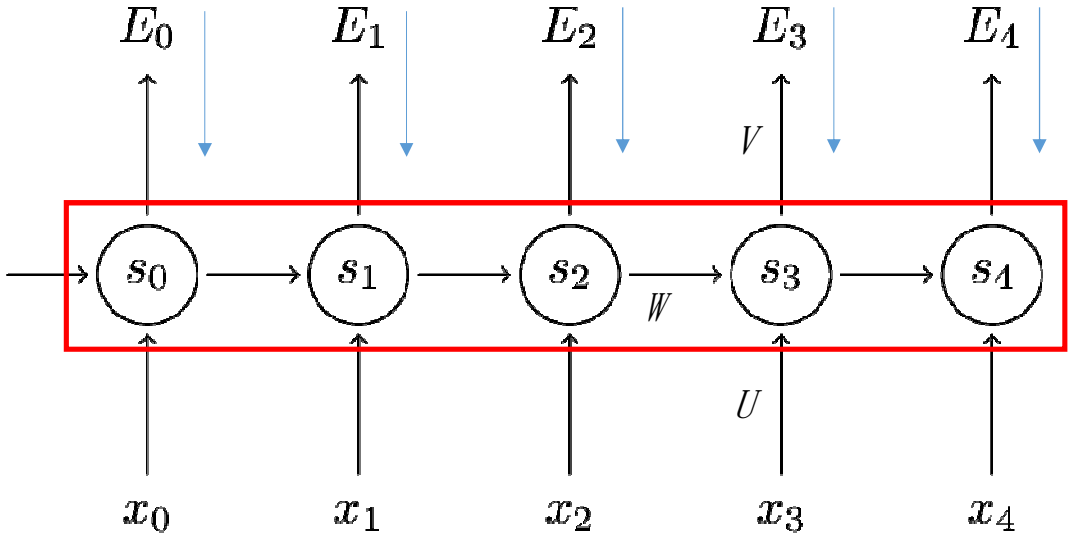
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

- 求$\boldsymbol{E}$对$\boldsymbol{V}$(隐状态到预测值)梯度

E_3对$\boldsymbol{V}$的梯度:

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V}$$

其中 $z_3 = V s_3$



对所有时刻重复左式求和可得总梯度 $\frac{\partial E}{\partial V}$

Recurrent Neural Network

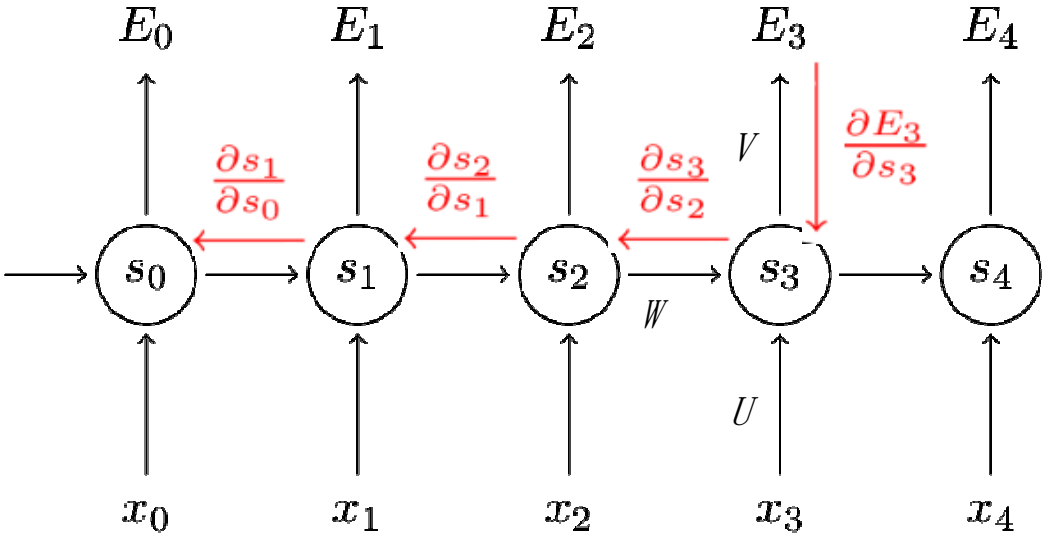
时序反向传播

- 求E对W(上一时刻隐状态的映射权重)的梯度。

E_3 对W的梯度:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

注意到: $s_3 = \tanh(Ux_t + Ws_2)$



其中: s_3 依赖于s_2，而 s_2 又依赖于 s_1 和W ，依赖关系一直传递到 t = 0 的时刻。**因此，当我们计算 s_t 对W的偏导时，不能把 s_2看作是常数项**

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{t=k+1}^{3}\frac{\partial s_t}{\partial s_{t-1}}\right)\frac{\partial s_k}{\partial W}$$

Recurrent Neural Network

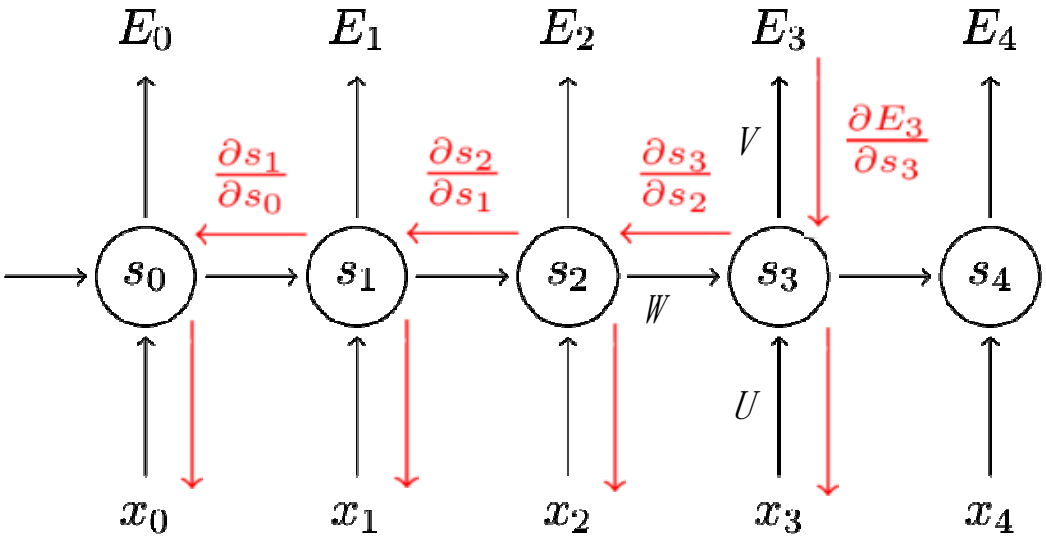
时序反向传播

- 求E对U(输入的映射权重)的梯度。

E_3 对U的梯度:

$$\frac{\partial E_3}{\partial U} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial U}$$

同样: $s_3 = \tanh(Ux_t + Ws_2)$



$$\frac{\partial E_3}{\partial U} = \sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{t=k+1}^{3}\frac{\partial s_t}{\partial s_{t-1}}\right)\frac{\partial s_k}{\partial U}$$

Recurrent Neural Network

时序反向传播

for t **from** T **downto** 1 **do**

$do_t \leftarrow g'(o_t) \cdot dL(z_t; y_t)/dz_t$

$db_o \leftarrow db_o + do_t$

$dW_{oh} \leftarrow dW_{oh} + do_t h_t^\top$

$dh_t \leftarrow dh_t + W_{oh}^\top do_t$

$dz_t \leftarrow e'(z_t) \cdot dh_t$

$dW_{hv} \leftarrow dW_{hv} + dz_t v_t^\top$

$db_h \leftarrow db_h + dz_t$

$dW_{hh} \leftarrow dW_{hh} + dz_t h_{t-1}^\top$

$dh_{t-1} \leftarrow W_{hh}^\top dz_t$

end for

Return $d\theta = [dW_{hv}, dW_{hh}, dW_{oh}, db_h, db_o, dh_0]$.

W_{hv}：输入层到当前隐藏层的权重参数

W_{hh}：上一时刻隐藏层到当前隐藏层的权重参数

W_{oh}：当前隐藏层到输出层的权重参数

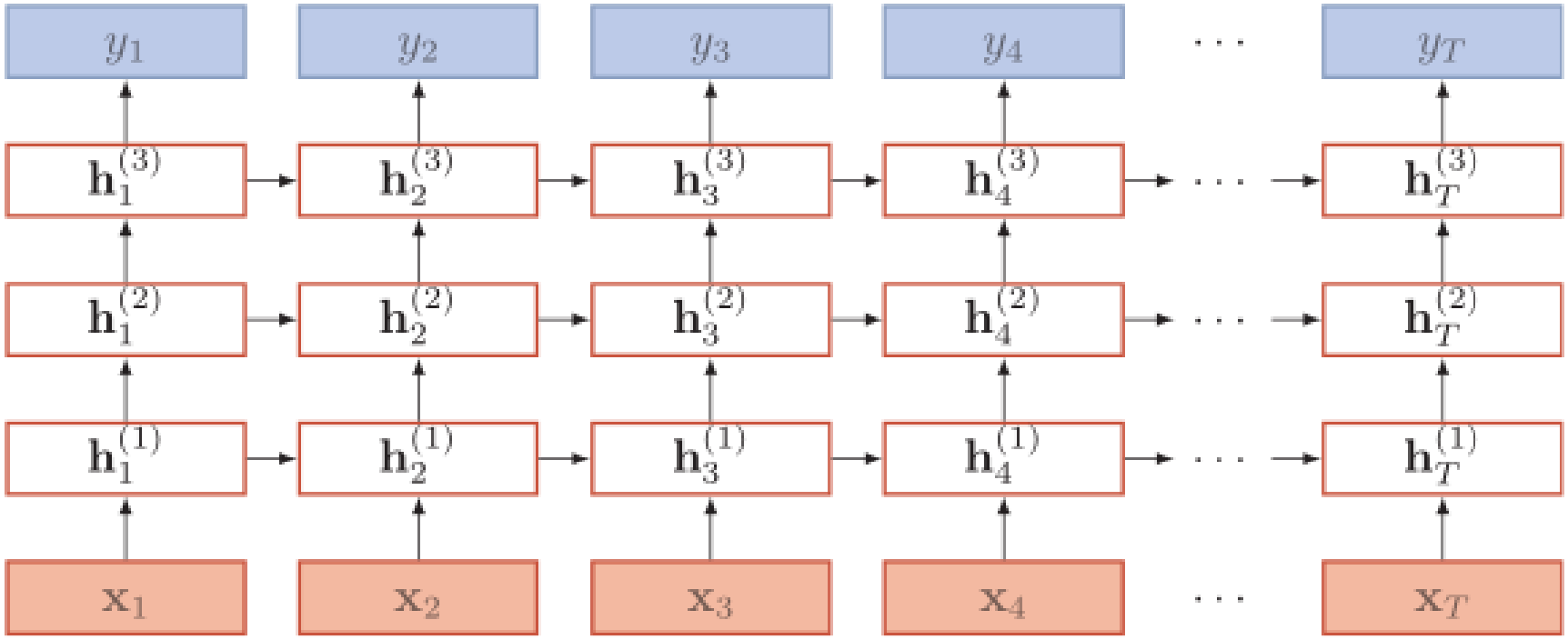
b_h ：当前隐藏层的偏置项

b_o ：输出层的偏置项

H_0 ：零时刻隐藏层的输出

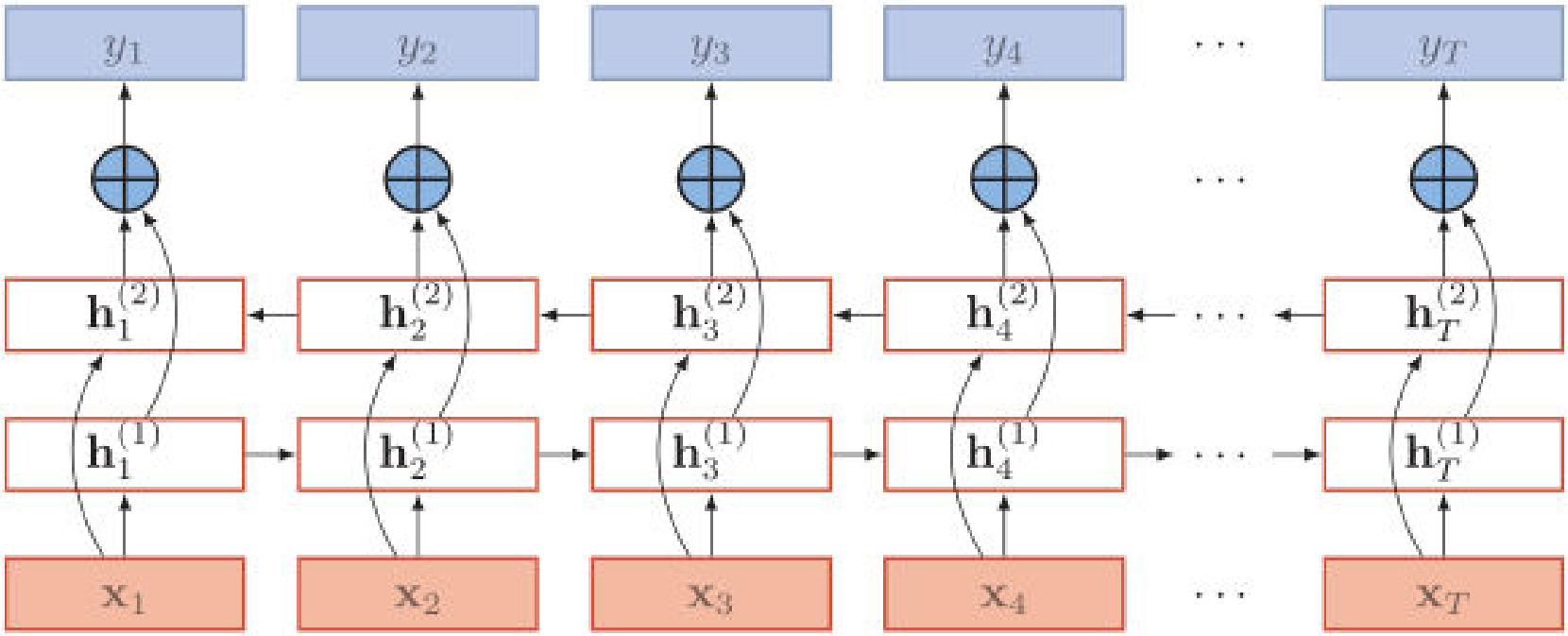
Recurrent Neural Network

堆叠RNN（Stacked RNN）



Recurrent Neural Network

双向RNN（Bidirectional RNN）

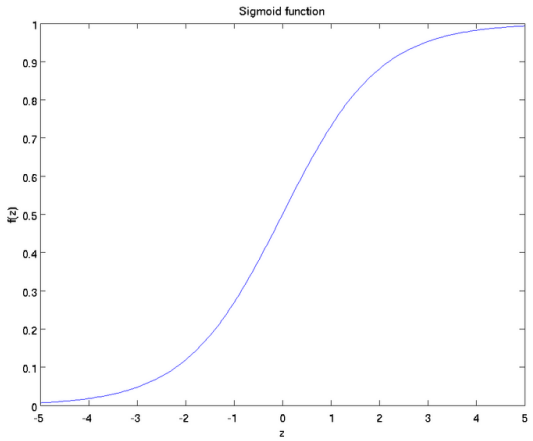


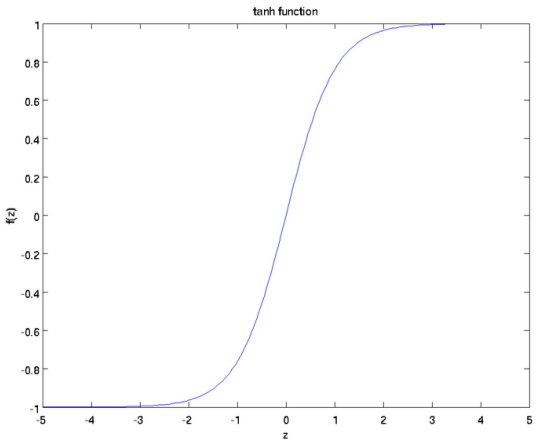
Recurrent Neural Network

时序反向传播

存在的问题：

- 梯度消失
- 梯度爆炸





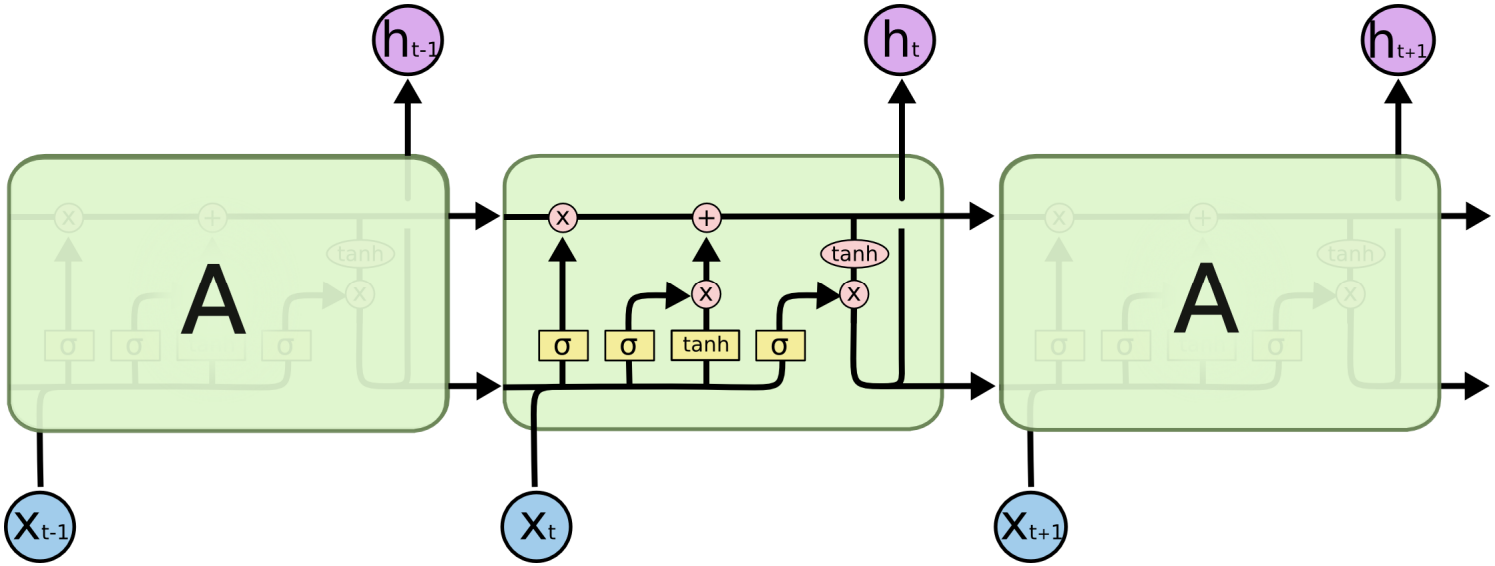
思考：这两个问题分别对应什么现象？可以从BPTT公式中的哪个地方看出来？

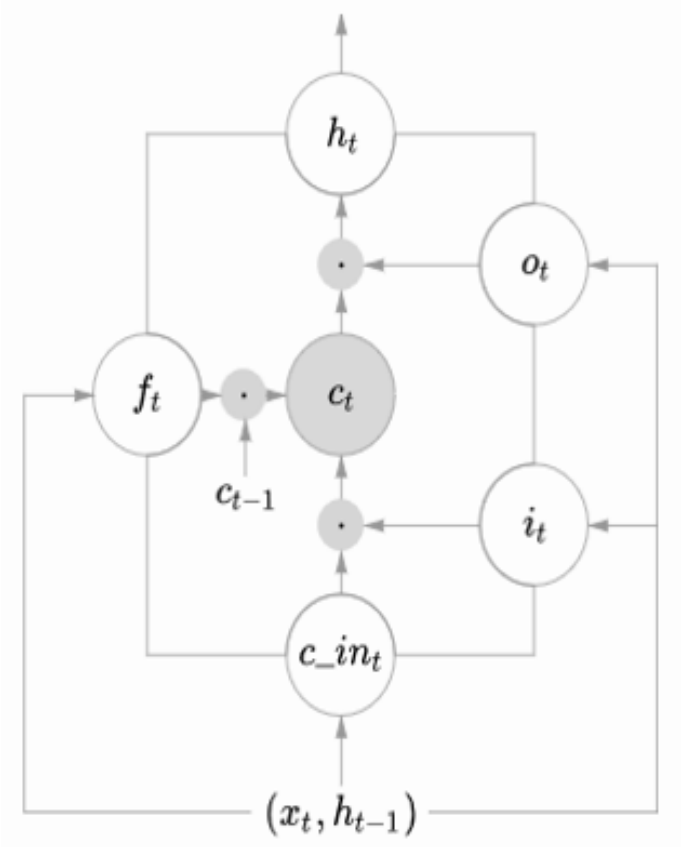
解决方案：

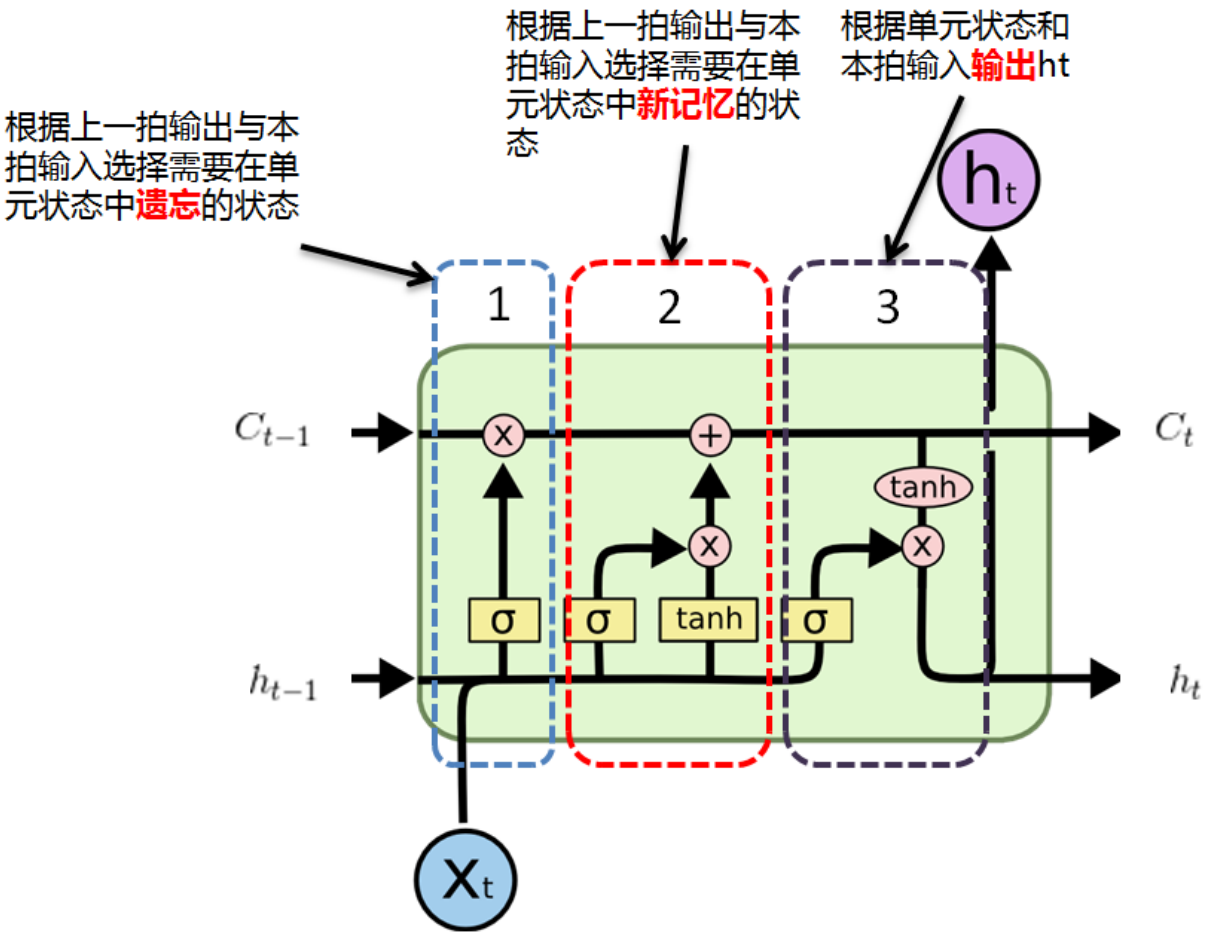
- 选择其他的激活函数。例如使用ReLU缓解梯度消失。
- 引入改进网络结构的机制，例如LSTM，GRU。

LSTM

长短期记忆网络，即Long Short-Term Memory Network，核心思想是引入“门”机制，没有像RNN一样直接使用多层感知机作为基本单元，并在一定程度上解决了梯度问题。







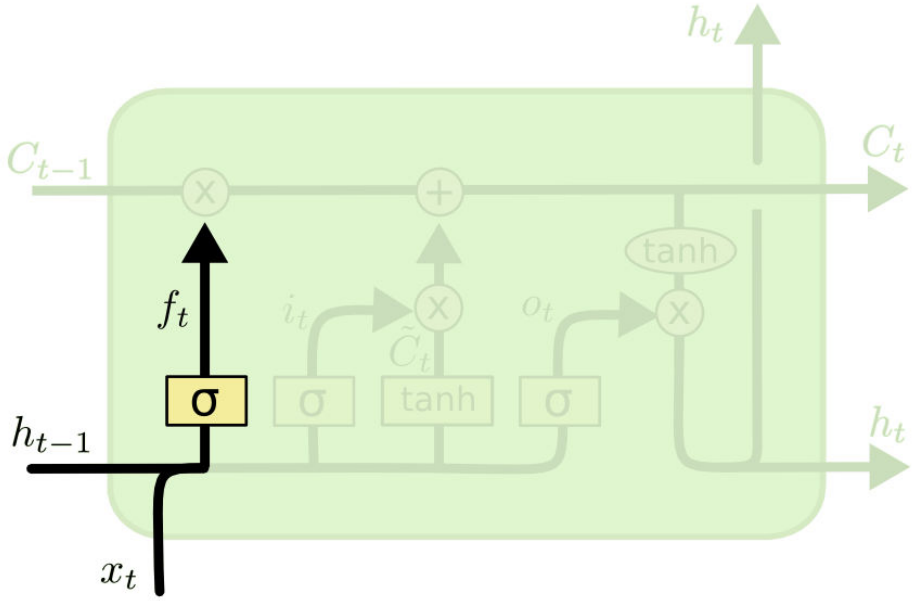
C_{t-1} C_t 单元状态 CellState

h_{t-1} h_t 单元输出

x_t 单元输入

Forget Gate

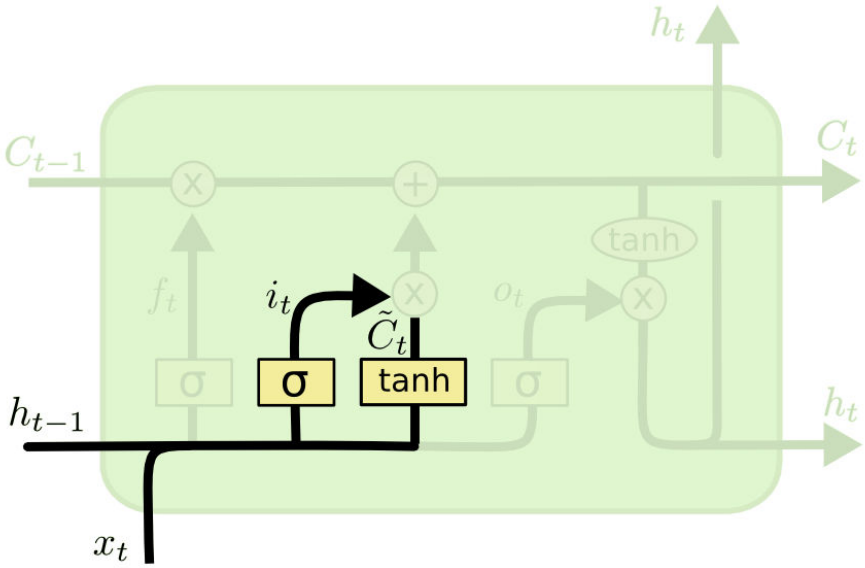
遗忘门的做法简单来说就是将当前时刻的输入通过激活函数，在每个位置映射成一个0到1的数，然后和LSTM用于保存历史信息的“细胞状态/cell state”做点乘，控制每个位置的保留程度。



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

Input Gate

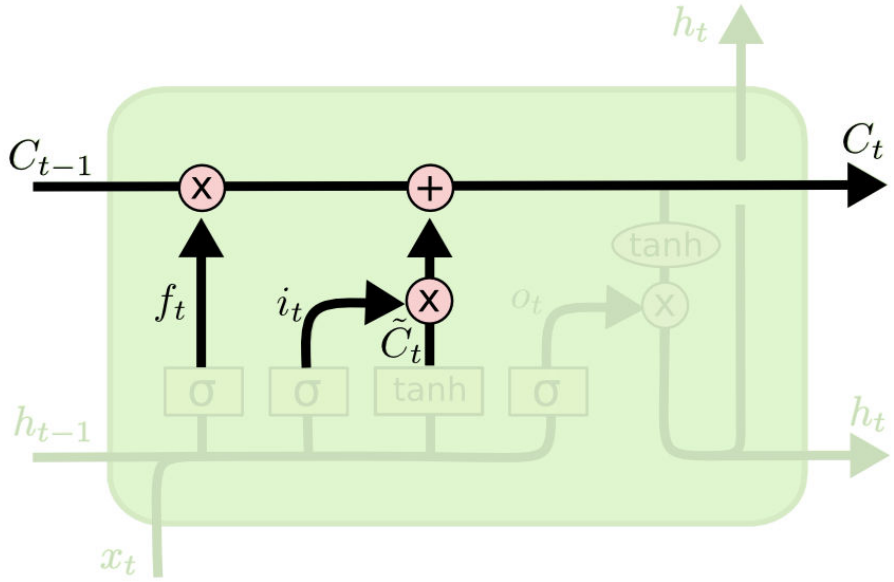
输入门同理，先将当前输入经过两个激活函数，然后点乘，表示当前产生的隐状态有多少需要被保留并加入到历史信息中。



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

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

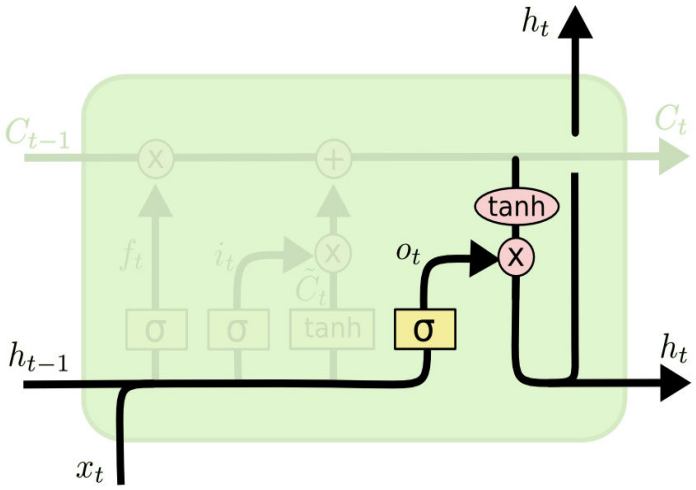
Cell State Update



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

Output Gate

输出门原理依旧相同，当前输入经过激活函数，和更新后的cell state点乘得到最终输出。



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

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

LSTM Overview

LSTM的训练方法依旧是BPTT，但是参数比一般的RNN要多。

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

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

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

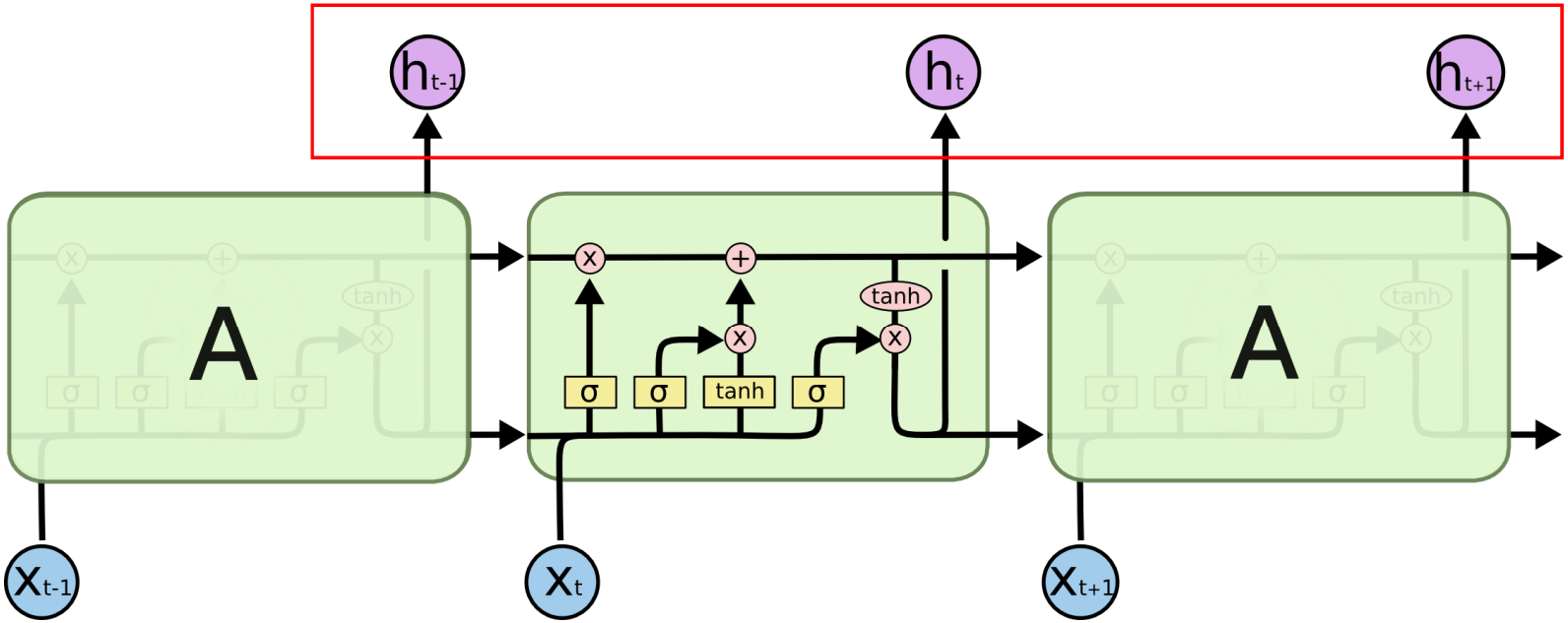
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

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

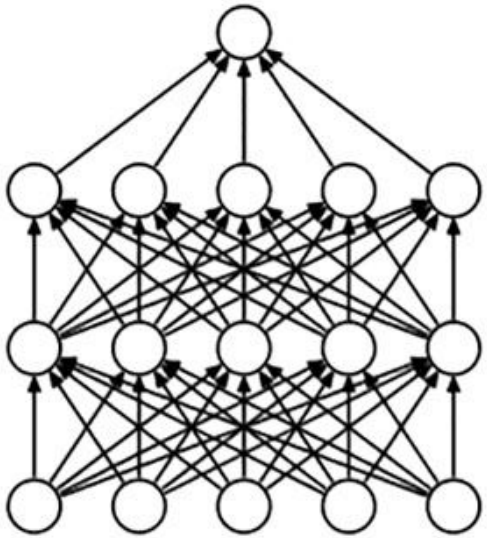
RNN for Practical Use

在tensorflow的RNN结构中，最终可访问的输出包括两部分，一是最后一个时刻的隐状态，二是每个时刻的隐状态。而LSTM则多了一个cell state。在实际中需要根据任务来选择使用RNN的最终隐状态（序列分类）或是所有时刻隐状态（序列生成）。

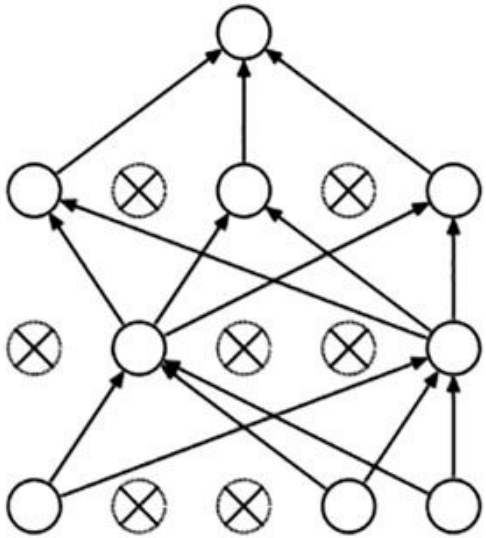


RNN for Practical Use

RNN（和其他深度学习模型）参数众多，非常容易过拟合，需要有效的正则化方法。其中一种简单且有效的技术就是Dropout。



(a) Standard Neural Net

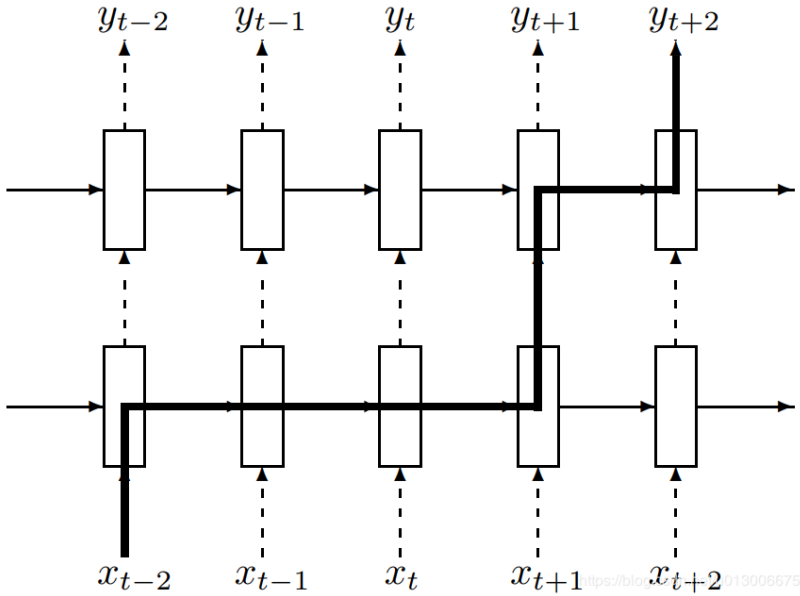


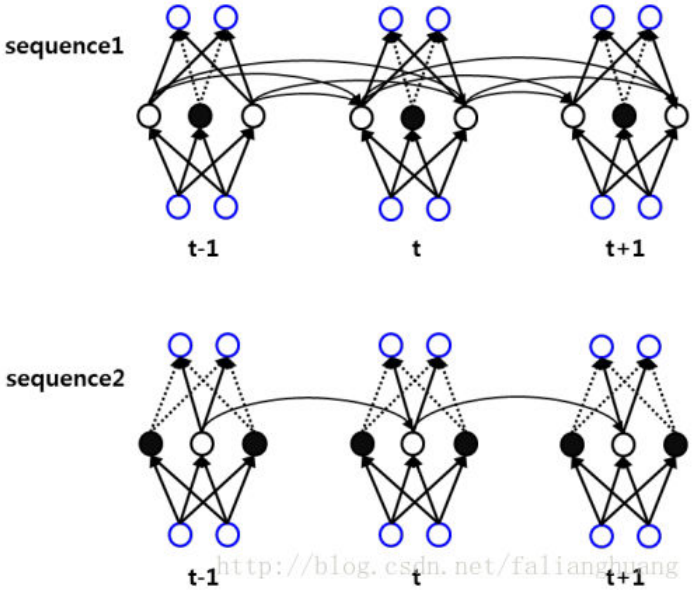
(b) After applying dropout.

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{l} + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

RNN for Practical Use

RNN中的Dropout不直接对网络结构进行更改，因为RNN属于自回归模型，会放大噪声，干扰自己的学习，而Dropout等正则化方法实际上等价于引入大量噪声。
Dropout一定要设置在网络的非循环部分，否则信息将会因循环步的进行而逐渐丢失。如果将Dropout设置在隐状态上(下图实线位置)，那么每经过一次循环，信息就被做随机丢弃一部分。对于长序列，循环到最后时信息已经所剩无几；反之，我们把Dropout设置在输入神经元上(下图虚线位置)，那么因Dropout造成的信息损失则与循环的次数（时间步）无关，仅仅与RNN单元的网络层数相关。

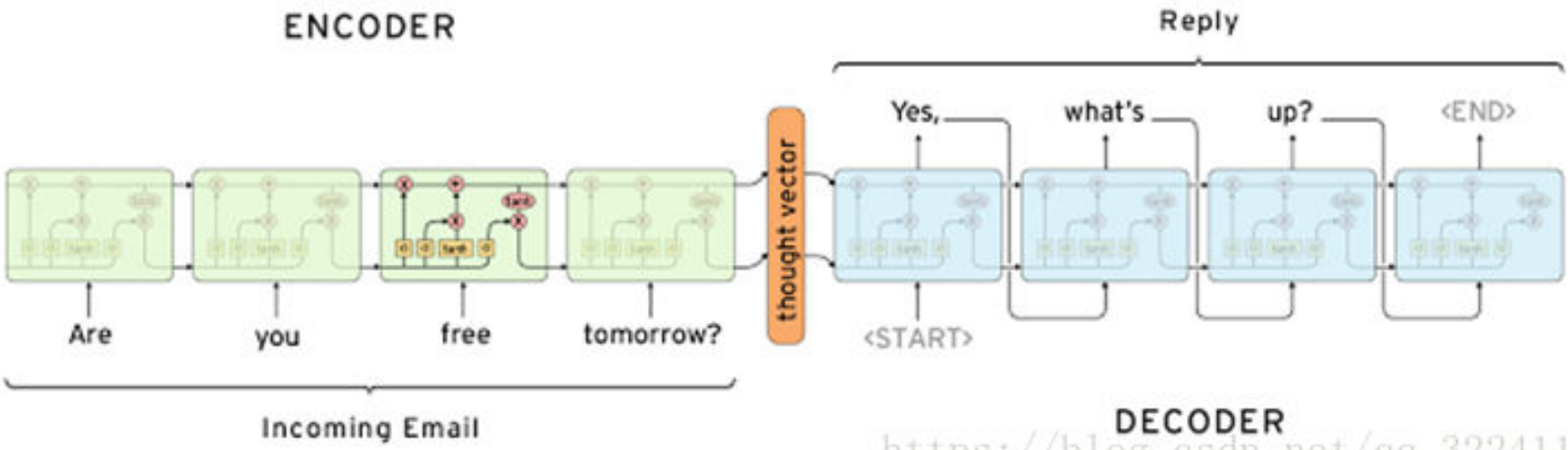




RNN for Practical Use

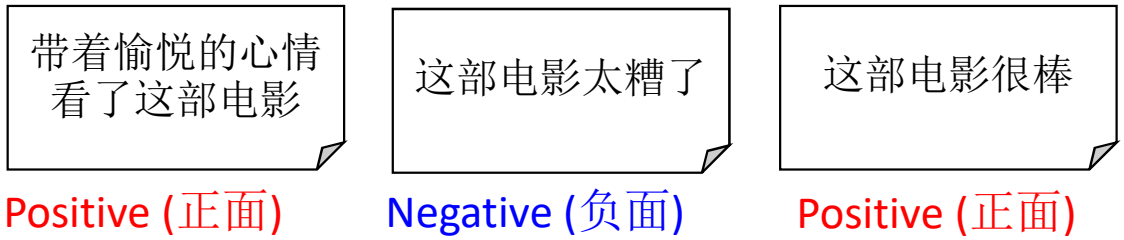
RNN在编码器-解码器（Encoder-Decoder）结构中的典型应用：序列到序列（seq2seq）模型

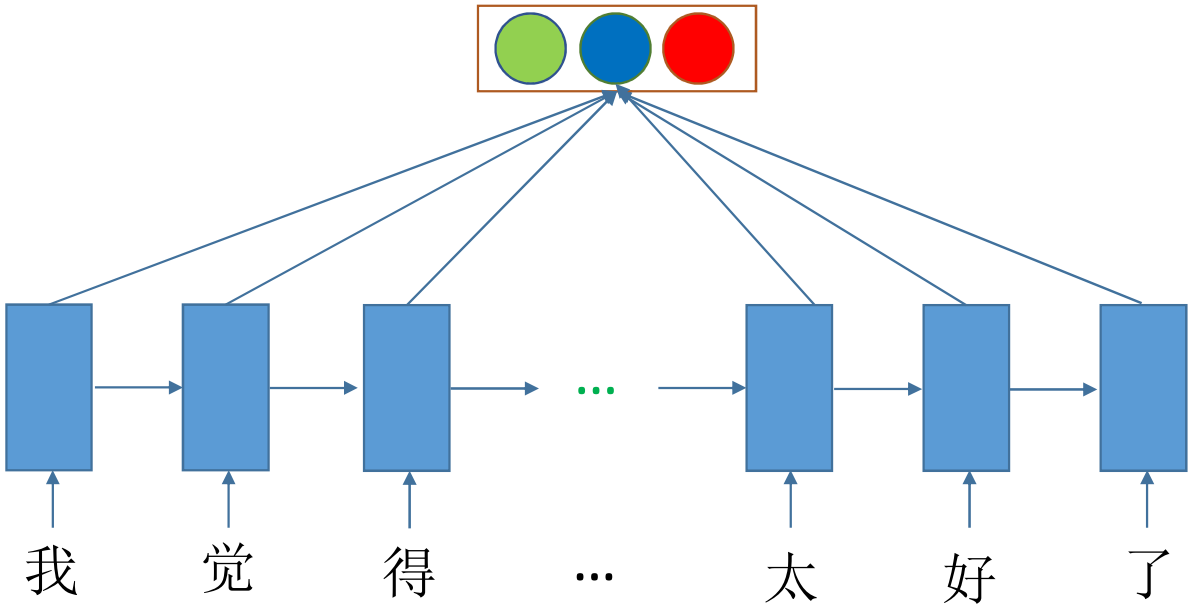
应用场景：机器翻译，文本摘要，对话生成，关键词/短语生成



RNN for Practical Use

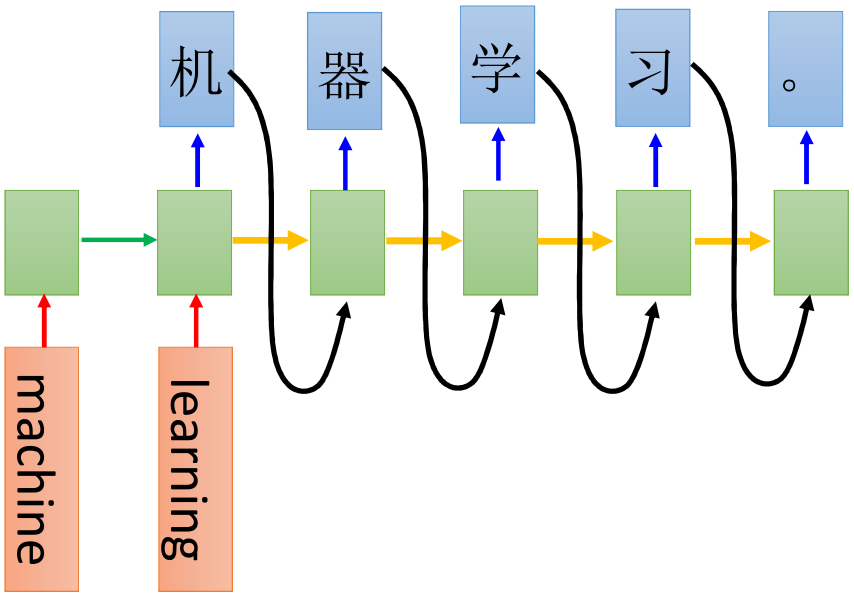
RNN做情感分类

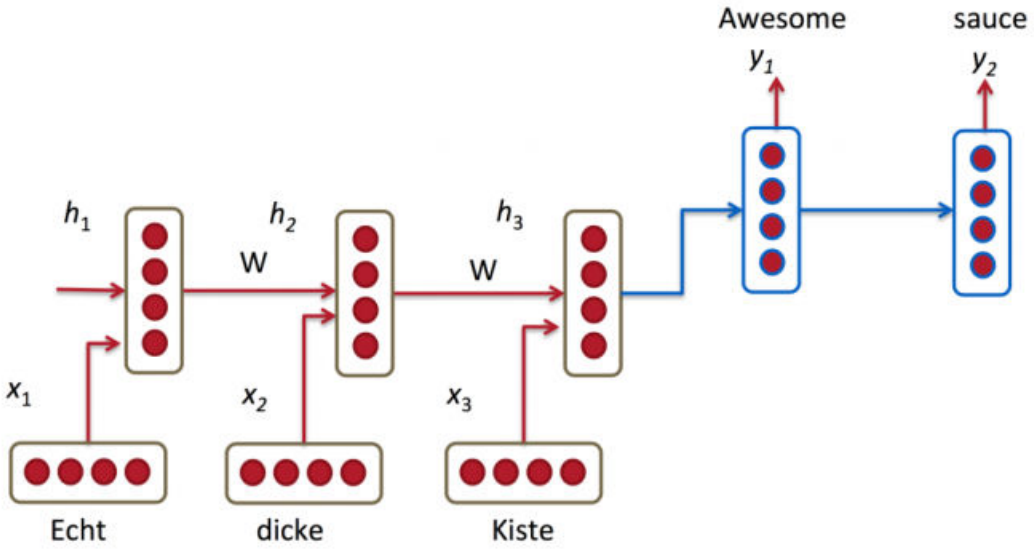




RNN for Practical Use

RNN做机器翻译





RNN for Practical Use

RNN做语音识别